On Generation of Context-Abstract Plans

Łukasz Mikulski¹, Artur Niewiadomski², Marcin Piątkowski¹, and Sebastian Smyczynski³

¹ Nicolaus Copernicus University, Chopina 12/18, 87-100 Toruń, Poland
lukasz.mikulski@mat.umk.pl
² Siedlce University, 3-Maja 54, 08-110 Siedlce, Poland
³ Simplito Computer Science Lab, Królowej Jadwigi 7/2, 87-100 Toruń, Poland

Abstract. This paper deals with an intermediate phase of resolving Web Service Composition Problem (WSCP) provided by PlanICS. The abstract planner discovers a set of abstract plans for a WSCP instance. The proposed algorithm utilizes the combinatorial structure of this set and, abstracting from object attributes, browses the space of all potential solutions taking into account only indistinguishable ones. Finally, the reported results are validated by checking the attributes valuation and presumed constraints.

1 Introduction

Following [6], the existing solutions of the Web Service Composition Problem (WSCP) are mostly based on automata theory [11], situation calculus [2], Petri nets [7], planning graphs [1], and model checking [16]. In the context of this paper we consider other approaches to WSCP exploiting partial orders or combinatorial algorithms.

Peer in [14] extends a Partial Ordering Planning (POP) by adding a set of causal link patterns that must be avoided by the planner. In the combination with the replanning algorithm, it is used to solve WSCP. Wang et al. in [18] construct partial order plans from a pool of atomic services described in OWL-S.

Most of combinatorial approaches in the Web Service domain is related to compositions based on Quality of Service. For example, Yu et al. in [19] model WSCP as Multiple-Choice Knapsack Problem and present an algorithm which maximizes the utility function satisfying the constraints, while Zou et al. in [5] focus on WSCP in multi-cloud environment. Finally, Höfner et al. in [8] present an algebraic structure of Web Services, assisting users in WS composition.

In this paper we follow the PlanICS [4] approach. One of its key ideas is to divide the composition process into several stages. The first phase, called abstract planning, deals with classes of services, where each class represents a set of real-world services, while the second works in the space of concrete services. The first stage produces an abstract plan, which becomes a concrete plan in the second phase. It reduces dramatically the number of concrete services to be considered.

This research was supported by the National Science Center under the grants No. 2011/01/B/ST6/01477 and No. 2013/09/D/ST6/03928.

© Springer International Publishing Switzerland 2015
DOI: 10.1007/978-3-319-15201-1_25
On Generation of Context-Abstract Plans

The main goal of the abstract planning phase is to find a number of abstract plans that potentially satisfy a user query. Our planners [12,15] deal with this problem by finding plans composed of the same service types that belong to different equivalence classes. From the efficiency point of view, the number of such classes can be exponentially smaller than the number of plans. Each equivalence class is defined by a multiset of service specifications (Fig. 1).

![PlanICS architecture diagram](image)

Fig. 1. PlanICS architecture

This paper focuses on the next step of the planning process, which consists in processing the multisets of service types obtained from abstract planners. The module addressing this issue is called Multiset Explorer. Our goal is to generate all significantly different Context Abstract Plans (CAPs) composed of the service types from the explored multiset that belong to different partial orders and the difference between utilized objects is more complex than a simple inheritance extension. The motivation behind this consists in the fact that, typically, we would like to have a large selection of plans at our disposal, but we do not want to distinguish between plans, which differ only in the ordering of their context independent services. This observation is strongly related to the original concept of Mazurkiewicz traces [9] – the sets of indistinguishable executions of sequences of partially ordered actions. In this paper we adapt the notion of traces to cut the set of all context-abstract plans.

The rest of the paper is structured as follows. We start with presenting some basic notions which are used in further considerations. In the next section, we build a bridge between assumptions and notations used in PlanICS system and our algorithm. It is followed by the main contribution of the paper – the results of our studies on the combinatorial structure of context-abstract plans. In the subsequent section we utilize this to develop an algorithm solving the problem of searching for the solutions overlooked by the abstract planner. The paper is summarized by short sections containing results of experiments and conclusions.
2 Basic Notions

Throughout the paper we use the standard notions of Set and Formal Languages Theories. In this section we recall the most important definitions.

By an alphabet we mean a nonempty finite set $\Sigma$, the elements of which are called letters. Finite sequences over $\Sigma$ are called words. The set of all words (including the empty word $\varepsilon$) is denoted by $\Sigma^*$. Let $w = a_1 \ldots a_n$ be a word. By the alphabet of $w$ we mean $\text{alp}(w) \subseteq \Sigma$ consisting of all letters contained in $w$. Namely, $\text{alp}(w) = \{a \in \Sigma \mid \exists_{0 < i \leq n} a = a_i\}$. Moreover, by $\#_w(a)$ we denote the number of occurrences of $a$ in $w$. We also introduce the notion of the letter occurrence. The set $\text{occ}(w)$ of letter occurrences in $w$ comprises all pairs $\langle a, i \rangle$ with $a \in \text{alp}(w)$ and $1 \leq i \leq \#_w(a)$. In what follows we move freely between sequences of letters and sequences of letter occurrences.

