A Conformant Planner Based On Approximation: CPA(H)

Dang-Vien Tran, Hoang-Khoi Nguyen, Tran Cao Son, Enrico Pontelli
Department of Computer Science
New Mexico State University
vtran|knguyen|tson|epontell@cs.nmsu.edu

April 18, 2010

Abstract

In this paper, we descibe the planner CPA(H), the recipient of the *Best Non-Observable Non-Deterministic Planner Award* in the "Uncertainty Part" of the 6th International Planning Competition (IPC) 2008. In particular, we present the various techniques that help CPA(H) to achieve the level of performance and scalability exhibiting in the competition. We also present experimental results indicating that the proposed techniques could be useful in other implementations. Furthermore, we provide possible explanations for the performance of CPA(H) in various domains and in comparison with `t0` the winner of the 2006 IPC.

1 Introduction and Motivation

Conformant planning is the problem of finding a sequence of actions that achieves the goal from every possible initial state of the world [18]. One of the main difficulties encountered in the process of determining a conformant plan is the high degree of uncertainty, due to the potentially large number of possible initial states of the problems.

The *Planning Domain Definition Language (PDDL)* introduces two constructs to express incomplete knowledge about the initial state of the world: *mutual-exclusion* statements (expressed using `one-of` clauses) and *disjunctive* statements (expressed using `or` clauses). Frequently, `one-of` clauses are used to specify the possible initial states and `or` clauses are used to eliminate infeasible states. Because of this, the number of possible initial states depends mainly on the number and the size of the `one-of` clauses—and these are often exponential in the number of constants present in the problem instances. For example, several domains in the 2006 and 2008 planning competition have this property (Table 1).

Effective methodologies and data structures are required to deal with the large number of possible initial states. Some conformant planners, such as POND [6] and KACMBP [8], employ an OBDD representation [5] of belief states, while others, such as CFF [4], adopt a CNF representation. These types of encodings avoid dealing directly with the exponential number of states, but they require extra work in determining the truth value of certain propositions after the execution of a sequence of actions in the initial belief state. For instance, CFF needs to make a call to a SAT-solver with the initial state and the sequence of actions; other planners need to recompute the OBDD representation, which could also be an expensive operation. Observe that the problem of determining the truth value of a proposition after the execution of a single action in a belief state is co-NP complete [1].

An alternative approach to deal with the large number of possible initial states is used by the planners `cf2cs(ff)` and CPA [13, 22], and further investigated in their successors `t0` and CPA+ [15, 20]. This

Instance	#Constants	# Initial States	Instance	# Constants	# Initial States
comm-10	25	2^{11}	coins-15	16	4×8^6
comm-15	35	2^{16}	coins-20	17	9×8^6
comm-20	85	2^{21}	coins-25	39	10^{20}
comm-25	140	2^{26}	coins-30	45	10^{25}
sortnet-10	11	2^{12}	sortnet-15	16	2^{16}
UTS-cycle-03	5	3×2^3	forest-02	20	2^2
UTS-cycle-04	6	4×2^4	forest-03	22	3^2
UTS-cycle-05	7	5×2^5	forest-04	24	4^2
UTS-cycle-10	12	10×2^{10}	forest-05	26	5^2
Raos-key-02	7	2^9	dispose-4-1	17	16
Raos-key-03	9	2^{13}	dispose-4-2	18	16^2
Raos-key-04	11	2^{17}	dispose-4-3	19	16^3
Raos-key-05	13	2^{21}	dispose-8-1	65	64
Raos-key-10	23	2^{41}	dispose-8-2	66	64^2
Raos-key-15	35	2^{61}	dispose-8-3	67	64^3
push-4-1	17	16	1-dispose-4-1	17	16
push-4-2	18	16^2	1-dispose-4-2	18	16^2
push-4-3	19	16^3	1-dispose-4-3	19	16^3
push-8-1	65	64	1-dispose-8-1	65	64
push-8-2	66	64^2	1-dispose-8-2	66	64^2
push-8-3	67	64^3	1-dispose-8-3	67	64^3

Table 1: Number of Constants/Possible Initial States

approach relies on an *approximation semantics* in reasoning with incomplete information [19]. The planners `cf2cs(ff)` and `t0` reduce the number of possible initial states to one by introducing additional propositions, transforming the original problem to a classical planning problem, and using FF, a classical planner [9], to find solutions. On the other hand, CPA and CPA+ reduce this number by dividing them into groups and using the intersection of each group as its representative during planning.

CPA+ and `t0` implement the idea of approximations differently. While CPA+ could be seen as a standard heuristic search forward planner, `t0` follows a translational approach. The performance of CPA+ depends on its heuristic function and its ability to approximate the initial belief state to a manageable set of partial states. On the other hand, the performance of `t0` largely depends on the performance of FF. `t0` was the winner of the 2006 planning competition.

In this paper, we describe the design and implementation of CPA(H), our entry[1] in the "Uncertainty Part" of the 6th International Planning Competition 2008 (IPPC), International Conference on Automated Planning and Scheduling (ICAPS), 2008, Sydney, Australia. We describe the basic reasoning mechanism of CPA(H) and present the various techniques novel techniques for improving efficiency and scalability. We begin with a short overview of the problem representation and the basic reasoning mechanism of the planner in Section 2.

2 Problem Representation

Following the notation in [13], we describe a *problem specification* as a tuple $P = \langle F, O, I, G \rangle$, where

- F is a set of propositions,
- O is a set of actions,

[1] Named CPA(H) to recognize its roots in the CPA+ system.

- I describes the initial state of the world and
- G describes the goal.

A *literal* is either a proposition $p \in F$ or its negation $\neg p$. $\bar{\ell}$ denotes the complement of a literal ℓ and is defined by $\bar{\ell} = \neg\ell$ where $\neg\neg p = p$ for $p \in F$. We say that ℓ and $\bar{\ell}$ are complementary literals. For a set of literals L, $\overline{L} = \{\bar{\ell} \mid \ell \in L\}$. In this paper, we will often represent a conjunction of literals by a set.

A set of literals X is *consistent* if there exists no $p \in F$ such that $\{p, \neg p\} \subseteq X$. A *state* s is a consistent and *complete* set of literals, i.e., s is consistent, and for each $p \in F$, either $p \in s$ or $\neg p \in s$. A *belief state* is a set of states. A set of literals X satisfies a literal ℓ (resp. a set of literals Y) iff $\ell \in X$ (resp. $Y \subseteq X$).

Each action a in O is associated with a precondition ϕ (denoted by $pre(a)$) and a set of conditional effects of the form $\psi \rightarrow \ell$ (also denoted by $a : \psi \rightarrow \ell$), where ϕ and ψ are sets of literals and ℓ is a literal. We will often write $a : \psi \rightarrow \ell_1, \ldots, \ell_n$ as a shorthand for the set $\{a : \psi \rightarrow \ell_1, \ldots, a : \psi \rightarrow \ell_n\}$.

The initial state of the world I is the union of three types of formulas and can be written as $I = I^d \cup I^o \cup I^r$ where

- I^d is a set of literals;
- I^o is a set of `one-of` clauses, each `one-of` clause is of the form $\texttt{one-of}(\phi_1, \ldots, \phi_n)$;
- I^r is a set of `or` clauses, each is of the form $\texttt{or}(\phi_1, \ldots, \phi_n)$

where each ϕ_i is a set of literals.

An `one-of` clause indicates that the ϕ_i's are mutually exclusive, while an `or` clause is a disjunctive normal form (DNF) representation of a formula. A set of literals X satisfies the `one-of` clause $\texttt{one-of}(\phi_1, \ldots, \phi_n)$ if there exists some $1 \leq i \leq n$ such that $\phi_i \subseteq X$ and for every $j \neq i$, $1 \leq j \leq n$, $\overline{\phi_j} \cap X \neq \emptyset$. X satisfies the `or` clause $\texttt{or}(\phi_1, \ldots, \phi_n)$ if there exists some $1 \leq i \leq n$ such that $\phi_i \subseteq X$. Given a `one-of`-clause or an `or`-clause o, we write $L \in o$ to denote that L is an element of o and $lit(o) = \bigcup_{L \in o}(L \cup \bar{L})$.

By $ext(I)$ we denote the set of all states satisfying I^d, every `one-of` clause in I^o, and every `or` clause in I^r. For example, if F=$\{g, f\}$ and I=$\{g, \texttt{one-of}(f, \neg f)\}$ then $ext(I) = \{\{g, f\}, \{g, \neg f\}\}$.

The goal of the prboblem, G, is a collection of literals and `or` clauses.

Example 1. Let us consider the 2×2 *dispose* problem with one object o_1 from [13]. The object is on the grid. The agent can move, one step at a time, within the grid. It can pick up an object, move along with the object, and drop it where the trashcan is. Different representations can be used for this domain. In this paper, we use the encoding in the IPC-08. In this setting, the grid is encoded by a set of locations $Loc = \{l_{11}, l_{12}, l_{21}, l_{22}\}$ with a predicate $adjacent(l, l')$ to denote the fact that the two locations l and l' are adjacent, indicating that one can move between l and l'. The goal is to collect all the objects on the grid and dispose them into the trashcan at the location l_{11}. Initially, the location of the objects are unknown.

Let us denote the problem by $D[2, 2, 1]$. The encoding of $D[2, 2, 1] = \langle P, O, I, G\rangle$ is given next. In this domain, the set of propositions is [2]

$$F = \left\{ \; obj\_at(o_1, l), trash\_at(l), at(l), holding(o_1), disposed(o_1) \;\middle|\; l \in Loc \; \right\}$$

where $obj\_at(o_1, l)$ says that object o_1 is at the location l (on the grid), $trash\_at(l)$ indicates that the trashcan is at the location l, $holding(o_1)$ states that the agent holds the object o_1, $disposed(o_1)$ states that the object o_1 has been disposed, and $at(l)$ means that the agent is at the location l.

[2] For simplicity, we omit the predicate $adjacent(l, l')$.

The set of actions with their conditional effects can be represented as follows:

$$O = \left\{ \begin{array}{ll} move(l_1, l_2): & true \rightarrow at(l_2), \neg at(l_1) \\ pickup(o_1, l): & obj\_at(o_1, l) \rightarrow holding(o_1), \neg obj\_at(o_1, l) \\ drop(o_1, l): & holding(o_1) \rightarrow disposed(o_1), \neg holding(o_1) \end{array} \right\}$$

where $l_1, l_2, l \in Loc$, $l_1 \neq l_2$. In addition, we have that

$$\begin{array}{rl} pre(move(l_1, l_2)) & = \{at(l_1)\} \\ pre(pickup(o_1, l)) & = \{at(l)\} \\ pre(drop(o_1, l)) & = \{at(l), trash\_at(l)\} \end{array}$$

The initial state of the problem can be given by $I = I^d \cup I^o$ where

$$I^d = \{trash\_at(l_{11}), at(l_{11}), \neg holding(o_1)\} \cup \{\neg trash\_at(l), \neg at(l) \mid l \in Loc \setminus \{l_{11}\}\}$$

and

$$I^o = \{\texttt{one-of}(obj\_at(o_1, l_{11}), obj\_at(o_1, l_{12}), obj\_at(o_1, l_{21}), obj\_at(o_1, l_{22}))\}.$$

Finally, the goal of the problem is given by

$$G = \{disposed(o_1)\}.$$

2.1 Conformant Planning

Given a state s and an action a, a is executable in s if $pre(a) \subseteq s$. The set of effects of a in s, denoted by $e_a(s)$, is defined by:

$$e_a(s) = \{l \mid \psi \rightarrow l \text{ is an effect of } a, \psi \subseteq s\}.$$

The execution of a in a state s results in a successor state $succ(a, s)$ which is defined by:

$$succ(a, s) = \left\{ \begin{array}{ll} s \cup e_a(s) \setminus \overline{e_a(s)} & \text{if } a \text{ is executable in } s \\ \\ failed & \text{otherwise} \end{array} \right.$$

$succ$ is extended to define $succ^*$, which computes the result of the execution of an action in a belief state, as follows:

$$succ^*(a, S) = \left\{ \begin{array}{ll} \{succ(a, s) \mid s \in S\} & \text{if } a \text{ is executable in every } s \in S \\ \\ failed & \text{otherwise} \end{array} \right. \quad (1)$$

Finally, we can define the function $\widehat{succ}$ to compute the final belief state resulting from the execution of a plan:

$$\widehat{succ}([a_1, \ldots, a_n], S) = \left\{ \begin{array}{lll} S & \text{if} & n = 0 \\ succ^*(a_n, \widehat{succ}([a_1, \ldots, a_{n-1}], S)) & \text{if} & n > 0 \end{array} \right.$$

An action sequence α is a *solution* of P iff $\widehat{succ}(\alpha, S_0) \neq failed$ and G is satisfied in every state belonging to $\widehat{succ}(\alpha, S_0)$.

