SMT-based Abstract Temporal Planning*

Artur Niewiadomski\textsuperscript{1} and Wojciech Penczek\textsuperscript{1,2}

\textsuperscript{1} ICS, Siedlce University, 3-Maja 54, 08-110 Siedlce, Poland
\texttt{artur.niewiadomski@uph.edu.pl}

\textsuperscript{2} ICS, Polish Academy of Sciences, Jana Kazimierza 5, 01-248 Warsaw, Poland
\texttt{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 $LTL^\infty$ 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)\textsuperscript{[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)\textsuperscript{[2, 1, 21]}. There are many various approaches to solve WSCP\textsuperscript{[14, 16]}, some of them we discuss in the next section.

In this paper, we follow the approach of the system Planics\textsuperscript{[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).
The branch of classes rooted at \textit{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 \textit{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 \textit{Service} class can produce the \textit{stamp} being an instance of any subtype of \textit{Stamp} and describing additional execution features. Note that each service produces exactly one confirmation object. The classes derived from \textit{Artifact} and \textit{Stamp} are called the \textit{object types}.

Each class derived from \textit{Service}, called a \textit{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 \textit{in}, \textit{inout}, and \textit{out}, and the Boolean formulas \textit{pre} and \textit{post} (\textit{pre} and \textit{post}, for short). These sets enumerate the objects, which are processed by the service. The objects of the \textit{in} set are read-only, i.e., they are passed unchanged to the next world. Each object of \textit{inout} can be modified - the service can change some values of its attributes. The objects of \textit{out} are produced by the service.

### 3.2 Basic definitions

Let $I$ denote the set of all \textit{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 $A \subseteq I$. An \textit{object type} is a pair $(t, Attr)$, where $t \in I$, and $Attr \subseteq A$. That is, an object type consists of the type name and a set of the attributes. By $P$ we mean a set of all object types.

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

Amongst all the objects we distinguish between the artifacts by type more attributes. An object 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 \( id \in I \) and \( type \in P \). By \( type(o) \) we denote the type of \( o \). The set of all objects is denoted by \( O \). 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 \( ST \), where \( ST \subseteq O \). Moreover, we define the function \( \text{attr} : O \rightarrow 2^A \) returning the set of all attributes for each object of \( O \).

**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 \( \text{isSet} \) and \( \text{isNull} \). In the abstract formulas used in APP, we assume that no abstract clause contains both \( \text{isSet}(o,a) \) and \( \text{isNull}(o,a) \), for the same \( o \in O \) and \( a \in \text{attr}(o) \). For example \( \text{isSet}(o,a) \land \text{isSet}(o,b) \lor \text{isNull}(o,a) \) 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 \( (\text{in}, \text{inout}, \text{out}, \text{pre}, \text{post}) \), where \( \text{in}, \text{inout}, \text{out} \) are pairwise disjoint sets of objects, and \( \text{pre} \) is an abstract formula defined over objects from \( \text{in} \cup \text{inout} \), while \( \text{post} \) is an abstract formula defined over objects from \( \text{in} \cup \text{inout} \cup \text{out} \).

A user query specification \( q \) or a service type specification \( s \) is denoted by \( \text{spec}_x = (\text{in}_x, \text{inout}_x, \text{out}_x, \text{pre}_x, \text{post}_x) \), 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 \( \alpha_i \) is the partial function \( v_{\alpha_i} : \bigcup_{o \in O} \{o\} \times \text{attr}(o) \rightarrow \{\text{true}, \text{false}\} \), where:

- \( v_{\alpha_i}(o,a) = \text{true} \) if \( \text{isSet}(o,a) \) is a literal of \( \alpha_i \), and
- \( v_{\alpha_i}(o,a) = \text{false} \) if \( \text{isNull}(o,a) \) is a literal of \( \alpha_i \), and
- \( v_{\alpha_i}(o,a) \) is undefined, otherwise.
We define the restriction of a valuation function \( v_\alpha \) to a set of objects \( O \subset \emptyset \) as \( v_\alpha(O) = v_\alpha\big|_{\bigcup_{o \in O} o \times \text{attr}(o)} \). 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 \( \text{total}(f) \) we denote the family of the total valuation functions over 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 \( V_\varphi \) over the abstract formula \( \varphi \) as the union of the sets of the consistent valuation functions over every abstract clause \( \alpha_i \), i.e., \( V_\varphi = \bigcup_{i=1}^{n} \text{total}(v_{\alpha_i}) \). The restriction of the family of functions \( V_x \) to a set of objects \( O \) and their attributes is defined as \( V_\varphi(O) = \bigcup_{i=1}^{n} \text{total}(v_{\alpha_i}(O)) \).

