SMT-based Abstract Temporal Planning*

Artur Niewiadomski[1] and Wojciech Penczek[1,2]

[1] ICS, Siedlce University, 3-Maja 54, 08-110 Siedlce, Poland
artur.niewiadomski@uph.edu.pl
[2] ICS, Polish Academy of Sciences, Jana Kazimierza 5, 01-248 Warsaw, Poland
penczek@ipipan.waw.pl

Abstract. An abstract planning is the first phase of the web service composition in the PlanICS framework. A user query specifies the initial and the expected state of a plan in request. The paper extends PlanICS with a module for temporal planning, by extending the user query with an LTL^{k}_{-X} formula specifying temporal aspects of world transformations in a plan. Our solution comes together with an example, an implementation, and experimental results.

Keywords: Web Service Composition, SMT, Abstract Planning, Temporal Planning, LTL

1 Introduction

Web service composition within Service-Oriented Architecture (SOA) [2] is still attracting a lot of interest, being a subject of many theoretical and practical approaches. The main idea consists in dealing with independent (software) components available via well-defined interfaces. As typically a simple web service does not satisfy the user objective, a composition is investigated in order to make the user fully satisfied. An automatic composition of Web services aims at relieving the user of a manual preparation of detailed execution plans, matching services to each other, and choosing optimal providers for all the components. The problem of finding such a satisfactory composition is NP-hard and well known in the literature as the Web Service Composition Problem (WSCP) [2, 1, 21]. There are many various approaches to solve WSCP [14, 16], some of them we discuss in the next section.

In this paper, we follow the approach of the system PlanICS [8, 9], which has been inspired by [1]. The main assumption is that all the web services in the domain of interest as well as the objects which are processed by the services, can be strictly classified in a hierarchy of *classes*, organised in an *ontology*. Another key idea consists in dividing planning into several stages. The first phase, called the *abstract planning*, deals with *classes of services*, where each class represents a set of real-world services. This phase has been implemented in PlanICS using

* This work has been supported by the National Science Centre under the grant No. 2011/01/B/ST6/01477.

two approaches: one based on a translation to SMT-solvers [17] and another one exploiting genetic algorithms [22]. The second phase, called *concrete planning*, deals with *concrete services*. Thus, while the first phase produces an *abstract plan*, it becomes a *concrete plan* in the second phase. Such an approach enables to reduce dramatically the number of concrete services to be considered as they are already eliminated in the abstract planning phase. This paper focuses on the abstract planning problem, but extends it to so called temporal planning. This extension together with the experimental results is the main contribution of the paper. The main idea behind this approach consists in providing the user with a possibility to specify not only the first and the expected state of a plan in request, but also to specify temporal aspects of state transformations in a plan. To this aim we introduce two general types of atomic properties for writing a temporal formula, namely *propositions* and *level constraints*. The propositions are used to describe (intermediate) states of a plan in terms of existence (or non-existence) of objects and abstract values of object attributes. The level constraints, built over a special set of objects, are used for influencing a service ordering within solutions. However, in order to express such restrictions the user has to rely on some knowledge about the planning domain. In order to get this knowledge, the planner can be first run without temporal constraints and then these restrictions can be added after a non-temporal planning results have been obtained.

We propose a novel approach based on applying SMT-solvers. Contrary to a number of other approaches, we focus not only on searching for a single plan, but we attempt to find all *significantly different* plans. We start with defining the abstract planning problem (APP, for short). Then, we present our original approach to APP based on a compact representation of abstract plans by multisets of service types. We introduce the language of LTL^{k}_{-X} for specifying the temporal aspects of the user query. This approach is combined with a reduction to a task for an SMT-solver. The encoding of blocking formulas allows for pruning the search space with many sequences which use the same multiset of service types in some plan already generated. Moreover, we give details of our algorithms and their implementations that are followed by experimental results. To the best of our knowledge, the above approach is novel, and as our experiments show it is also very promising.

The rest of the paper is organized as follows. Related work is discussed in Section 2. Section 3 deals with the abstract planning problem. In Section 4 the temporal planning is presented. An example of an abstract temporal planning is shown in Section 5. Section 6 discusses the implementation and the experimental results of our planning system. The last section summarizes this paper and discusses a further work.

2 Related Work

A classification matrix aimed at the influence on the effort of Web service composition is presented in [14]. According to [14], situation calculus [6], Petri nets [11], theorem proving [20], and model checking [23] among others belongs to AI plan-

ning. A composition method closest to ours based on SMT is presented in [16], where the authors reduce WSCP to a reachability problem of a state-transition system. The problem is encoded by a propositional formula and tested for satisfiability using a SAT-solver. This approach makes use of an ontology describing a hierarchy of types and deals with an inheritance relation. However, we consider also the states of the objects, while [16] deals with their types only. Moreover, among other differences, we use a multiset-based SMT encoding instead of SAT.

Most of the applications of SMT in the domain of WSC is related to the automatic verification and testing. For example, a message race detection problem is investigated in [10], the paper [4] takes advantage of symbolic testing and execution techniques in order to check behavioural conformance of WS-BPEL specifications, while [15] exploits SMT to verification of WS-BPEL specifications against business rules.

Recently, there have also appeared papers dealing with temporal logics in the context of WSC. Bersani et al. in [5] present a formal verification technique for an extension of LTL that allows the users to include constraints on integer variables in formulas. This technique is applied to the substitutability problem for conversational services. The paper [13] deals with the problem of automatic service discovery and composition. The authors characterize the behaviour of a service in terms of a finite state machine, specify the user's requirement by an LTL formula, and provide a translation of the problem defined to SAT. However, the paper does not specify precisely experimental results and such important details as, e.g., the number of services under consideration. An efficient application of the authors method is reported for plans of length up to 10 only. The authors of [3] address the issue of verifying whether a composite Web services design meets some desirable properties in terms of deadlock freedom, safety, and reachability. The authors report on automatic translation procedures from the automata-based design models to the input language of the NuSMV verification tool. The properties to be verified can be expressed as LTL or CTL formulae.

Searching for plans meeting temporal restrictions is also a topic of interest of a broad planning community. The PDDL language [12] has been also extended with LTL-like modal operators, but for planning automata-based methods are used instead of SMT-based symbolic ones.

3 Abstract Planning

This section introduces APP as the first stage of WSCP in the PlanICS framework. First, the PlanICS ontology is presented. Next, we provide some basic definitions and explain the main goals of APP.

3.1 PlanICS Ontology

The OWL language [19] is used as the PlanICS ontology format. The concepts are organized in an inheritance tree of *classes*, all derived from the base class - *Thing*. There are 3 children of *Thing*: *Artifact*, *Stamp*, and *Service* (Fig. 1).

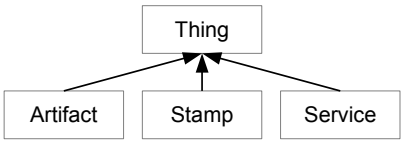

Fig. 1. The base classes in PlanICS ontology

The branch of classes rooted at *Artifact* is composed of the types of the objects, which the services operate on. Each object consists of a number of attributes, whereas an attribute consists of a name and a type. Note that the types of the attributes are irrelevant in the abstract planning phase as they are not used by the planner. The values of the attributes of an object determine its state, but in the abstract planning it is enough to know only whether an attribute does have some value (i.e., is set), or it does not (i.e., it is null). The *Stamp* class and its descendants define special-purpose objects, often useful in constructing a user query, and in the planning process. A stamp is a specific type aimed at a confirmation of the service execution. The specialized descendants of the *Service* class can produce the *stamp* being an instance of any subtype of *Stamp* and describing additional execution features. Note that each service produces exactly one confirmation object. The classes derived from *Artifact* and *Stamp* are called the *object types*.