Given $R \subseteq X \times X$, $R^0 = \emptyset$ and $R^n = R^{n−1} \circ R$, for all $n \geq 1$. Then (i) the inverse of $R$ is given by $R^{-1} = \{(b, a) \mid (a, b) \in R\}$; (ii) the symmetric closure by $R^{\text{sym}} = R \cup R^{-1}$; (iii) the reflexive closure of $R$ is defined by $R \cup \emptyset$; (iv) the transitive closure by $R^+ = \bigcup_{i \geq 1}R^i$; (v) the reflexive transitive closure by $R^* = \emptyset \cup R^+$; (vi) the transitive reduction by $R^{\text{red}} = \bigcap_{Q \subseteq R^+}Q$; and (vii) the largest equivalence relation contained in $R^*$ by $R^\circ = R^* \cap (R^*)^{-1}$.

A relation $R \subseteq X \times X$ is: (i) symmetric if $R = R^{-1}$; (ii) asymmetric if $R \cap R^{-1} \subseteq \emptyset$; (iii) reflexive if $\emptyset \subseteq R$; (iv) irreflexive if $\emptyset \cap R = \emptyset$; (v) transitive if $R \circ R \subseteq R$; and (vi) total if $R \cup R^{-1} = X \times X$; (vii) equivalence relation if it is symmetric, transitive and reflexive. A set of all equivalence classes of $R$ is denoted by $X/R$. An equivalence class containing an element $x \in X$ is denoted by $[x]_R$, or simply by $[x]$ if $R$ is clear from the context.

A relation $R \subseteq X \times X$ is (i) a (weak) partial order if it is asymmetric, reflexive and transitive; (ii) a strict partial order if it is irreflexive and $R \cup \emptyset$ is a weak partial order. A pair $PS = \langle X, R \rangle$ is a (strict) partially ordered set (poset in short). A (strict) poset $PS = \langle X, R \rangle$ is called well-founded if there is no infinite sequence $(x_1, x_2, \ldots)$ of distinct elements from $X$ such that $(x_{i+1}, x_i) \in R$.

Let $R \subseteq X \times X$ be a partial order. $F \subseteq X$ is called a filter if (i) $F \neq \emptyset$, (ii) $x \in F$ and $\langle x, y \rangle \in R$ implies $y \in F$, and (iii) $x, y \in F$ implies existence of $z \in F$ such that $\langle z, x \rangle \in R$ and $\langle z, y \rangle \in R$. A filter $F$ is principal if there exists a minimal element $x \in F$ (called the principal element). Note that if poset is well-founded, all its filters are principal.

A concurrent alphabet is a pair $\Psi = \langle \Sigma, \text{dep} \rangle$, where $\Sigma$ is an alphabet and $\text{dep} \subseteq \Sigma \times \Sigma$ is a reflexive and symmetric dependence relation. The corresponding independence relation is given by $\text{ind} = (\Sigma \times \Sigma) \setminus \text{dep}$. $\Psi$ defines an equivalence relation $\equiv_\Psi$ identifying words which differ only by the order of independent letters. Equivalence classes of $\equiv_\Psi$ are called Mazurkiewicz traces.

With every word $w = a_1 \ldots a_n$ we can associate a strict poset $\langle \text{occ}(w), \subseteq_w \rangle$ induced by the dependence relation over occurrence labels. The relation $\subseteq_w$ is defined as the transitive closure of the relation consisting of all pairs $\langle a_i, a_j \rangle$ such that $\langle a_i, a_j \rangle \in \text{dep}$ and $i < j$. Since an induced poset is a constitutive invariant of a trace, we lift this notion to the level of traces (see [10]).
We emphasize the special case of traces over concurrent alphabets with \( \text{dep} = \mathbb{I} \). In this case the only important information constituting a trace are the numbers of letters contained in it. This gives a natural correspondence with multisets over \( \Sigma \) and a trace \([w]\) is usually called the Parikh vector (of a word \(w\)).

Let \( X \) be a finite nonempty set. The relational structure over \( X \) is a triple \( \langle X, ME, SO \rangle \), where (i) \( ME \subseteq X \times X \) is a symmetric and irreflexive relation called mutual exclusion, and (ii) \( SO \subseteq ME \) is an asymmetric and irreflexive relation called skeleton order. A relational structure \( S = \langle X, ME, SO \rangle \) is separable if \( SO^\circ = \mathbb{I} \). A strict partial order \( PO \subseteq X \times X \) is consistent with a relational structure \( RS = \langle X, ME, SO \rangle \) if \( SO \subseteq PO \) and \( PO^{\text{red}} \subseteq ME \subseteq PO^{\text{sym}} \). It is easy to prove that for a given relational structure \( S = \langle X, ME, SO \rangle \) there exists a partial order \( PO \) consistent with it if and only if \( S \) is separable.