**Definition 3.** A world \( w \) is a pair \( (O_w, v_w) \), where \( O_w \subseteq \emptyset \) 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 \( V = \{v_1, \ldots, v_n\} \) is a family of total valuation functions and \( O \subseteq \emptyset \) is a set of objects, then \( (O, V(O)) \) means the set \( \{(O, v_i(O)) \mid 1 \leq i \leq n\} \), 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 \( \text{spec}_x = (\text{in}_x, \text{inout}_x, \text{out}_x, \text{pre}_x, \text{post}_x) \) be a user query or a service type specification, where \( x \in \{q, s\} \), resp. An interpretation of \( \text{spec}_x \) is a pair of world sets \( x = (W^x_{\text{pre}}, W^x_{\text{post}}) \), where:
- \( W^x_{\text{pre}} = (\text{in}_x \cup \text{inout}_x, V^x_{\text{pre}}) \), where \( V^x_{\text{pre}} \) is the family of the valuation functions over \( \text{pre}_x \),
- \( W^x_{\text{post}} = (\text{in}_x \cup \text{inout}_x \cup \text{out}_x, V^x_{\text{post}}) \), where \( V^x_{\text{post}} \) is the family of the valuation functions over \( \text{post}_x \).

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

For a service type \( (W^s_{\text{pre}}, W^s_{\text{post}}) \), \( W^s_{\text{pre}} \) is called the input world set, while \( W^s_{\text{post}} \) - the output world set. The set of all the service types defined in the ontology is denoted by \( \mathbb{S} \). For a user query \( (W^q_{\text{pre}}, W^q_{\text{post}}) \), \( W^q_{\text{pre}} \) is called the initial world set, while \( W^q_{\text{post}} \) - the expected world set, and denoted by \( W^q_{\text{init}} \) and \( W^q_{\text{exp}} \) 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 \( a, a' \in \mathcal{O} \) and \( v \) and \( v' \) be valuation functions. We say that \( v'(a') \) is compatible with \( v(a) \), denoted by \( v'(a') \triangleright \text{obj} v(a) \), iff the types of both objects are the same, or the type of \( a' \) is a subtype of type of \( a \), i.e., \( \text{type}(a) = \text{type}(a') \) or \( (\text{type}(a'), \text{type}(a)) \in \mathcal{Ext} \), and for all attributes of \( a \), we have that \( v' \) agrees with \( v \), i.e., \( \forall a \in \text{attr}(o) \; v'(a', a) = v(a, a) \). Intuitively, an object of a richer type \( (a') \) is compatible with the one of the base type \( (a) \), 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 \( w' \triangleright \text{wrl} w \), iff there exists a one-to-one mapping \( \text{map} : O \mapsto O' \) such that \( \forall o \in O \; v'(\text{map}(o)) \triangleright \text{obj} v(o) \). 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 \( w' \triangleright \text{swrl} w \) 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 \( \text{spec} s \), 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 \( \text{out} s \) appear, and, as well as the objects of \( \text{inout} s \), 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 \( \text{ctx}_s^O : \text{in}_s \cup \text{inout}_s \cup \text{out}_s \mapsto O \) 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 \( \text{in}_s \), \( \text{inout}_s \), and \( \text{out}_s \).

Now, we can define a world transformation.

**Definition 6.** Let \( w, w' \in \mathcal{W} \) be worlds, called a world before and a world after, respectively, and \( s = (W_s, W'_s) \) be a service type. Assume that \( w = (O, v) \), \( w' = (O', v') \), where \( O \subseteq O' \subseteq \mathcal{O} \), and \( v, v' \) are valuation functions. Let \( \text{ctx}_s^O \) be a context function, and the sets \( \text{IN}, \text{IO}, \text{OU} \) be the \( \text{ctx}_s^O \) images of the sets \( \text{in}_s \), \( \text{inout}_s \), and \( \text{out}_s \), respectively, i.e., \( \text{IN} = \text{ctx}_s^O(\text{in}_s) \), \( \text{IO} = \text{ctx}_s^O(\text{inout}_s) \), and \( \text{OU} = \text{ctx}_s^O(\text{out}_s) \). Moreover, let \( \text{IN} \subseteq (O' \setminus O) \) and \( \text{OU} = (O \cap O') \).