Each class derived from *Service*, called a *service type*, stands for a description of a set of real-world services. It contains a formalized information about their activities. A service type affects a set of objects and transforms them into a new set of objects. The detailed information about this transformation is contained in the attributes of a service type: the sets *in*, *inout*, and *out*, and the Boolean formulas *pre* and *post* (*pre* and *post*, for short). These sets enumerate the objects, which are processed by the service. The objects of the *in* set are read-only, i.e., they are passed unchanged to the next world. Each object of *inout* can be modified - the service can change some values of its attributes. The objects of *out* are produced by the service.

3.2 Basic definitions

Let $\mathbb{I}$ denote the set of all *identifiers* used as the type names, the objects, and the attributes. In APP we deal with abstract values only, the types of the attributes are irrelevant, and we identify the attributes with their names. Moreover, we denote the set of all attributes by A, where $\mathrm{A} \subset \mathbb{I}$. An *object type* is a pair $(t, Attr)$, where $t \in \mathbb{I}$, and $Attr \subseteq \mathrm{A}$. That is, an object type consists of the type name and a set of the attributes. By $\mathbb{P}$ we mean a set of all object types.

Example 1. Consider the following exemplary ontology containing in addition to *Thing* also the class *Artifact* and *Stamp*. The class *Artifact* corresponds to the object type $(Artifact, \{id\})$ (the only attribute is an identifier) while the class

Stamp corresponds to the object type $(Stamp, \{serviceClass, serviceId, level\})$, introducing the attributes describing the service generating the stamp, and the position of this service in an execution sequence we consider.

We define also a transitive, irreflexive, and antisymmetric *inheritance* relation $Ext \subseteq \mathbb{P} \times \mathbb{P}$, such that $\big((t_1, A_1), (t_2, A_2)\big) \in Ext$ iff $t_1 \neq t_2$ and $A_1 \subseteq A_2$. That is, a subtype contains all the attributes of a base type and optionally introduces more attributes. An *object* o is a pair $o = (id, type)$, where $id \in \mathbb{I}$ and $type \in \mathbb{P}$. By $type(o)$ we denote the type of o. The set of all objects is denoted by $\mathbb{O}$. Amongst all the objects we distinguish between the *artifacts* (the instances of the *Artifact* type) and the *stamps* (the instances of the *Stamp* type). The set of all the stamps is denoted by $\mathbb{ST}$, where $\mathbb{ST} \subseteq \mathbb{O}$. Moreover, we define the function $attr : \mathbb{O} \longmapsto 2^{\mathrm{A}}$ returning the set of all attributes for each object of $\mathbb{O}$.

Service types and user queries. The service types available for composition are defined in the ontology by *service type specifications*. The user goal is provided in a form of a *user query specification*, which is then extended by a temporal formula. Before APP, all the specifications are reduced to sets of objects and *abstract formulas* over them. An **abstract formula** over a set of objects O and their attributes is a DNF formula without negations, i.e., the disjunction of clauses, referred to as *abstract clauses*. Every abstract clause is the conjunction of literals, specifying abstract values of object attributes using the functions $isSet$ and $isNull$. In the abstract formulas used in APP, we assume that no abstract clause contains both $isSet(o.a)$ and $isNull(o.a)$, for the same $o \in O$ and $a \in attr(o)$. For example $(isSet(o.a) \wedge isSet(o.b)) \vee isNull(o.a)$ is a correct abstract formula. The syntax of the specifications of the user queries and of the service types is the same and it is defined below.

Definition 1. *A* **specification** *is a 5-tuple* $(\mathrm{in}, \mathrm{inout}, \mathrm{out}, \mathrm{pre}, \mathrm{post})$*, where* in*,* inout*,* out *are pairwise disjoint sets of objects, and* pre *is an abstract formula defined over objects from* $\mathrm{in} \cup \mathrm{inout}$*, while* post *is an abstract formula defined over objects from* $\mathrm{in} \cup \mathrm{inout} \cup \mathrm{out}$*.*

A user query specification q or a service type specification s is denoted by $spec_x = (in_x, inout_x, out_x, pre_x, post_x)$, where $x \in \{q, s\}$, resp. In order to formally define the *user queries* and the *service types*, which are interpretations of their specifications, we need to define the notions of *valuation functions* and *worlds*.

Definition 2. *Let* $\varphi = \bigvee_{i=1..n} \alpha_i$ *be an abstract formula. A* **valuation of the attributes** *over* α_i *is the partial function* $v_{\alpha_i} : \bigcup_{o \in \mathbb{O}} \{o\} \times attr(o) \longmapsto \{true, false\}$*, where:*

- $v_{\alpha_i}(o, a) = true$ *if* $isSet(o.a)$ *is a literal of* α_i*, and*
- $v_{\alpha_i}(o, a) = false$ *if* $isNull(o.a)$ *is a literal of* α_i*, and*
- $v_{\alpha_i}(o, a)$ *is undefined, otherwise.*

We define the restriction of a valuation function v_{α_i} to a set of objects $O \subset \mathbb{O}$ as $v_{\alpha_i}(O) = v_{\alpha_i}\big|_{\bigcup_{o \in O}\{o\} \times attr(o)}$. The undefined values appear when the interpreted abstract formula does not specify abstract values of some attributes, which is a typical case in the WSC domain. The undefined values are used also for representing families of total valuation functions. Next, for a partial valuation function f, by $total(f)$ we denote the family of the total valuation functions on the same domain, which are consistent with f, i.e., agree on the values defined of f. Moreover, we define a *family of the valuation functions* $\mathcal{V}_\varphi$ over the abstract formula φ as the union of the sets of the consistent valuation functions over every abstract clause α_i, i.e., $\mathcal{V}_\varphi = \bigcup_{i=1}^{n} total(v_{\alpha_i})$. The restriction of the family of functions $\mathcal{V}_\varphi$ to a set of objects O and their attributes is defined as $\mathcal{V}_\varphi(O) = \bigcup_{i=1}^{n} total(v_{\alpha_i}(O))$.

Definition 3. *A* **world** *w is a pair (O_w, v_w), where $O_w \subseteq \mathbb{O}$ and $v_w = v(O_w)$ is a total valuation function equal to some valuation function v restricted to O_w. The* size *of w, denoted by $|w|$ is equal to $|O_w|$.*

That is, a world represents a state of a set of objects, where each attribute is either set or null. By a *sub-world* of w we mean a world built from a subset of O_w and v_w restricted to the objects from the chosen subset. Moreover, a pair consisting of a set of objects and a family of total valuation functions defines a *set of worlds*. That is, if $\mathcal{V} = \{v_1, \ldots, v_n\}$ is a family of total valuation functions and $O \subseteq \mathbb{O}$ is a set of objects, then $\big(O, \mathcal{V}(O)\big)$ means the set $\{\big(O, v_i(O)\big) \mid 1 \leq i \leq n\}$, for $n \in \mathbb{N}$. Finally, the set of all worlds is denoted by $\mathbb{W}$.

Now, we are in a position to define a *service type* and a (basic) *user query* as an interpretation of its specification. In the next section the user query is extended to a temporal version.