Let \( RS_1 = \langle X_1, ME_1, SO_1 \rangle \) and \( RS_2 = \langle X_2, ME_2, SO_2 \rangle \) be relational structures. We say that \( RS_1 \) is equivalent by \( rsm : X_1 \rightarrow X_2 \) to \( RS_2 \) (denoted by \( RS_1 \equiv_{rsm} RS_2 \)) if \( rsm \) is a bijective function such that \( \langle x_1, x_2 \rangle \in ME_1 \) iff \( \langle rsm(x_1), rsm(x_2) \rangle \in ME_2 \), and \( \langle x_1, x_2 \rangle \in SO_1 \) iff \( \langle rsm(x_1), rsm(x_2) \rangle \in SO_2 \). We say that \( RS_1 \) is equivalent to \( RS_2 \) (denoted by \( RS_1 \equiv RS_2 \)) if there exists a function \( rsm \) such that \( RS_1 \equiv_{rsm} RS_2 \).

### 3 Planics Specification

The OWL language [13] is used as the Planics ontology format. The concepts are organized in an inheritance tree of classes, all derived from the base class, called Thing. There are three descendants of Thing: Artifact, Stamp and Service.

The branch of classes rooted at Artifact is composed of the object types, which the services operate on. The Stamp class and its descendants define special-purpose objects, often useful in constructing a user query. Classes derived from Artifact and Stamp are called the object types. Each class derived from Service, called the service type, stands for a description of a set of real-world services. It contains a formalized information about their activities. A service affects a set of objects (world before), and transforms it into a new set of objects (world after).

In this section we provide all definitions necessary to formalize the problem of generating CAPs from a multiset. As we operate on types of objects only, in what follows we abstract from the object attributes and stamps. The validation of the found CAP (taking into account object attributes and stamps) is done at the very end. It is realized by calling the external Planics library function.

**Types and Objects.** Let \( O, P \) be nonempty sets of objects and object types, respectively. Over the set \( P \) we define a binary inheritance relation \( Ext \subseteq P \times P \) which is transitive and irreflexive (hence also asymmetric and acyclic). Semantically \( \langle p_2, p_1 \rangle \in Ext \) means that the type \( p_2 \) is extended by the type \( p_1 \) (i.e. \( p_1 \) is the subtype of \( p_2 \)). Moreover, we assume that for any triple of types \( p_1, p_2, p_3 \in P \) we have that \( \langle p_2, p_1 \rangle \in Ext \) and \( \langle p_3, p_1 \rangle \in Ext \) implies \( p_2 = p_3 \) or \( \langle p_2, p_3 \rangle \in Ext \) or \( \langle p_3, p_2 \rangle \in Ext \) (hence multiple inheritance is excluded). In the complete Planics semantics, two types are in the relation \( Ext \) if the set of attributes of \( p_2 \) is a subset of the set of attributes of \( p_1 \).
By an object we mean a labelled instance of a type, namely a pair \( \langle \text{id}, t \rangle \), where \( t \in P \) and \( \text{id} \) is a unique object identifier. Following [4] we define a function \( \text{type} : O \to P \) such that \( \text{type}(\langle \text{id}, t \rangle) = t \). In the context of the system state, the finite set of objects is called a world. The set of all worlds is denoted by \( \mathbb{W} \).

For a technical reason we define a function \( \text{obj} : O^N \to 2^O \) which assigns to every finite sequence of objects the set of objects that are elements of this sequence. We extend this function to the sets of sequences of objects, sequences of sequences of objects, functions with a set of sequences as a codomain and so on. Intuitively, \( \text{obj}(A) \) means the set of all objects somehow “occurring” in \( A \). We use similar constructions for all maps that transform sets of objects.

Let \( X,Y \subseteq O \) and \( A \subseteq X \) be sets of objects. We say that \( \text{map} : X \to Y \) is \( A \)-invariant if restricted to \( A \) it is an identity function.

**Services and Their Specifications.** An abstract service specification is a quadruple \( \text{spec} = \langle \text{name}, \text{in}, \text{inout}, \text{out} \rangle \), where \( \text{in}, \text{out} \) and \( \text{inout} \) are multisets of object types, while \( \text{name} \) is a service name. Semantically, \( \text{in} \) is the multiset of types of read-only objects, \( \text{inout} \) is the multiset of types of objects whose state may be changed, while \( \text{out} \) is the multiset of types of newly created objects. We also assume that \( \text{inout} \) and \( \text{out} \) multisets may not be simultaneously empty. The set of all service specifications is denoted by \( \mathbb{S} \).

For a multiset of object types we define a (partial) context function \( \text{ctx} : \mathbb{N}^P \to (O^N)^P \), namely a function that to a given multiset \( M \) of types assigns sequences of objects. For \( p \in P \) we have \( \text{ctx}(M)(p)(i) = o_i^p \), for \( 1 \leq i \leq M(p) \), and undefined otherwise. For every defined value of \( \text{ctx}(M)(p)(i) \) we require that \( \langle p, \text{type}(o_i^p) \rangle \in \text{Ext} \) (i.e. to \( p \) we assign a sequence of objects which types are subtypes of \( p \)). Moreover, for \( i, j \leq M(p) \) we assume that \( i \neq j \) implies \( o_i^p \neq o_j^p \).