Example 2. Consider the planning problem in Example 1, assuming that $adjacent(l_{11}, l_{12})$, $adjacent(l_{11}, l_{21})$, $adjacent(l_{21}, l_{22})$, and $adjacent(l_{22}, l_{12})$ are true and the relation $adjacent$ is symmetric. We can easily check that the sequence of action

$$\alpha = \left[\begin{array}{l} pick(o_1, l_{11}), drop(o_1, l_{11}), \\ move(l_{11}, l_{12}), pick(o_1, l_{12}), move(l_{12}, l_{11}), drop(o_1, l_{11}), \\ move(l_{11}, l_{12}), move(l_{12}, l_{22}), pick(o_1, l_{22}), move(l_{22}, l_{12}), move(l_{12}, l_{11}), drop(o_1, l_{11}), \\ move(l_{11}, l_{21}), pick(o_1, l_{21}), move(l_{21}, l_{11}), drop(o_1, l_{11}) \end{array} \right]$$

would achieve the goal $disposed(o_1)$ from $ext(I)$, i.e., α is a solution for the planning problem $D[2, 2, 1]$.

2.2 Conformant Planning Using Approximation

As we have mentioned earlier, the size of the initial belief state, i.e., the number of states in $ext(I)$ provides a challenge for conformant planners. In [22], a new approach to conformant planning is proposed. This approach extends on the notion of approximation proposed in [19] to languages with state constraints. The original approximation proposes to approximate a belief state by a set of literals that are true in all the states belonging to it and develops a transition function to reason about the effects of actions on approximation states. The advantage of this approach lies in that conformant planning using approximation is NP-complete while it is Σ_P^2 with respect to the complete semantics [1]. The trade-off is its incompleteness. This issue has been addressed in [20].

Formally, we refer to a consistent set of literals as a *partial state*. A set of partial states is called a *cs-state*. For a partial state δ, by $ext(\delta)$ we denote the belief state $\{s \mid \delta \subseteq s\}$. Intuitively, a partial state δ approximates the belief state $ext(\delta)$ and a cs-states $\{\delta_1, \ldots, \delta_n\}$ approximates the belief state $\bigcup_{i=1}^{n} ext(\delta_i)$.

The reasoning with respect to partial states is characterized by a function ($succ_A$) that maps an action and a partial state to a partial state. The *possible effects* of a in a partial state δ are given by

$$pc_a(\delta) = \{l \mid \psi \rightarrow l \text{ is an effect of } a, \overline{\psi} \cap \delta = \emptyset\}. \tag{2}$$

The successor partial state from the execution of a in δ is defined by

$$succ_A(a, \delta) = \begin{cases} (\delta \cup e_a(\delta)) \setminus \overline{pc_a(\delta)} & \text{if } a \text{ is executable in } \delta \\ \\ succ_A(a, \delta) = failed & \text{otherwise} \end{cases}$$

Similarly to $succ^*$ and $\widehat{succ}$, $succ_A$ can be extended to define $succ_A^*$ (mapping cs-states to cs-states) and $\widehat{succ_A}$ for computing the result of the execution of an action sequence starting from a cs-state as follows.

$$succ_A^*(a, \Delta) = \begin{cases} \{succ(a, \delta) \mid \delta \in \Delta\} & \text{if } a \text{ is executable in every } \delta \in \Delta \\ \\ succ_A^*(a, \Delta) = failed & \text{otherwise} \end{cases}$$

and

$$\widehat{succ_A}([a_1, \ldots, a_n], \Delta) = \begin{cases} \Delta & \text{if} \quad n = 0 \\ \\ succ_A^*(a_n, \widehat{succ_A}([a_1, \ldots, a_{n-1}], \Delta)) & \text{if} \quad n > 0 \end{cases}$$

The soundness of the approximation states that for every partial state δ, an action sequence α, and a formula φ, if φ is satisfied by $\widehat{succ_A}(\alpha, \delta)$ then φ is satisfied in $\widehat{succ}(\alpha, ext(\delta))$ [19]. This allows us to establish the following property:

Proposition 1. *Let P be a planning problem and $\{\Delta_1, \ldots, \Delta_k\}$ be a set of cs-states such that $\bigcup_{i=1}^{k} ext(\Delta_i) = ext(I)$. Then, a sequence of actions $\alpha = [a_1, \ldots, a_n]$ is a solution of a problem P if G holds in $\widehat{succ_A}(\alpha, \Delta_i)$ for every i.*

Proof. Trivially follows from the result in [19] and the fact $\bigcup_{i=1}^{k} ext(\Delta_i) = ext(I)$. □

The above proposition shows that $succ_A$ (or $\widehat{succ_A}$) can be employed in the implementation of a conformant planner, provided that we can select an appropriate set $\Omega = \{\Delta_1, \ldots, \Delta_k\}$. Because $S \subseteq ext(\bigcap_{u \in S} u)$ for every belief state S, we could use $\Omega = \{\{\delta_0\}\}$ where $\delta_0 = \cap_{s \in ext(I)} s$. This choice was used in early implementations of conformant planners using approximation [21, 22, 26]. The advantage of this choice is that the size of the cs-state is small (one), a significant reduction from $2^{|F|}$, the number of possible states of a domain. This choice, however, does not guarantee completeness due to the incompleteness of the approximation when actions have conditional effects. This can be seen in the following example.

Example 3. Given the problem $P = \langle \{f, h\}, \{a : f \to h, a : \neg f \to h\}, \emptyset, \{h\} \rangle$, we can easily check that

$$succ^*(a, ext(I)) = \{\{f, h\}, \{\neg f, h\}\}$$

and

$$succ_A^*(a, \{\emptyset\}) = \{\emptyset\}$$

The first equation indicates that a is a solution of P. On the other hand, the second one states that if the initial belief state is approximated to $\emptyset$ then a is not a plan with respect to the approximation.

To guarantee the completeness of approximation based planners and still exploit the advantages of the approximaiton, whenever it is possible, we need to identify an appropriate partitions $\Omega = \{\Delta_1, \ldots, \Delta_k\}$ of $ext(I)$ such that

- $\bigcup_{i=1}^{k} \Delta_i = ext(I)$,
- $\Delta_i \cap \Delta_j = \emptyset$ for each $i \neq j$, and
- for each formula φ and sequence of actions α, $\widehat{succ_A}(\alpha, \{\delta_1, \ldots, \delta_k\}) \models \varphi$ iff $\widehat{succ}(\alpha, ext(I)) \models \varphi$, where δ_i is the intersection of the states in Δ_i.

Observe that the partition $\Omega = \{\{s\} \mid s \in ext(I)\}$ satisfies the aforementioned conditions. This guarantees the existence of such a partition. On the other hand, it is not always necessary to consider such an extreme situation. For example, the partition $\{\{\{f, h\}, \{f, \neg h\}\}, \{\{\neg f, h\}, \{\neg f, \neg h\}\}\}$ satisfies these conditions for the problem in Example 3. Observe that this means that the approximation planner will start with the cs-states $\{\{f\}, \{\neg f\}\}$.

Research has been conducted to provide sufficient syntactical conditions to identify valid partitions—based on the identification of fluents that should be explicitly distinguished in different partitions (see, e.g., [20, 25]). The planner CPA(H) employs this technique.

3 A Competitive Conformant Planner: Design

The internal organization of the proposed planner, called CPA(H), is illustrated in Fig. 1. The planner is composed of two modules.

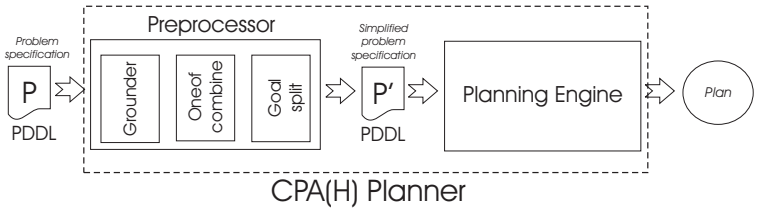

Figure 1: Internal organization of CPA(H)

The first module (*Preprocessor*) is a static analyzer that performs a number of transformations of the problem specification. Along with a grounder (which also applies standard simplifications, such as forward reachability), the preprocessor applies some novel transformations (`one-of` clause combination and goal splitting) aimed at drastically reducing the size of the search space. The second module (*Planning engine*) is a heuristic search engine implementing forward planning. The next subsections present the design of these modules.

3.1 The Planning Engine

CPA(H) employs the $succ^*_A$ function in the context of a planning algorithm which implements forward planning using a traditional best-first heuristic search. For the completeness of the paper, the algorithm is included (Algorithm 1).

Algorithm 1 FWDPLAN(P)

Require: a planning problem $P = \langle P, O, I, G \rangle$
Ensure: a solution of P if P is solvable or FAILURE otherwise

```
S := Make_Initial(O, I, G)                          {Algorithm on Page 47 [25]}
if G is true in S then
   return [ ]
end if
Enqueue(PriorityQueue, (S, [ ]))
VisitedStates := {S}
while PriorityQueue is not empty do
   (S′,path) := Extract(PriorityQueue)              {element with best heuristic value}
   for each action a executable in S′ do
      S″ := succ*_A(a, S′)
      if S″ satisfies G then
         return [path, a]
      else
         if S″ ∉ VisitedStates then
            compute heuristic for S″
            Enqueue(PriorityQueue, (S″, [path, a]))
            VisitedStates := VisitedStates ∪ {S″}
         end if
      end if
   end for
end while
return FAILURE
```

CPA(H) computes the following basic heuristics:

- *The cardinality heuristic:* Since the initial cs-state could potentially be the same as the initial belief state $ext(I)$, its cardinality can be used to guide the search. We prefer cs-states that have a smaller cardinality. In other words, $h_{card}(S) = |S|$ where S is a cs-state. Note that we use this heuristic in a forward fashion, and this is different from its use in [2, 7]. The intuition behind this is that planning with complete information is "easier" than planning with incomplete information and a lower cardinality implies a lower degree of uncertainty.
- *The relaxed graphplan heuristic:* For a cs-state S, we define $h_{rgp}(S) = \sum_{\delta \in S} d(\delta)$, where $d(\delta)$ is the well-known sum heuristic value given that the initial state is $\delta \cup \{\neg p \mid p \in F,\ p \notin \delta,\ \neg p \notin \delta\}$ [12].
- *The number of satisfied subgoals:* This heuristic counts the number of satisfied subgoals and is denoted by $h_{gc}(S)$. Formally, $h_{gc}(S) = \sum_{\delta \in S} sat(\delta)$ where $sat(\delta)$ is the number of subgoals satisfied by δ.

CPA(H) implements a combination of these heuristics:

$$h_{css}(\Sigma) = (h_{card}(\Sigma), h_{gc}(\Sigma), h_{rgp}(\Sigma)).$$

The measures are compared according to their lexicographic order.

3.2 Design of the Preprocessor

The main goal of the preprocessor is to simplify the planning problem using a variety of techniques. This section presents the basic definitions underlying these techniques. Among them, some are standard (e.g., reachability analysis); others are specific to CPA(H). The key to the analysis in the preprocessor is the notion of *dependence* between actions and propositions—similar to the notion of dependence between actions and literals explored in [20]. In what follows, we denote with P an arbitrary but fixed planning problem.

