Generating None-Plans in Order to Find Plans*

M. Knapik[1], A. Niewiadomski[2], and W. Penczek[1,2]

[1] Institute of Computer Science, PAS, Warsaw, Poland
{mknapik,penczek}@ipipan.waw.pl
[2] University of Natural Sciences and Humanities, ICS, Siedlce, Poland
artur.niewiadomski@uph.edu.pl

Abstract We put forward a brand new approach to planning. The method aims at simplifying the task of planning in an abstract object-oriented domain where entities are added only and never removed. Our approach is based on the synthesis of a family of all sets of actions that cannot be composed into a plan (called none-plans) in order to prune the state space searched for plans. We show how to build a propositional formula describing a set of the none-plans and how to approximate this set when the task of planning becomes too complex. A preliminary evaluation of the application of the none-plans synthesis to the generation of plans in the PlanICS framework is shown. The experimental results show a high potential of the approach.

1 Introduction

Planning tasks appear in many applications of Artificial Intelligence [14], for example in automated composition of web services [13,9] or in automated organization of the robot activities required to achieve a desired goal [5]. It is known that planning in a domain involving objects is a difficult computational problem, which belongs to the class of NP-hard problems [10].

In this paper we put forward a brand new method aiming at simplifying the planning process by applying an abstraction to the planning problem. The presented results are inspired by the research of the authors in the field of approximative parametric model checking [7,8]. Our method reduces the number of sets of actions to be considered while looking for plans and improves the efficiency of their automated selection. Informally, the idea is as follows. We translate the planning problem to the abstract affine planning problem in such a way that each action corresponds to some abstract action, and each plan corresponds to some abstract plan, but not necessarily the other way round. This means that in the abstract planning domain there could be abstract plans that do not correspond to any plan in the original planning domain. However, if one identifies a set of abstract actions which cannot be composed into an abstract plan (call it a none-plan), then the actions corresponding to these abstract actions also cannot

* This work has been partly supported by the National Science Centre under the grant No. 2011/01/B/ST6/01477. Michał Knapik is supported by DEC-2012/07/N/ST6/03426 NCN Preludium 4 grant.

be composed into a plan. Our aim is to characterize none-plans in the abstract planning domain in order to reduce the search of plans to these sets of actions that do not correspond to none-plans.

Our abstract planning domain has been selected in such a way that it is affine, i.e., the states and the operations can be represented as vectors of natural numbers, and the problem of finding a single plan can be solved in a polynomial time. To this order we assume that there is no consumption of objects, so that an increase of their number is visible only. There are several other reasons motivating our choices:

- More complicated plans can be divided into smaller fragments, where the objects are not consumed;
- Consumption of objects can be encoded by modifying their attributes (without removing them);
- In many programming languages, a garbage collector is activated only after some limit has been reached, as objects are collected in a dynamic way and removed altogether in a specified moment of time;
- There are planners exploiting directly affine planning domains, where actions cannot remove objects (see Planics [2]).

We show how to build a propositional formula describing a set of the none-plans and how to approximate this set when the task of planning becomes too complex. A preliminary evaluation of the application of the none-plans synthesis to the SMT-based generation of plans [10,11] in the Planics framework [2] is discussed. The experimental results of the presented framework for the synthesis of all the none-plans suggest that this task is of acceptable average complexity[1] and show a high potential of the approach. In most of the experiments the total time of finding the first plan and all the plans is much shorter when the SMT-solver gets also a formula blocking all the none-plans.

The rest of the paper is organized as follows. In the next subsection we discuss the related work. Section 2 introduces an abstract planning domain, while the none-plan synthesis is described in Section 3. The experimental results are discussed in Section 4. Section 5 concludes the paper.

1.1 Related Work

Since decades automatic planning is in the scope of interest of a broad scientific community. Currently, one of the leading standards in this field is the PDDL language [3]. However, due to the high complexity of the planning problem there are number of abstraction-based approaches to deal with it. For example, in [12] Nourbakhsh proposes an abstraction enabling interleaved planning and action execution. The paper [4] introduces the Dynamic Abstract Planning (DAP) technique that improves the efficiency of state-enumeration planners for real-time embedded systems. The intuition behind DAP is similar to the case of the abstract planning stage in the web service composition framework Planics [2]

[1] Pessimistic complexity is still prohibitive.

– sometimes certain world features are just not important. On the other hand, DAP allows for the application of different levels of abstraction in different parts of the search space.

Nevertheless, many of the existing approaches concentrate either on (1) *action-based* or (2) *state-based* abstraction [12], while our solution combines both: (1) we treat equally (to some point) multiple executions of the same actions, and (2) we do not distinguish between object instances, while we are interested only in their types and total count. Moreover, to our best knowledge, there is no approach exploiting the none-plans concept.

2 Abstract Planning Domain

As we outlined in the introduction, our task is to provide at least a partial characterization of the collection of the sets of actions such that a plan cannot be composed from any of these sets, i.e., we aim at characterizing none-plans. The idea is strongly influenced by some practical applications, as we have been looking for a method to optimize an abstract phase of service composition in an object-oriented domain. This is the reason that we preserve an object-oriented nomenclature in this paper.

In the abstract planning domain introduced in this section we only track the quantity of the instances of each type and the services are modeled by the actions that transform the current world state solely by adding objects. The actions prerequisites are rather rudimentary, as the action is enabled only in the worlds where certain resources are provided and can only add new objects when executed. The strength of this abstraction lies in its simplicity. As we show, the proposed domain is affine, thus it can be explored and analysed using many AI-related approaches. In this work we lay the groundwork for the approach based on formal methods. We start with introducing the basic concepts of the theory.

We assume a simple, set-theoretic understanding of the notion of a class and the class inheritance. Let $\mathcal{A}$ be a finite set of attributes and $\mathcal{U} \subseteq 2^{\mathcal{A}}$. The pair $\mathcal{H} = (\mathcal{U}, \subseteq)$ is called a *Class Hierarchy* and the elements of $\mathcal{U}$ are called *Classes*. Let $u, u' \in \mathcal{U}$. If $u \subseteq u'$ then we say that u' *extends* u. If for every $u'' \in \mathcal{U}$ we have $u \subseteq u'' \subseteq u' \implies u'' = u' \vee u'' = u$, then u' is called a *successor* of u. The set of all the successors of u is denoted by $\mathrm{succ}(u)$. The set of the *ancestors* of u is defined as $\mathrm{anc}(u) = \{u' \in U \mid u' \neq u \wedge u' \subseteq u\}$.