We extend the definition of \( \text{ctx} \) to the case of service specifications assigning to \( \text{spec} = \langle \text{name}, \text{in}, \text{inout}, \text{out} \rangle \) a quadruple \( \langle \text{name}, \text{IN}, \text{IO}, \text{OU} \rangle \), where \( \text{IN} = \text{ctx}(\text{in}) \), \( \text{IO} = \text{ctx}(\text{inout}) \) and \( \text{OU} = \text{ctx}(\text{out}) \). We require that the sets \( \text{obj}(\text{IN}) \), \( \text{obj}(\text{IO}) \) and \( \text{obj}(\text{OU}) \) are pairwise disjoint. The specified instance of a service specification is simply called a service. The set of all services is denoted by \( \mathbb{S} \).

We also define an abstraction function \( \text{abs} : \mathbb{S} \to \mathbb{S} \), complementary to the context function \( \text{ctx} \). For a given service \( \bar{s} = \langle \text{name}, \text{IN}, \text{IO}, \text{OU} \rangle \) it returns its specification \( \text{spec} = \langle \text{name}, \text{in}, \text{inout}, \text{out} \rangle \), where for every \( p \in P \) \( \text{in}(p) = |\text{obj}(\text{IN}(p))| \), \( \text{inout}(p) = |\text{obj}(\text{IO}(p))| \), and \( \text{out}(p) = |\text{obj}(\text{OU}(p))| \). As the same service (with the same specification) may occur many times in a transformation sequence, we use the notion of a service occurrence.

A service \( \langle \text{name}, \text{IN}, \text{IO}, \text{OU} \rangle \) may be used to transform a world \( w_1 \) into the world \( w_2 = w_1 \cup \text{obj}(\text{OU}) \) if \( \text{obj}(\text{IN}) \cup \text{obj}(\text{IO}) \subseteq w_1 \) and \( \text{obj}(\text{OU}) \cap w_1 = \emptyset \), which means that the difference between \( w_1 \) and \( w_2 \) is precisely the set of newly created objects. This formalizes the assumption that function \( \text{ctx} \) assigns globally new and unique names to newly created objects. We extend the transforming operation to sequences of services in a natural way.

Let \( w_1, w_2 \in \mathbb{W} \) be sets of objects. We say that \( w_1 \) is compatible by \( \text{map} \) (or simply compatible) with \( w_2 \) if \( \text{map} : w_1 \to w_2 \) is a bijective function such that
∀o∈w₁ \{\text{type}(o), \text{type}(\text{map}(o))\} ∈ \text{Ext}. We say that \(w₁\) is subcompatible with \(w₂\) if there exists a set of objects \(w₃ ⊆ w₂\) such that \(w₁\) is compatible with \(w₃\).

**User Queries.** In the approach presented in [4] user query specification has the same structure as a service specification. The core of the user query is a triple of multisets of types \(qs = \langle in_q, inout_q, out_q \rangle\). The interpretation of \(qs\) is a pair \(q = \langle W_{\text{init}}, W_{\text{exp}} \rangle\), where \(W_{\text{init}}\) and \(W_{\text{exp}}\) are sets of worlds called the *initial worlds* and the *expected worlds*, respectively. Since we disregard attributes, \(W_{\text{init}}\) and \(W_{\text{exp}}\) became singletons \(w_{\text{init}}\) and \(w_{\text{exp}}\), which have to be consistent with \(qs\), namely \(w_{\text{init}} = \text{obj}(\text{ctx}(in_q) \cup \text{ctx}(inout_q))\) and \(w_{\text{exp}} = \text{obj}(\text{ctx}(in_q) \cup \text{ctx}(inout_q) \cup \text{ctx}(out_q))\).

Let \(q = \langle w_{\text{init}}, w_{\text{exp}} \rangle\) be a user query. We say that a transformation sequence \(s\) satisfies \(q\) if \(s\) transforms a world \(w_{\text{init}}\) into a world \(w_{\text{fin}}\) and \(w_{\text{exp}}\) is subcompatible with \(w_{\text{fin}}\). The definition of subcompatibility is existential, and matching \(\text{map}\) used in it may be not unique and each of appropriate substitutions is interesting in terms of the full solution. We have to distinguish between two solutions obtained by two different matchings. Hence, a transformation sequence \(s\) satisfies the user query \(q = \langle w_{\text{init}}, w_{\text{exp}} \rangle\) by \(\text{map} : w_{\text{exp}} \to w_{\text{fin}}\) if \(s\) transforms \(w_{\text{init}}\) into \(w_{\text{fin}}\) and \(∀o∈w_{\text{exp}} \{\text{type}(o), \text{type}(\text{map}(o))\} ∈ \text{Ext}\).

A solution is presented as a *transformation sequence* \(s\), namely a sequence of services, which transforms \(w_{\text{init}}\) into \(w_{\text{fin}}\). We denote the set of all transformation sequences which start in \(w_{\text{init}}\) by \(\tilde{S}\). Note that all objects used by services contained in \(\tilde{s}\) must be present in \(w_{\text{init}}\) or be created before their first use.

