Automated Planning of Process Models:

Design of a Novel Approach to Construct Exclusive Choices

Authors:

Heinrich, Bernd, Department of Management Information Systems, University of Regensburg, Universitätsstrasse 31, D-93040 Regensburg, Germany, bernd.heinrich@ur.de

Mathias, Klier, Department of Management Information Systems, University of Regensburg, Universitätsstrasse 31, D-93040 Regensburg, Germany, mathias.klier@ur.de

Steffen Zimmermann, Department of Information Systems, Production and Logistics Management, University of Innsbruck, Universitätsstrasse 15, A-6020 Innsbruck, Austria, Steffen.Zimmermann@uibk.ac.at

Citation: Bernd Heinrich, Mathias Klier, Steffen Zimmermann, Automated Planning of Process Models: Design of a Novel Approach to Construct Exclusive Choices, Decision Support Systems, Volume 78, October 2015, Pages 1–14, ISSN 0167-9236, http://dx.doi.org/10.1016/j.dss.2015.07.005. (http://www.sciencedirect.com/science/article/pii/S0167923615000299)

Automated Planning of Process Models:

Design of a Novel Approach to Construct Exclusive Choices

Abstract

In times of dynamically changing markets, companies are forced to (re)design their processes quickly and frequently, which typically implies a significant degree of time-consuming and cost-intensive manual work. To alleviate this drawback, we envision the automated planning of process models. More precisely, we propose a novel algorithm for an automated construction of the control flow pattern 'exclusive choice', which constitutes an essential step toward an automated planning of process models. The algorithm is built upon an abstract representation language that provides a general and formal basis and serves as the vocabulary to define the planning problem. As part of our evaluation, we find that, based on a given planning problem, our algorithm is not subject to potential modeling failures. We further implement the approach in process planning software and analyze not only its feasibility and applicability by means of several real-world processes from different application contexts and companies but also its practical utility based on the criteria flexibility by definition, modeling costs, and modeling time.

Keywords: process model; automated planning; exclusive choice; algorithm; design science research

1. INTRODUCTION

In times of dynamically changing markets, companies must frequently (re)design their business processes to adapt them to new market conditions such as shifting customer needs and new offers of emerging competitors. At the same time, companies are increasingly embedded in interorganizational, process-based collaborations, a fact that makes process (re)designs all the more complex. For instance, we are involved in an extensive project including several process (re)designs with a European bank in which over 600 core business processes and 1,500 support processes are modeled in different departments and areas. These process models, which are composed of actions and corresponding control flows [33], are modeled using the ARIS toolset and documented in a company-wide process repository to support the standardization of processes and to have a common base for process (re)design projects. To keep the process models updated, it is necessary to frequently (re)design process models due to changing market conditions such as new products, new distribution channels, and new regulatory obligations. Several interviews with IT and business executives of the bank highlighted the fact that today's process (re)design projects are more cost-intensive and time-consuming than such projects

were 10 years ago due to their higher complexity. This change also became evident in interviews with executives of other branches such as insurance and engineering. The most frequently mentioned reasons for increasing costs and duration are the growing frequency and complexity of such process (re)design projects, which involve a significant degree of manual work (cf. also [32]).

The research strand of Semantic Business Process Management (SBPM) aims to alleviate this drawback by using semantic technologies to enable a higher level of automation when designing, processing, executing, and analyzing processes and process models [26]. Wetzstein et al. [58] structure the scope of SBPM in their SBPM lifecycle and differentiate four phases: SBP modeling, SBP implementation, SBP execution, and SBP analysis. In our research, we aim to contribute to SBP modeling. The objectives in this phase are the semantic annotation, the design, and the adaptation of process models in an automated manner and their evaluation to ensure feasibility and (practical) utility [58].

Focusing on the SBP modeling phase, we envision the automated planning of process models. We aim to develop a planning approach that automatically arranges semantically annotated actions in a control flow leading from an initial state to desired goal states. When applying such an approach, the (re)design of process models is no longer performed manually but by an algorithm that uses semantic concepts and automated reasoning. With this research, we aim to increase the flexibility by definition (cf. [49]) of the resulting process models and to (re)design process models - for processes that must be frequently (re)designed – to be more cost-efficient and less time-consuming compared with manual process modeling. For automated process planning, it is insufficient to construct sequences of actions because entire process models include control flow patterns [43]. The specific research goal of this paper is the automated construction of one of the most important control flow patterns, namely exclusive choice.

Therefore, we initially define an abstract representation language to express the preconditions (comprising everything an action requires to be applied, including input parameters) and effects (how an action affects the state of the world, including output parameters) of actions and belief states (possibly infinite sets of world states that may exist before and after applying an action). Using this abstract representation language, we define our planning problem and, most importantly, design a novel algorithm for the automated construction of exclusive choices. As part of the evaluation, we find that, based on a given planning problem, our algorithm is not subject to potential modeling failures. We further implement the approach in SBPM process planning software and analyze not only its feasibility and applicability by means of several real-world processes but also its practical

utility based on the criteria flexibility by definition, modeling costs, and modeling time.