Example 1. Let $\mathcal{A} = \{\mathit{seatNo}, \mathit{trunkSize}, \mathit{waterSpeed}, \mathit{roadSpeed}\}$ and $\mathcal{H} = (\mathcal{U}, \subseteq)$ be a hierarchy such that $\mathcal{U} = \{\mathit{Body}, \mathit{Car}, \mathit{Boat}, \mathit{Amphibian}\}$, where $\mathit{Body} = \{\mathit{seatNo}, \mathit{trunkSize}\}$, $\mathit{Car} = \mathit{Body} \cup \{\mathit{roadSpeed}\}$, $\mathit{Boat} = \mathit{Body} \cup \{\mathit{waterSpeed}\}$, and $\mathit{Amphibian} = \mathit{Body} \cup \{\mathit{roadSpeed}, \mathit{waterSpeed}\}$. We have $\mathrm{anc}(\mathit{Amphibian}) = \{\mathit{Body}, \mathit{Car}, \mathit{Boat}\}$, $\mathrm{anc}(\mathit{Car}) = \mathrm{anc}(\mathit{Boat}) = \{\mathit{Body}\}$, and $\mathrm{anc}(\mathit{Body}) = \emptyset$. We depict the inheritance relation in Fig. 1, where each arrow head points to the class being extended.

In what follows let us assume a fixed hierarchy $\mathcal{H} = (\mathcal{U}, \subseteq)$. A function $\omega \colon \mathcal{U} \to \mathbb{N}$ is called an *abstract world* and we denote the set of all the abstract

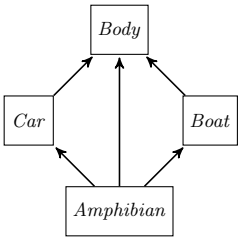

Figure 1: A simple class hierarchy.

worlds by $\mathcal{W}_{\mathcal{H}}$. An abstract world associates a class with the total number of its instances. A world is transformed by adding objects. By $\omega + k \cdot u$ we mean the addition of k instances of the class u to the world ω, as defined below.

Definition 1. *Let $\omega \in \mathcal{W}_{\mathcal{H}}$, $u \in \mathcal{U}$, and $k \in \mathbb{N}$. We define the following function $\omega + k \cdot u \colon \mathcal{U} \to \mathbb{N}$:*

$$(\omega + k \cdot u)(u') = \begin{cases} \omega(u') + k & \text{if } u' = u \text{ or } u' \in \mathrm{anc}(u), \\ \omega(u') & \text{otherwise.} \end{cases}$$

Intuitively, $\omega + k \cdot u$ means extending ω with k instances of u and k instances of each ancestor of u. This way we encode the part of the hierarchy of classes (rooted at u) in ω. Observe that we have $(\omega + k \cdot u) + k' \cdot u' = (\omega + k' \cdot u') + k \cdot u$ for all $u, u' \in \mathcal{U}$ and $k, k' \in \mathbb{N}$, therefore we can omit the brackets whenever it is convenient. In what follows let ω^0 denote the abstract world such that $\omega^0(u) = 0$ for all $u \in \mathcal{U}$.

Example 2. If $\mathcal{H}$ is the hierarchy from Example 1, then we have:

$$\omega^0 + 1 \cdot \mathit{Amphibian} + 2 \cdot \mathit{Car} = \begin{cases} \mathit{Body} \mapsto 3, \\ \mathit{Car} \mapsto 3, \\ \mathit{Boat} \mapsto 1, \\ \mathit{Amphibian} \mapsto 1. \end{cases}$$

To unify the notation we also use the abstract worlds to represent the actions constraints and results. To this end we extend the addition operation as follows. Let $\omega, \omega' \in \mathcal{W}_{\mathcal{H}}$ and $\omega' = \{(u_1, k_1), (u_2, k_2), \ldots, (u_n, k_n)\} \subseteq \mathcal{U} \times \mathbb{N}$ for some $n \in \mathbb{N}$. Now, we define $\omega + \omega' := \omega + k_1 \cdot u_1 + k_2 \cdot u_2 + \ldots + k_n \cdot u_n$. For any relation $\sim \in \{\leq, <, =, >, \geq\}$ we write $\omega \sim \omega'$ if $\forall_{u \in \mathcal{U}}\ \omega(u) \sim \omega'(u)$. In particular, if $\omega \geq \omega'$, then we say that ω *covers* ω'. We say that $\omega \in \mathcal{W}_{\mathcal{H}}$ is *realisable* iff for some $n \in \mathbb{N}$ there exist $k_1, k_2, \ldots, k_n \in \mathbb{N}$ and $u_1, u_2, \ldots, u_n \in \mathcal{U}$ such that $\omega = \omega^0 + \sum_{i=1}^{n} k_i \cdot u_i$. Intuitively, ω is realisable if it can be built from the empty world by adding new objects. All the transformations of the abstract worlds considered in this section preserve realisability.

Definition 2. *Let* $\mathrm{pre}, \mathrm{eff} \in \mathcal{W}_{\mathcal{H}}$*, where* pre *is realisable. An ordered pair* $\mathrm{act} = (\mathrm{pre}, \mathrm{eff})$ *is called an* action*,* pre *is its* enabling condition *and* eff *is its* effect*.*

We also use the notation pre(act) and eff(act) to denote the enabling condition and the effect of act, respectively. An action act can be executed in $\omega \in \mathcal{W}_\mathcal{H}$ iff $\omega \geq \text{pre(act)}$; the result of its execution is $\omega' = \omega + \text{eff(act)}$. This is denoted by $\omega \xrightarrow{\text{act}} \omega'$. Next, we introduce planning domains system used in this paper.

Definition 3 (Planning Domain). *By a* Planning Domain *we mean a four-tuple* $\mathcal{P} = (\mathcal{W}_\mathcal{H}, F_I, F_G, Act)$*, where:*

- $\mathcal{W}_\mathcal{H}$ *is the set of the abstract worlds,*
- F_I, F_G *are finite sets of the* initial *and the* final *abstract worlds, respectively, where all the worlds from* F_I *are realisable,*
- $Act = \{\text{act}_1, \ldots, \text{act}_k\}$*, where* $k \in \mathbb{N}$*, is a finite set of actions.*

Let us assume throughout this section that we have a fixed planning domain $\mathcal{P} = (\mathcal{W}_\mathcal{H}, F_I, F_G, Act)$. The following concepts are needed to present the theory of plan synthesis. A *path* is a sequence $(\omega_0, \text{act}_1, \omega_1, \text{act}_2, \ldots, \text{act}_n, \omega_n)$ for some $n \in \mathbb{N}$, such that $\forall_{0 \leq i \leq n} \omega_i \in \mathcal{W}_\mathcal{H}$ and $\forall_{0<i\leq n}(\text{act}_i \in Act \wedge \omega_{i-1} \xrightarrow{\text{act}_i} \omega_i)$. One can observe that a path is uniquely determined by its first world and the sequence of actions, thus we can introduce a convenient short-hand notation $\pi = \omega_0 \text{act}_1 \circ \text{act}_2 \circ \ldots \circ \text{act}_n$. By $|\pi| = n$ we mean that the length of the path π is n and by $\pi_i = \omega_i$ we denote its i-th state for $0 \leq i \leq n$. The set $Acts(\pi) \subseteq Act$ consists of all the actions of π; we call it the *support* of π. For $A \subseteq Act$ and $\omega, \omega' \in \mathcal{W}_\mathcal{H}$, we denote by $\Pi(\omega, A, \omega')$ the set of all the paths π starting from ω, such that $Acts(\pi) \subseteq A$ and the final state of π covers ω, i.e., $\pi_{|\pi|} \geq \omega$. By an *abstract plan* we mean a path $\pi \in \Pi(\omega_I, A, \omega_F)$, where $\omega_I \in F_I$ and $\omega_F \in F_G$. Notice that an abstract plan is a path that starts in an initial state and covers a final state.