Definition 4. *Let $spec_x = (\mathrm{in}_x, \mathrm{inout}_x, \mathrm{out}_x, \mathrm{pre}_x, \mathrm{post}_x)$ be a user query or a service type specification, where $x \in \{q, s\}$, resp. An interpretation of $spec_x$ is a pair of world sets $x = (W^x_{pre}, W^x_{post})$, where:*

- *$W^x_{pre} = \big(\mathrm{in}_x \cup \mathrm{inout}_x, \mathcal{V}^x_{pre}\big)$, where $\mathcal{V}^x_{pre}$ is the family of the valuation functions over* pre_x*,*
- *$W^x_{post} = \big(\mathrm{in}_x \cup \mathrm{inout}_x \cup \mathrm{out}_x, \mathcal{V}^x_{post}\big)$, where $\mathcal{V}^x_{post}$ is the family of the valuation functions over* post_x*.*

An interpretation of a user query (service type) specification is called simply a user query *(*service type*, resp.).*

For a service type (W^s_{pre}, W^s_{post}), W^s_{pre} is called the *input world set*, while W^s_{post} - the *output world set*. The set of all the service types defined in the ontology is denoted by $\mathbb{S}$. For a user query (W^q_{pre}, W^q_{post}), W^q_{pre} is called the *initial world set*, while W^q_{post} - the *expected world set*, and denoted by W^q_{init} and W^q_{exp}, respectively.

Abstract Planning Overview. The main goal of APP is to find a composition of service types satisfying a user query, which specifies some initial and some expected worlds as well as some temporal aspects of world transformations.

Intuitively, an initial world contains the objects owned by the user, whereas an expected world consists of the objects required to be the result of the service composition. To formalize it, we need several auxiliary concepts.

Let $o, o' \in \mathbb{O}$ and v and v' be valuation functions. We say that $v'(o')$ is compatible with $v(o)$, denoted by $v'(o') \succ^{obj} v(o)$, iff the types of both objects are the same, or the type of o' is a subtype of type of o, i.e., $type(o) = type(o')$ or $(type(o'), type(o)) \in Ext$, and for all attributes of o, we have that v' agrees with v, i.e., $\forall_{a \in attr(o)} v'(o', a) = v(o, a)$. Intuitively, an object of a richer type (o') is compatible with the one of the base type (o), provided that the valuations of all common attributes are equal.

Let $w = (O, v)$, $w' = (O', v')$ be worlds. We say that the world w' is compatible with the world w, denoted by $w' \succ^{wrl} w$, iff there exists a one-to-one mapping $map : O \longmapsto O'$ such that $\forall_{o \in O} v'(map(o)) \succ^{obj} v(o)$. Intuitively, w' is compatible with w if both of them contain the same number of objects and for each object from w there exists a compatible object in w'. The world w' is called *sub-compatible* with the world w, denoted by $w' \succ^{swrl} w$ iff there exists a sub-world of w' compatible with w.

World transformations. One of the fundamental concepts in our approach concerns a world transformation. A world w, called a *world before*, can be transformed by a service type s, having specification $spec_s$, if w is sub-compatible with some input world of s. The result of such a transformation is a world w', called a *world after*, in which the objects of out_s appear, and, as well as the objects of $inout_s$, they are in the states consistent with some output world of s. The other objects of w do not change their states. In a general case, there may exist a number of worlds possible to obtain after a transformation of a given world by a given service type, because more than one sub-world of w can be compatible with an input world of s. Therefore, we introduce a *context function*, which provides a strict mapping between objects from the worlds before and after, and the objects from the input and output worlds of a service type s.

Definition 5. *A* **context function** $ctx^s_O : in_s \cup inout_s \cup out_s \longmapsto O$ *is an injection, which for a given service type s and a set of objects O assigns an object from O to each object from in_s, $inout_s$, and out_s.*

Now, we can define a world transformation.

Definition 6. *Let $w, w' \in \mathbb{W}$ be worlds, called a* world before *and a* world after*, respectively, and $s = (W^s_{pre}, W^s_{post})$ be a service type. Assume that $w = (O, v)$, $w' = (O', v')$, where $O \subseteq O' \subseteq \mathbb{O}$, and v, v' are valuation functions. Let $ctx^s_{O'}$ be a context function, and the sets IN, IO, OU be the $ctx^s_{O'}$ images of the sets in_s, $inout_s$, and out_s, respect., i.e., $IN = ctx^s_{O'}(in_s)$, $IO = ctx^s_{O'}(inout_s)$, and $OU = ctx^s_{O'}(out_s)$. Moreover, let $IN, IO \subseteq (O \cap O')$ and $OU = (O' \setminus O)$.*

We say that a service type s transforms *the world w into w'* in the context *$ctx^s_{O'}$, denoted by $w \overset{s, ctx^s_{O'}}{\rightarrow} w'$, if for some $v^s_{pre} \in \mathcal{V}^s_{pre}$ and $v^s_{post} \in \mathcal{V}^s_{post}$, all the following conditions hold:*

1. $\big(IN, v(IN)\big) \succ^{wrl} \big(in_s, v^s_{pre}(in_s)\big)$,

2. $(IO, v(IO)) \succ^{wrl} \big(inout_s, v^s_{pre}(inout_s)\big)$,
3. $(IO, v'(IO)) \succ^{wrl} \big(inout_s, v^s_{post}(inout_s)\big)$,
4. $(OU, v'(OU)) \succ^{wrl} \big(out_s, v^s_{post}(out_s)\big)$,
5. $\forall_{o \in (O \setminus IO)}\ \forall_{a \in attr(o)}\ v(o,a) = v'(o,a)$.

Intuitively, (1) the *world before* contains a sub-world built over IN, which is compatible with a sub-world of some input world of the service type s, built over the objects from in_s. (2) The *world before* contains a sub-world built over IO, which is compatible with a sub-world of the input world of the service type s, built over the objects from $inout_s$. (3) After the transformation the state of objects from IO is consistent with $post_s$. (4) The objects produced during the transformation (OU) are in a state consistent with $post_s$. (5) The objects from IN and the objects not involved in the transformation do not change their states.

In the standard way we extend a world transformation to a sequence of world transformations *seq*. We say that a world w_0 *is transformed by the sequence seq* into a world w_n, denoted by $w_0 \overset{seq}{\rightsquigarrow} w_n$, iff there exists a sequence of worlds $\rho = (w_0, w_1, \ldots, w_n)$ such that $\forall_{1 \le i \le n}\ w_{i-1} \overset{s_i, ctx^{s_i}_{O_i}}{\rightarrow} w_i = (O_i, v_i)$ for some v_i. Then, the sequence $seq = (s_1, \ldots, s_n)$ is called a *transformation sequence* and ρ is called a *world sequence*.

Having the transformation sequences defined, we introduce the concept of *user query solutions* or simply *solutions*, in order to define a plan.

Definition 7. *Let* seq *be a transformation sequence,* $q = (W^q_{init}, W^q_{exp})$ *be a user query. We say that* seq *is a* solution *of* q*, if for* $w \in W^q_{init}$ *and some world* w' *such that* $w \overset{\text{seq}}{\rightsquigarrow} w'$*, we have* $w' \succ^{swrl} w^q_{exp}$*, for some* $w^q_{exp} \in W^q_{exp}$*. The world sequence corresponding to* seq *is called a* world solution*. The set of all the (world) solutions of the user query* q *is denoted by* $QS(q)$ *(*$WS(q)$*, resp.).*

Intuitively, by a solution of q we mean any transformation sequence transforming some initial world of q to a world sub-compatible to some expected world of q.