We say that a service type \( s \) transforms the world \( w \) into \( w' \) in the context \( \text{ctx}_s^O \), denoted by \( w \xrightarrow{s, \text{ctx}_s^O} w' \), if for some \( v_{\text{pre}}^s \in \mathcal{V}_{\text{pre}}^s \) and \( v_{\text{post}}^s \in \mathcal{V}_{\text{post}}^s \), all the following conditions hold:

1. \( (\text{IN}, v(\text{IN})) \triangleright \text{wrl} (\text{in}_s, v_{\text{pre}}^s(\text{in}_s)) \),
2. $(\mathit{IO}, v(\mathit{IO})) \xrightarrow{\mathit{wrI}} (\mathit{in}_{\mathit{st}}, v^*_{\mathit{pre}}(\mathit{in}_{\mathit{st}}))$.
3. $(\mathit{IO}, v'(\mathit{IO})) \xrightarrow{\mathit{wrI}} (\mathit{in}_{\mathit{st}}, v^*_{\mathit{post}}(\mathit{in}_{\mathit{st}}))$.
4. $(\mathit{OU}, v'(\mathit{OU})) \xrightarrow{\mathit{wrI}} (\mathit{out}_{\mathit{st}}, v^*_{\mathit{post}}(\mathit{out}_{\mathit{st}}))$.
5. $\forall \mathit{s} \in (\mathit{O} \setminus \mathit{IO}) \forall \mathit{a} \in \mathit{attr}(\mathit{a}) \ v(\mathit{a}, \mathit{a}) = v'(\mathit{a}, \mathit{a})$.

Intuitively, (1) the world before contains a sub-world built over $\mathit{IN}$, which is compatible with a sub-world of some input world of the service type $\mathit{s}$, built over the objects from $\mathit{in}_{\mathit{st}}$. (2) The world before contains a sub-world built over $\mathit{IO}$, which is compatible with a sub-world of the input world of the service type $\mathit{s}$, built over the objects from $\mathit{in}_{\mathit{st}}$. (3) After the transformation the state of objects from $\mathit{IO}$ is consistent with $\mathit{post}_{\mathit{st}}$. (4) The objects produced during the transformation ($\mathit{OU}$) are in a state consistent with $\mathit{post}_{\mathit{st}}$. (5) The objects from $\mathit{IN}$ 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 $\mathit{seq}$. We say that a world $\mathit{w}_0$ is transformed by the sequence $\mathit{seq}$ into a world $\mathit{w}_n$, denoted by $\mathit{w}_0 \xrightarrow{\mathit{seq}} \mathit{w}_n$, iff there exists a sequence of worlds $\rho = (\mathit{w}_0, \mathit{w}_1, \ldots, \mathit{w}_n)$ such that $\forall 1 \leq i \leq n \ w_{i-1} \xrightarrow{s_{i}, e_{i}, t_{i}} \mathit{w}_i = (\mathit{O}_i, \mathit{v}_i)$ for some $\mathit{v}_i$. Then, the sequence $\mathit{seq} = (s_1, \ldots, s_n)$ is called a transformation sequence and $\rho$ 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 $\mathit{seq}$ be a transformation sequence, $q = (\mathit{W}^q_{\mathit{init}}, \mathit{W}^q_{\mathit{exp}})$ be a user query. We say that $\mathit{seq}$ is a solution of $q$, if for $\mathit{w} \in \mathit{W}^q_{\mathit{init}}$ and some world $\mathit{w}'$ such that $\mathit{w} \xrightarrow{\mathit{seq}} \mathit{w}'$, we have $\mathit{w}' \xrightarrow{\mathit{wrI}} \mathit{w}^q_{\mathit{exp}}$, for some $\mathit{w}^q_{\mathit{exp}} \in \mathit{W}^q_{\mathit{exp}}$. The world sequence corresponding to $\mathit{seq}$ is called a world solution. The set of all the (world) solutions of the user query $q$ is denoted by $\mathit{QS}(q)$ ($\mathit{WS}(q)$, 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 $\mathit{seq} \in \mathit{QS}(q)$ be a solution of some user query $q$. An abstract plan is a set of all the solutions equivalent to $\mathit{seq}$, denoted by $[\mathit{seq}]_\sim$. 
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 $\varphi$ specifying temporal aspects of world transformations in a solution. Then, the temporal solutions are these solutions for which the world sequences satisfy $\varphi$. 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 consecutive 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 $o \in O$ be an object, $a \in \text{attr}(o)$. The set of propositional variables $PV = \{\text{pEx}(o), \text{pSet}(o,a), \text{pNull}(o,a) \mid o \in O, a \in \text{attr}(o)\}$. Intuitively, $\text{pEx}(o)$ holds in each world, where the object $o$ exists, $\text{pSet}(o,a)$ holds in each world, where the object $o$ exists and the attribute $a$ is set, and $\text{pNull}(o,a)$ 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 $\text{LC}$ over the stamps $ST$, defined by the following grammar:

\[
\text{le} ::= \text{lexp} \sim \text{lexp} \\
\text{lexp} ::= c \mid s.\text{level} \mid \text{lexp} \oplus \text{lexp}
\]

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

Intuitively, $s.\text{level} < c$ holds in each world, where the stamp $s$ exists and the value of its level is smaller than $c$. 
4.2 Syntax of $\text{LTL}_{k,X}^k$

The $\text{LTL}_{k,X}^k$ formulae are defined by the following grammar:

$$\varphi ::= p \mid \neg p \mid \text{le} \mid \text{¬le} \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid \varphi U_k \varphi \mid \varphi R_k \varphi.$$ 

where $p \in PV$, $\text{le} \in \text{LC}$, and $k \in \mathbb{N}$.