The notions introduced above have their convenient vector counterparts. We employ a slight notational abuse and use the same symbol for the vector addition and the addition of abstract worlds. It is easy to identify the operation from the type of the operands. Assume that $\mathcal{U} = \{u_1, \ldots, u_n\}$ for some $n \in \mathbb{N}$ and $\prec$ is a fixed linear order on $\mathcal{U}$ such that $u_1 \prec \ldots \prec u_n$. A vector $\boldsymbol{\omega} \in \mathbb{N}^n$ satisfying $\forall_{1 \leq i \leq n}\ \boldsymbol{\omega}^i = \omega(u_i)$ is the *vector representation* of $\omega \in \mathcal{W}_\mathcal{H}$. The arithmetic relations between the abstract worlds are naturally extended to their vector representations using the lexicographical ordering of vectors. Let $u \in \mathcal{U}$ and define $e_u \in \mathbb{N}^n$ such that:

$$e_u^i = \begin{cases} 1 & \text{if } u_i \in \text{anc}(u) \text{ or } u_i = u, \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to see that $\omega + k \cdot u$ is represented by $\boldsymbol{\omega} + k \cdot e_u$ for any $k \in \mathbb{N}$. To extend this to the general case of the addition of abstract worlds, assume that $\omega, \omega' \in \mathcal{W}_\mathcal{H}$ and $\omega' = \{(u_1, k_1), (u_2, k_2), \ldots, (u_n, k_n)\} \subseteq \mathcal{U} \times \mathbb{N}$. The vector representation of the sum $\omega + \omega'$ is given by $\boldsymbol{\omega} + \boldsymbol{\omega'} := \boldsymbol{\omega} + k_1 \cdot e_{u_1} + k_2 \cdot e_{u_2} \ldots + k_n \cdot e_{u_n}$. The *linear representation* of an action act = (pre, eff) consists of the linear representations of its enabling condition and the effect, and it is denoted by $\mathbf{act} = (\mathbf{pre}, \mathbf{eff})$. Consequently, we use the following notations $\text{pre}(\mathbf{act}) = \mathbf{pre}$ and $\text{eff}(\mathbf{act}) = \mathbf{eff}$.

We illustrate the concepts introduced above in the following example.

Example 3. We build an exemplary planning domain $\mathcal{P}' = (\mathcal{W}_{\mathcal{H}}, F_I, F_G, Act)$, where the hierarchy $\mathcal{H}$ is defined in Example 1 and $F_I = \{\omega_I\}$ and $F_G = \{\omega_F\}$ satisfy $\omega_I = \omega^0$, $\omega_F(Amphibian) = 1$ and $\omega_F(u) = 0$ for $u \in \mathcal{U} \setminus \{Amphibian\}$. In our domain we start with the empty world and the goal is to produce at least one *Amphibian*. To this end we can employ the actions from the set $Act = \{makeBody, makeCar, makeBoat, makeAmphibian, tinker\}$ having the following specifications:

- for each $O \in \{Body, Car, Boat, Amphibian\}$ we have $\mathrm{eff}(makeO)(O) = 1$ and $\mathrm{eff}(makeO)(u) = 0$ for $u \in \mathcal{U} \setminus \{O\}$,
- $\forall_{u \in \mathcal{U}} \mathrm{pre}(makeBody)(u) = 0$,
- $\mathrm{pre}(makeCar)(Body) = 1$ and $\forall_{u \in \mathcal{U} \setminus \{Body\}} \mathrm{pre}(makeCar)(u) = 0$,
- $\mathrm{pre}(makeBoat)(Body) = 1$ and $\forall_{u \in \mathcal{U} \setminus \{Body\}} \mathrm{pre}(makeBoat)(u) = 0$,
- $\mathrm{pre}(makeAmphibian)(Body) = 2$, $\mathrm{pre}(makeAmphibian)(Car) = 1$, $\mathrm{pre}(makeAmphibian)(Boat) = 1$, $\mathrm{pre}(makeAmphibian)(Amphibian) = 0$,
- $\mathrm{pre}(tinker)(Body) = 2$, $\mathrm{pre}(tinker)(Car) = 2$, $\mathrm{pre}(tinker)(Boat) = 1$, $\mathrm{pre}(tinker)(Amphibian) = 1$,
- $\mathrm{eff}(tinker)(Amphibian) = 2$ and $\forall_{u \in \mathcal{U} \setminus \{Amphibian\}} \mathrm{eff}(tinker)(u) = 0$.

Intuitively, a *Car* or a *Boat* can be produced from a *Body*, and an *Amphibian* from a pair of a *Car* and a *Boat* (observe that these two make two *Bodies*). The production of a *Body* does not have any prerequisites. We can also *tinker* and build two *Amphibians* from an *Amphibian* and a *Car*. Let us build the vector representation of the domain by assuming the order $Body \prec Car \prec Boat \prec Amphibian$. The initial state is represented by $\boldsymbol{\omega_I} = (0, 0, 0, 0)$ and $\boldsymbol{\omega_F} = (0, 0, 0, 1)$ is the representation of the goal state. We also have $e_{Body} = (1, 0, 0, 0)$, $e_{Car} = (1, 1, 0, 0)$, $e_{Boat} = (1, 0, 1, 0)$, and $e_{Amphibian} = (1, 1, 1, 1)$, therefore we represent the actions from Act as follows:

- $\mathrm{pre}(\boldsymbol{makeBody}) = (0, 0, 0, 0)$ and $\mathrm{eff}(\boldsymbol{makeBody}) = (1, 0, 0, 0)$,
- $\mathrm{pre}(\boldsymbol{makeCar}) = (1, 0, 0, 0)$ and $\mathrm{eff}(\boldsymbol{makeCar}) = 1 \cdot e_{Car} = (1, 1, 0, 0)$,
- $\mathrm{pre}(\boldsymbol{makeBoat}) = (1, 0, 0, 0)$ and $\mathrm{eff}(\boldsymbol{makeBoat}) = 1 \cdot e_{Boat} = (1, 0, 1, 0)$,
- $\mathrm{pre}(\boldsymbol{makeAmphibian}) = (2, 1, 1, 0)$ and $\mathrm{eff}(\boldsymbol{makeAmphibian}) = 1 \cdot e_{Amphibian} = (1, 1, 1, 1)$,
- $\mathrm{pre}(\boldsymbol{tinker}) = (2, 2, 1, 1)$ and $\mathrm{eff}(\boldsymbol{tinker}) = 2 \cdot e_{Amphibian} = (2, 2, 2, 2)$.

It is now easy to see that to create an abstract plan we need all the actions from the set $A_{\mathcal{P}} = \{makeBody, makeCar, makeBody, makeBoat, makeAmphibian\}$. The action *tinker* is redundant, as it can be fired only after the goal is enabled.