Plans. Basing on the definition of a solution to the user query q, we can now define the concept of an (abstract) plan, by which we mean a non-empty set of solutions of q. We define a plan as an equivalence class of the solutions, which do not differ in the service types used. The idea is that we do not want to distinguish between solutions composed of the same service types, which differ only in the ordering of their occurrences or in their contexts. So we group them into the same class. There are clearly two motivations behind that. Firstly, the user is typically not interested in obtaining many very similar solutions. Secondly, from the efficiency point of view, the number of equivalence classes can be exponentially smaller than the number of the solutions. Thus, two user query solutions are equivalent if they consist of the same number of the same service types, regardless of the contexts.

Definition 8. *Let* $\text{seq} \in QS(q)$ *be a solution of some user query* q*. An abstract plan is a set of all the solutions equivalent to seq, denoted by* $[\text{seq}]_\sim$.

It is important to notice that all the solutions within an abstract plan are built over the same *multiset* of service types, so a plan is denoted using a multiset notation, e.g., the plan $[2S + 4T + 3R]$ consists of 2 services S, 4 services T, and 3 services R.

In order to give the user a possibility to specify not only the initial and expected states of a solution, we extend the user query with an LTL^k_{-X} formula φ specifying temporal aspects of world transformations in a solution. Then, the temporal solutions are these solutions for which the world sequences satisfy φ. This is formally introduced in the next section.

4 Temporal Abstract Planning

In this section we extend the user query by an LTL^k_{-X} temporal formula and a solution to a temporal solution by requiring the temporal formula to be satisfied. The choice of linear time temporal logic is quite natural since our user query solutions are defined as sequences of worlds. The reason for disallowing a direct use of the operator X (by removing it from the syntax) is twofold. Firstly, we still aim at not distinguishing sequences which differ only in the ordering of independent service types. Secondly, if the user wants to introduce the order on two consequtive service types he can use formulas involving level constraints. On the other hand our language and the temporal planning method can be easily extended with the operator X.

We start with defining the set of propositional variables, the level constraints, and then the syntax and the semantics of LTL^k_{-X}.

4.1 Propositional variables

Let $o \in \mathbb{O}$ be an object, $a \in attr(o)$. The set of propositional variables $PV = \{\mathbf{pEx}(o), \mathbf{pSet}(o.a), \mathbf{pNull}(o.a) \mid o \in \mathbb{O}, a \in attr(o)\}$. Intuitively, $\mathbf{pEx}(o)$ holds in each world, where the object o exists, $\mathbf{pSet}(o.a)$ holds in each world, where the object o exists and the attribute a is set, and $\mathbf{pNull}(o.a)$ holds in each world, where the object o exists and the attribute a is null.

In addition to PV we use also the set of *level constraints* **LC** over the stamps $\mathbb{ST}$, defined by the following grammar:

$$\begin{aligned} \mathbf{lc} &::= \mathbf{lexp} \sim \mathbf{lexp} \\ \mathbf{lexp} &::= c \mid s.level \mid \mathbf{lexp} \oplus \mathbf{lexp} \end{aligned} \tag{1}$$

where $s \in \mathbb{ST}$, $c \in \mathbb{Z}$, $\oplus \in \{+, -, \cdot, /, \%\}$, $\sim \in \{\leq, <, =, >, \geq\}$, and $/, \%$ stand for integer division and modulus, respectively.

Intuitively, $s.level < c$ holds in each world, where the stamp s exists and the value of its level is smaller than c.

4.2 Syntax of LTL^k_{-X}

The LTL^k_{-X} formulae are defined by the following grammar:

$$\varphi ::= p \mid \neg p \mid \mathbf{lc} \mid \neg \mathbf{lc} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \mathrm{U}_{<k} \varphi \mid \varphi \mathrm{R}_{<k} \varphi.$$

where $p \in PV$, $\mathbf{lc} \in \mathbf{LC}$, and $k \in \mathbb{N}$.

Observe that we assume that the LTL^k_{-X} formulae are given in the *negation normal form* (NNF), in which the negation can be only applied to the propositional variables and the level constraints. The temporal modalities $\mathrm{U}_{<k}$ and $\mathrm{R}_{<k}$ are named as usual k-restricted *until* and *release*, respectively. Intuitively, $\varphi \mathrm{U}_{<k} \psi$ means that eventually, but in less than k steps, ψ holds and always earlier φ holds. The formula $\varphi \mathrm{R}_{<k} \psi$ expresses that either for the next $k-1$ states ψ holds or in less than k steps, φ holds and always earlier ψ holds.
The derived basic temporal modalities are defined as follows: $\mathrm{F}_{<k} \varphi \stackrel{def}{=} true \mathrm{U}_{<k} \varphi$ and $\mathrm{G}_{<k} \varphi \stackrel{def}{=} false \mathrm{R}_{<k} \varphi$.

4.3 Semantics of LTL^k_{-X}

We start with defining models over the world solutions, which are finite sequences of worlds.

Definition 9. *A* model *is a pair $M = (\rho, V_\rho)$, where $\rho = (w_0, w_1, \ldots, w_n)$ is a world solution with $w_i = (O_i, v_i)$ for $0 \leq i \leq n$, and $V_\rho : \bigcup_{i=0}^{n} \{w_i\} \times \mathbb{ST} \longrightarrow \mathbb{N} \cup \{\infty\}$ is the function over the worlds of ρ valuating the expressions of the form stamp.level, defined as follows:*

- $V_\rho(w_i, s.level) = \infty$ *if* $s \notin O_i$,
- $V_\rho(w_i, s.level) = 0$ *if* $s \in O_0$,
- $V_\rho(w_i, s.level) = j$ *if* $s \in O_j$ *and* $s \notin O_{j-1}$, *for some* $1 \leq j \leq i$.

The intuition behind the definition of V_ρ is as follows. If a stamp s is not an element of a world w, then the value of $s.level$ in w does not exist, and this is denoted by ∞. If a stamp s is an element of the world w_0, then the value of $s.level$ is 0 in all the worlds. If w_j is the world, where s appears for the first time, then the value of $s.level$ is equal to j in w_j as well as in all further worlds.

Before defining the semantics of LTL^k_{-X} we extend the stamp.level valuation function V_ρ from $\mathbb{ST}$ to the level expressions as follows:

- $V_\rho(w_i, c) = c$,
- $V_\rho(w_i, \mathbf{lexp} \oplus \mathbf{lexp}') = V_\rho(w_i, \mathbf{lexp}) \oplus V_\rho(w_i, \mathbf{lexp}')$ if $V_\rho(w_i, \mathbf{lexp}) \neq \infty \neq V_\rho(w_i, \mathbf{lexp}')$,
- $V_\rho(w_i, \mathbf{lexp} \oplus \mathbf{lexp}') = \infty$ if $V_\rho(w_i, \mathbf{lexp}) = \infty$ or $V_\rho(w_i, \mathbf{lexp}') = \infty$,

We say that an LTL^k_{-X} formula φ is true in $M = (\rho, V_\rho)$ (in symbols $M \models \varphi$) iff $w_0 \models \varphi$, where for $m \leq n$ we have:

- $w_m \models \mathbf{pEx}(o)$ iff $o \in O_m$.