The research presented in this paper is based on the Design Science Research paradigm [18, 27]. In the introduction, we motivated the research problem - the automated construction of exclusive choices. In Section 2, we discuss contributions addressing related research problems (prescriptive knowledge) and elaborate the research gap. In Section 3, we present a general approach for an automated planning of process models to inform our research problem (descriptive knowledge). In section 4, we introduce a running example to illustrate the basic idea of our approach as well as each design step in the remainder of the paper. In Section 5, we present our approach for an automated construction of exclusive choices. Section 6 is dedicated to the evaluation of our approach. In Section 7, we discuss limitations and directions for future research before we conclude with a summary of our key findings in Section 8.

2. RELATED LITERATURE

Works addressing research problems that are related to the automated construction of exclusive choices are found in the research fields of Automated Planning and SBPM. Beginning with the literature in Automated Planning, the planning problem addressed in this paper can be characterized as a nondeterministic planning problem with initial state uncertainty. Algorithms that can cope with nondeterminism and initial state uncertainty [4, 8, 28] are called conditional planning approaches. When constructing exclusive choices in process models, an approach must cope with large data types (e.g., double) and possibly infinite sets of world states. However, according to Geffner [15], a key problem in large state spaces – resulting from large data types and possibly infinite sets of world states – is representing belief states and enabling mapping of one belief state onto another. In this context, Bertoli et al. [4] propose the use of Binary Decision Diagrams. Another possibility is the implicit representation of a belief state by an initial state in combination with a sequence of actions that leads to the belief state to be represented [29]. Further planners explicitly enumerate all world states that may occur after applying an action [7]. However, in the context of planning process models and constructing exclusive choices, it is essential to cope with large data types accompanied by infinite sets of world states. This issue has not thus far been addressed by existing approaches. Moreover, existing conditional planning approaches operate with so-called observations, which are points in the plan at which it is necessary to validate some logical expression to define how to proceed. However, these observations are encoded separately in the form of observation variables and observation actions making them both part of the given planning domain. Here, the observations in the domain description constitute the only points in the plan in which the control flow

might branch (e.g., to construct exclusive choices). Thus, it is possible to consider exclusive choices using existing conditional planners [8, 28], but they must be "hard-coded" in the domain (e.g., by sensing actions) and are additionally restricted to Boolean variables. However, in the context of planning process models, the points in the plan at which exclusive choices appear are not given; rather the corresponding conditions for which the control flow branches must be planned considering large data types. This challenge has so far also not yet been addressed by existing planning approaches.

In addition to the literature on Automated Planning, we discuss related approaches in the field of SBPM structured according to the phases of the SBPM lifecycle by Wetzstein et al. [58]:

SBP analysis: This phase comprises process mining and the validation of existing process models. The goal of process mining algorithms is to deduce process models from event logs representing recorded information about (many) former executions of the considered process [50, 52]. The deduced process models can then be compared with the deployed process models and thus be used for conformance checking and optimization purposes [53]. Existing process mining algorithms are able to identify control flow patterns based on dependency relationships observed in the event logs [3, 14, 17, 50, 51, 56, 57]. However, because these approaches focus on dependencies among actions, they do not aim at deriving the conditions of exclusive choices, which is an indispensable step toward our goal of planning exclusive choices in process models. Moreover, to derive the conditions of exclusive choices in the case of large data types, the event log would have to contain information about a possibly infinite number of process executions, which is rather unrealistic. Another major difference between process mining and the automated planning of process models refers to the fact that process mining aims at *re*constructing models for *as-is* processes to capture the processes as they are actually being executed [57]. In contrast, the automated planning of process models focuses on the construction of *to-be* process models for a given planning problem. Further related work in the SBP analysis phase aims at examining the consistency of existing process models [13, 34, 55]. These approaches validate whether the actions (within a process model) are consistent both among themselves and with respect to the control flow patterns used. Thus, they check whether exclusive choices are consistently constructed in given process models but do not focus on elaborating a planning domain or an algorithm to construct exclusive choices.

SBP implementation and execution: Within these phases, (web) services that are required to execute processes are composed in an automated manner. For that purpose, multiple service composition approaches were developed in recent years that are motivated by a problem definition related to planning process models