As we have already mentioned, a planning domain is an abstraction of a possibly more complex system. In particular, the fact that the objects are only produced and never consumed means that some plans found in the planning domain may have no counterpart in the original system. However, referring to Example 3, observe that as none of the proper subsets of $A_{\mathcal{P}'}$ can be a support of an abstract plan, none of them can be used for generating a plan in the underlying system. We exploit this observation in the next section.

3 Action Classification and None-Plan Synthesis

As we show, in our planning domain a greedy approach to the synthesis of the abstract plans is guaranteed to succeed, and we obtain a certain classification of the actions as a byproduct of the computations. This partition can be used for the identification of the core sequence of the sets of actions necessary in every abstract plan.

In what follows assume that we have a single initial and a single goal state, i.e., $F_I = \{\omega_I\}$ and $F_G = \{\omega_F\}$. Moreover, we can assume that $\omega_I = \omega^0$. This premise is possible without a loss of generality, as we can replace every action that has a vector representation $\mathbf{act} = (\mathbf{pre}, \mathbf{eff})$ with the action represented by $\mathbf{act} = (\mathbf{pre} - \boldsymbol{\omega_I}, \mathbf{eff})$, and the final world ω_F with a world represented by $\boldsymbol{\omega_F} - \boldsymbol{\omega_I}$. Note that as a result of this operation some negative numbers may appear in enabling conditions and the final state. This is not a problem as the translation does not change the results of the arithmetic comparisons.

Action Classification We classify the actions from a fixed set $A \subseteq Act$. Let $V_{\max}$ be the largest number present in the enabling conditions of the vector representations of all the actions from Act. For each $B \subseteq A$ we denote:

$$\mathrm{enact}(B) = \{\mathrm{act} \in A \mid \sum_{\mathrm{act}' \in B} V_{\max} \cdot \mathrm{eff}(\mathbf{act}') \geq \mathrm{pre}(\mathbf{act})\}.$$

Observe that the set $\mathrm{enact}(B)$ consists of all those actions from A that can be enabled by firing actions from B, possibly each action more than once.

As before assume that $\omega, \omega' \in \mathcal{W}_{\mathcal{H}}$. In order to partition the set A we define the ascending sequence $\{G_i^\omega\}_{i \in \mathbb{N}}$ of subsets of A such that:

$$G_0^\omega = \{\mathrm{act} \in A \mid \mathrm{act} \geq \omega\},$$
$$G_{i+1}^\omega = \mathrm{enact}(G_i^\omega) \ \text{ for } i > 0.$$

We also define the derived sequence $\{H_i^\omega\}_{i \in \mathbb{N}}$ such that $H_0^\omega = G_0^\omega$ and $H_{i+1}^\omega = G_{i+1}^\omega \setminus G_i^\omega$ for all $i \in \mathbb{N}$. The set G_{i+1}^ω consists of the actions enabled by executing all the actions G_i^ω and out of these H_{i+1}^ω selects those actions that are newly enabled. Now, let $\mathrm{klimit}(\omega) \in \mathbb{N}$ be the smallest number such that $H_{\mathrm{klimit}(\omega)}^\omega = \emptyset$; this definition is correct as the set A is finite. By $\mathrm{kgoal}(\omega, \omega') \in \mathbb{N}$ we mean the smallest number such that the effect of the actions from G_{i+1}^ω covers ω'. Formally:

$$\mathrm{kgoal}(\omega, \omega') = min(\{k \in \mathbb{N} \mid \sum_{act \in G_k^\omega} V_{\max} \cdot \mathrm{eff}(\mathbf{act}) \geq \boldsymbol{\omega'}\})$$

and if there is no such a number, then we set $\mathrm{kgoal}(\omega, \omega') = \infty$.

Lemma 1. *The following condition holds:*

$$\mathrm{kgoal}(\omega, \omega') < \infty \ \ \textit{iff} \ \ \Pi(\omega, A, \omega') \neq \emptyset.$$

The value of $\mathrm{kgoal}(\omega, \omega')$ *can be computed in time* $O(|A|^2 \cdot |\mathcal{U}|)$.

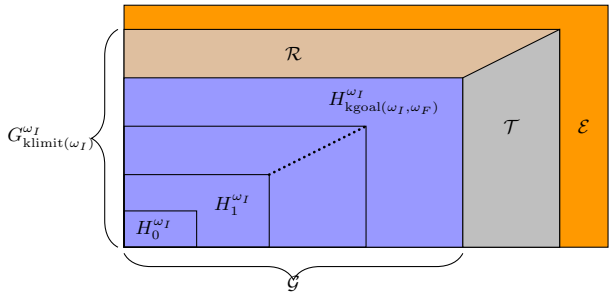

Figure 2: A partition of the actions in A.

Proof. The first part of the thesis follows immediately from the definition. A procedure for finding $\text{kgoal}(\omega, \omega')$ is based on building the sequence $\{G^{\omega}_i\}_{i=0}^{\text{klimit}(\omega)}$. In order to find $G^{\omega}_{i+1} = \text{enact}(G^{\omega}_i)$ given G^{ω}_i we need to compare the computed value $\sum_{\text{act}' \in G^{\omega}_i} V_{\max} \cdot \text{eff}(\mathbf{act}')$ with the enabling conditions of all the actions from the set $A \setminus G^{\omega}_i$. In the worst case of $\text{klimit}(\omega) = |A|$, we need $O(|A|^2)$ comparisons of the vectors of length $|\mathcal{U}|$ to build the whole sequence. □

Assume that $\text{kgoal}(\omega_I, \omega_F) < \infty$, which, from Lemma 1, is equivalent to the existence of an abstract plan. Define the following subsets of A:

- $\mathcal{E} = A \setminus G^{\omega_I}_{\text{klimit}(\omega_I)}$ - the set of the *useless* actions that cannot be enabled,
- $\mathcal{G} = G^{\omega_I}_{\text{kgoal}(\omega_I,\omega_F)}$ - the set of the actions *sufficient* to build an abstract plan,
- $\mathcal{R} = \{\text{act} \in G^{\omega_I}_{\text{klimit}(\omega_I)} \mid \text{pre}(\mathbf{act}) \geq \boldsymbol{\omega_F}\}$ - the *redundant* actions that can be enabled only after the goal is covered,
- $\mathcal{T} = G^{\omega_I}_{\text{klimit}(\omega_I)} \setminus (G^{\omega_I}_{\text{kgoal}(\omega_I,\omega_F)} \cup \mathcal{R})$ - the set of the *potentially useful* actions that are not a part of the greedily built set of the sufficient actions.

As established in the following corollary, these sets partition A, i.e., they are pairwise disjoint and their union is equal to A. We illustrate this in Fig. 2.

Corollary 1. *If* $\text{kgoal}(\omega_I, \omega_F) < \infty$, *then:*

- $\{\mathcal{E}, \mathcal{G}, \mathcal{R}, \mathcal{T}\}$ *is a partition of* A,
- $\{H^{\omega_I}_0, \ldots, H^{\omega_I}_{\text{kgoal}}(\omega_I, \omega_F)\}$ *is a partition of* $\mathcal{G}$.