- $w_m \models \mathbf{pSet}(o.a)$ iff $o \in O_m$ and $v_m(o,a) = true$,
- $w_m \models \mathbf{pNull}(o.a)$ iff $o \in O_m$ and $v_m(o,a) = false$,
- $w_m \models \neg p$ iff $w_m \not\models p$, for $p \in PV$,
- $w_m \models (\mathbf{lexp} \sim \mathbf{lexp'})$ iff $V_\rho(w(m), \mathbf{lexp}) \sim V_\rho(w_m, \mathbf{lexp'})$ and $V_\rho(w_m, \mathbf{lexp}) \neq \infty \neq V_\rho(w_m, \mathbf{lexp'})$,
- $w_m \models \neg \mathbf{lc}$ iff $w_m \not\models \mathbf{lc}$, for $\mathbf{lc} \in \mathbf{LC}$,
- $w_m \models \varphi \wedge \psi$ iff $w_m \models \varphi$ and $w_m \models \psi$,
- $w_m \models \varphi \vee \psi$ iff $w_m \models \varphi$ or $w_m \models \psi$,
- $w_m \models \varphi \mathrm{U}_{<k} \psi$ iff $(\exists_{min(m+k,n)>l\geq m})(w_l \models \psi$ and $(\forall_{m\leq j<l})w_j \models \varphi)$,
- $w_m \models \varphi \mathrm{R}_{<k} \psi$ iff $(\forall_{min(m+k,n)>l\geq m})\ w_l \models \psi)$ or $(\exists_{min(m+k,n)>l\geq m})(w_l \models \varphi$ and $(\forall_{m\leq j\leq l})w_j \models \psi)$.

The semantics of the propositions follows their definitions, for the level constraints the semantics is based on the valuation function V_ρ, whereas for the temporal operators the semantics is quite standard. Note that we interpret our language over finite sequences as the solutions we are dealing with are finite.

Now, by a *temporal query* we mean a query (as defined in the former section) extended with an LTL^k_{-X} formula φ. The temporal solutions are these solutions for which the world sequences satisfy φ. A *temporal plan* is an equivalence class of the temporal solutions, defined over the same multiset of services.

5 Example of Temporal Abstract Planning

This section contains an example showing how the abstract temporal planning can be used in practice for a given ontology and user (temporal) queries.

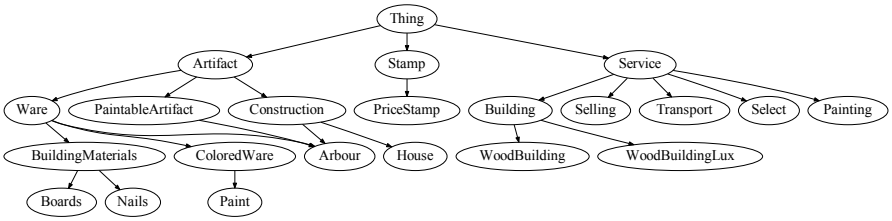

Fig. 2. Example ontology

Consider the ontology depicted in Fig. 2. In the $Artifact$ branch one can see several types of objects, like, e.g., *Arbour* (the main point of interest of this example), which is a subclass of $Ware$, $PaintableArtifact$, and $Construction$. At the left hand side the *Service* branch and its subclasses are located. The service $Select$ (St) is able to search any $Ware$, $Selling$ (Sg) allows to purchase it, while $Transport$ (T) can be used to change its location. The $Painting$ (P) service is able to change colour of any $PaintableArtifact$, but it needs to use some $Paint$. The $Building$ (B) service can be used to obtain some $Construction$, but

it needs $BuildingMaterials$. Finally, two subclasses of $Building$ are specialized in production of wooden constructions using the supplied boards and nails. The services $WoodBuilding$ (Wb) and $WoodBuildingLux$ (Wbx) are similar, but the latter also paints the product to the chosen colour using their own paint, however for a higher price.

Assume the user wants to get a wooden arbour painted in blue. He formulates the query as follows: $in = inout = \emptyset$, $pre = true$, $out = \{Arbour\ a\}$, $post = (a.colour = blue\ \wedge\ a.owner = Me\ \wedge\ a.location = MyAddress)$. The $post$ formula is automatically translated to its abstract form, that is $(isSet(a.colour) \wedge isSet(a.owner) \wedge isSet(a.location))$. The shortest plans are $[St + Sg]$ and $[St + Sg + T]$. The former satisfies the user query only if the $Selling$ service is located in a close proximity of the user's address.

Assume that during the next planning steps (i.e., the offer collecting and the concrete planning) those plans turn out to have no realization acceptable by the user. Perhaps, there are no blue arbours in nearby shops or they are too expensive. Then, the alternative plan is to buy and transport an arbour in any colour, as well as some blue paint, and then use the $Painting$ service: $[2St+2Sg+2T+P]$, where one triple of services (St, Sg, T) provides the arbour, and the other a blue paint.

However, it could be the case that, e.g., the transport price of such a big object like an arbour exceeds the budget. If so, the possible solution is to buy boards, nails, and paint, transport them to the destination address, then to assembly the components with an appropriate building service, and paint, finally. This scenario is covered, for example, by the following plan: $[3St + 3Sg + 3T + Wb + P]$, where the triples of services (St, Sg, T) provide and transport boards, nails, and the paint.

Although, there are over eight hundred abstract plans of length from 2 to 11 satisfying the above user query, including these with multiple transportations of the arbour, or painting it several times. In order to restrict the plans to more specific ones, the user can refine the query demanding of specific types of services to be present in the plan using stamps. Thus, for example, by adding the following set of stamps to out: $\{Stamp\ t_1, Stamp\ t_2, Stamp\ t_3\}$ and extending $post$ by: $\bigwedge_{i=1..3}(t_i.serviceClass\ instanceOf\ Transport)$, the number of possible abstract plans (of length from 2 to 11) can be reduced below two hundred. Then, if instead of buying a final product the user wants to buy and transport the components, in order to build and paint the arbour, he can add two more stamps and conditions to the query. That is, by adding to out the set $\{Stamp\ b, Stamp\ p\}$, and by extending $post$ by the expression $(\wedge\ b.serviceClass\ instanceOf\ Building \wedge t_3.serviceClass\ instanceOf\ Painting)$, one can reduce the number of resulting plans to 2 only: $[3St+3Sg+3T+Wb+P]$ and $[3St+3Sg+3T+Wbx+P]$.

However, even 2 abstract plans only can be realized in a number of different ways, due to possible many transformation contexts, and the number of different partial orders represented by a single abstract plan. If the user wants to further reduce the number of possible plan realizations by interfering with an order of services, he should specify some temporal restrictions. For example, if the user

wants to ensure that all the transports are executed before the building starts, he can express it as a formula:

$$\varphi_1 = \mathrm{F}\big((b.level > t_1.level) \wedge (b.level > t_2.level) \wedge (b.level > t_3.level)\big)$$

Moreover, if the intention of the user is to proceed with some service directly after another one, for example, to start building just after the third transport, one can express such a constraint as:

$$\varphi_2 = \mathrm{F}(b.level = t_3.level + 1)$$

Moreover, using a temporal query the user can prevent some services from occurring in the plan. For example, using the following formula:

$$\varphi_3 = \neg\mathbf{pEx}(a)\ \mathrm{U}\ \mathbf{pNull}(a.colour),$$

which means that just after the arbour has been produced, its colour is not set, the user excludes the $WoodBuildingLux$ service (which builds and paints the arbour).

The other possibility of extending the user query by a temporal component includes using the k-restricted versions of modal operators. For example, consider the following formula:

$$\varphi_4 = \mathrm{F}_{<10}(\mathbf{pEx}(t_1) \wedge \mathbf{pEx}(t_2) \wedge \mathbf{pEx}(t_3)),$$

which states that three transportations should be executed in the first nine steps of the plan.

6 Implementation, Experiments, and Conclusions

In this section we sketch the implementation of the propositions and the level constraints, and then we evaluate the efficiency of our tool using several scalable benchmarks.