Definition 1. *An action a* depends *on a literal ℓ if*

1. *$\ell \in pre(a)$, or*
2. *there exists an effect $a : \phi \rightarrow h$ in P and $\ell \in \phi$, or*
3. *there exits an action b that depends on ℓ and a depends on some of the effects of b, i.e., b depends on ℓ and there exists $b : \phi \rightarrow h$ such that a depends on h.*

By $preact(\ell)$ we denote the set of actions depending on ℓ. For a set of literals L, we define $preact(L) = \bigcup_{\ell \in L} preact(\ell)$. Intuitively, the fact that a depends on ℓ indicates that the truth value of ℓ could influence the result of the execution of a. For example, the action $drop(o_1, l)$ (Example 1) depends on $holding(o_1)$, $at(l)$, and $trash\_at(l)$; $preact(at(l)) = \{move(l_1, l_2) \mid l_1 \neq l_2 \in Loc\} \cup \{pickup(o_1, l), drop(o_1, l) \mid l \in Loc\}$; $preact(obj\_at(l)) = \{pickup(o_1, l), drop(o_1, l)\}$; and $preact(\neg obj\_at(l)) = \emptyset$; and the action a (Example 3) depends on f and $\neg f$.

Definition 2. *Two literals ℓ and ℓ' are* distinguishable *if $\ell \neq \ell'$ and there is no action that depends on both ℓ and ℓ', i.e., $preact(\ell) \cap preact(\ell') = \emptyset$.*

Obviously, the distinguishable relation is symmetric and irreflexive. Two sets of literals L_1 and L_2 are distinguishable if $preact(L_1) \cap preact(L_2) = \emptyset$.

The dependence between a literal and an action, often used in *reachability analysis*, is defined next.

Definition 3. *A literal* ℓ depends *on an action* a *if*

1. $a : \psi \to \ell$ *is in* P*, or*
2. *there exist an action* b *with* $b : \psi \to \ell$ *in* P *and some* $\ell' \in \psi \cup pre(b)$ *such that* ℓ' *or* $\bar{\ell}'$ *depends on* a*.*

Intuitively, ℓ depends on a implies that the truth value of ℓ may be affected by the execution of the action a. In other words, to change the truth value of ℓ, we might need to execute the action a. By $deps(a)$ we denote the set of literals that depend on a. $postact(\ell) = \{a \mid \ell \in deps(a)\}$ is the set of actions which ℓ depends on. For a set of actions A and a set of literals L, $deps(A) = \bigcup_{a \in A} deps(a)$ and $postact(L) = \bigcup_{\ell \in L} postact(\ell)$. It is easy to see that the following holds.

Observation 1. *Let* a *be an action with* $a : \varphi \to \ell'$ *in* P *and* b *be an action with* $\ell \in deps(b)$ *and* $\ell' \in pre(b)$ *or* $\ell' \in \varphi'$ *for some* $b : \varphi' \to h$ *in* P*. Then* $\ell \in deps(a)$*.*

We now define the notion of independence between literals and actions.

Definition 4. *Two literals* ℓ_1 *and* ℓ_2 *are* independent *if* $\ell_1 \neq \ell_2$ *and there exists no action on which both* ℓ_1 *and* ℓ_2 *depend, i.e.,* $postact(\ell_1) \cap postact(\ell_2) = \emptyset$*.*

Two actions a *and* b *are* independent *if there exists no literal* ℓ *which depends on both* a *and* b*, i.e.,* $deps(a) \cap deps(b) = \emptyset$*.*

We say that two sets of literals L_1 and L_2 are independent if $postact(L_1) \cap postact(L_2) = \emptyset$. Two set of actions A_1 and A_2 are independent if $deps(A_1) \cap deps(A_2) = \emptyset$. The following lemma discusses the relationship between $deps$ and $preact$.

Lemma 1. *Let* A *be a set of actions and* a *is an action with a conditional effect* $a : \varphi, \ell' \to \ell$ *such that* $\ell' \in deps(A)$*. Then,* $\ell \in deps(A)$*.*

Proof. $\ell' \in deps(A)$ implies that there exists some $b \in A$ such that ℓ' depends on b. By Definition 3, ℓ depends on b. Thus, $\ell \in deps(A)$. □

3.2.1 Standard Transformations

The preprocessor starts its operations with a number of basic normalization steps, aimed at reducing the number of propositions and the number of actions present in the problem specification. These steps have been implemented by several other planners. More precisely, the preprocessor implements the traditional *forward reachability* simplification aimed at detecting

1. propositions whose truth value cannot be affected by the actions in the problem specification (with respect to the extended initial state, see below); and
2. actions whose execution cannot be triggered with respect to the extended initial state.

This process can be modeled as a fixpoint computation and easily be implemented using Prolog. As the forward reachability analysis was developed originally for domains with complete information, the preprocessor of CPA(H) starts the computation with an *extended intitial state*, defined by:

$$I_0 = I^d \cup \bigcup_{o \in I^o} lit(o) \cup \bigcup_{o \in I^r} lit(o).$$

The set of forward applicable actions, fw_a, and relevant propositions, fw_p, are defined by

$$\begin{array}{rcl} fw_p^0 & = & I_0 \\ fw_a^0 & = & \{a \mid a \in preact(\ell),\ \ell \in fw_p^0\} \\ fw_p^{k+1} & = & fw_p^k \cup \{\ell \mid a \in fw_a^k,\ a : \psi \rightarrow \ell \in O\} \\ fw_a^{k+1} & = & \{a \mid a \in preact(\ell),\ \ell \in fw_p^{k+1}\} \end{array}$$

and

$$fw_a = \bigcup_{i=0}^{\infty} fw_a^i \qquad\qquad fw_p = \bigcup_{i=0}^{\infty} fw_p^i$$

The preprocessor computes fw_a and fw_p. It then removes the actions and propositions not belonging to fw_a and fw_p, respectively, from the problem.

3.2.2 Combination of `one-of` Clauses

The idea of this technique is based on the *non*-interaction between actions and propositions in different subproblems of a conformant planning problem. As the proposed technique can be applied in any conformant planners, the discussion in this subsection will be based on the reasoning method using the function $succ$ (Subsection 2.1). The results are valid for CPA(H) due to its completeness. We will begin with an example illustrating this idea.

Example 4. Let consider a modified version of the 2×2 *dispose* problem from Example 1 with two objects o_1 and o_2, $D[2,2,2]$, with $D[2,2,2] = \langle P, O, I, G \rangle$. The set of propositions and actions are similar to their counterparts in $D[2,2,1]$ and are omitted for brevity. Assume that the initial state of the problem is given by $I = I^d \cup I^o$ where

$$I^d = \{trash\_at(l_{11}), at(l_{11})\} \cup \{\neg trash\_at(l), \neg at(l) \mid l \in Loc \setminus \{l_{11}\}\} \cup \{\neg holding(o_i) \mid i \in \{1,2\}\}$$

and

$$I^o = \{\texttt{one-of}(obj\_at(o_1, l_{11}), obj\_at(o_1, l_{12})), \texttt{one-of}(obj\_at(o_2, l_{11}), obj\_at(o_1, l_{12}))\}.$$

The goal of the problem is given by

$$G = \{disposed(o_1), disposed(o_2)\}.$$

We can check that that the sequence

$$\alpha = \left[\begin{array}{l} pickup(o_1, l_{11}), drop(o_1, l_{11}), move(l_{11}, l_{12}), pickup(o_1, l_{12}), move(l_{12}, l_{11}), drop(o_1, l_{11}), \\ pickup(o_2, l_{11}), drop(o_2, l_{11}), move(l_{11}, l_{12}), pickup(o_2, l_{12}), move(l_{12}, l_{11}), drop(o_2, l_{11}) \end{array} \right]$$

is a solution of $D[2,2,2]$. Furthermore, the search should start from the initial belief state $S_0 = \{\delta_i \cup I^d \mid i = 1, \ldots, 4\}$ where

$$\begin{aligned}
\delta_1 &= \{obj_at(o_1, l_{11}), \neg obj_at(o_1, l_{12}), obj_at(o_2, l_{11}), \neg obj_at(o_2, l_{12})\} \\
\delta_2 &= \{obj_at(o_1, l_{11}), \neg obj_at(o_1, l_{12}), \neg obj_at(o_2, l_{11}), obj_at(o_2, l_{12})\} \\
\delta_3 &= \{\neg obj_at(o_1, l_{11}), obj_at(o_1, l_{12}), \neg obj_at(o_2, l_{11}), obj_at(o_2, l_{12})\} \\
\delta_4 &= \{\neg obj_at(o_1, l_{11}), obj_at(o_1, l_{12}), obj_at(o_2, l_{11}), \neg obj_at(o_2, l_{12})\}
\end{aligned}$$

Let $D[2,2,2]'$ be the problem obtained from $D[2,2,2]$ by replacing I^o with $I^{o'}$, where

$$I^{o'} = \{\texttt{one-of}(obj\_at(o_1, l_{11}) \wedge obj\_at(o_2, l_{11}), obj\_at(o_1, l_{12}) \wedge obj\_at(o_2, l_{12}))\}.$$

We can easily check that α is also a solution of $D[2,2,2]'$. Furthermore, each solution of $D[2,2,2]'$ is also a solution of $D[2,2,2]$. This transformation in interesting since the initial belief state of $D[2,2,2]'$ is $S'_0 = \{\delta_1 \cup I^d, \delta_3 \cup I^d\}$. In other words, the number of states in the initial belief state that a conformant planner has to consider in $D[2,2,2]'$ is 2, while it is 4 in $D[2,2,2]$. This transformation is possible because the set of actions that are "activated" by $obj\_at(o_1, l_{11})$ and $obj\_at(o_1, l_{12})$ is disjoint from the set of actions that are "activated" by $obj\_at(o_2, l_{11})$ and $obj\_at(o_2, l_{12})$, i.e., $preact(\{obj\_at(o_1, l_{11}), obj\_at(o_1, l_{12})\}) \cap preact(\{obj\_at(o_2, l_{11}), obj\_at(o_2, l_{12})\}) = \emptyset$.

The above example shows that different `one-of` clauses can be combined into a single `one-of` clause, which effectively reduces the size of the initial belief state that the planner needs to consider in its search for a solution. Theoretically, if the size of the two `one-of` clauses in consideration is m and n, then it is possible to achieve a reduction in the number of possible states from $m \times n$ to $\max(m, n)$. Since in many problems the size of the `one-of` clauses increases with the number of objects, being able to combine the `one-of` clauses could provide a significant advantage for the planner. We will present the precise definition of this transformation. We need some extra notations.

Definition 5. *Two disjoint sets of literals L_1 and L_2 are* combinable *if*

- $preact(lit(L_1)) \cap preact(lit(L_2)) = \emptyset$*; and*
- $deps(preact(lit(L_i))) \cap (deps(preact(lit(L_j))) \cup L_j) = \emptyset$ *for* $i \neq j$.

Let Ω_1 and Ω_2 be two combinable sets of literals. For a state s, the (Ω_1, Ω_2)-representation of s is a tuple $(s^1, \ldots, s^5)$ of sets of literals, denoted by $s = (s^1, s^2, s^3, s^4, s^5)$, where

- $s^1 = s \cap lit(\Omega_1)$,
- $s^2 = s \cap lit(\Omega_2)$,
- $s^3 = [s \cap deps(preact(\Omega_1))] \setminus lit(\Omega_1)$,
- $s^4 = [s \cap deps(preact(\Omega_2))] \setminus lit(\Omega_2)$, and
- $s^5 = s \setminus (\bigcup_{i=1}^4 s^i)$.

It is easy to see that the (Ω_1, Ω_2)-representation of s is unique and is a partition of s.