Observe that we assume that the $\text{LTL}_{k,X}^k$ 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 $U_k$ and $R_k$ are named as usual $k$-restricted until and release, respectively. Intuitively, $\varphi U_k \psi$ means that eventually, but in less than $k$ steps, $\psi$ holds and always earlier $\varphi$ holds. The formula $\varphi R_k \psi$ expresses that either for the next $k - 1$ states $\psi$ holds or in less than $k$ steps, $\varphi$ holds and always earlier $\psi$ holds.

The derived basic temporal modalities are defined as follows: $F_{k} \varphi \overset{\text{def}}{=} \text{true} U_{k} \varphi$ and $G_{k} \varphi \overset{\text{def}}{=} \text{false} R_{k} \varphi$.

4.3 Semantics of $\text{LTL}_{k,X}^k$

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 $V_{\rho} : \bigcup_{i=0}^{n}\{w_{i}\} \times \text{ST} \rightarrow \mathbb{N} \cup \{\infty\}$ is the function over the worlds of $\rho$ valuating the expressions of the form $\text{stamp.level}$, defined as follows:

- $V_{\rho}(w_{i}, s.\text{level}) = \infty$ if $s \notin O_{i}$,
- $V_{\rho}(w_{i}, s.\text{level}) = 0$ if $s \in O_{0}$,
- $V_{\rho}(w_{i}, s.\text{level}) = j$ if $s \in O_{j}$ and $s \notin O_{j-1}$, for some $1 \leq j \leq i$.

The intuition behind the definition of $V_{\rho}$ is as follows. If a stamp $s$ is not an element of a world $w$, then the value of $s.\text{level}$ in $w$ does not exist, and this is denoted by $\infty$. If a stamp $s$ is an element of the world $w_{0}$, then the value of $s.\text{level}$ is 0 in all the worlds. If $w_{j}$ is the world, where $s$ appears for the first time, then the value of $s.\text{level}$ is equal to $j$ in $w_{j}$ as well as in all further worlds.

Before defining the semantics of $\text{LTL}_{k,X}^k$ we extend the stamp level valuation function $V_{\rho}$ from $\text{ST}$ to the level expressions as follows:

- $V_{\rho}(w_{i}, c) = c$,
- $V_{\rho}(w_{i}, \text{lexp} \oplus \text{lexp}') = V_{\rho}(w_{i}, \text{lexp}) \oplus V_{\rho}(w_{i}, \text{lexp}')$ if $V_{\rho}(w_{i}, \text{lexp}) \neq \infty \neq V_{\rho}(w_{i}, \text{lexp}')$,
- $V_{\rho}(w_{i}, \text{lexp} \oplus \text{lexp}') = \infty$ if $V_{\rho}(w_{i}, \text{lexp}) = \infty$ or $V_{\rho}(w_{i}, \text{lexp}') = \infty$.

We say that an $\text{LTL}_{k,X}^k$ formula $\varphi$ is true in $M = (\rho, V_{\rho})$ (in symbols $M \models \varphi$) iff $w_{0} \models \varphi$, where for $m \leq n$ we have:

- $w_{m} \models \text{pEx}(o)$ iff $o \in O_{m}$.
\[ w_m \models \text{pSet}(o,a) \text{ iff } o \in O_m \text{ and } v_m(o,a) = true, \]
\[ w_m \models \text{pNull}(o,a) \text{ iff } o \in O_m \text{ and } v_m(o,a) = false, \]
\[ w_m \models \neg \varphi \text{ iff } w_m \notin le, \text{ for } le \in LC, \]
\[ w_m \models \varphi \land \psi \text{ iff } w_m \models \varphi \text{ and } w_m \models \psi, \]
\[ w_m \models \varphi \lor \psi \text{ iff } w_m \models \varphi \text{ or } w_m \models \psi, \]
\[ w_m \models \varphi \U_k \psi \text{ iff } (\exists_{\min(m+k,n)>l \geq m})(w_l \models \psi \text{ and } (\forall_{m \leq j<l})w_j \models \varphi), \]
\[ w_m \models \varphi \R_k \psi \text{ iff } (\forall_{\min(m+k,n)>l \geq m})w_l \models \psi \text{ or } \]
\[ (\exists_{\min(m+k,n)>l \geq m})(w_l \models \varphi \text{ and } (\forall_{m \leq j \leq l})w_j \models \psi). \]

The semantics of the propositions follows their definitions, for the level constraints the semantics is based on the valuation function \( V_p \), 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 \( \varphi \). The temporal solutions are these solutions for which the word sequences satisfy \( \varphi \). 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.