### 4 Partitioning the Solution Domain

In the presented approach, the set of all possible transformation sequences is the solution domain. In this section we discuss the concept of grouping them into sets of indistinguishable solutions. The main idea of this partitioning of the solution domain is depicted on Fig. 2. The components presented there are described in detail in the rest of this section. We fix the user query \(q = \langle w_{\text{init}}, w_{\text{exp}} \rangle\), hence the set \(\tilde{S}\) of transformation sequences starting from \(w_{\text{init}}\) is also fixed.

We start from defining two independent notions of indistinguishability of transformation sequences. Their sources are the partial (not total) ordering of concurrent computations leading to the same result and the inheritance relation, which causes the multiplication of a single solution by replacing intermediate objects by their extended substitutes.

**Order Indistinguishability.** The first identification is based on the theory of Mazurkiewicz traces. It requires the proper definition of a dependence relation on services contained in a given transformation sequence. Let \(s, t\) be services and \(w₁\) is transformed by \(s t\) into \(w₂\). If \(s\) and \(t\) operate on disjoint sets of objects the order of their execution does not matter and so \(w₁\) is transformed into \(w₂\) also by \(t s\). We observe the same behaviour when the objects shared by distinct services \(s\) and \(t\) are accessed in read-only mode. In both situations mentioned above, it is safe to assume that \(s t\) and \(t s\) leads to the same change of the values of objects.
attributes. This assumption in all other situations is unjustified. Without taking into account the impact of changes made on the values of object attributes it is impossible to determine whether we can harmlessly change the order of $\bar{s}$ and $\bar{t}$.

Formally, let $dep$ be a dependence relation defined as specified above. Then $Maz = \langle \bar{S}, dep \rangle$ is a concurrent alphabet and $\equiv_{Maz}\subseteq \bar{S} \times \bar{S}$ is an equivalence relation. Being a valid transformation sequence is an invariant for equivalence classes of the relation $\equiv_{Maz}$.

**Proposition 1.** Let $\bar{s}, \bar{t} \in \bar{S}$. If $\bar{s} \equiv_{Maz} \bar{t}$ and $\bar{s}$ satisfies a user query $q$ by map then $\bar{t}$ satisfies $q$ by map.

**Filter Indistinguishability.** The source of redundant valid solutions is also the freedom in choosing names of intermediate objects. We say that transformation sequences $\bar{s}$ and $\bar{t}$ are indistinguishable if $\bar{s}$ arises from $\bar{t}$ by one-to-one changing of the object names; we denote it by $\bar{s} \cong \bar{t}$. In such a case we shall report at most one transformation sequence. If $\bar{s}$ differs from $\bar{t}$ not only by objects names but also objects types (with the inheritance preservation) we shall report the more general one only. It leads to the definition of the inheritance relation over transformation sequences.

Let $\bar{s}$ and $\bar{t}$ be transformation sequences. We say that $\bar{s}$ is extended by $\bar{t}$ (or $\bar{t}$ inherits from $\bar{s}$) if $\text{obj}(\bar{s})$ is compatible by a $\text{inh}$ with $\text{obj}(\bar{t})$. Recall that formally it means that $\forall o \in \text{obj}(\bar{s}) \langle \text{type}(o), \text{type}(\text{inh}(o)) \rangle \in \text{Ext}$. We denote it by $\bar{s} \preceq \bar{t}$.

**Fact 1.** Let $\bar{s}, \bar{t}, \bar{u}$ be three transformation sequences. Then (i) if $\bar{s} \preceq \bar{t}$ and $\bar{t} \preceq \bar{u}$ then $\bar{s} \preceq \bar{u}$ (relation $\preceq$ is transitive), (ii) if $\bar{s} \equiv \bar{t}$ then $\bar{s} \preceq \bar{t}$, and (iii) if $\bar{s} \preceq \bar{t}$ and $\bar{t} \preceq \bar{s}$ then $\bar{s} \equiv \bar{t}$. Hence the relation $\equiv \subseteq \bar{S} \times \bar{S}$ is an equivalence relation.

The above fact justify quotienting the set $\bar{S}$ by the relation $\equiv$ and extending $\preceq$ to the case of equivalence classes of $\bar{S} / \equiv$ (we keep the notation $\preceq$). The structure of $\bar{S} / \equiv$ helps in reporting only essential transformation sequences that satisfy user query. Namely, the following hold:

**Proposition 2.** Let $\bar{s}, \bar{t}$ be two transformation sequences. If $\bar{s} \equiv \bar{t}$ and $\bar{s}$ satisfies user query $q$ then $\bar{t}$ satisfies user query $q$. 

![Fig. 2. The partitions of the solution domain.](image-url)
Theorem 1. Let $F \subseteq \mathbb{S}/\sim$ be a filter with a principal element $[\hat{s}]$. If $\hat{s}$ satisfies user query $q$ then for all $[t] \in F$, $t$ satisfy user query $q$.

The two methods of clustering indistinguishable solutions presented above can be combined. As a result we obtain filters of traces (or traces of filters if we look from the opposite direction). We summarize this facts by the theorem, which is one of the central points of the presented solution.