Lemma 2. *Let Ω_1 and Ω_2 be two combinable sets of literals. Let s be a state with the (Ω_1, Ω_2)-representation $(s^1, \ldots, s^5)$. Then, for every action $a \in preact(\Omega_1)$ and conditional effect $a : \varphi \rightarrow \ell$ in O, it holds that $\varphi \cap (s^2 \cup s^4) = \emptyset$ and $\ell \notin (s^2 \cup s^4 \cup s^5)$.*

Proof. We have that $\varphi \cap s^2 = \emptyset$ because $preact(\Omega_1) \cap preact(\Omega_2) = \emptyset$. $\varphi \cap s^4 = \emptyset$ because of Lemma 1 and the combinability of Ω_1 and Ω_2. $\ell \notin (s^2 \cup s^4)$ since $\ell \in deps(preact(\Omega_1))$. $\ell \notin s^5$ by definition of the (Ω_1, Ω_2)-representation. □

Lemma 3. *Let Ω_1 and Ω_2 be two combinable sets of literals. Let s be a state with the (Ω_1, Ω_2)-representation $(s^1, \ldots, s^5)$. Then, for every action $a \notin (preact(\Omega_1) \cup preact(\Omega_2))$ and conditional effect $a : \varphi \rightarrow \ell$, the following hold:*

- $\varphi \cap (s^1 \cup s^2) = \emptyset$
- *Either* $\varphi \cap s^3 = \emptyset$ *or* $\varphi \cap s^4 = \emptyset$.
- *If* $\varphi \subseteq s^5$ *then* $\ell \in lit(s^5 \cup \Omega_1 \cup \Omega_2)$
- *If* $\varphi \subseteq (s^5 \cup s^3)$ *and* $\varphi \setminus s^3 \neq \emptyset$ *then* $\ell \in deps(preact(\Omega_1))$.
- *If* $\varphi \subseteq (s^5 \cup s^4)$ *and* $\varphi \setminus s^4 \neq \emptyset$ *then* $\ell \in deps(preact(\Omega_2))$.

Proof. The first item holds because $a \notin preact(\Omega_1)$ and $a \notin preact(\Omega_2)$. The second item holds because of the combinability of Ω_1 and Ω_2. The third item and fourth item follow from Lemma 1. □

Definition 6. *Let S be a set of states and s be a state. We say that S covers s with respect to two combinable set of literals, Ω_1 and Ω_2, if S contains two states u and v such that $u = (s^1, u^2, s^3, u^4, v^5)$ and $v = (v^1, s^2, v^3, s^4, s^5)$.*

In the next lemma, we show that the coverage property between belief states and states is an invariant over the execution of actions.

Lemma 4. *Let S be a set of states and s be a state such that S covers s with respect to be two combinable set of literals Ω_1 and Ω_2. Let a be an executable action in S. Then, a is also executable in S and $succ^*(a, S)$ covers $succ(a, s)$ with respect to Ω_1 and Ω_2.*

Proof. Since S covers s with respect to Ω_1 and Ω_2, there exist u and v in S such that $u = (s^1, u^2, s^3, u^4, v^5)$ and $v = (v^1, s^2, v^3, s^4, s^5)$. This shows that $u \cap v \subseteq s$. Thus, a is executable in s since a is executable in u and v.

We consider the following cases:

- $a \in preact(\Omega_1)$. Lemma 2 implies that $e_a(s) = e_a(u)$ and $e_a(s) \cap (s^2 \cup s^4 \cup s^5) = \emptyset$. Thus, $succ(a, s) = (s^1_1, s^2, s^3_1, s^4, s^5)$ where $s^1_1 = (s^1 \setminus \overline{e_a(s)} \cup e_a(s)) \cap lit(\Omega_1)$ and $s^3_1 = (s^3 \setminus \overline{e_a(s)} \cup e_a(s)) \setminus lit(\Omega_1)$. Using the same arguments and the (Ω_1, Ω_2)-representations of u and v, we can show that $succ(a, u) = (s^1_1, u^2, s^3_1, u^4, s^5)$, and $succ(a, v) = (v^1_1, s^2, v^3_1, s^4, s^5)$. Since $succ(a, u) \in succ^*(a, S)$ and $succ(a, v) \in succ^*(a, S)$, we have that $succ^*(a, S)$ covers $succ(a, s)$.
- $a \in preact(\Omega_2)$. This is similar to the previous case due to the symmetry of u and v with respect to Ω_1 and Ω_2.
- $a \notin preact(\Omega_1)$ and $a \notin preact(\Omega_2)$. Let

$$\begin{array}{lcl} C_1 & = & \{a : \varphi \rightarrow \ell \mid \varphi \subseteq s^5\} \\ C_2 & = & \{a : \varphi \rightarrow \ell \mid \varphi \subseteq s^5 \cup s^3 \wedge \varphi \cap s^3 \neq \emptyset\} \\ C_3 & = & \{a : \varphi \rightarrow \ell \mid \varphi \subseteq s^5 \cup s^4 \wedge \varphi \cap s^4 \neq \emptyset\} \\ C_4 & = & \{a : \varphi \rightarrow \ell \mid \varphi \subseteq s^5 \cup u^4 \wedge \varphi \cap u^4 \neq \emptyset\} \\ C_5 & = & \{a : \varphi \rightarrow \ell \mid \varphi \subseteq s^5 \cup v^3 \wedge \varphi \cap v^3 \neq \emptyset\} \end{array}$$

From the construction of s^i, u^i, and v^i we have that $C_1 \cap C_2 = C_1 \cap C_3 = C_2 \cap C_3 = \emptyset$. Furthermore, $C_1 \cap C_4 = C_1 \cap C_2 = C_2 \cap C_4 = \emptyset$ and $C_1 \cap C_5 = C_1 \cap C_3 = C_3 \cap C_5 = \emptyset$. Let $e_i = \{\ell \mid a : \varphi \rightarrow \ell \in C_i\}$. From Lemma 3, we can conclude $e_a(s) = e_1 \cup e_2 \cup e_3$, $e_a(u) = e_1 \cup e_2 \cup e_4$, and $e_a(v) = e_1 \cup e_3 \cup e_5$. Lemma 3 also implies that $e_2 \subseteq deps(preact(\Omega_1))$ and thus $e_2 \cap (deps(preact(\Omega_2)) \cup lit(\Omega_2)) = \emptyset$. Similarly, $e_3 \subseteq deps(preact(\Omega_2))$ and thus $e_3 \cap (deps(preact(\Omega_1)) \cup lit(\Omega_1)) = \emptyset$. This allows us to derive the following

$$succ(a, s) = \left(\begin{array}{c} (s^1 \setminus \overline{(e_1 \cup e_2)} \cup (e_1 \cup e_2)) \cap lit(\Omega_1), \\ (s^2 \setminus \overline{(e_1 \cup e_3)} \cup (e_1 \cup e_3)) \cap lit(\Omega_2), \\ (s^3 \setminus \overline{(e_1 \cup e_2)} \cup (e_1 \cup e_2)) \cap deps(preact(\Omega_1)) \setminus lit(\Omega_1), \\ (s^4 \setminus \overline{(e_1 \cup e_3)} \cup (e_1 \cup e_3)) \cap deps(preact(\Omega_2)) \setminus lit(\Omega_2), \\ (s^5 \setminus \overline{e_1} \cup e_1 \setminus lit(\Omega_1 \cup \Omega_2)) \end{array} \right)$$

Similarly, we have that

$$succ(a, u) = \left(\begin{array}{c} (s^1 \setminus \overline{(e_1 \cup e_2)} \cup (e_1 \cup e_2)) \cap lit(\Omega_1), \\ (u^2 \setminus \overline{(e_1 \cup e_4)} \cup (e_1 \cup e_4)) \cap lit(\Omega_2), \\ (s^3 \setminus \overline{(e_1 \cup e_2)} \cup (e_1 \cup e_2)) \cap deps(preact(\Omega_1)) \setminus lit(\Omega_1), \\ (u^4 \setminus \overline{(e_1 \cup e_4)} \cup (e_1 \cup e_4)) \cap deps(preact(\Omega_2)) \setminus lit(\Omega_2), \\ (s^5 \setminus \overline{e_1} \cup e_1 \setminus lit(\Omega_1 \cup \Omega_2)) \end{array} \right)$$

and

$$succ(a, v) = \left(\begin{array}{c} (v^1 \setminus \overline{(e_1 \cup e_5)} \cup (e_1 \cup e_5)) \cap lit(\Omega_1), \\ (s^2 \setminus \overline{(e_1 \cup e_3)} \cup (e_1 \cup e_3)) \cap lit(\Omega_2), \\ (v^3 \setminus \overline{(e_1 \cup e_5)} \cup (e_1 \cup e_5)) \cap deps(preact(\Omega_1)) \setminus lit(\Omega_1), \\ (s^4 \setminus \overline{(e_1 \cup e_3)} \cup (e_1 \cup e_3)) \cap deps(preact(\Omega_2)) \setminus lit(\Omega_2), \\ (s^5 \setminus \overline{e_1} \cup e_1 \setminus lit(\Omega_1 \cup \Omega_2)) \end{array} \right)$$

This shows that $succ^*(a, S)$ covers $succ(a, s)$.

□

Definition 7. *Let $P = \langle F, O, I, G \rangle$ be a planning problem. Two* `one-of` *clauses o_1 and o_2 are* combinable *if $lit(o_1)$ and $lit(o_2)$ are combinable.*

As noted, the `one-of` clauses in Example 4 are combinable. Let $o_1 =$ `one-of`$(L_1, \ldots, L_n)$ and $o_2 =$ `one-of`$(S_1, \ldots, S_m)$. Assume that $n \geq m$. A *combination* of o_1 and o_2, denoted by $o_1 \oplus o_2$ (or $o_2 \oplus o_1$) is the clause

$$\texttt{one-of}(L_1 \wedge S_1, \ldots, L_m \wedge S_m, L_{m+1} \wedge S_1, \ldots, L_n \wedge S_1)$$

Intuitively, a combination of o_1 and o_2 is a `one-of` clause whose elements are pairs obtained by composing one element of o_1 with exactly one element of o_2.

Proposition 2. *Let $P = \langle F, O, I, G \rangle$ be a planning problem, where G is a conjunction of literals and o_1 and o_2 are two combinable* `one-of` *clauses in P. Let $P' = \langle F, O, I', G \rangle$, where I' is obtained from I by replacing o_1 and o_2 by $o_1 \oplus o_2$. Every solution of P' is also a solution of P and vice versa.*

Proof. We consider two cases:

- α is a solution of P. The conclusion of the proposition follows from the following facts:
  - $ext(I') \subseteq ext(I)$,
  - if $\widehat{succ}(\alpha, ext(I')) = failed$ then $\widehat{succ}(\alpha, ext(I)) = failed$, and
  - $\widehat{succ}(\alpha, ext(I')) \subseteq \widehat{succ}(\alpha, ext(I))$.
- Let $\Omega_1 = lit(o_1)$ and $\Omega_2 = lit(o_2)$. We observe that $ext(I')$ covers every $s \in ext(I) \setminus ext(I')$. We prove by induction that if α is a solution of P' then α is a solution of P. This follows immediately from Lemma 4 since the coverage of $ext(I')$ over states belonging to $ext(I) \setminus ext(I')$ is maintained through the execution of executable actions.

□

Observe that the above proposition may not hold if P contains disjunctive goals, as shown next.

Example 5. Let $P = \langle \{q, g, h, p, i, j\}, O, I, G \rangle$ where

$$I = \{\texttt{one-of}(h, g), \texttt{one-of}(p, q), \neg i, \neg j\} \text{ and } G = \texttt{or}(i, j)$$

and O consists of $a : p, \neg q \rightarrow i$, $c : p, q \rightarrow i$, $b : g, \neg h \rightarrow j$, and $d : \neg g, h \rightarrow j$.

It is easy to check that $\texttt{one-of}(h, g)$ and $\texttt{one-of}(q, p)$ are combinable. Let P' be the problem obtained from P by replacing I with $I' = \{\texttt{one-of}(g \wedge q, h \wedge p), \neg i, \neg j\}$. Then, $[a, b]$ is a solution of P' but not a solution of P.

The *combinable* notion can be generalized as follows.

Definition 8. *A set of* one-of*-clauses $\{o_1, \ldots, o_k\}$ is combinable if o_i and o_j are combinable for each $1 \leq i \neq j \leq k$.*

Let $\oplus(o_1, \ldots, o_k)$ be the shorthand for $(((o_1 \oplus o_2) \oplus \ldots) \oplus o_k)$. Proposition 2 can be generalized as follows.

Proposition 3. *Let $P = \langle F, O, I, G \rangle$ be a planning problem, where G is a conjunction of literals. Let $\{o_1, \ldots, o_k\}$ be a combinable set of* one-of*-clauses in P. Let $P' = \langle F, O, I', G \rangle$, where I' is obtained from I by replacing $\{o_1, \ldots, o_k\}$ with $\oplus(o_1, \ldots, o_k)$. We have that each solution of P' is a solution of P and vice versa.*

Proof. By induction. Straightforwards due to Definition 5, the definition of $deps$ and $preact$, and combinability between the one-of clauses □

We developed an algorithm for reducing the size of the initial cs-state by composing combinable sets of one-of clauses in a planning problem P. We implemented a greedy algorithm, whose running time is polynomial in the size of P, for detecting sets of combinable one-of clauses and replacing them with their corresponding combination. This is possible since testing if two set of literals are combinable can be done in polynomial time in the size of P, and the number of pairs that need this test is quadratic in the number of propositions. Algorithms 2 shows the procedure to detect combinable groups of one-of clauses.