6.1 Implementation

The implementation of the propositions and the level constraints exploits our symbolic representation of world sequences. The objects and the worlds are represented by sets of *variables*, which are first allocated in the memory of an SMT-solver, and then used to build formulas mentioned in Section 4. The representation of an object is called a *symbolic object*. It consists of an integer variable representing the type of an object, called a *type variable*, and a number of Boolean variables to represent the object attributes, called the *attribute variables*. In order to represent all types and identifiers as numbers, we introduce a function $num : \mathrm{A} \cup \mathbb{P} \cup \mathbb{S} \cup \mathbb{O} \longmapsto \mathbb{N}$, which with every attribute, object type, service type, and object assigns a natural number.

A *symbolic world* consists of a number of symbolic objects. Each symbolic world is indexed by a natural number from 0 to n. Formally, the i-th symbolic object from the j-th symbolic world is a tuple: $\mathbf{o}_{i,j} = (\mathbf{t}_{i,j}, \mathbf{a}_{i,0,j}, ..., \mathbf{a}_{i,max_{at}-1,j})$, where $\mathbf{t}_{i,j}$ is the type variable, $\mathbf{a}_{i,x,j}$ is the attribute variable for $0 \leq x < max_{at}$, where max_{at} is the maximal number of the attribute variables needed to represent the object.

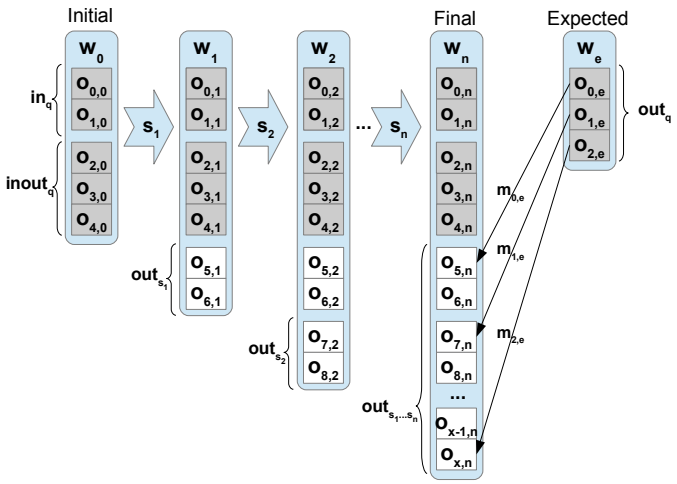

Fig. 3. Symbolic worlds of a transformation sequence

Note that actually a symbolic world represents *a set of worlds*, and only a *valuation* of its variables makes it a single world. The j-th symbolic world is denoted by $\mathbf{w}_j$, while the number of the symbolic objects in $\mathbf{w}_j$ - by $|\mathbf{w}_j|$. Note that the set of the initial worlds of the query q (W^q_{init}) is represented by a symbolic world $\mathbf{w}_0$. Fig. 3 shows subsequent symbolic worlds of a transformation sequence.

One of the important features of our encoding is that for a given index of a symbolic object i we are able to determine the step of a solution, in which the object was produced. This is done by the function $lev_q : \mathbb{N} \mapsto \mathbb{N}$, such that for a given query q:

$$lev_q(i) = \begin{cases} 0 \text{ for } i < |\mathbf{w}_0| \\ \lfloor \frac{(i-|\mathbf{w}_0|)}{max_{out}} \rfloor + 1 \text{ for } i \geq |\mathbf{w}_0| \end{cases} \tag{2}$$

where max_{out} is the maximal number of the objects produced by a single service.

Another important feature of our encoding is that the objects of out_q need to be identified among the objects of the symbolic world $\mathbf{w}_n$ (of indices greater than $|\mathbf{w}_0|$). To this aim, we allocate a new symbolic world $\mathbf{w}_e$ (with $e = n + 1$), containing all the objects from out_q. Note that the world $\mathbf{w}_e$ is not a part of a

world solution, but it provides a set of additional, helper variables. Finally, we need a mapping between the objects from a final world $\mathbf{w}_n$ produced during the subsequent transformations and the objects from $\mathbf{w}_e$. To this aim we allocate p additional *mapping variables* in the symbolic world $\mathbf{w}_e$, where $p = |out_q|$. These variables, denoted by $\mathbf{m}_{0,e}, \ldots, \mathbf{m}_{p-1,e}$, are intended to store the indices of the objects from a final world, which are compatible with the objects encoded over $\mathbf{w}_e$. Thus, we encode the state of the expected worlds of the query q (W^q_{exp}), imposed by $post_q$, using two sets of symbolic objects. The objects of $in_q \cup inout_q$ are encoded directly over the (final) symbolic world $\mathbf{w}_n$. The state of the objects from out_q are encoded over $\mathbf{w}_e$, and since their indices are not known, all possible mappings between objects from $\mathbf{w}_e$ and $\mathbf{w}_n$ are considered, by encoding a disjunction of equalities between objects from $\mathbf{w}_e$ and $\mathbf{w}_n$. See [18] for more details.

The translation of the propositions defined over the objects and their attributes of a user query q in a symbolic world $\mathbf{w}_m$ $(0 \leq m \leq n)$ is as follows:

$$[\mathbf{pEx}(o)]^m = \begin{cases} true, \text{ for } o \in in_q \cup inout_q, \\ false, \text{ for } o \in out_q, m = 0, \\ lev_q(\mathbf{m}_{num(o),e}) \leq m, \text{for } o \in out_q, m > 0. \end{cases} \tag{3}$$

That is, the objects from the initial world exist in all the subsequent worlds, the objects from the *out* set do not exist in the world $\mathbf{w}_0$, and they appear in some subsequent world. Then, since the index of the object o is stored as the value of corresponding mapping variable $\mathbf{m}_{num(o),e}$, we can determine if it exists in the world $\mathbf{w}_m$ using the lev_q function.

The proposition $\mathbf{pSet}(o.a)$ is encoded over the symbolic world $\mathbf{w}_m$ as:

$$[\mathbf{pSet}(o.a)]^m = [\mathbf{pEx}(o)]^m \wedge \begin{cases} \mathbf{a}_{j,x,m}, \text{for } o \in in_q \cup out_q, \\ \bigvee_{i=|\mathbf{w}_0|}^{|\mathbf{w}_m|-1} (\mathbf{m}_{j,e} = i \wedge \mathbf{a}_{i,x,m}), \text{for } o \in out_q \end{cases} \tag{4}$$

where $j = num(o)$ and $x = num(a)$.

It follows from our symbolic representation that the indices of objects from an initial world are known, and we can get the value of the appropriate attribute variable directly. However, in the case of objects from out_q we have to consider all possible mappings between objects from $\mathbf{w}_e$ and $\mathbf{w}_m$. Note that the encoding of the proposition $\mathbf{pNull}(o.a)$ over the symbolic world $\mathbf{w}_m$ (i.e., $[\mathbf{pNull}(o.a)]^m$) is very similar. The only change is the negation of $\mathbf{a}_{i,x,m}$ in the above formula.