Theorem 2. Let $\vec{s}, \vec{t} \in \vec{S}$ be transformation sequences.

(i) Let $\vec{u} \in \vec{S}$ be such that $\vec{u} \equiv_{\text{Maz}} \vec{s}$. If $\vec{s}$ is extended by $\vec{t}$ then there exists $\vec{v} \in \vec{S}$ such that $\vec{v} \equiv_{\text{Maz}} \vec{t}$ and $\vec{u}$ is extended by $\vec{v}$.

(ii) Let $\vec{v} \in \vec{S}$ be such that $\vec{v} \equiv_{\text{Maz}} \vec{t}$. If $\vec{s}$ is extended by $\vec{t}$ then there exists $\vec{u} \in \vec{S}$ such that $\vec{u} \equiv_{\text{Maz}} \vec{s}$ and $\vec{u}$ is extended by $\vec{v}$.

Abstract Partitions of $\vec{S}$. In the process of browsing all classes of indistinguishable solutions we utilize more abstract partitions of the set of all transformation sequences. We pay attention to maintain compliance with expected form of results. In other words, we do not want to partition any filters of traces.

As a first coarse cut we present an abstract topology which can be seen as a specification for traces. By an abstract topology we mean relations between single service occurrences, which are determined by their common objects. In contrast to traces, we would like to leave as much flexibility as possible (indicating the dependence between two services but leaving their order unspecified).

Formally, we utilize relational structures. We define mutual exclusion relation $ME_{\vec{x}}$ identical as dependence in the case of traces. Two occurrences of services $\langle \bar{x}, i \rangle$ and $\langle \bar{y}, j \rangle$ are in $ME_{\vec{x}}$ if $\bar{x} = \bar{y}$ or $(\text{obj}(\bar{x}) \cap \text{obj}(\bar{y})) \neq (\text{obj}(\text{IN}_{\bar{x}}) \cap \text{obj}(\text{IN}_{\bar{y}}))$. On the other hand, we specify the order on service occurrences only if it is invariant for all transformation sequences containing the same sets of occurrences. Namely, two occurrences of services $\langle x, i \rangle$ and $\langle y, j \rangle$ are in $SO_{\vec{x}}$ if $x = y$ and $i < j$ or $\text{obj}(\text{OUT}_{\bar{x}}) \cap \text{obj}(\bar{y}) \neq \emptyset$. Note that $SO_{\vec{x}} \subseteq ME_{\vec{x}}$ and all other conditions (e.g. irreflexivity of $ME_{\vec{x}}$ and $SO_{\vec{x}}$) are satisfied.

Proposition 3. Let $\vec{s}$ and $\vec{t}$ be transformation sequences. If $\vec{s}$ is extended by $\vec{t}$ or $\vec{s} \equiv_{Maz} \vec{t}$ then $\vec{s}$ and $\vec{t}$ (hence a single reported result) have the same topology.

The second coarse cut is based on the Parikh vectors of service specifications. It is utilized in [4] where a single valid solution found inside a class causes its exclusion from the further search. Therefore, the cut presented below is very important and embeds Multiset Explorer inside the Planics project. The algorithm presented in this paper describes a method of browsing such single class.

Two transformation sequences $\vec{s}$ and $\vec{t}$ are Parikh specification-equivalent if the abstractions of their alphabets are equal (denoted as $\vec{s} \equiv_{sPar} \vec{t}$). The relation $\equiv_{sPar}$ is reflexive, symmetric and transitive, hence it is an equivalence relation and we can quotient $\vec{S}$ by $\equiv_{sPar}$. The obtained equivalence classes are precisely the sets of transformation sequences skipped by the approach from [4] (which may overlook some nontrivially distinct solutions). In what follows we consider only a single equivalence class from $\vec{S}/\equiv_{sPar}$ and search it for other possible solutions. Hereby, the procedure of browsing all solutions became complete.
Proposition 4. Let $\vec{s}, \vec{t} \in \vec{S}$. If $\vec{s} \equiv_{sPar} \vec{t}$ then $|obj(\vec{s})| = |obj(\vec{t})|$.

**Parikh Equivalence.** To support the last level of partition, we present the context version of Parikh equivalence, based on the inheritance relation (instead of service specification only).

Two transformation sequences $\vec{s}$ and $\vec{t}$ are Parikh equivalent if their topology is not only the same but also implied in the same way by corresponding sets of objects. To formalize that, we first define Parikh compatibility of two transformation sequences. We say that $\vec{s}$ is Parikh compatible by $pmap$ with $\vec{t}$ if $obj(\vec{s})$ is compatible by $w_{init}$-invariant $pmap$ with $obj(\vec{t})$ and $occ(\vec{t}) = pmap(occ(\vec{s}))$. We denote it by $\vec{s} \prec_{Par} pmap \vec{t}$ (or $\vec{s} \prec_{Par} \vec{t}$ if $pmap$ is not important).