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: \( in = \text{inout} = \emptyset, \text{pre} = \text{true}, \text{out} = \{\text{Arbour} a\}, \text{post} = (a.\text{colour} = \text{blue} \land a.\text{owner} = \text{Me} \land a.\text{location} = \text{MyAddress}) \). The post formula is automatically translated to its abstract form, that is \((\text{isSet}(a.\text{colour}) \land \text{isSet}(a.\text{owner}) \land \text{isSet}(a.\text{location}))\). 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 assemble 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 those 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 \(\text{out} = \{\text{Stamp t}_1, \text{Stamp t}_2, \text{Stamp t}_3\}\) and extending \(\text{post}\) by: \(\bigwedge_{i=1,3} (t_i.\text{serviceClass instanceOf} \ \text{Transport})\), 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 \(\text{out}\) the set \(\{\text{Stamp b, Stamp p}\}\), and by extending \(\text{post}\) by the expression \((\land b.\text{serviceClass instanceOf} \ \text{Building} \land t_3.\text{serviceClass instanceOf} \ \text{Painting})\), 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:

$$\varphi_1 = F((h_{level} > t_1_{level}) \land (h_{level} > t_2_{level}) \land (h_{level} > t_3_{level}))$$

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:

$$\varphi_2 = F(h_{level} = t_3_{level} + 1)$$

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

$$\varphi_3 = \neg p\text{Ex}(a) \lor p\text{Null}(a, \text{colour}),$$

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:

$$\varphi_4 = F_{<10}(p\text{Ex}(t_1) \land p\text{Ex}(t_2) \land p\text{Ex}(t_3)),$$

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 \(\text{num} : A \cup P \cup S \cup O \rightarrow \mathbb{N}\), 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: \( o_{i,j} = (t_{i,j}, a_{i,0,j}, \ldots, a_{i,maxat-1,j}) \), where \( t_{i,j} \) is the type variable, \( a_{i,x,j} \) is the attribute variable for \( 0 \leq x < maxat \), where \( maxat \) is the maximal number of the attribute variables needed to represent the object.

![Figure 3. Symbolic worlds of a transformation sequence](image)

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 \( w_j \), while the number of the symbolic objects in \( w_j \) is \( |w_j| \). Note that the set of the initial worlds of the query \( q(W_{init}) \) is represented by a symbolic world \( w_0 \). 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 \( lev_q : \mathbb{N} \rightarrow \mathbb{N} \), such that for a given query \( q \):

\[
lev_q(i) = \begin{cases} 0 & \text{for } i < |w_0| \\ \lfloor \frac{(i - |w_0|)}{maxout} \rfloor + 1 & \text{for } i \geq |w_0| \end{cases}
\]  

(2)

where \( maxout \) 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 \( w_n \) (of indices greater than \( |w_0| \)). To this aim, we allocate a new symbolic world \( w_e \) (with \( e = n + 1 \)), containing all the objects from \( out_q \). Note that the world \( w_e \) is not a part of a
null

of the proposition where to the world variables. That is, for every stamp is very similar. The only change is the negation of all possible mappings between objects from world directly. However, in the case of objects from initial world are known, and we can get the value of the appropriate attribute to store the level value of the stamp world corresponding mapping variable. Then, since the index of the object the following formula:

That is, the objects from the initial world exist in all the subsequent worlds, the objects from final world do not exist in the world $w_0$, and they appear in some subsequent world. Then, since the index of the object $o$ is stored as the value of corresponding mapping variable $m_{\text{num}(o),e}$, we can determine if it exists in the world $w_m$ using the $\text{lev}_q$ function.

The proposition $\text{pSet}(o,a)$ is encoded over the symbolic world $w_m$ as:

where $j = \text{num}(o)$ and $x = \text{num}(a)$.

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 $\text{out}_q$ we have to consider all possible mappings between objects from $w_e$ and $w_m$. Note that the encoding of the proposition $\text{pNull}(o,a)$ over the symbolic world $w_m$ (i.e., $[\text{pNull}(o,a)]^m$) is very similar. The only change is the negation of $a_{j,x,m}$ 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 $w_e$ an additional integer variable $l_{i,e}$, where $i = \text{num}(s)$, intended to store the level value of the stamp $s$. The level value is assigned to $l_{i,e}$ using the following formula:

for $i = \text{num}(s)$, where $q$ is a user query. Then, for every stamp $s$ used in a level constraint we add the corresponding $[\text{bind}(\text{num}(s))]$ formula as an SMT assertion. Thus, the encoding of the level constraints is as follows:
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, $[\varphi]^m_o$ denotes the translation of the formula $\varphi$ at the state $w_m$ of the world sequence of length $n + 1$.