We then implemented Algorithm 3 to combine the combinable one-of clauses.

Algorithm 2 : COMBINABLE(P: planning problem)

Require: $\{o_1, \ldots, o_n\}$: `one-of` clauses in P

$S = \{o_1, \ldots, o_n\}$
$Q = \emptyset$
while ($S \neq \emptyset$) **do**
  pick $o \in S$ and set s=$\{o\}$
  for all ($o' \in S \ \wedge \ o' \notin s$) **do**
    if (o' is combinable with every $r \in s$) **then**
      $s = s \cup \{o'\}$
    end if
  end for
  $S = S \setminus s$
  if $|s| > 1$ **then**
    $Q = Q \cup \{s\}$
  end if
end while
return Q

Algorithm 3 Composition of `one-of` clauses

Require: $\{o(L^i_1, \ldots, L^i_{n_i})\}_{i=1}^{k}$ combinable `one-of` clauses

$o = \emptyset$
$d[1, \ldots, k] = [1, \ldots, 1]$
for (i=1 **TO** max($n_1, \ldots, n_k$)) **do**
  $c = true$
  for (j=1 **TO** k) **do**
    $c = c \wedge L^j_{d[j]}$
    if ($j < n_j$) **then**
      $j = j + 1$
    end if
  end for
  $o = o \cup \{c\}$
end for
return o

3.2.3 Goal Splitting

Reducing the size of the initial state only helps the planner to start the search. It does not necessarily imply that the planner can find a solution. Furthermore, the technique is not always applicable as shown in Tables 4 and 8. In this section, we present another technique, called *goal-splitting*, which can be used in conjunction with the combination of `one-of` to deal with large planning problems. This technique can be seen as a variation of the goal ordering technique in [10] and it relies on the notion of dependence proposed in Definition 4. The key idea is that if a problem P contains a subgoal whose truth value cannot be negated by the actions used to reach the other goals, then the problem can be decomposed into smaller problems with different goals, whose solutions can be combined to create a solution of the original problem. This is illustrated in the following example.

Example 6. *Consider the problem $P = D[2,2,2]$ of Example 4. It is easy to see that the two goals $disposed(o_1)$ and $disposed(o_2)$ are independent in that we can solve the problem by computing the plan to achieve each subgoal independently, one after another. More formally, P can be decomposed into two sub-problems*

$$P_1 = \langle F, O_0 \cup O_1, I, disposed(o_1)\rangle$$

and

$$P_2 = \langle F, O_0 \cup O_2, I_2, disposed(o_2)\rangle$$

where $O_0 = \{move(l_1,l_2) \mid l_1, l_2 \in Loc\}$, $O_1 = \{pickup(o_1,l), drop(o_1,l) \mid l \in Loc\}$, and $O_2 = \{pickup(o_2,l), drop(o_2,l) \mid l \in Loc\}$ with the following property: if α is a solution of P_1 and β is a solution of P_2 where $I_2 = \widehat{succ_A}(\alpha, I_1)$, then $\alpha;\beta$ is a solution of P.[3] □

Let us start with a definition capturing the condition that allows the splitting of goals.

Definition 9. *Let $P = \langle F, O, I, G\rangle$ be a planning problem and let $\ell \in G$. We say that ℓ is G-separable if, for each $\ell' \in G \setminus \{\ell\}$ we have that $\bar{\ell}$ and ℓ' are independent.*

For a planning problem P and a literal $\ell \in G$, a splitting of P with respect to ℓ is a pair $(P_\ell, P_{G\setminus\{\ell\}})$ of planning problems where $P_\ell = \langle F, postact(\ell), I, \ell\rangle$ and $P_{G\setminus\{\ell\}} = \langle F, postact(G\setminus\{\ell\}), *, G\setminus\{\ell\}\rangle$ and the '' in $P_{G\setminus\{\ell\}}$ denotes an unspecified initial state.*

According to this definition, the subgoals in Example 4 are G-seperatable since $postact(\neg disposed(o_i)) \cap postact(disposed(o_j)) = \emptyset$ for $i \neq j$ and $i, j \in \{1,2\}$, i.e., once $disposed(o_i)$ is achieved, it cannot be made false by the actions which may be necessary to achieve $disposed(o_j)$. We prove that G-separable is sufficient for goal spliiting in the next proposition.

Proposition 4. *Let $P = \langle F, O, I, G\rangle$ be a planning problem. Assume that $\ell \in G$ is G-separable. Let $P_\ell = \langle F, postact(\ell), I, \ell\rangle$ and α be a solution of P_ℓ. Let $P_{G\setminus\{\ell\}} = \langle F, postact(G\setminus\{\ell\}), I', G\setminus\{\ell\}\rangle$, where $I' = \widehat{succ}(\alpha, I)$, and β be a solution of $P_{G\setminus\{\ell\}}$. Then, $\alpha;\beta$ is a solution of P.*

Proof. Trivial since $postact(G\setminus\{\ell\})$ does not contain any action that can make $\bar{\ell}$ true. □

Proposition 4 guarantees the soundness of the splitting technique. On the other hand, it is easy to see that not every plan of P can be splitted into two parts α and β such that α is a solution of P_ℓ and β is a solution of $P_{G\setminus\{\ell\}}$. It is also possible that not every decomposition of P can be used to search for a solution of P even if the goals are separable . This can be seen in the following example.

[3] $\alpha;\beta$ denotes the concatenation of two sequences of actions.

Example 7. *Consider the problem* $P = \langle\{f, g, h, k\}, \{a : \top \to \neg g, h, b : g, f \to k\}, \{f, g, \neg h, \neg k\}, h \wedge k\rangle$.

Clearly, h *and* $\neg k$ *are independent because* $postact(\neg k) = \emptyset$. *Similarly,* $\neg h$ *and* k *are independent because* $postact(\neg h) = \emptyset$. *Therefore both* h *and* g *are* G*-separable with respect to* $h \wedge g$. *Furthermore, the problem has a solution* $[b, a]$.

Splitting the problem into two problems $P_k = \langle\langle\{f, g, h, k\}, \{b : g, f \to k\}, \{f, g, \neg h, \neg k\}, k\rangle$ *and* $P_h = \langle\{f, g, h, k\}, \{a : \top \to \neg g, h\}, \{f, g, \neg h, k\}, h\rangle$ *does indeed allow us to find the plan* $[b, a]$.

On the other hand, the splitting of P *into* $P'_h = \langle\langle\{f, g, h, k\}, \{a : \top \to \neg g, h\}, \{f, g, \neg h, \neg k\}, h\rangle$ *and* $P'_k = \langle\{f, g, h, k\}, \{b : g, f \to k\}, I', k\rangle$ *will yield no solution.*

The above example shows that the goal splitting technique is only sound. To guarantee its completeness, we need to be able to identify an appropriate order among the goals. However, this problem is a well-known hard problem. On the other hand, checking for G-separable according to Definition 9 is only polynomial in the size of P. We note that the splitting proposed in Definition 9 can be improved by also splitting the propositions and initial states into different theories. We have implemented a generalized version of Definition 9 to split a problem into a sequence of problems. This implementation runs in polynomial time in the size of P. In our experiments, whenver a solvable problem can be splitted, the sequence of sub-problem does yield a solution.

4 Implementation of CPA(H)

CPA(H) is built from the C++ source code of CPA+ [20]. The main differences between CPA+ and CPA(H) lie in the use of a more complex heuristic and the application of the `one-of` combination technique and the goal splitting technique by CPA(H). Since both techniques are characterized by syntactical conditions, they can be processed by a static analyzer. We will now elaborate on the implementations of the two components of CPA(H): preprocessor and the planning engine.

4.1 Implementation of the Preprocessor

The preprocessor is implemented in Prolog (specifically, SICStus Prolog). The choice of Prolog was natural, as it provides several features needed by the problem at hand:

- The components of a problem specification have an obvious representation as Prolog terms and clauses; PDDL actions and fluents have parameters and they can be encoded as complex terms, e.g., the action `go_up` with parameters elevator, floor, floor, is naturally represented by the term `go_up(Elev,Floor1,Floor2)`.
- PDDL statements can be readily mapped to a collection of Prolog rules; in particular, it allows us to keep a non-ground representation, and offers a quick access to the various components of the domain specification. For example, the PDDL action specification of the action `go_up` is translated to the Prolog rules in Fig. 2. Grounding can be obtained for free by simply collecting all valid instances of an action (e.g., using `setof`). Unification allows us to easily select components of the problem specification that meet desired requirements—e.g., a simple goal like `executable(go_up(e0,X,Y),L)` gives us access to the executability conditions of any instance of the action `go_up` targeting elevator `e0`.

```
(:action go-up                          action(go_up(E,F,NF)):-
 :parameters (?e - elevator               elevator(E), floor(F),
              ?f ?nf - floor)             floor(NF).
 :precondition (dec_f ?nf ?f)            executable(go_up(E,F,NF),
 :effect (when (in ?e ?f)                           [dec_f(NF,F)]):-
     (and (in ?e ?nf)                      elevator(E),
          (not (in ?e ?f)))) )             floor(F), floor(NF).
                                         causes(go_up(E,F,NF),
                                                [in(E,NF),neg(in(E,F))],
                                                              [in(E,F)]) :-
                                          elevator(E), floor(F), floor(NF).
```

Figure 2: PDDL action and Prolog representation

- Most of the proposed transformations described are fixpoint computations, and these can be elegantly encoded in Prolog.
- Viewing action specifications as Prolog clauses, allows us to write elegant meta-interpreters to perform abstract executions; for example, if we represent an approximate state as an ordered list `L` of terms (representing the fluent literals that hold in that partial state), then determining the executable actions and the derived consequences from applying such actions can be reduced to simple Prolog statements, a `findall` applied to the goal `executable(A,C), ord_subset(C,L), causes(A,Cons,_)`. Meta-interpreters allow us to simulate both progression (i.e., if action is applicable, applied it and repeat) and regression (i.e., from the goal find actions that produce the goal and replace goal with their preconditions). Note that abstractions of progressions and regression are needed to compute forward reachability and goal relevance.

The preprocessor maps the input PDDL theory to a collection of Prolog clauses. This mapping nicely avoids the need of explicitly grounding the problem specification a priori. The transformations are implemented as fixpoint computations on the Prolog clauses representing the problem specification.

4.2 Implementation of the Planning Engine

As we have mentioned earlier, the planning engine is built from the source code of CPA+ [20] and is implemented as a C++ program, running on a Linux, gcc 4.2.1 version, with STL library. A partial state is implemented as a set (a basic data structure in STL) of literals. The engine implements the best first search over the search space of cs-states (Algorithm 1). Each cs-state is a data structure consisting of a set of partial states, a plan to reach that cs-state, and the heuristic values: h_{card}, h_{gc}, and h_{rpg}. A modified version of the algorithm presented in [11] is implemented to compute h_{rpg}.

$succ^*_A$ is used to compute the next cs-state. A hash table (resp. priority queue) is used to store the visited (resp. unvisited) cs-states. A special module is developed to compute the initial cs-state, which consists of the set of initial partial states and guarantees the completeness of CPA(H). Each initial partial state δ satisfies the following conditions:

1. $\{p, \neg p\} \cap \delta \neq \emptyset$ for each proposition p appears in I;
2. $I^d \subseteq \delta$;
3. for each `one-of`$(\phi_1, \ldots, \phi_n) \in I^o$, there exists an i such that $\phi_i \subseteq \delta$ and for all $j \neq i$, $\overline{\phi_j} \cap \delta \neq \emptyset$;
4. for each `or`$(\phi_1, \ldots, \phi_n) \in I^r$, there exists an i such that $\phi_i \subseteq \delta$; and

5. δ is consistent.