Now we introduce Parikh equivalence of $\vec{s}, \vec{t} \in \vec{S}$ as their simultaneous Parikh compatibility (denote by $\vec{s} \equiv_{Par} \vec{t}$). We also define Parikh inheritance-equivalence of $\vec{s}$ and $\vec{t}$ as an existence of $\vec{u}$ Parikh compatible with both $\vec{s}$ and $\vec{t}$ (denoted by $\vec{s} \equiv_{iPar} \vec{t}$). Both $\equiv_{Par}$ and $\equiv_{iPar}$ are equivalence relations. The elements of $\vec{s} / \equiv_{Par}$ and $\vec{s} / \equiv_{iPar}$ are called process templates and processes respectively.

The elements of $\vec{s} / \equiv_{Par}$ are fully compatible with all previously presented partitionings. Each process template is completely contained both in classes of $\equiv_{sPar}$ and classes of transformation sequences with the same topology. Every filter, trace and process is completely contained in a process template. The following theorem is crucial from the point of view of our algorithm correctness.

**Theorem 3.** Let $\vec{s}$ and $\vec{t}$ be transformation sequences.

(i) If $\vec{s} \equiv_{iPar} \vec{t}$ then $\vec{s} \equiv_{sPar} \vec{t}$, and $\vec{s}$ and $\vec{t}$ have the same topology.
(ii) If $\vec{s} \equiv_{Par} \vec{t}$ or $\vec{s} \equiv_{Par} \vec{t}$ then $\vec{s} \equiv_{iPar} \vec{t}$.
(iii) If $\vec{s}$ is extended by $\vec{t}$ then $\vec{s} \equiv_{Par} \vec{t}$.

We utilize Parikh compatibility to equip process templates with the algebraic structure. According to Theorem 3, each process template decomposes into processes. The following facts allows to extend the notion of Parikh compatibility to processes and to define a poset on process templates seen as sets of processes.

**Proposition 5.** Let $p$ and $q$ be two processes minimal in the sense of $\prec_{Par}$. If $p$ and $q$ belong to the same process template then $p = q$.

Finally, we take into account the expected world $w_{exp}$ and matchings between $w_{exp}$ and $w_{fin}$ separating distinguishable solutions. Each solution we are interested in has a form of a filter of traces and is totally contained in a process template (see Theorem 3). It remains to show that relations $\preceq$ and $\prec_{Par}$ are consistent. We show that each matching identifies a single filter of processes inside a process template which decomposes to the set of expected results.

**Proposition 6.** Let $\vec{s}, \vec{t} \in \vec{S}$. If $\vec{s} \prec_{Par} \vec{t}$ then $[\vec{s}] \equiv_{Par} \vec{t}$, while if $\vec{s}$ satisfies user query $q$ by map and $\vec{s} \prec_{pmap} \vec{t}$ then $\vec{t}$ satisfies $q$ by $pmap \circ map$.

**Theorem 4.** Let $\vec{s}, \vec{t}, \vec{u} \in \vec{S}$ be such that $\vec{u} \prec_{Par} \vec{s}$ and $\vec{u} \prec_{Par} \vec{t}$, and $\vec{s}$ satisfies user query $q$ by map. If $\vec{t}$ satisfies $q$ by $pmap_{2} \circ pmap_{1}$ then there exists $\vec{v} \in \vec{S}$ such that $\vec{v} \prec_{Par} \vec{s}$ and $\vec{v} \prec_{Par} \vec{t}$ and $\vec{v}$ satisfies $q$ by $pmap_{3} \circ map$.
5 Algorithm

The main goal of the procedure developed in this paper is to browse all transformation sequences that satisfy a given user query and have the same Parikh vector of service specifications. We present a module Multiset Explorer utilizing the notions presented heretofore. As an input we take the object types from the ontology, the user query \( q = (w_{\text{init}}, w_{\text{exp}}) \), and a multiset of service names.

Algorithm 1. Multiset Explorer

| Input: A CAP \( cp \) (context abstract plan) |
| Output: List of CAPs equivalent with \( cp \) |
1. Initialize a class \( scp \) of \( \equiv_{spar} \) with \( cp \);
2. foreach each process template \( tp \) do
   3. Construct bipartite graph \( G \) using \( \text{obj}(tp) \) and \( w_{\text{exp}} \);
   4. foreach process \( p \) do
      5. Use the topology to construct a dependence relation \( \text{dep} \);
      6. foreach trace \( \tau(p, \text{dep}) \) do
         7. report \( \text{minlex}(\tau) \);

Having an element of \( \vec{S}/\equiv_{spar} \) fixed, we start the partitioning by determining a single representative for each subclass. In the first step we divide \( \vec{S} \) into process templates. The chosen representatives are \( \prec_{\text{Par}} \)-maximal. Next we fix the names of the objects produced by the provided services \( O_{OU} \) (the objects from OU’s) and we determine the most abstract types which they may have. We use those objects, together with \( w_{\text{init}} \), to fill all IN’s and IO’s of provided services respecting the type inheritance (and updating the types of objects from \( O_{OU} \)).