**Definition 10 (Translation of the $LTL^k_X$ formula to SMT).** Let $\varphi$ be an $LTL^k_X$ formula, $(w_0, \ldots, w_n)$ be a sequence of symbolic worlds, and $0 \leq m \leq n$.

- $[p]^m := [p]^m$, for $p \in PV$,
- $\lnot[p]^m := \lnot[p]^m$, for $p \in PV$,
- $[\text{lexp}'] \sim [\text{lexp}'']^m := [\text{lexp}'] \sim [\text{lexp}''] \land_{s \in st(\text{lexp})}[\text{pEx}(s)]^m \land_{s \in st(\text{lexp}'')}[\text{pEx}(s)]^m$,
- $[\lnot lc]^m := \lnot[lc]^m$, for $lc \in LC$,
- $[\varphi \land \psi]^m := [\varphi]^m \land [\psi]^m$,
- $[\varphi \lor \psi]^m := [\varphi]^m \lor [\psi]^m$,
- $[\varphi \land_{\leq k} \psi]^m := \bigwedge_{i=m}^{\min(m+k,n)}(\bigwedge_{j=m}^{\min(n,m+k)}[\varphi]^j_i) \lor \bigvee_{i=m}^{\min(m+k,n)}(\bigvee_{j=m}^{\min(n,m+k)}[\varphi]^j_i)$,
- $[\varphi \land_{\leq k} \psi]^m := \bigwedge_{i=m}^{\min(m+k,n)}(\bigwedge_{j=m}^{\min(n,m+k)}[\psi]^j_i) \land \bigvee_{i=m}^{\min(m+k,n)}(\bigvee_{j=m}^{\min(n,m+k)}[\psi]^j_i)$,

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

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

**Proof.** First, we show that encoding of propositional variables is correct, i.e., it faithfully corresponds to the defined semantics. We begin with a proposition expressing the existence of an object in a world. Let us recall from the Definition 9 that we interpret propositions over a world solution $\rho = (w_0, w_1, \ldots, w_n)$ with $w_m = (o_m, v_m)$ for $0 \leq m \leq n$. Thus, according to our semantics, $w_m \models \text{pEx}(o)$ iff $o \in o_m$. We show that $o \in o_m$ iff $[\text{pEx}(o)]^m_i$, i.e., the object $o$ exists in the world $w_m$ iff the formula $[\text{pEx}(o)]^m_i$ is satisfiable. Its encoding involves three cases (see Formula 3). The first and the second one follow straightforward from the Planics semantics, while the third case exploits the features of symbolic representation of the transformation sequences.

The first case unconditionally assigns the whole formula the value true. It concerns situations when the object $o$ is given by the user query $q$ as a member of the initial worlds, i.e., $o$ belongs to $\text{in}_q$ or $\text{inout}_q$. Thus, $o$ exists in $w_0$, the initial state of the world solution. The feature of Planics semantics crucial for this proof is that the objects can not be removed from the worlds, but only their states can be modified, and new objects can be created. Therefore, since the object $o$ exists in the world $w_0$, it still exists in all worlds $w_i \in \rho$, and hence, the formula $[\text{pEx}(o)]^m$ is always satisfiable for $o \in \text{in} \cup \text{inout}$. 

$$[\text{lexp}] = \begin{cases} c & \text{for } \text{lexp} = c \\ 1_i, c & \text{for } \text{lexp} = s\.level, i = \text{num}(s) \\ [\text{lexp}'] \oplus [\text{lexp}''] & \text{for } \text{lexp} = \text{lexp}' \oplus \text{lexp}'' \end{cases}$$ (6)
The second and the third cases concern the objects specified as the members of the set out$_q$. According to Planics semantics, the objects from out$_q$ do not exist in the initial worlds, but they have to be produced during the plan execution. Therefore, the second case of Formula 3 assigns $false$ to $[\text{pEx}(o)]^m$, when $m = 0$, i.e., the initial state of the world solution is under consideration.

Finally, the last case concerns the existence of the objects from out$_q$ in all states of $\rho$ except the initial one. In all these situations the valuation of $[\text{pEx}(o)]^m$ formula depends on the function lev$_q$ which a given index of a symbolic object assigns the index of the state of $\rho$ when this object appears for the first time.