Choosing to implement the initial cs-state as a set (of the set of initial partial states) makes the computation of the successor cs-state (the result of $succ^*_A$) easier. The main disadvantage of this choice is that the size of the initial cs-state can be exponential in the size of the number of object constants in the problem. This is also the main reason why reducing the size of the initial cs-state is critical to our planner.

5 Experimental Evaluation

We compare CPA(H) with several other conformant planners: Conformant-FF (CFF) [4], KACMBP[8], POND [6], `t0` [15], and DNF [23]. These are some of the fastest conformant planners on most of the benchmark domains in the literature. [4] We would like to mention that, in this paper, we only compare CPA(H) with other conformant planners with the same capabilities. Among other things, the representation language employed in the discussed planners is, in one way or another, a propositional language with limited expressive power. Furthermore, all these planners are complete. For this reason, we do not compare CPA(H) with the PKS system [16] (improved in [17]) which employs a richer representation language and the knowledge-based approach to reason about effects of actions in the presence of incomplete information and is incomplete.

To make the comparison between the different planning systems as fair as possible, we try our best in not altering the problem specifications obtained from the different repositories. Nevertheless, this is almost impossible since the collected planners do generally disagree on input format. For example, KACMBP requires the input in a different format, all other systems receive the same PDDL input in all domains and `t0` does not support negative preconditions. The experiments have been conducted on a Linux platform, based on an Intel Pentium 4 3.06GHz chipset and 1GB of RAM. We tested the performance of CPA(H) using four collections of benchmarks in our experimentation. [5]

5.1 IPC-05 Domains

The IPC-05 domains [3] consists of six domains used in the 2006 planning competition. The `adder` domain is the synthesis of an adder Boolean circuit. The `coins` domain is similar to the well-known transportation domain where the goal is to collect coins from different, initially unknown, positions. The `sortnet` domain is a synthesis of sorting networks which has disjunctive goals and a large number of possible initial states. The `comm` domain encodes a communication protocol whose difficulty lies in the huge size of the initial state. The `uts` domain is the computation of universal transversal sequence for graphs whose number of actions and uncertainty are more manageable comparing to other domains. The test suite also contains some problems in the `block-world` domain. Table 2 describes a few parameters of these problems—the table describes the number of actions, propositions, goals and `oneof` statements; the table also indicates the reduction in these numbers achieved by the previously mentioned simplifications. For example, in the `comm-10` instance, the theoretical numbers of actions and propositions are 529 and 419, respectively. The simplification reduces these number to 203 and 69, respectively. The most significant reduction obtained is, however, the number of partial states in the initial state, a reduction from 2^{11} to 2. Observe, however, that the reduction of number of initial partial states is achieved only in three out of six domains.

[4] The authors of [23] recently developed two other conformant planners whose performance is comparable to DNF. For this reason, it only present the results of comparing to DNF.

[5] The encodings of the instances are available at `www.cs.nmsu.edu/~tson/CpA`.

Instance	# Actions		# Propositions		# Goal	oneof	
	Theoretical Number	After Simplification	Theoretical Number	After Simplification		Theoretical Number	After Simplification
comm-10	529	203	419	69	11	2^{11}	2
comm-15	1004	373	764	99	16	2^{16}	2
comm-20	5710	1968	4070	189	40	2^{21}	2
comm-25	15515	5153	10900	214	65	2^{26}	2
coins-10	144	56	78	34	4	2^{10}	16
coins-15	432	136	208	76	6	2^{20}	32
coins-20	660	206	271	86	6	$2^{18}\times 9$	72
coins-25	5500	1870	1920	320	15	10^{20}	10^5
coins-30	6000	2370	2425	376	20	10^{25}	10^5
uts-20	420	420	441	41	20	20	20
uts-25	110	110	121	21	10	10	10
uts-30	420	420	441	41	20	20	20
sortnet-5	36	15	42	42	5	2^6	2^6
sortnet-10	121	55	132	11	10	2^{11}	2^{11}
sortnet-15	256	12	272	16	15	2^{16}	2^{16}
adder-1	3100	1800	30	20	17	4	4
adder-2	8428	4810	42	28	31	16	16
adder-3	17820	9996	54	36	45	64	64
blw-1	12	12	11	11	2	5	5
blw-2	24	24	19	19	3	18	18
blw-3	40	40	29	29	4	125	125

Table 2: Characteristics of the IPC-05 Domains

Table 3 contains some representative results [6] of our experiments with the IPC-05 domains. Our planner compete well with the other planners on most domains. There is only one domain `adder` where our planner fails to find a solution (time-out) while some other planners succeed. Our planners can find solutions in several instances where others fail (e.g., in `sortnet`, where `t0` and CFF cannot be used, in `comm` where KACMBP times out).

`t0` is more consistent and has better performance in most of the domains that are applicable to it. CPA(H) is comparable to CFF in most instances, except for `comm-25`. In genral, DNF performs better than CPA(H). One of the main reason for this is the compact belief state representation used by DNF. POND tends to be faster in smaller instances but it does not seem to scale up well in larger instances, compared to CPA(H). It should be mentioned that the combined heuristics h_c is not an admissible heuristic and this is reflected in the length of the solutions found by CPA(H)—they are often longer than those found by other planners. This can also be seen from the results of DNF, which employs similar heuristic to CPA. It is also interesting to note that only KACMBP can solve the `adder-01` and `adder-02` instance. This domain has a very large number of actions whose preconditions are empty, and the cardinality heuristic does not help—since the number of states is constant in every step of the computation. Furthermore, the goal is a huge formula with disjunction, which might indicate that the treatment of formulas in KACMBP is better than other planners.

5.2 Challenging Domains

This test suite consists of the domains that seem to be challenging for conformant planners, as detailed in [14]. These are variations of the grid problems. `dispose` is about retrieving objects whose initial location

[6] Some of the numbers have been rounded up. The complete experimental results are included in the Appendix.

Instance	CPA(H)		t0		CFF		POND		KACMBP		DNF	
	Time	Len	Time	Len	Time	Len	Time	Len	Time	Len	Time	Len
adder-01	-	TO	-	NA	-	NA	-	TO	65.62	3	6.343	3
adder-02	-	TO	-	NA	-	NA	-	TO	1650.87	59		TO
blw-01	0.199	4	0.056		-	NA	0.01	6	0.052	5	0.196	7
blw-02	0.424	33	0.22		-	NA	0.12	34	-	AB	0.93	40
blw-03	20.475	205	48.51		-	NA	7.69	80	-	AB	307.441	325
coins-01	0.02	11	0.008	9	0	12	0.02	11	1.476	15	0.21	10
coins-05	0.006	11	0.012	11	0.01	13	0.04	13	1.592	16	0.22	11
coins-10	0.031	48	0.036	26	1.02	38	1.07	46		TO	0.205	27
coins-15	0.396	191	0.1	81	7.35	79	21.1	124		TO	0.537	67
coins-20	0.737	195	0.152	108	38.19	143	211.19	153		TO	0.970	99
comm-02	0.087	17	0.024	19	0	17	0.04	17	-	TO	0.21	19
comm-05	0.178	35	0.028	40	0.01	35	0.21	35	-	TO	0.309	45
comm-10	0.662	65	0.56	75	0.06	65	1.46	65	-	TO	1.104	80
comm-15	2.290	95	0.092	110	0.22	95	23.34	98	-	TO	3.426	125
comm-20	56.878	239	0.484	278	13.33	239		TO	-	TO	145.065	296
comm-25	1222.63	389	1.552	453	109.49	389		TO	-	TO	1797.777	501
sortnet-01	0.061	1	-	NA	-	NA	0.01	1	0.012	1	0.064	1
sortnet-05	0.068	13	-	NA	-	NA	0.01	12	0.160	13	0.082	11
sortnet-10	2.373	39	-	NA	-	NA	0.05	38	1.708	41	1.447	54
sortnet-15	740.00	74	-	NA	-	NA	0.28	65	13.900	46	35.303	118
uts-01	0.002	4	0.012	4	0	4	0	4	0.024	4	0.082	4
uts-05	0.330	34	0.132	29	0.34	28	0.74	33	-	TO	0.473	31
uts-10	14.33	89	0.88	59	55.49	58	26.16	68	-	TO	2.656	66
uts-15	0.204	53	0.072	47	0.04	29	1.19	46	-	TO	0.25	59
uts-20	8.936	156	0.56	85	1.64	59	111.54	88	-	TO	1.651	109
uts-25	0.185	33	0.092	34	1.51	33	0.72	32	-	TO	0.245	39
uts-30	4.905	74	0.792	67	25.65	66	39.88	68	-	TO	1.39	73

Table 3: IPC-05 domains (Time in *seconds*, TO-Time out (30 min), NA-Not Applicable, AB-Out of Memory)

is unknown and placing them in a trash can at a given location. `push-to` is a variation where objects can be picked up only at two designated positions in the grid to which all objects have to be pushed to. `1-dispose` is a variation of `dispose` where the robot hand being empty is a condition necessary for the pick-up action. `look-n-grab` is about picking up the objects that are sufficiently close, if there are any, and each object picked has to be dropped in the trash can before continuing. Table 4 summarizes the characteristics of these benchmarks.

Table 5 contains the results of our experiments with the challenging domains from [14]. We obtained the scripts for generating these domains from the authors of `t0`. As described in [14], other planners cannot handle these domains. For these reasons, we did not compare CPA(H) with other planners on these domains.

As it can be seen, CPA(H) is faster than `t0` in most of the challenge domains. It can also solve more problems compared to `t0`. In these domains, the cardinality heuristic does very well. Yet, CPA(H) tends to produce longer plan than `t0`.

As in other domains, DNF performs better than CPA(H) and can solve more problems than `t0`. It is interesting to observe that the solutions produced by DNF are shorter than those produced by `t0` and CPA(H). DNF also displays better scalability as it is the only one that can solve `1-dispose-8-2` or `look-n-grab-8-1-2`.

The main reason for the better performance of CPA(H) vs. `t0` has been discussed in detail by the authors of `t0` in [15].

Instance	# Actions		# Propositions		# Goal	`oneof`	
	Theoretical Number	After Simplification	Theoretical Number	After Simplification		Theoretical Number	After Simplification
dispose-4-1	288	65	646	83	1	16	16
dispose-4-2	320	82	720	101	2	16^2	16
dispose-4-3	352	99	798	119	3	16^3	16
dispose-8-1	4224	289	8710	355	1	64	64
dispose-8-2	4352	354	8976	421	2	64^2	64
dispose-8-3	4480	419	9246	487	3	64^3	64
dispose-12-1	21024	673	42630	819	1	144	144
push-4-1	528	98	305	83	1	16	16
push-4-2	800	148	322	100	2	16^2	16
push-4-3	1072	198	339	117	3	16^3	16
push-8-1	8256	450	4289	355	1	64	64
push-8-2	12416	676	4354	420	2	64^2	64
push-8-3	16576	902	4419	485	3	64^3	64
push-12-1	41616	1058	21169	819	1	144	144
1-dispose-4-1	288	80	613	82	1	16	16
1-dispose-4-2	288	80	685	99	2	16^2	16^2
1-dispose-4-3	288	80	761	116	3	16^3	16^3
1-dispose-8-1	4224	352	8581	354	1	64	64
1-dispose-8-2	4224	352	8845	419	2	64^2	64^2
look-n-grab-4-1-1	288	80	613	83	1	16	16
look-n-grab-4-2-1	288	80	613	82	1	16	16
look-n-grab-4-3-1	288	80	613	82	1	16	16
look-n-grab-8-1-1	4224	352	8581	354	1	64	64
look-n-grab-8-1-2	4224	352	8845	419	2	64^2	64^2
look-n-grab-8-2-1	4224	352	8581	354	1	64	64

Table 4: Characteristics of the Challenging Domains

5.3 IPC-06 domains

This test suite consists of six domains used in the 2008 planning competition. The `blockworld` and `adder` from the previous competition are reused without any change. A modified version of the `uts`, which were also used in the 2006 competition, is used in the 2008 competition. In particular, the topology of the mobile ad-hoc network is fully known in the IPC-05 domain while in this version, the topology is partially known. The `raos-keys` domain is to find keys behind locked gates under different lights. It is not known which key opens which gate and which key is located under which light. The difficulty in this domain is the size of the initial state, which increases exponential in the number of lights and keys. The `forest` domain is a hierachical structure domain. The top level is an unobservable grid navigation problem with classical sub-problem at each grid point. The classical sub-problem for each grid node is randomly generated from the set: a two city logistics problem, the Sussman anomaly blockworld, or a small grid navigation problem. In this domain, the size of the grid influences the size of the initial belief state. The characteristics of these domains are detailed in Table 6.