Having the single process template \( T \) we take one of its maximal element \( \vec{s} \) and \( w_{\text{init}} \)-invariant matching \( \text{map} \) of \( w_{\text{exp}} \) with \( \text{obj}(\vec{s}) \) which can be realized for one of the elements of \( T \) (as in Theorem 4). For each such matching we compute a filter of processes. Those filters contain transformation sequences which completely cover the set of context-abstract plans satisfying user query. To browse all fitting \( w_{\text{init}} \) invariant matchings which guaranties subcompatibility with \( w_{\text{exp}} \) we construct bipartite graph \( G \) between objects from \( w_{\text{fin}} \setminus w_{\text{init}} \) produced by the chosen representative and \( w_{\text{exp}} \setminus w_{\text{init}} \). There is an edge \( E \) between \( o_1 \in w_{\text{fin}} \) and \( o_2 \in w_{\text{exp}} \) if \( (\text{type}(o_1), \text{type}(o_2)) \in \text{Exp} \) or \( (\text{type}(o_2), \text{type}(o_1)) \in \text{Exp} \). We enumerate all maximal matching in the constructed graph using the algorithm based on [17].

We make some final cuts in the obtained filters of processes \( FP \) to report the solution in the assumed form. We start by computing the relational structure \( RS \) (topology) which is common (by Theorem 3) for all transformation sequences from \( FP \). Finally, we compute all traces over achieved from \( RS \) concurrent alphabet. They induce strict partial orders consistent with \( RS \). We utilize an
algorithm from [10] generating all traces with a given Parikh vector and dispose those, which are not consistent with \( RS \). Every obtained transformation sequence (the principal element of a filter of traces) satisfies user query (in a context-abstract sense) and is a final result of Multiset Explorer. The reported results are a subject of further verification made by the external procedure provided in PlanIcs which validates them by attributes valuation.

**Experimental Results**

We implemented Multiset Explorer as a standalone application and made some tests using data randomly generated by Ontology Generator provided by PlanIcs.

The experiments have been performed on a computer equipped with 32 GB RAM and 4-core Intel Xeon 2.9 GHz CPU running CentOS 6.5 system. We used SMT solver Z3 [3] as an external procedure called by Abstract Planner. In all experiments we first search for context-abstract plans using Abstract Planner and SMT solver Z3. We fix the maximal number of reported CAPs to 100. Then we make use of all of those CAPs to explore the space of transition sequences Parikh specification-equivalent with solutions found in the first phase.

**Table 1.** Experiments. In subsequent columns we put a case name, length of solutions we search for, minimal length of existing solution, number of CAPs (at most 100) found by AP and all equivalent with them CAPs founded by Multiset Explorer.

<table>
<thead>
<tr>
<th>File</th>
<th>dpt</th>
<th>min</th>
<th>AP + Z3</th>
<th>ME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>CAPs</td>
<td>sec.</td>
</tr>
<tr>
<td>nn</td>
<td>5</td>
<td>10</td>
<td>22.6</td>
<td>1</td>
</tr>
<tr>
<td>nn</td>
<td>7</td>
<td>100</td>
<td>97.7</td>
<td>503</td>
</tr>
<tr>
<td>box</td>
<td>5</td>
<td>4</td>
<td>9.37</td>
<td>85760</td>
</tr>
<tr>
<td>box</td>
<td>6</td>
<td>4</td>
<td>25.96</td>
<td>2938662</td>
</tr>
<tr>
<td>ont5</td>
<td>5</td>
<td>11</td>
<td>18.0</td>
<td>1.0</td>
</tr>
<tr>
<td>ont5</td>
<td>6</td>
<td>5</td>
<td>100</td>
<td>39.4</td>
</tr>
</tbody>
</table>

The results listed in Table 1 allows to conclude that for solutions of minimal length, Multiset Explorer only confirms their uniqueness. This phenomenon may be caused by the implementation of Ontology Generator. For longer solutions, there are additional distinguishable context abstract plans reported by Multiset Explorer. Their numbers significantly depends on processed ontology. The main reason of noticed blowup is probably the large number of degrees of freedom in choosing additional services and objects which are not used in the creation of expected world. In the case of confirmatory behaviour of Multiset Explorer, the computation times are significantly better than the times of original procedures.

**6 Conclusions**

The main contribution of the paper is the partitioning of the solution domain for the WSCP. It is used in the presented Multiset Explorer (ME) – a module
of Planics. ME fills the gap left by abstract planning, i.e., the first stage of resolving WSCP.

To join indistinguishable service sequences into equivalence classes we use the notion of traces. ME utilizes an efficient algorithm for enumeration of all traces with a given Parikh vector (see [10]). This algorithm generates additional traces without any interpretation in WSCP. Improving this part of the module by its full adaptation is one of the straightforward future plans.

Multiset Explorer is intended to be the middle part of Planics. The input is taken from abstract planner, while the results are passed to the external validator. Performed tests show that ME has possibilities and limitations comparable to those of Abstract Planner. While at the moment ME is a standalone application, we work on incorporating it into the Planics system. As a first step of this integration we have implemented wrappers allowing exchange of data between Abstract Planner and Multiset Explorer.

References