The encoding of lev$_q$ function (see Formula 2) corresponds faithfully to its definition. For a given query $q$ the size of the initial worlds $|w_0|$ is known, and it is equal to the number of objects belonging to $\text{in} \cup \text{inout}$. The parameter max$_{out}$ is also known, since it depends on the ontology, and it corresponds to the maximal number of objects produced by a single service. The symbolic encoding of the subsequent states of the world solution relies on adding max$_{out}$ new symbolic objects in each step. If a service produces less than max$_{out}$ objects, than some of them remains empty by setting the types of unused objects to null. Thus, the indices of the objects produced in the $i$-th step of the plan come from the interval $([|w_0| + (i - 1) \cdot \text{max}_{out}], [|w_0| + i \cdot \text{max}_{out} - 1])$, where $i > 0$. Moreover, the object indices remain constant, that is, the same object has the same index in next, and all the subsequent worlds till the final world. Therefore, knowing the object index, we can find out in which step of the plan it has been created. Thus, in the third case the formula $[\text{pEx}(o)]^m$ is satisfiable if the index of $o$ is sufficiently low, i.e., when $o$ has been produced in the step $m$ or earlier. If so, then $o$ exists in the world $w_m$.

Actually, we could introduce the fourth case (which is currently covered by the third one): $[\text{pEx}(o)]^m = true$ for $o \in \text{out}_q$ and $m = n$, which follows from satisfaction of non-temporal query, i.e., definition of a solution. That is, in the final world the objects (compatible with those) from out$_q$ exist, because otherwise we do have neither a world solution nor a model.

The next propositional variable under consideration is pSet$(o,a)$ for $o \in \text{O}$ and $a \in \text{attr}(o)$. According to the semantics we have $w_m \models \text{pSet}(o,a)$ iff $o \in O_m$ and $v_m(o,a) = true$. We show that $[\text{pSet}(o,a)]^m$ is satisfiable iff $o \in O_m$ and $v_m(o,a) = true$. Indeed, this conjunction is reflected directly in encoding of $[\text{pSet}(o,a)]^m$ (see Formula 4). It is also the conjunction of $[\text{pEx}(o)]^m$, corresponding to $o \in O_m$, and the two-case subformula encoding the abstract value of the attribute $a$. Assuming that the object $o$ exists in the world $w_m$ and the first conjunct is satisfied, in both cases the value of $[\text{pSet}(o,a)]^m$ depends on the boolean variable representing the attribute $a$. In the first case, when $o$ is an object from inout$_q$ its index is known, and the encoding is straightforward. However, if $o$ is an object from out$_q$ we have to consider all possible mappings between the objects produced in all steps of the plan and the objects from the expected world. Thus, we build the disjunction of expressions $(m_{x,e} = i \land a_{i,x,m})$, where $i$ iterates over indices of all objects produced until the $m$-th step of the plan. Because the object $o$ exists in $w_m ([\text{pEx}(o)]^m$ evaluates to true) and this is
an object from \( out_{o} \), there exists an appropriate mapping, and one of the conditions \((m_{j,e} = i)\) is satisfied. If so, the value of \([pSet(o.a)]_{m}\) is equal to a valuation of the attribute variable \(a_{i,x,m}\). The correctness of encoding of \(pNull(o.a)\) can be proved analogously.

Besides the propositions, we use also the set of level constraints \(LC\) over the stamps \(ST\). They are interpreted over a model \( M = (\rho, V_{\rho}) \), that is, a world solution \( \rho \) augmented with the level valuation function \( V_{\rho} \). According to the semantics, we have: \( w_{m} \models (\text{lexp} \sim \text{lexp}'_{m}) \) iff \( V_{\rho}(w_{m}, \text{lexp}) \sim V_{\rho}(w_{m}, \text{lexp}') \) and \( V_{\rho}(w_{m}, \text{lexp}) \neq \infty \neq V_{\rho}(w_{m}, \text{lexp}') \). We show that the formula \([\text{lexp} \sim \text{lexp}']_{m}\) is satisfiable iff \( V_{\rho}(w_{m}, \text{lexp}) \sim V_{\rho}(w_{m}, \text{lexp}') \) and \( V_{\rho}(w_{m}, \text{lexp}) \neq \infty \neq V_{\rho}(w_{m}, \text{lexp}') \).

Thus, the level constraints are predicates over level expressions. In a simplest case a level expression is a (integer) constant, so its encoding is straightforward. Another possible form of a level expression is a stamp.level expression, for \( \text{stamp} \in ST \). In this case, the encoding involves an integer variable \( l_{i,e} \) bounded with a value equals to the step of the plan in which a given stamp has been created, computed by \( lev_{q} \) function (see Formulae 5, 6). Because once created, the stamp has a constant value of its level attribute in all subsequent worlds, it is enough to introduce only one variable \( l_{i,e} \) for each stamp used to construct the level constraints.