Table 7 contains the results of our experiments with the domains from IPC-06. We report here only the results on the domains that were not used in the IPC-05 or the challenging domains.

The result of CPA(H)'s `uts-cycle` is different from the report in the IPC-06 final result. The reason is that there was a bug in the preprocessor that generates incorrect initial state for the planner. With the bug fixed, CPA(H) can solve 9 instances instead of 2.

It is noted that `t0` performed much better than other planners in `forest`. We suspect that the cardinality

Instance	CPA(H)		t0		DNF	
	Time	Len	Time	Len	Time	Len
dispose-4-1	0.266	84	0.05	59	0.476	45
dispose-4-2	0.605	198	0.100	110	0.617	78
dispose-4-3	0.406	314	0.212	122	0.931	185
dispose-8-1	15.004	741	1.668	426	47.118	150
dispose-8-2	96.377	1480	12.724	639	54.426	302
dispose-8-3	224.676	2227	133.92	761	34.417	629
push-4-1	.0.328	66	0.116	78	0.525	41
push-4-2	0.621	146	0.316	136	0.821	118
push-4-3	1.669	224	1.706	208	1.576	194
push-8-1	17.765	465	61.85	464	51.212	163
push-8-2	212.9	1099	-	AB	68.030	903
push-8-3	-	AB	-	AB	103.820	1477
1-dispose-4-1	0.764	112	12.908	148	0.507	68
1-dispose-4-2	4.473	108	-	AB	1.45	256
1-dispose-4-3	61.63	108	-	AB	17.307	64
1-dispose-8-1	613.653	1456	-	AB	60.218	246
1-dispose-8-2	-	AB	-	AB	85.807	256
look-n-grab-4-1-1	1.058	44	0.260	14	0.530	16
look-n-grab-4-2-1	1.350	4	0.428	4	0.681	4
look-n-grab-4-3-1	1.567	4	0.512	4	0.794	4
look-n-grab-8-1-1	182.36	554	109.17	145	51.276	99
look-n-grab-8-1-2	-	AB	-	AB	396.164	144
look-n-grab-8-2-1	104.253	314	28.84	48	48.412	73

Table 5: Challenging Domains (time in *seconds*, AB-Out of Memory)

heuristic is not of use here since the initial belief state is small (in size). On the other hand, this implies that the size of the problem obtained by the transtaiton of t0 is small, i.e., the input to FF by t0 is small. This could be the reason that t0 performs much better in this domain.

5.4 Mixed Domains

The fourth test suite contains some domains from the distribution of CFF and t0, such as the ring, safe, and logistics domains, and were used in previous IPCs. In the ring domain, one can move in a cyclic fashion (either forward or backward) around a n-room building to lock windows. Each room has a window and the window can be locked only if it is closed. The uncertainty lies in the lack of knowledge about the

Instance	# Actions		# Propositions		# Goal	oneof	
	Theoretical Number	After Simplification	Theoretical Number	After Simplification		Theoretical Number	After Simplification
uts-cycle-03	2	2	24	24	3	3×2^3	3×2^3
uts-cycle-04	2	2	40	40	4	4×2^4	4×2^4
uts-cycle-05	2	2	60	60	5	5×2^5	5×2^5
uts-cycle-10	2	2	60	60	10	10×2^{10}	10×2^{10}
forest-02	1612	1276	296	296	1	2^2	2
forest-03	2416	1912	454	454	1	3^2	3
forest-04	3220	2548	628	628	1	4^2	4
raos-keys-02	15	13	27	27	2	2^9	2^9
raos-keys-03	27	24	47	47	3	2^{13}	2^{13}

Table 6: Characteristics of the IPC-06 Domains

Instance	CPA(H)		t0		DNF	
	Time	Len	Time	Len	Time	Len
uts-cycle-03	0.005	3	0.136	3	0.009	3
uts-cycle-04	0.027	6	0.448	7	0.043	6
uts-cycle-05	0.123	10	1.844	10	0.160	10
uts-cycle-10	34.609	71	-	AB	85.194	89
uts-cycle-12	-	AB	-	AB	895.534	134
forest-02	25.947	84	0.124		3.055	55
forest-03	-	AB	0.624		-	TO
forest-08	-	AB	64.488		-	TO
raos-keys-02	0.26	32	0.016		0.095	39
raos-keys-03	4.207	152	0.216		0.801	153

Table 7: IPC-06 Domains (time in *seconds*, AB-Out of Memory)

initial state of the windows. The goal is to have all windows locked. In the `safe` domain, a safe has one out of n possible combinations, and one must try all combinations in order to open the safe. The `logistics` domain is the 'incomplete version' of the well-known logistics domain. The uncertainty is in the initial position of each package within its origin city. We add the `cleaner` domain to this test suite. It is a modified version of the `ring` domain. The difference is that instead of locking the window, the robot has to clean objects. Each room has p objects to be cleaned. Initially, the robot is at the first room and does not know whether or not objects are cleaned. The goal is to have all objects cleaned. The properties of these benchmarks are summarized in Table 8.

Table 9 reports the results of our experiment with well-known domains from previous IPCs, available from CFF's test suite. In these domains, CFF provides the best results. However, it times out in `safe-50`, while both `t0` and CPA(H) can solve this instance. CPA(H) and POND provide comparable performance, though CPA(H) can solve a larger number of instances.

6 Conclusions and Future Work

In this paper, we presented the complete design and implementation of an efficient conformant planner, called CPA(H). In particular, we discussed the theoretical basis of two main techniques that help CPA(H) achieves the level of performance exhibited in the 2008 planning competition.

We would like to note that we have conducted a preliminary investigation on the usefulness of the proposed techniques in other planners and the results are promising. In particular, applying the `one-of` combination technique could improve the performance of POND and employing the goal-splitting technique allows CFF to scale up [24]. This suggests that these two techniques could be useful in the development of conformant planners.

The future developments of this project include exploring whether alternative methods for the internal implementation of cs-states (e.g., OBDD) can further enhance performance and whether the proposed techniques could be strengthened to deal with domains where the current techniques are inapplicable. For example, the current simplification techniques do not take into consideration about the initial states. This could be a source for improvement. Another interesting question would be the applicability of other heuristics.

Finally, let us observe that we did not focus on the efficiency of the preprocessor in this version of CPA(H). More precisely, the current implementation of the preprocessor is composed of fairly unoptimized fixpoint computations—and as such it is not particularly fast (especially for certain large instances, as in the

comm domain). An optimized code for this module could improve the overall performance of CPA(H).

References

[1] 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.

[2] Piergiorgio Bertoli, Alessandro Cimatti, and Marco Roveri. Heuristic search + symbolic model checking = efficient conformant planning. In Bernhard Nebel, editor, *IJCAI*, pages 467–472. Morgan Kaufmann, 2001.

[3] B. Bonet and B. Givan. Results of the conformant track of the 5th planning competition, 2006. http://www.ldc.usb.ve/~bonet/.

[4] Ronen Brafman and Jörg Hoffmann. Conformant planning via heuristic forward search: A new approach. In Sven Koenig, Shlomo Zilberstein, and Jana Koehler, editors, *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-04)*, pages 355–364, Whistler, Canada, 2004. Morgan Kaufmann.

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

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

[7] Daniel Bryce and Subbarao Kambhampati. Heuristic Guidance Measures for Conformant Planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*, pages 365–375. AAAI, 2004.

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

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

[10] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *J. Artif. Intell. Res. (JAIR)*, 22:215–278, 2004.

[11] Derek Long and Maria Fox. Efficient implementation of the plan graph in stan. *Journal of Artificial Intelligence Research*, 10:87–115, 1999.

[12] X.L Nguyen, S. Kambhampati, and R. Nigenda. Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *Artificial Intelligence*, 135(1-2):73–123, 2002.

[13] 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*, 2006.

[14] 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*, 2007.

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

[16] Ronald P. A. Petrick and Fahiem Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, April 23-27, 2002, Toulouse, France*, pages 212–222. AAAI, 2002.

[17] Ronald P. A. Petrick and Fahiem Bacchus. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the Sixth International Conference on Automated Planning and Scheduling, 2004*, pages 2–11, 2004.

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

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

[20] 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.

[21] Tran Cao Son, Phan Huy Tu, Michael Gelfond, and Ricardo Morales. An Approximation of Action Theories of $\mathcal{AL}$ and its Application to Conformant Planning. In *Proceedings of the the 7th International Conference on Logic Programming and NonMonotonic Reasoning*, pages 172–184, 2005.

[22] 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.

[23] Son Thanh To, Enrico Pontelli, and Tran Cao 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*. AAAI, 2009.

[24] Dang-Vien Tran, Hoang-Khoi Nguyen, Enrico Pontelli, and Tran Cao 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.

[25] Phan Huy Tu. *Reasoning AND Planning With Incomplete Information In The Presence OF Static Causal Laws*. PhD thesis, New Mexico State University, 2007.

[26] Phan Huy Tu, Tran Cao Son, and Chitta Baral. Reasoning and Planning with Sensing Actions, Incomplete Information, and Static Causal Laws using Logic Programming. *Theory and Practice of Logic Programming*, 7:1–74, 2006.

Appendix: Experimental Results

This appendix contains the complete results for the IPC-05 domains. In all the table, time is in *seconds*, TO indicates *Time out* (30 min), AB means *Out of Memory*, and /NA stands for *Not Applicable*.

Instance	# Actions		# Propositions		# Goal	`oneof`	
	Theoretical Number	After Simplification	Theoretical Number	After Simplification		Theoretical Number	After Simplification
ring-2	3	3	16	8	2	18	18
ring-3	4	4	24	12	3	81	81
ring-4	4	4	32	16	4	324	324
ring-5	4	4	40	20	5	1215	1215
safe-5	5	5	6	6	1	5	5
safe-10	10	10	11	11	1	10	10
safe-30	30	30	31	31	1	30	30
safe-50	50	50	51	51	1	50	50
logistics-2-2-2	21296	30	429	37	2	4	2
logistics-2-3-3	108086	90	969	969	3	8	2
logistics-3-2-2	52136	56	672	672	2	9	3
logistics-3-3-3	200000	120	1320	1320	3	27	3
logistics-4-3-3	340676	156	1725	1725	3	64	4
bomb-5-1	42	6	24	24	5	2^5	2
bomb-10-1	132	11	44	44	10	2^{10}	2
bomb-20-1	462	21	84	84	20	2^{20}	2
bomb-50-1	2652	51	204	204	50	2^{50}	2
bomb-100-1	10302	101	404	404	100	2^{100}	2
bomb-5-5	110	30	40	40	5	2^5	2
bomb-10-5	240	55	60	60	10	2^{10}	2
bomb-20-5	650	105	100	100	20	2^{20}	2
bomb-50-5	3080	255	220	220	50	2^{50}	2
bomb-5-10	240	60	60	60	5	2^5	2
bomb-10-10	420	110	80	80	10	2^{10}	2
bomb-20-10	930	210	120	120	20	2^{20}	2
bomb-50-10	3660	510	240	240	50	2^{50}	2
bomb-100-10	12210	1010	404	404	100	2^{100}	2
cleaner-2-5	51	12	70	70	10	2^{10}	2
cleaner-2-10	146	22	180	180	20	2^{20}	2
cleaner-2-20	486	42	550	550	40	2^{40}	2
cleaner-2-50	2706	102	2860	2860	100	2^{100}	2
cleaner-5-5	102	27	130	130	25	2^{25}	2
cleaner-5-10	227	52	270	270	50	2^{50}	2
cleaner-5-20	627	102	700	700	100	2^{100}	2
cleaner-5-50	3027	252	3190	3190	250	2^{250}	2