Example 4. Let $\mathcal{P}$ be the planning domain from Example 3, where $V_{\max} = 2$. We perform the classification of the set of all the actions act. For clarity we omit the initial and the final states from the sub- and superscripts. The action *makeBody* is the only one that is enabled in the initial state ω_I, thus we have $H_0 = G_0 = \{\textit{makeBody}\}$. Firing *makeBody* enables the actions *makeCar* and *makeBoat*, therefore $G_1 = \{\textit{makeBody}, \textit{makeCar}, \textit{makeBoat}\}$ and $H_1 = \{\textit{makeCar}, \textit{makeBoat}\}$. Now, firing all the actions from H_1 enables the action *makeAmphibian* that is needed to cover the final state ω_F; we therefore have $\mathcal{G} = G_2 = \{\textit{makeBody}, \textit{makeCar}, \textit{makeBoat}, \textit{makeAmphibian}\}$ and

$H_2 = \{makeAmphibian\}$ and kgoal = 2. The action *tinker* can be enabled by G_2, therefore $G_3 = \{makeBody, makeCar, makeBoat, makeAmphibian, tinker\}$ and $H_3 = \{tinker\}$. No more actions can be enabled, which means that klimit = 3. The action *tinker* needs an *Amphibian*, it is therefore redundant and $\mathcal{R} = \{tinker\}$. As there are no useless actions, we have $\mathcal{E} = \emptyset$; also, there are no potentially useful actions, i.e., $\mathcal{T} = \emptyset$.

We know from Lemma 1 that the sequence $\{H_i^{\omega}\}_{i=0}^{\mathrm{kgoal}(\omega,\omega')}$ can be built in a polynomial time. In the next lemma we observe that the support of an abstract plan contains at least one action from each of the elements of this sequence. By a *simple plan* we mean a plan such that only its final state covers ω_F.

Lemma 2. *Assume that* $\mathrm{kgoal}(\omega_I, \omega_F) < \infty$ *and let* $\pi = \omega_I \mathrm{act}_1 \circ \mathrm{act}_2 \circ \ldots \circ \mathrm{act}_n$ *be a simple plan such that* $\pi \in \Pi(\omega_I, A, \omega_F)$*. There exists a prefix* $\pi' \in \Pi(\omega_I, A)$ *of* π *such that* $Acts(\pi') \subseteq \mathcal{G}$ *and* $H_i^{\omega_I} \cap Acts(\pi') \neq \emptyset$ *for all* $0 \leq i \leq \mathrm{kgoal}(\omega_I, \omega_F)$*.*

Proof. Let us start with two observations. Firstly, immediately from the definition, for each $i \in \mathbb{N}$ if $B \subseteq G_i^{\omega_I}$, then $\mathrm{enact}(B) \subseteq G_{i+1}^{\omega_I}$. Secondly, for a path $\pi = \omega_I \mathrm{act}_1 \circ \mathrm{act}_2 \circ \ldots \circ \mathrm{act}_j \circ \ldots \circ \mathrm{act}_n$ such that $\mathrm{act}_j \in \mathcal{T}$ we have $Acts(\pi) \cap H^{\omega_I}_{\mathrm{kgoal}(\omega_I,\omega_F)} \neq \emptyset$. Now, let us move to the proof of the lemma. Let π' be a maximal prefix of π such that $Acts(\pi') \subseteq G^{\omega_I}_{\mathrm{kgoal}(\omega_I,\omega_F)}$. From both the observations we have that $Acts(\pi') \cap H^{\omega_I}_{\mathrm{kgoal}(\omega_I,\omega_F)} \neq \emptyset$, as otherwise π' would either not cover ω_I or would not enable any action from $\mathcal{T}$, or would not be maximal. Therefore, $Acts(\pi')$ needs to contain an element from $H^{\omega_I}_{\mathrm{kgoal}(\omega_I,\omega_F)-1}$, as otherwise by the first observation none of the actions from $H^{\omega_I}_{\mathrm{kgoal}(\omega_I,\omega_F)}$ would become enabled. The rest of the proof follows by the induction. □

The results of Lemma 2 can be applied in pruning the space of the possible plans. Observe that if we put $A = Act$, then the collection $\{Act \setminus H_0^{\omega_I}, \ldots, Act \setminus H^{\omega_I}_{\mathrm{kgoal}}(\omega_I, \omega_F)\}$ consists of the sets of actions insufficient to form an abstract plan. We call such sets of actions *none-plans*. Formally, assume that $A \subseteq Act$ and $\omega, \omega' \in \mathcal{W}_{\mathcal{H}}$ and define the set of none-plans $\mathcal{Z}(\omega, A, \omega') \subseteq 2^A$ as the collection of sets of the actions such that $B \in \mathcal{Z}(\omega, A, \omega')$ iff $B \subseteq A$ and $\Pi(\omega, B, \omega') = \emptyset$.

None-Plan Synthesis We now propose a method for the synthesis of the set of none-plans. Firstly (Item 1 of Lemma 3), we show that an abstract plan covers the final state iff it covers all the non-zero valued coordinates of its vector representation. Secondly (Item 3 of Lemma 3), we prove that a set of actions is a none-plan iff for each action a from this set, if $\mathrm{eff}(a)$ covers the final state, then a cannot be enabled. We apply these properties to obtain the recursive formula characterizing the none-plans in Theorem 1.

Let $\mathbb{I}(\omega_F)$ be the set of all the worlds such that their vector representations are unitary and covered by $\boldsymbol{\omega_F}$, formally:

$$\mathbb{I}(\omega_F) = \{\omega \in \mathcal{W}_{\mathcal{H}} \mid \exists_{0 \leq i \leq n}(\boldsymbol{\omega}_i = 1 \wedge \boldsymbol{\omega_F}_i > 0 \wedge \forall_{j \neq i} \omega_j = 0)\}.$$

Let $\omega \in \mathbb{I}(\omega_F)$ be such a world that $\boldsymbol{\omega}_i = 1$ for $i \in \mathbb{N}$. We define $\bar{A}(\omega) = \{\text{act} \in A \mid \text{eff}(\mathbf{act})_i = 0\}$. The set $\bar{A}(\omega)$ consists of those actions from A whose effect vector representation assigns 0 to the non-zero coordinate of $\boldsymbol{\omega}$.

To avoid the convoluted notation, the notions introduced in this section so far have depended on the fixed set of actions. From now on, we implicitly designate the set of actions whenever we need to refer to the concepts that depend on $A \subseteq Act$, i.e., we write: $\{G_i^{\omega}(A)\}_{i\in\mathbb{N}}$, klimit$(\omega, A)$, and kgoal$(\omega, A, \omega')$.