In order to encode the level constraints, we introduce a set of the special *level variables*. That is, for every stamp s used in some level constraint we introduce to the world $\mathbf{w}_e$ an additional integer variable $\mathbf{l}_{i,e}$, where $i = num(s)$, intended to store the level value of the stamp s. The level value is assigned to $\mathbf{l}_{i,e}$ using the following formula $[bind(i)] := (\mathbf{l}_{i,e} = lev_q(\mathbf{m}_{i,e}))$ for $i = num(s)$, where q is a user query. Then, for every stamp s used in a level constraint we add the corresponding $[bind(num(s))]$ formula as an SMT assertion. Thus, the encoding of the level constraints is as follows:

$$[\mathbf{lexp}] = \begin{cases} c \text{ for } \mathbf{lexp} = c \\ \mathbf{l}_{i,e}, \text{for } \mathbf{lexp} = s.level, i = num(s) \\ [\mathbf{lexp}'] \oplus [\mathbf{lexp}''] \text{ for } \mathbf{lexp} = \mathbf{lexp}' \oplus \mathbf{lexp}'' \end{cases} \quad (5)$$

The encoding of arithmetic operators is straightforward, since they are supported by theories built in SMT-solvers, like, e.g., Linear Integer Arithmetic or Bitvector theory. In what follows, $[\varphi]_n^m$ denotes the translation of the formula φ at the state w_m of the world sequence of length $n+1$.

Definition 10 (Translation of the LTL_{-X}^k formulae to SMT). *Let φ be an LTL_{-X}^k formula, $(\mathbf{w}_0, \ldots, \mathbf{w}_n)$ be a sequence of symbolic worlds, and $0 \leq m \leq n$.*

- $[p]_n^m := [p]^m$, *for* $p \in PV$,
- $[\neg p]_n^m := \neg[p]^m$, *for* $p \in PV$,
- $[\mathbf{lexp}' \sim \mathbf{lexp}'']_n^m := [\mathbf{lexp}'] \sim [\mathbf{lexp}''] \bigwedge_{s \in st(\mathbf{lexp}')} [\mathbf{pEx}(s)]^m \bigwedge_{s \in st(\mathbf{lexp}'')} [\mathbf{pEx}(s)]^m$,
- $[\neg \mathbf{lc}]_n^m := \neg[\mathbf{lc}]_n^m$, *for* $\mathbf{lc} \in \mathbf{LC}$,
- $[\varphi \wedge \psi]_n^m := [\varphi]_n^m \wedge [\psi]_n^m$,
- $[\varphi \vee \psi]_n^m := [\varphi]_n^m \vee [\psi]_n^m$,
- $[\varphi \mathrm{U}_{<k} \psi]_n^m := \bigvee_{i=m}^{min(m+k,n)} ([\psi]_n^i \wedge \bigwedge_{j=m}^{i-1} [\varphi]_n^j)$,
- $[\varphi \mathrm{R}_{<k} \psi]_n^m := \bigwedge_{i=m}^{min(m+k,n)} [\psi]_n^i \vee \bigvee_{i=m}^{min(m+k,n)} ([\varphi]_n^i \wedge \bigwedge_{j=m}^{i} [\psi]_n^j)$,

where $st(\mathbf{lexp})$ returns the set of the stamps over which the expression **lexp** *is built.*

Theorem 1. *The encoding of the temporal query is correct.*

Proof. This can be shown by induction on the length of a formula. Omitted here because of lack of space[3].

6.2 Experimental Results

In order to evaluate the efficiency of our approach we performed several experiments using standard PC with 2GHz CPU and 8GB RAM, and Z3 [7] version 4.3 as an SMT-solver. The results are summarized in Table 1. Using our Ontology Generator (OG) we generated 15 ontologies, each of them consisting of 150 object types and from 64 to 256 service types (the column named **n** of Tab. 1). For each ontology a query has been generated in such a way that it is satisfied by exactly 10 plans of length from 6 to 18 (the parameter **k**). The queries demand at least two objects to be produced, and impose restrictions on (abstract) values of some of their attributes. A generated query example is as follows: $in = \{Rjlbp\ rjlbp1\}$, $inout = \{Bozwd\ bozwd1\}$, $out = \{Opufo\ opufo1, Ehxjb\ ehxjb2\}$, $pre = isSet(bozwd1.avg) \wedge isSet(rjlbp1.ppw)$,

[3] The full version of this paper is available at http://artur.ii.uph.edu.pl/pnse14ltl.pdf.

$post = isSet(opufo1.epv) \wedge isNull(bozwd1.dyn) \wedge isSet(ehxjb2.zdv) \wedge isNull(ehxjb2.rxz) \wedge isSet(bozwd1.fsl)$.

First, we ran our planner for each ontology and each query instance without a temporal query (column ψ_1), in order to collect statistics concerning the time needed to find the first plan ($\mathbf{P}_1$), all 10 plans ($\mathbf{P}_{10}$), as well as the total time (column **T**) and the memory consumed (**M**) by the SMT-solver in order to find all the plans and checking that no more plans of length k exist. We imposed the time limit of 1000 seconds for the SMT-solver. Each time-out is reported in the table by TO. It is easy to observe that during these experiments as many as 9 instances ran out of time.

Table 1. Experimental results

		$\psi_1 = true$				ψ_3				ψ_4				ψ_5			
n	**k**	$\mathbf{P}_1$ [s]	$\mathbf{P}_{10}$ [s]	**T** [s]	**M** [MB]	$\mathbf{P}_1$ [s]	$\mathbf{P}_{10}$ [s]	**T** [s]	**M** [MB]	$\mathbf{P}_1$ [s]	$\mathbf{P}_{10}$ [s]	**T** [s]	**M** [MB]	$\mathbf{P}_1$ [s]	$\mathbf{P}_{10}$ [s]	**T** [s]	**M** [MB]
64	**6**	7.23	9.73	19.5	19.7	2.34	7.17	8.49	14.1	2.19	4.14	4.21	11.6	3.51	4.7	6.48	10.8
	9	23.5	53.1	177	173	20.1	34.5	47.0	38.3	15.9	18.8	19.2	22.3	14.8	40.1	44.7	36.0
	12	165	479	TO	-	101	216	354	117	60.5	85.2	87.6	59.3	95.1	122	127	68.1
	15	305	TO	TO	-	329	762	TO	-	119	216	241	105	195	345	351	129
	18	TO	TO	TO	-	TO	TO	TO	-	461	TO	TO	-	604	TO	TO	-
128	**6**	16.0	28.1	55.7	42.7	9.75	19.8	22.6	21.3	7.68	10.1	10.2	16.8	10.9	12.7	15.1	16.3
	9	53.1	94.9	270	250	55.5	98.2	104	44.7	21.8	34.0	34.2	31.0	38.1	54.7	62.6	44.9
	12	136	677	TO	-	134	428	474	96.3	76.8	99.9	102	61.9	93.1	114	117	47.9
	15	TO	TO	TO	-	456	780	TO	-	116	199	202	86	183	258	263	79.8
	18	TO	TO	TO	-	TO	TO	TO	-	381	556	573	143	383	708	714	130
256	**6**	16.1	30.6	41.5	35.4	20.3	25.7	30.0	26.1	11.1	13.9	14.1	21.7	14.9	18.3	21	21.6
	9	84.4	137	374	466	63.7	99.3	131	53.4	26.2	38.4	39.1	37.5	88.5	119	156	74.9
	12	267	TO	TO	-	242	584	692	112	114	181	183	80.6	250	315	321	73.4
	15	685	TO	TO	-	562	TO	TO	-	198	304	309	86.8	472	582	592	120
	18	TO	TO	TO	-	TO	TO	TO	-	574	919	937	137	942	TO	TO	-

The next experiments involve temporal queries using level constraints. To this aim we extended the *out* set of the generated queries by the appropriate stamp set. Moreover, the *post* formulas of the queries have been also extended with the expression: $\bigwedge_{i=1}^{\lfloor \frac{k}{2} \rfloor}(s_i.serviceClass\ instanceOf\ C_i)$, where $s_i \in \mathbb{ST}$, while $C_i \in \mathbb{S}$ are service types occurring in the solutions generated by OG.