Table 8: Characteristics of the Mixed Domains

Instance	CPA(H)		t0		CFF		POND		KACMBP		DNF	
	Time	Len	Time	Len	Time	Len	Time	Len	Time	Len	Time	Len
ring-2	0.012	7	0.016	5	.0	7	.01	6	0.00	5	0.16	7
ring-3	0.128	8	0.012	8	.11	15	.12	13	0.00	8	0.136	11
ring-4	0.199	17	0.02	13	2.0	25	4.56	16	0.02	11	0.228	15
ring-5	0.585	21	0.02	17	46	45	282	20	0.02	14	0.812	19
safe-05	0.142	5	0.004	5	.0	5	.02	10	.02	5	0.144	5
safe-10	0.160	10	0.02	10	.01	10		TO	.02	10	0.13	10
safe-30	2.215	30	0.088	30	4.26	30		TO	.01	30	0.228	30
safe-50	19.505.591	50	0.224	50		TO		TO	.03	50	0.597	100
logistics-2-2-2	0.252	27	0.02	16	.04	16	.13	16	.276	14	0.234	17
logistics-2-3-3	1.074	27	0.048	24	.06	24	1.4	30	164.94	34	1.049	55
logistics-3-2-2	0.575	23	0.024	20	.09	20	.78	22	.112	14	0.59	34
logistics-3-3-3	2.184	51	0.06	34	.11	34	18.48	39	.220	40	2.638	104
logistics-4-3-3	3.298	55	0.072	36	.17	37	29.55	37	.288	39	3.52	193
bomb-5-1	0.07	0.138	1	.0	9	.03	9	.02	10	0.075		
bomb-10-1	0.153	19	0.012	20	.01	19	.13	19	.036	20	0.055	
bomb-50-1	0.604	99	0.076	100	2.8	99		AB	.034	100	0.496	
bomb-100-1	1.737	199	0.236		84.03	199		AB	2.48	200	1.372	
bomb-20-1	0.181	39	0.024	40	.13	39	1.13	39	.04	40	0.106	
bomb-5-5	0.154	5	0.02	10	.0	5	.12	5	.072	10	0.097	5
bomb-10-5	0.194	15	0.02	20	.01	15	.89	15	.148	20	0.11	15
bomb-20-5	0.42	35	0.036	40	.09	35	4.38	35	.188	40	0.293	35
bomb-50-5	2.64	95	0.116	100	2.38	95		AB	1.00	100	1.491	95
bomb-5-10	0.189	5	0.016	10	.0	5	.48	5	.18	10	0.121	5
bomb-10-10	0.284	10	0.028	20	.0	10	1.69	10	.19	20	0.198	10
bomb-20-10	0.717	30	0.052	40	.06	30	11.85	30	.56	40	0.598	30
bomb-50-10	3.184	90	0.0176	100	. 1.89	90		AB	3.04	100	3.937	90
bomb-100-10	17.9	190	0.736	200	71.53	190		AB	20.26	200	4.010	190
bomb-100-100		AB	6.260	200	1.92	100		AB		TO		TO
cleaner-2-5	0.0	11	0.008	11	.0	11	.09	11	.04	11	0.078	11
cleaner-2-10	0.08	21	0.012	21	.0	21	.83	21	.14	21	0.072	21
cleaner-2-20	0.14	41	0.012	41	.04	41	8.99	41	.4	41	0.012	41
cleaner-2-50	0.937	101	0.032	101	.37	101		AB	7.61	101	0.148	101
cleaner-5-5	0.06	29	0.016	29	.0	29	.86	29	.116	34	0.086	33
cleaner-5-10	0.14	54	0.028	54	.03	54	6.14	54	.59	56		TO
cleaner-5-20	0.115	104	0.052	5	.33	103	113.85	104	4.56	106		TO
cleaner-5-50	2.957	254	0.013	254	9.86	254		AB	128.78	256		TO

Table 9: Mixed Domains

Instance	CPA(H)	t0	CFF	POND	KACMBP	DNF
coins-01	0.02/11	0.08/10	0/12	0.02/11	1.476/15	0.21/10
coins-02	0.006/11	0.016/10	0/12	0.03/11	1.424/9	0.19/10
coins-03	0.0006/14	0.016/10	0/13	0.04/13	1.3/15	0.19/11
coins-04	0.006/11	0.012/10	0/12	0.02/11	1.272/9	0.19/10
coins-05	0.006/11	0.012/10	0.01/13	0.04/13	1.592/16	0.22/11
coins-06	0.030/45	0.04/28	0.04/31	0.38/31	/TO	0.21/31
coins-07	0.031/48	0.024/26	0.12/34	0.7/38	/TO	0.20/27
coins-08	0.045/49	0.032/28	0.03/28	0.26/29	/TO	0.226/27
coins-09	0.029/45	0.028/26	0.02/26	0.42/28	/TO	0.204/26
coins-10	0.031/48	0.036/26	1.02/38	1.07/46	/TO	0.205/27
coins-11	0.035/188	0.080/74	1.58/78	14.91/88	/TO	0.621/74
coins-12	0.326/194	0.092/67	1.39/72	11.04/76	/TO	0.576/68
coins-13	0.359/195	0.096/68	6.13/95	24.86/94	/TO	0.547/75
coins-14	0.383/198	0.096/76	1.07/76	17.62/88	/TO	0.655/75
coins-15	0.396/191	0.100/79	7.35/79	21.1/124	/TO	0.537/67
coins-16	0.857/202	0.216/113	64.67/145	145.96/136	/TO	1.013/101
coins-17	0.925/207	0.144/96	2.21/94	69.08/120	/TO	1.028/110
coins-18	1.292/204	0.116/97	11.37/118	61.76/115	/TO	0.978/100
coins-19	0.736/205	0.172/105	32.14/128	151.56/147	/TO	0.901/93
coins-20	0.737/195	0.152/107	38.19/143	211.19/153	/TO	0.970/99

Table 10: IPC-05 coins domains

Instance	CPA(H)	t0	CFF	POND	KACMBP	DNF
sortnet-01	0.061/1	/NA	/NA	0.01/1	0.012/1	0.064/1
sortnet-02	0.031/3	/NA	/NA	0/3	0.02/3	0.034/3
sortnet-03	0.032/5	/NA	/NA	0/5	0.056/5	0.036/6
sortnet-04	0.046/9	/NA	/NA	0.01/9	0.092/9	0.051/10
sortnet-05	0.068/13	/NA	/NA	0.01/12	0.160/13	0.082/11
sortnet-06	0.099/16	/NA	/NA	0.02/16	0.328/19	0.108/21
sortnet-07	0.206/21	/NA	/NA	0.01/19	0.316/17	0.231/28
sortnet-08	0.405/28	/NA	/NA	0.02/26	0.616/27	0.368/36
sortnet-09	0.945/32	/NA	/NA	/AB	0.956/28	1.006/44
sortnet-10	2.373/39	/NA	/NA	0.05/38	1.708/41	1.447/54
sortnet-11	5.989/46	/NA	/NA	0.07/43	2.348/36	2.008/65
sortnet-12	15.392/50	/NA	/NA	0.13/50	4.00/49	6.058/75
sortnet-13	39.766/56	/NA	/NA	0.18/55	5.564/48	8.486/90
sortnet-14	97.778/65	/NA	/NA	0.27/61	9.784/30	27.230/103
sortnet-15	740.00/74	/NA	/NA	0.28/65	13.900/46	35.303/118

Table 11: IPC-05 sortnet domains

Instance	CPA(H)	t0	CFF	POND	KACMBP	DNF
comm-02	0.087/17	0.03/19	0/17	0.04/17	/TO	0.21/19
comm-03	0.091/23	0.024/26	0/23	0.1/23	/TO	0.181/26
comm-04	0.134/29	0.016/33	0/29	0.12/29	/TO	0.259/36
comm-05	0.178/35	0.028/40	0.01/35	0.21/35	/TO	0.309/45
comm-06	0.201/41	0.032/47	0/41	0.32/41	/TO	0.404/52
comm-07	0.327/47	0.036/54	0.02/47	0.54/47	/TO	0.53/59
comm-08	0.406/53	0.036/61	0.04/53	0.72/53	/TO	0.677/66
comm-09	0.568/59	0.040/68	0.04/59	1.06/59	/TO	0.855/73
comm-10	0.662/65	0.056/75	0.06/65	1.46/65	/TO	1.104/80
comm-11	0.918/71	0.072/82	0.08/71	2.51/72	/TO	1.402/87
comm-12	1.137/77	0.088/89	0.12/77	3.15/77	/TO	1.806/104
comm-13	1.471/83	0.096/96	0.15/83	5.22/83	/TO	2.275/101
comm-14	1.900/89	0.096/103	0.18/89	9.42/89	/TO	2.777/108
comm-15	2.290/95	0.092/110	0.22/95	23.34/98	/TO	3.426/125
comm-16	2.144/119	0.152/138	0.54/119	221.86/119	/TO	6.219/160
comm-17	3.004/149	0.192/173	1.68/149	/TO	/TO	16.259/190
comm-18	11.327/179	0.26/208	3.59/179	/TO	/TO	41.138/213
comm-19	24.690/209	0.35/243	7.37/209	/TO	/TO	78.956/248
comm-20	56.878/239	0.48/278	13.33/239	/ TO	/TO	149.105/296
comm-21	194.666/269	0.60/313	22.47/269	/TO	/TO	278.929/357
comm-22	329.015/299	0.80/348	35.69/299	/TO	/TO	452.738/388
comm-23	531.432/329	1.01/418	53.82/329	/TO	/TO	775.246/439
comm-24	796.437/359	1.24/453	77.12/359	/TO	/TO	1167.18/469
comm-25	1222.63/389	1.55/453	109.49/389	/ TO	/TO	1797.77/501

Table 12: IPC-05 comm domains

Instance	CPA(H)	t0	CFF	POND	KACMBP	DNF
uts-01	0.002/4	0.012/5	0/4	0/4	0.024/4	0.082/4
uts-02	0.007/81	0.012/11	0/10	0.02/12	0.2/11	0.154/11
uts-03	0.029/92	0.036/17	0.02/16	0.11/19	12.108/25	0.197/17
uts-04	0.082/111	0.064/23	0.09/22	0.25/26	/TO	0.286/24
uts-05	0.330/86	0.132/29	0.34/28	0.74/33	/TO	0.473/31
uts-06	0.714/134	0.192/35	1.21/34	1.76/40	/TO	0.648/38
uts-07	3.296/126	0.288/41	3.73/40	3.88/47	/TO	0.994/45
uts-08	3.891/129	0.424/47	10.86/46	7.8/54	/TO	1.412/52
uts-09	11.76/155	0.636/53	23.72/52	15/61	/TO	1.978/59
uts-10	14.33/194	0.660/59	55.49/58	26.16/68	/TO	2.656/66
uts-11	0.002/4	0.004/5	0/4	0.01/4	/TO	0.073/4
uts-12	0.007/16	0.02/13	0/11	0.03/14	/TO	0.137/13
uts-13	0.042/24	0.036/26	0.01/17	0.09/23	/TO	0.124/26
uts-14	0.083/34	0.006/30	0.01/23	0.34/33	/TO	0.203/43
uts-15	0.204/47	0.07/44	0.04/29	1.19/46	/TO	0.25/59
uts-16	0.316/59	0.14/55	0.1/35	AB	/TO	0.363/69
uts-17	0.879/71	0.188/70	0.2/41	7.84/64	/TO	0.558/79
uts-18	2.003/88	0.252/80	0.43/47	18.19/70	/TO	0.794/89
uts-19	4.375/104	0.504/93	0.85/53	45.16/82	/TO	1.131/99
uts-20	8.936/120	0.560/97	1.64/59	111.54/88	/TO	1.651/109
uts-23	0.022/23	0.036/19	0.01/17	0.08/17	/TO	0.123/19
uts-24	0.058/33	0.048/25	0.04/26	0.25/24	/TO	0.185/27
uts-25	0.185/50	0.092/30	1.51/33	0.72/32	/TO	0.245/39
uts-26	0.398/51	0.140/37	9.56/38	AB	/TO	0.320/41
uts-27	0.968/57	0.236/47	2.5648	4.45/53	/TO	0.479/48
uts-28	1.949/100	0.320/51	8.17/48	11.25/54	/TO	0.719/59
uts-29	3.302/112	0.408/58	7.76/62	18.73/64	/TO	0.953/66
uts-30	4.905/109	0.792/65	25.65/66	39.88/68	/TO	1.390/73

Table 13: IPC-05 uts domains