Lemma 3. *Let $A \subseteq Act$ and $\omega, \omega' \in \mathcal{W}_{\mathcal{H}}$. The following conditions hold:*

1. *$\Pi(\omega, A, \omega') = \emptyset$ iff $\exists_{\omega'' \in \mathbb{I}(\omega')} \Pi(\omega, A, \omega'') = \emptyset$,*
2. *$\Pi(\omega, A, \omega') = \emptyset$ iff $\exists_{\omega'' \in \mathbb{I}(\omega')} G^{\omega}_{\text{klimit}(\omega,A)} \subseteq \bar{A}(\omega'')$,*
3. *if ω' is represented by a unitary vector, then we have $\Pi(\omega, A, \omega') = \emptyset$ iff $\omega \not\geq \omega' \wedge \forall_{\text{act}\in A}\big(\text{eff}(\text{act}) \geq \omega' \implies \Pi(\omega, A \setminus \{\text{act}\}, \text{pre}(\text{act})) = \emptyset\big)$.*

Proof. Let us start with Item 1 of the lemma. If there exists $\omega'' \in \mathbb{I}(\omega')$ such that $\Pi(\omega, A, \omega'') = \emptyset$, then from the definition of $\mathbb{I}(\omega')$ there is a none-zero valued coordinate of $\boldsymbol{\omega'}$ that is not covered by any plan. Furthermore, no plan can cover $\boldsymbol{\omega'}$. For the other side of the proof, assume that $\mathbb{I}(\omega') = \{\omega''_1, \ldots, \omega''_m\}$ for some $m \in \mathbb{N}$ and that for every $\omega'' \in \mathbb{I}(\omega')$ we have $\Pi(\omega, A, \omega'') \neq \emptyset$, i.e., there exists a sequence $\pi_{\omega''} = \omega \text{act}_1^{\omega''} \circ \ldots \circ \text{act}_{n_{\omega''}}^{\omega''} \in \Pi(\omega, A, \omega'')$. Now, from the fact that the actions stay enabled once becoming so, we obtain that the sequence $\pi = \omega \text{act}_1^{\omega''_1} \circ \ldots \circ \text{act}_{n_{\omega''_1}}^{\omega''_1} \circ \ldots \circ \text{act}_1^{\omega''_m} \circ \ldots \circ \text{act}_{n_{\omega''_m}}^{\omega''_m}$ belongs to $\Pi(\omega, A, \omega')$.

To prove Item 2, we fix any $\omega'' \in \mathbb{I}(\omega')$ and observe that $\Pi(\omega, A, \omega'') = \emptyset$ iff the actions present in the sequence $\{G_i^{\omega}(A)\}_{i\in\mathbb{N}}$ do not cover ω''. This is equivalent to $G^{\omega}_{\text{klimit}(\omega,A)} \subseteq \bar{A}(\omega'')$ thus we conclude by applying Item 1.

Let us move now to Item 3. If $\Pi(\omega, A, \omega') = \emptyset$, then $\omega \not\geq \omega'$. If there were an action act $\in A$ with eff(act) $\geq \omega'$, and a plan $\pi \in \Pi(\omega, A \setminus \{\text{act}\}, \text{pre}(\text{act}))$, then we would have had $\pi \circ \text{act} \in \Pi(\omega, A, \omega')$, which contradicts with the emptiness of this set. Let us assume now that $\pi \in \Pi(\omega, A, \omega')$ (hence $\Pi(\omega, A, \omega') \neq \emptyset$) and $\omega \not\geq \omega'$. As $\boldsymbol{\omega'}$ is a unitary vector, there exists an action act $\in Acts(\pi)$ such that eff(act) $\geq \omega'$. Let π' be the prefix of π that ends immediately before the first occurrence of act. Notice that we have $\pi' \in \Pi(\omega, A \setminus \{\text{act}\}, \text{pre}(\text{act}))$, and the non-emptiness of $\Pi(\omega, A \setminus \{\text{act}\}, \text{pre}(\text{act}))$ concludes the proof of this case and the whole lemma. □

It can be shown that the assumption that ω' is unitary, made in Item 3 of Lemma 3, is essential. The following theorem provides a recursive characterization of none-plans.

Theorem 1. *Let $A \subseteq Act$ and $\omega, \omega' \in \mathcal{W}_{\mathcal{H}}$. The set $\mathcal{Z}$ of the none-plans is characterized as follows:*

$$\mathcal{Z}(\omega, A, \omega') = \bigcup_{\{\omega'' \in \mathbb{I}(\omega') \mid \omega \not\geq \omega''\}} \bigcap_{\{\text{act}\in A \mid \text{eff}(\text{act}) \geq \omega''\}} \big(\mathcal{D}(\omega, A, \text{act}) \cup 2^{A\setminus\{\text{act}\}}\big),$$

where $\mathcal{D}(\omega, A, \text{act}) = \{B \cup \{\text{act}\} \mid B \in \mathcal{Z}(\omega, A \setminus \{\text{act}\}, \text{pre}(\text{act}))\}$.

Proof. Firstly, observe that from Item 1 of Lemma 3 and the fact that if $\omega \geq \omega''$, then $\mathcal{Z}(\omega, A, \omega'') = \emptyset$, we have:

$$\mathcal{Z}(\omega, A, \omega') = \bigcup_{\omega'' \in \mathbb{I}(\omega')} \mathcal{Z}(\omega, A, \omega'') = \bigcup_{\{\omega'' \in \mathbb{I}(\omega') | \omega \not\geq \omega''\}} \mathcal{Z}(\omega, A, \omega''). \qquad (\star)$$

Assume that $\omega \not\geq \omega''$ and ω'' have unitary vector representations. By Item 3 of Lemma 3 we know that $B \in \mathcal{Z}(\omega, A, \omega'')$ iff B consists of all those actions a that satisfy (1) $\mathrm{eff}(a)$ does not cover ω'', or (2) $\mathrm{eff}(a)$ covers ω'', but a is not enabled by the remaining actions of B. Therefore, we have the following equality:

$$\mathcal{Z}(\omega, A, \omega'') = \bigcap_{\{\mathrm{act} \in A | \mathrm{eff}(\mathrm{act}) \geq \omega''\}} \big(\{B \cup \{\mathrm{act}\} | B \in \mathcal{Z}(\omega, A \setminus \{\mathrm{act}\}, \mathrm{pre}(\mathrm{act}))\} \cup 2^{A \setminus \{\mathrm{act}\}}\big),$$

which applied to $(\star)$ concludes the proof. □

The results of Theorem 1 can be applied to generate a propositional formula $\phi_{\mathcal{P}}$ that describes all the none-plans over a planning domain. This formula has such a property that the set of its models is in a one-to-one correspondence with the set of none-plans in $\mathcal{P}$. To build the representation of $\phi_{\mathcal{P}}$ we firstly recursively unfold the set of none-plans $\mathcal{Z}(\omega, A, \omega')$, as given in Theorem 1, into a meet-join graph with the power sets of certain sets of actions as leaves. This graph is then transformed into an and-or graph with disjunctions of the propositions representing the respective actions in the leaves. Due to the limited space we omit here further details.

In the next example we perform by hand computations the first step of the recursive computation of the set of none-plans for an exemplary planning domain.

Example 5. Let $\mathcal{P}'' = (\mathcal{W}_{\mathcal{H}}, F'_I, F'_G, Act')$ be the planning domain obtained by modifying the planning domain $\mathcal{P}'$ of Example 3 as follows:

- $Act' = \{makeBody, makeCar, makeBoat, makeAmphibian\}$, i.e., *tinker* is removed from the set of actions,
- $F'_I = \{\omega'_I\}$ and $F'_G = \{\omega'_F\}$ are such that:
  - $\omega'_I(Body) = \omega'_I(Boat) = 1$ and $\omega'_I(Car) = \omega'_I(Amphibian) = 0$,
  - $\omega'_F(Car) = 1$ and $\omega'_F(Car) = \omega'_F(Boat) = \omega'_F(Amphibian) = 0$.