Moreover, we deal also with arithmetic over level expressions. We use arithmetic operators built in SMT-solver theories (like, e.g. Linear Integer Arithmetic), and we just rewrite the composite level expressions to the syntax accepted by the SMT-solver making use of simple level expressions, discussed above. We do the same while building level constraint predicates. To this aim we exploit standard comparison operators also built in SMT-solver theories. Finally, to encode the condition \( V_{\rho}(w_{m}, \text{lexp}) \neq \infty \neq V_{\rho}(w_{m}, \text{lexp}') \), we introduce the conjunction of of formulas encoding the existence of each stamp used to construct the level constraints. Thus, the formula \([\text{lexp} \sim \text{lexp}']_{m}\) is satisfiable iff the level constraint is satisfied, and the valuation function \( V_{\rho} \) all involved stamps does not assign \( \infty \).

Then, having the propositions and level constraints correctly encoded, let us proceed with \( LTL_{k}^{X} \) formulae. The negations of propositions and level constraints are obvious. Similarly, the conjunctions and disjunctions of \( LTL_{k}^{X} \) formulae are just conjunctions, and disjunctions, respectively, of their encoded counterparts.

Finally, the correctness of the encoding of \( Until \) and \( Release \) modalities can be shown by induction on the length of a formula.

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
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:

\[
in = \{R_{jlb}p \, r_{jlb}p\}, \quad inou = \{B_{ozwd} \, b_{ozwd}\}, \quad out = \{O_{pufo} \, o_{pufo}\}, \quad \text{pre} = isSet(b_{ozwd}.avg) \land isSet(r_{jlb}p.ppw), \quad \text{post} = isSet(o_{pufo}.e_{pufo}) \land isNull(b_{ozwd}.d_{ozwd}) \land isNull(e_{jlb}b.z_{jlb}e) \land isSet(b_{ozwd}.f_{ozwd})\]

First, we ran our planner for each ontology and each query instance without a temporal query (column \( \psi_1 \)), in order to collect statistics concerning the time needed to find the first plan (\( P_1 \)), all 10 plans (\( P_{10} \)), 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>
<thead>
<tr>
<th>( n \times k \times (\psi) )</th>
<th>( P_1 )</th>
<th>( P_{10} )</th>
<th>( T )</th>
<th>( M )</th>
<th>( P_1 )</th>
<th>( P_{10} )</th>
<th>( T )</th>
<th>( M )</th>
<th>( P_1 )</th>
<th>( P_{10} )</th>
<th>( T )</th>
<th>( M )</th>
<th>( P_1 )</th>
<th>( P_{10} )</th>
<th>( T )</th>
<th>( M )</th>
</tr>
</thead>
<tbody>
<tr>
<td>6x23</td>
<td>7.3</td>
<td>9.3</td>
<td>19.1</td>
<td>19.7</td>
<td>2.3</td>
<td>1.7</td>
<td>5.4</td>
<td>4.9</td>
<td>2.1</td>
<td>1.4</td>
<td>1.2</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>9x33</td>
<td>5.1</td>
<td>14.7</td>
<td>17.3</td>
<td>20.1</td>
<td>34.5</td>
<td>47.0</td>
<td>58.3</td>
<td>45.9</td>
<td>38.8</td>
<td>19.2</td>
<td>22.3</td>
<td>11.8</td>
<td>10.1</td>
<td>14.7</td>
<td>13.5</td>
<td>11.0</td>
</tr>
<tr>
<td>12x42</td>
<td>101</td>
<td>354</td>
<td>117</td>
<td>60.5</td>
<td>85.2</td>
<td>87.6</td>
<td>95.3</td>
<td>95.1</td>
<td>122</td>
<td>127</td>
<td>68.1</td>
<td>78.6</td>
<td>82.5</td>
<td>89.2</td>
<td>89.2</td>
<td>89.2</td>
</tr>
<tr>
<td>15x52</td>
<td>329</td>
<td>762</td>
<td>381</td>
<td>601</td>
<td>136</td>
<td>136</td>
<td>136</td>
<td>136</td>
<td>136</td>
<td>136</td>
<td>136</td>
<td>136</td>
<td>136</td>
<td>136</td>
<td>136</td>
<td>136</td>
</tr>
<tr>
<td>18x62</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

The table 1. Experimental results

The next experiments involve temporal queries using level constraints. To this aim we extended the \( \text{out} \) set of the generated queries by the appropriate stamp set. Moreover, the \( \text{post} \) formulas of the queries have been also extended with the expression: \( \bigwedge_{i=1}^{\lfloor \frac{n}{k} \rfloor} (s_i.\text{serviceClassInstanceOf} \, C_i) \), where \( s_i \in ST \), while \( C_i \in S \) are service types occurring in the solutions generated by OG.
Our second group of experiments involved the temporal formula $\psi_2$:

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

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:

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

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 $\psi_3$ 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 $\psi_4$ specifying the strict ordering of several services in the solution using stamp-based level constraints:

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

The analysis of the results (given in the column $\psi_4$ 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:

$$\psi_5 = \neg pEx(o) U_{<\lfloor k/2 \rfloor} pSet(o.a)$$

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 $\psi_5$ 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