Our second group of experiments involved the temporal formula ψ_2:

$$\psi_2 = \mathrm{F}(\bigwedge_{i=1}^{\lfloor \frac{k}{2} \rfloor - 1} s_i.level < s_{i+1}.level),$$

which expresses that about a half of the stamps being effects of the solution execution, should be produced in the given order. We do not present detailed

results, because they are in general comparable with the performance in the former experiments. Similarly, there are 9 time-outs, but the time and the memory consumption varies a bit - for some cases the results are slightly better, while for others are a little worse.

In the third group of the experiments we imposed stronger restrictions on the possible service orders of the solutions using the following formula:

$$\psi_3 = \mathrm{F}(\bigwedge_{i=1}^{\lfloor \frac{k}{2} \rfloor} s_i.level < i + 2).$$

This formula still leaves a certain degree of freedom in a service ordering, however its encoding as an SMT-instance is more compact, since the constant values are introduced in place of some level variables. Thus, probably, it is also easier to solve. The results are summarized in the column ψ_3 in Table 1. It is easy to observe that the time and the memory consumption is significantly lower. Moreover, the number of time-outs dropped to 6. Thus, this is an example showing an improvement in the planning efficiency using a temporal query.

Our next experiment involves the formula ψ_4 specifying the strict ordering of several services in the solution using stamp-based level constraints:

$$\psi_4 = \mathrm{F}(\bigwedge_{i=1}^{\lfloor \frac{k}{2} \rfloor} s_i.level = i).$$

The analysis of the results (given in the column ψ_4 of Table 1) indicates a dramatic improvement of our planner efficiency, in terms of time and memory consumption by the SMT-solver. Moreover, in this experiments group the planner has been able to terminate its computations in the given time limit for all but one instances.

Finally, we want to confront the planner behaviour with other kind of temporal formulae. Using the *Until* modality, we demand that one of the objects from out_q has to be produced no later than in the middle of the plan. Moreover, knowing the structure of the generated queries, we impose that after the object appears one of its attributes should already be set. This is expressed by the following formula:

$$\psi_5 = \neg\mathbf{pEx}(o)\ \mathrm{U}_{<\lceil \frac{k}{2} \rceil}\ \mathbf{pSet}(o.a)$$

where $o \in out_q$, $a \in attr(o)$, and the expression $isSet(o.a)$ is not contradictory with $post_q$. The results has been summarised in column ψ_5 of Table 1. It is easy to observe that also this time the plans have been found faster and using less memory than in the case when no temporal formula is involved.

7 Conclusions

In this paper we have applied the logic LTL^k_{-X} to specifying temporal queries for temporal planning within our tool PlanICS. This is a quite natural extension

of our web service composition system, in which the user gets an opportunity to specify more requirements on plans. These requirements are not only declarative any more. The overall conclusion is that the more restrictive temporal query, the more efficient planning, given the same ontology. Assuming that the more restrictive temporal queries, the longer formulas expressing them, the above conclusion shows a difference with model checking, where the complexity depends exponentially on the length of an LTL^k_{-X} formula.

Our temporal planner is the first step towards giving the user even more freedom by defining a so-called parametric approach. We aim at having a planner which in addition to the current capabilities, could also suggest what extensions to the ontology or services should be made in order to get better or unrealizable plans so far. This is going to be a subject of our next paper.

References

1. S. Ambroszkiewicz. Entish: A language for describing data processing in open distributed systems. *Fundam. Inform.*, 60(1-4):41–66, 2004.
2. M. Bell. *Introduction to Service-Oriented Modeling.* John Wiley & Sons, 2008.
3. J. Bentahar, H. Yahyaoui, M. Kova, and Z. Maamar. Symbolic model checking composite web services using operational and control behaviors. *Expert Systems with Applications*, 40(2):508 – 522, 2013.
4. L. Bentakouk, P. Poizat, and F. Zaidi. Checking the behavioral conformance of web services with symbolic testing and an SMT solver. In *Tests and Proofs*, volume 6706 of *LNCS*, pages 33–50. Springer, 2011.
5. M. M. Bersani, L. Cavallaro, A. Frigeri, M. Pradella, and M. Rossi. SMT-based verification of LTL specification with integer constraints and its application to runtime checking of service substitutability. In *SEFM*, pages 244–254, 2010.
6. V. Chifu, I. Salomie, and E. St. Chifu. Fluent calculus-based web service composition - from OWL-S to fluent calculus. In *Proc. of the 4th Int. Conf. on Intelligent Computer Communication and Processing*, pages 161 –168, 2008.
7. L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Proc. of TACAS'08*, volume 4963 of *LNCS*, pages 337–340. Springer-Verlag, 2008.
8. D. Doliwa, W. Horzelski, M. Jarocki, A. Niewiadomski, W. Penczek, A. Półrola, and J. Skaruz. HarmonICS - a tool for composing medical services. In *ZEUS*, pages 25–33, 2012.
9. D. Doliwa, W. Horzelski, M. Jarocki, A. Niewiadomski, W. Penczek, A. Półrola, M. Szreter, and A. Zbrzezny. PlanICS - a web service compositon toolset. *Fundam. Inform.*, 112(1):47–71, 2011.
10. M. Elwakil, Z. Yang, L. Wang, and Q. Chen. Message race detection for web services by an SMT-based analysis. In *Proc. of the 7th Int. Conference on Autonomic and Trusted Computing*, ATC'10, pages 182–194. Springer, 2010.
11. V. Gehlot and K. Edupuganti. Use of colored Petri nets to model, analyze, and evaluate service composition and orchestration. In *System Sciences, 2009. HICSS '09.*, pages 1 –8, jan. 2009.
12. 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(5–6):619 – 668, 2009. Advances in Automated Plan Generation.

13. S. Hao and L. Zhang. Dynamic web services composition based on linear temporal logic. In *Information Science and Management Engineering (ISME), 2010 International Conference of*, volume 1, pages 362–365, Aug 2010.
14. 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.
15. G. Monakova, O. Kopp, F. Leymann, S. Moser, and K. Schäfers. Verifying business rules using an SMT solver for BPEL processes. In *BPSC*, pages 81–94, 2009.
16. W. Nam, H. Kil, and D. Lee. Type-aware web service composition using boolean satisfiability solver. In *Proc. of the CEC'08 and EEE'08*, pages 331–334, 2008.
17. A. Niewiadomski and W. Penczek. Towards SMT-based Abstract Planning in PlanICS Ontology. In *Proc. of KEOD 2013 – International Conference on Knowledge Engineering and Ontology Development*, pages 123–131, September 2013.
18. A. Niewiadomski, W. Penczek, and A. Półrola. Abstract Planning in PlanICS Ontology. An SMT-based Approach. Technical Report 1027, ICS PAS, 2012.
19. OWL 2 web ontology language document overview. http://www.w3.org/TR/owl2-overwiew/, 2009.
20. J. Rao, P. Küngas, and M. Matskin. Composition of semantic web services using linear logic theorem proving. *Inf. Syst.*, 31(4):340–360, June 2006.
21. 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, 2004.
22. J. Skaruz, A. Niewiadomski, and W. Penczek. Automated abstract planning with use of genetic algorithms. In *GECCO (Companion)*, pages 129–130, 2013.
23. P. Traverso and M. Pistore. Automated composition of semantic web services into executable processes. In *The Semantic Web – ISWC 2004*, volume 3298 of *LNCS*, pages 380–394. 2004.