Intuitively, we start with a *Boat* and wish to build a *Car*. For convenience we use the vector representations and the order established in Example 3, thus we have $\boldsymbol{\omega'_I} = (1, 0, 1, 0)$ and $\boldsymbol{\omega'_F} = (0, 1, 0, 0)$. Let us compute $\mathcal{Z}(\omega'_I, Act', \omega'_F)$. Firstly, observe that $\{\boldsymbol{\omega''} \in \mathbb{I}(\omega'_F) \mid \omega'_I \not\geq \omega''\} = \{(0, 1, 0, 0)\}$ and $\{\mathrm{act} \in Act' \mid \mathrm{eff}(\mathbf{act}) \geq (0, 1, 0, 0)\} = \{makeCar, makeAmphibian\}$, we therefore have:

$$\mathcal{Z}(\omega'_I, Act', \omega'_F) = \big(\mathcal{D}(\omega'_I, Act', makeCar) \cup 2^{Act' \setminus \{makeCar\}}\big) \cap \big(\mathcal{D}(\omega'_I, Act', makeAmphibian) \cup 2^{Act' \setminus \{makeAmphibian\}}\big).$$

Now, to compute $\mathcal{D}(\omega'_I, Act', makeCar)$ and $\mathcal{D}(\omega'_I, Act', makeAmphibian)$ we need to find $\mathcal{Z}(\omega'_I, \{makeBody, makeBoat, makeAmphibian\}, \mathrm{pre}(makeCar))$ and

$\mathcal{Z}(\omega'_I, \{makeBody, makeCar, makeBoat\}, \text{pre}(makeAmphibian))$, resp. We apply the equivalence from Theorem 1 until we obtain the result consistent with the intuition, i.e., the set of none-plans consisting of all the subsets of Act' that do not contain $makeCar$.

In theory, such an approach, based on the straightforward unfolding of the characterisation of none-plans, may become intractable for very large planning domains. Observe that in the equivalence from Theorem 1, in the worst case the outer sum ranges over all the unitary vectors and the inner join ranges over all the currently considered actions. Thus, applying the recursive equivalence k times can require $O\big(|\mathcal{U}|^k \cdot \frac{|Act|!}{(|Act|-k)!}\big)$ operations in the worst case. Since it can be at most $|Act|$ steps until the unfolding is complete, the time complexity of the synthesis of all the none-plans is in $O(|\mathcal{U}|^{|Act|} \cdot |Act|!)$. This is a rough estimate of the pessimistic complexity, but the experimental results allow to conjecture that the average complexity is much better. Nevertheless, we propose a method of approximating the result when the exhaustive computations are too time-consuming.

Let $\omega, \omega' \in \mathcal{W}_{\mathcal{H}}$ and $A \subseteq Act$. Recall that if ω'' has a unitary vector representation, then $\bar{A}(\omega'')$ is the set of the actions which effects cannot cover ω''.Let $\mathcal{E}_F(\bar{A}(\omega'')) = A \setminus G^{\omega}_{\text{klimit}(\omega, \bar{A}(\omega''))}$ denote the set of all the actions from A that are not enabled by $\bar{A}(\omega'')$. We define:

$$\mathcal{F}(\omega, A, \omega') = \bigcup_{\omega'' \in \mathbb{I}(\omega')} \{X \cup Y \mid X \subseteq \bar{A}(\omega'') \wedge Y \subseteq \mathcal{E}_F(\bar{A}(\omega''))\}.$$

As shown in the following lemma $\mathcal{F}(\omega, A, \omega')$ is a lower approximation of the set of the none-plans $\mathcal{Z}(\omega, A, \omega')$.

Lemma 4. *For all $A \subseteq Act$ and $\omega, \omega' \in \mathcal{W}_{\mathcal{H}}$ we have $\mathcal{F}(\omega, A, \omega') \subseteq \mathcal{Z}(\omega, A, \omega')$.*

Proof. Let $B \in \mathcal{F}(\omega, A, \omega')$ and $\omega'' \in \mathbb{I}(\omega')$ be such that $B = X \cup Y$ for some $X \subseteq \bar{A}(\omega'')$ and $Y \subseteq \mathcal{E}_F(\bar{A}(\omega''))$. By the definition B consists of the actions that are either useless or their effects do not cover ω'', therefore by Item 1 of Lemma 3 their effects do not cover ω' as well. □

We can therefore stop the unfolding of the characterisation of the set of the none-plans performed by the consecutive application of the equivalence from Theorem 1 at any given depth and obtain an approximate result.

This can be done by replacing at the selected depth the recursive call to the $\mathcal{Z}(\omega, A, \omega')$ computation with the results of computing $\mathcal{F}(\omega, A, \omega')$.

4 Evaluation

The search space reduction described above has been evaluated in practice. We have implemented the SpaceCut (SC) tool which generates a formula describing none-plans using the SMT-LIB v2 format [1], as well as partitioning the service types into sets. The open source tool can be downloaded from [6].

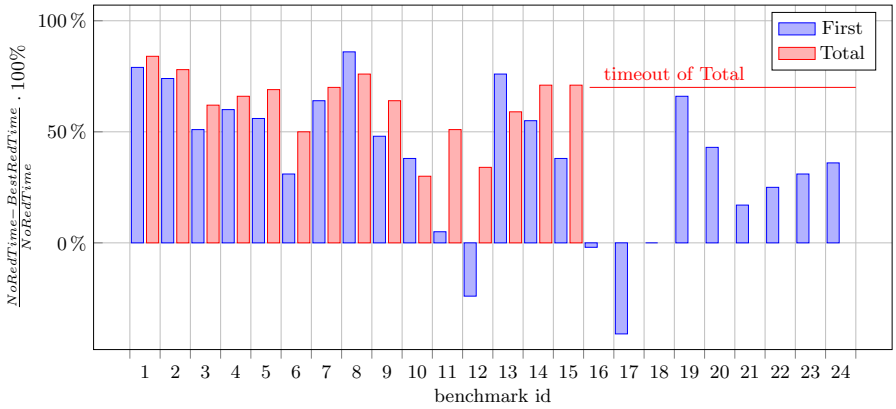

Figure 3: The summary of the reduction efficiency.

We have also implemented the dedicated version of the SMT-based abstract planner on top of PlanICS [2] which makes use of the none-plans description and the service types partitioning. The implemented changes are threefold. Firstly, we ignored the services classified by SC as useless. Overall, we could remove out of the planning scope about 7% of the service types (from 0 to 34 such service types, depending on the particular benchmark).

The second improvement consists in using the formula describing none-plans. We have simply added $\neg\phi_{\mathcal{P}}$ as an SMT-solver assertion. However, our original SMT-based planner [11] in the low-level encoding deals with *sequences* rather than with *sets* of service types. Thus, a "bridge" formula is needed in order to join the "old" encoding with the $\phi_{\mathcal{P}}$ formula.

Finally, we also make use of the service types partitioning. As the first service in the sequence belongs to $G_0^{\omega_I}$, the second one is from $G_1^{\omega_I}$, the third is from $G_2^{\omega_I}$ and so on, we simply encoded this dependency as a propositional formula. Moreover, assuming that our search for the plans is incremental, and we have found nothing so far, we can add one more constraint. Namely, we can assume that the last service in the sequence is from the set $H^{\omega_I}_{\text{kgoal}}(\omega_I, \omega_F) \cup \mathcal{T}$.

We report in Fig. 3 the experimental results performed using 24 benchmarks where the search space reduction has been applied on various depths, compared to the standard SMT-based abstract planner of the PlanICS toolset. We imposed the 2000 sec. time-out for every experiment. We have focused on two performance indicators: the time needed to find the first plan (Fig. 3: First) and the time consumed by the SMT-solver in order to search the whole space (Fig. 3: Total), i.e., to make sure that all the plans (of a given length) have already been found. Due to the space limit the detailed results are presented in Appendix. The experiments have been conducted on a standard PC computer with 2GHz CPU and 8GB RAM.

The benchmarks have been randomly generated by our tool Ontology Generator [15]. Table 1 presents the benchmarks characteristics. The consecutive rows

ont	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
N	2^6	2^7	2^8	2^6	2^7	2^8	2^6	2^7	2^8	2^6	2^7	2^8	2^6	2^7	2^8	2^6	2^7	2^8	2^6	2^7	2^8	2^6	2^7	2^8
k	6						9						12						15					
sol	1			10			1			10			1			10			1			10		
$H_0^{\omega_I}$	35	67	131	44	76	140	35	67	131	40	76	140	35	67	131	28	76	140	35	67	131	16	76	140
$H_1^{\omega_I}$	5	21	84	14	23	91	5	26	94	12	22	100	3	26	98	12	25	92	6	19	106	12	16	80
$H_2^{\omega_I}$	1	6	35		3	21	6	10	21	12	14	16	3	6	14	12	13	12	3	9	13	12	12	12
$H_3^{\omega_I}$					1	1							3	3	3	12	12	12	3	6	3	12	12	12
$H_4^{\omega_I}$																			3	4	3	12	12	12
$\mathcal{E}$	23	34	6	6	25	3	18	25	10	0	16	0	20	26	10	0	2	0	14	23	0	0	0	0
$\mathcal{R}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\mathcal{T}$	1	6	35	0	4	22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1: Benchmark features

contain the following data, starting from the top: the benchmark id (ont), the number of service types (actions) available, the length of the shortest solutions (k), and the number of plans of the length k. The next rows show the results of the service types partitioning, i.e., the size of particular sets.

Observe that in most cases (20 out of 24), the time needed for finding the first solution has been reduced. Moreover, in 18 cases the improvement is significant, i.e., we obtain a speedup by over 25%. Only for two benchmarks the reduction worsens the performance of finding the first plan. We plan to further investigate these anomalies. In the remaining cases we obtain only a minor difference between the approaches, as the benefits of the reduction probably have been balanced by the overhead resulting from more complex formula to be solved.

However, while taking into account the total computation time, an application of the reduction has been superior for all the cases which we could measure, i.e., for those that do not exceeded the time limit. The improvements vary from 30% to 84%. This means that the computation combined with the reduction may take up to 84% less time than the one without the reductions.

5 Conclusions and Future Work

In this paper we presented a novel framework aimed at the optimization of planning in an object-oriented domain. The problem is inspired by our earlier applications of planning to web service composition [2,10,11]. We have shown how to represent a simplified class hierarchy together with the actions that operate on classes of objects as an affine transition system. Our main contribution consists in synthesising none-plans, i.e., the sets of actions that are not sufficient to forming a monotone plan. This synthesis can be either exhaustive or approximative. The none-plans are then used to reduce the space of the possible web service compositions in PlanICS.

The experimental results are quite promising as we have achieved a substantial speedup in most cases. A parallelized approach where the separate threads

attempt to synthesise solutions having different degrees of approximation seems to be the best strategy for attaining the optimal efficiency. While the rough estimates of the theoretical complexity of the task of none-plan synthesis are discouraging, the experimental results suggest that in practice the algorithm behaves much better. As far as the future work is concerned, we plan to investigate the theoretical complexity of the task of the none-plan synthesis in more detail. We are also going to design criteria for the selection of the optimal degree of the approximation of the set of none-plans for reducing the time of the synthesis of the first plan.

References

1. D. R. Cook. The SMT-LIBv2 Language and Tools: A Tutorial. http://www.grammatech.com/resource/smt/SMTLIBTutorial.pdf, 2012.
2. D. Doliwa et al. PlanICS - a web service compositon toolset. *Fundam. Inform.*, 112(1):47–71, 2011.
3. A. E. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5):619–668, 2009.
4. R. P. Goldman, D. J. Musliner, K. D. Krebsbach, and M. S. Boddy. Dynamic abstraction planning. In *AAAI/IAAI*, pages 680–686, 1997.
5. J. V. Gómez, A. Lumbier, S. Garrido, and L. Moreno. Planning robot formations with fast marching square including uncertainty conditions. *Robot. Auton. Syst.*, 61(2):137–152, Feb. 2013.
6. M. Knapik. SpaceCut - a tool for none-plan generation. https://github.com/MichalKnapik/SpaceCut, 2015.
7. M. Knapik and W. Penczek. Bounded model checking for parametric timed automata. *T. Petri Nets and Other Models of Concurrency (ToPNoC)*, 5:141–159, 2012.
8. M. Knapik and W. Penczek. SMT-based parameter synthesis for L/U automata. In *Proc. of Int. Workshop on Petri Nets and Software Engineering, 2012 (PNSE'12)*, pages 77–92, 2012.
9. Z. Li, L. O'Brien, J. Keung, and X. Xu. Effort-oriented classification matrix of web service composition. In *Proc. of the Fifth International Conference on Internet and Web Applications and Services*, pages 357–362, 2010.
10. A. Niewiadomski and W. Penczek. Towards SMT-based Abstract Planning in PlanICS Ontology. In *Proc. of Int. Conf. on Knowledge Engineering and Ontology Development (KEOD)*, pages 123–131, 2013.
11. A. Niewiadomski and W. Penczek. SMT-based abstract temporal planning. In *Proc. of Int. Workshop on Petri Nets and Software Engineering*, pages 55–74, 2014.
12. I. Nourbakhsh. Using abstraction to interleave planning and execution. In *Proceedings of the Third Biannual World Automation Congress*, volume 2, 1998.
13. J. Rao and X. Su. A survey of automated web service composition methods. In *Proc. of SWSWPC'04*, volume 3387 of *LNCS*, pages 43–54. Springer, 2005.
14. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.
15. J. Skaruz, A. Niewiadomski, and W. Penczek. Evolutionary algorithms for abstract planning. In *Parallel Processing and Applied Mathematics (PPAM)*, volume 8384 of *LNCS*, pages 392–401. Springer, 2013.