[1, 5, 6, 11, 30, 36, 37, 38, 40, 42, 45, 46, 48, 54, 59] (for a current overview of research on web service composition see e.g., [44]). However, few of them consider conditions that are required to construct exclusive choices. For instance, Meyer and Weske [38] propose to extend an enforced Hill-Climbing algorithm to support the construction of alternative control flows. They add an or-split to the service composition "if subsequent services cannot be invoked in all states". However, they do not consider belief states able to consider possibly infinite sets of world states. Bertoli et al. [6] (and other authors such as Wu et al. [59]) propose a planning framework to create a composite service that can handle services specified and implemented using industrial standard languages for business process execution. However, they do not focus on planning conditions for exclusive choices but on identifying one feasible service composition based on a search tree. Wang et al. [54] aim at integrating conditional branch structures in automated web service composition to represent users' diverse and personalized needs in combination with dynamic environmental changes. They propose algorithms that are based on formalized user preferences (e.g., $P = AccountBalance \geq Payment$? – *PayInFull*: *PayByInstalments*; i.e., the user will pay in full, if (s)he has sufficient money; otherwise, (s)he will pay by instalments). These preferences are explicitly specified and given as part of the web service composition problem. Therefore, in contrast to constructing exclusive choices and conditions, the approach of Wang et al. [54] somehow predefines the points at which the control flow might branch (e.g., to construct an exclusive choice) and the corresponding conditions in the problem definition.

SBP modeling: This phase is about the automated construction of process models but has thus far been much less extensively researched compared with the other phases. Hoffmann et al. [31] aim at leveraging synergies with model-based software development and propose a heuristic that can be used to reduce additional modeling overhead caused by planning process models. They adapt a well-known deterministic planning system to allow for nondeterministic actions that are characterized by multiple possible disjunctive effects predefined using finite-domain variables. Based on predefined possible effects and corresponding case distinctions (one case for each possible outcome), exclusive choices can be defined after a nondeterministic action. However, Hoffmann et al. [31] aim at deriving neither the partitions (conditions) of exclusive choices nor multiple feasible process models differing in their exclusive choices. Other approaches that are associated with the SBP modeling phase do not address the construction of exclusive choices.

In summary, to the best of our knowledge, none of the existing approaches aims to cope with the following set of necessary characteristics when constructing exclusive choices in process models:

(1) Consider *large data types* and *possibly infinite sets of world states*;

(2) Construct exclusive choices in an automated manner without "hard-coded" observations in the given planning domain by *creating partitions and corresponding conditions*;

(3) Construct *multiple feasible process models* (*feasible solutions*) differing in their exclusive choices; and

(4) Construct exclusive choices in *to-be* process models *in an automated manner*.

Thus, we address an important research gap by constructing exclusive choices and enabling the planning of all feasible process models for a specific planning problem in an automated manner.

3. AUTOMATED PROCESS PLANNING APPROACH

The automated construction of exclusive choices is necessary but not sufficient to plan entire process models. Therefore, we started to design a planning approach of which the proposed algorithm to construct exclusive choices constitutes a fundamental part. Our planning approach is based on semantically annotated actions stored in an action library. The annotation of an action includes a semantic annotation of its logical *preconditions* and *effects*. In addition to the annotated actions, our starting point comprises an initial state representing the overall process input and one or more goal states representing the necessary process output. Thus, our planning approach constructs feasible process models in an automated manner. The three steps ① to ③ of the planning approach include the following:

① *Semantic-based reasoning of dependencies between actions*: To identify dependencies between actions, we use semantic reasoning. In other words, we analyze the semantic matching between preconditions and effects of actions in the action library. The dependencies are represented in a Dependency Graph. To construct this graph and exclude actions that cannot be part of feasible process models (feasible solutions), we apply a backward search algorithm starting from the goal state(s) and ending in the initial state.

② *Planning feasible sequences of actions*: The Dependency Graph describes no direct predecessor-successor-relationships among actions. Hence, in the second step, a forward search algorithm is applied to determine all sequences of actions leading from the initial state to the goal state(s). Thus, we obtain a Search Graph that is an acyclic, bipartite directed graph and comprises all feasible sequences of actions for the corresponding planning problem.

③ *Construction of control flow patterns and feasible process models*: To obtain process models (e.g., UML Activity Diagrams), planning only sequences of actions such as those represented by the Search Graph is insufficient. Rather, in the third step, control flow patterns [43] provided by process modeling languages

and describing the control flow must be constructed in an automated manner [21].

In the literature, there are approaches that can serve as an initial basis to design algorithms that address the steps of our planning approach. Concerning the *semantic-based reasoning of dependencies between actions* (cf. step ①**)**, existing approaches that are also used in web service composition can be considered [20, 23]. To *plan feasible sequences of actions* (cf. step ②), existing planning techniques [4, 8, 28] can be enhanced [25]. The *construction of control flow patterns and feasible process models* based on a Search Graph (cf. step ③) is especially challenging and innovative. In this step, the automated construction of exclusive choices is essential. This is supported by our analyses of several processes from different application contexts and companies (cf. Section 6). Indeed, we found that all considered processes contain one to many simple and nested exclusive choices, which illustrates the importance of exclusive choices for specifying the control flow in process models.

4. RUNNING EXAMPLE

The example is taken from the security order management of a European bank. We will illustrate our approach and each design step based on an excerpt of the order execution process that is part of the core business of the bank. In the past, this process had to be frequently (re)designed due to repeated launches of new products and regulations. The actions *validate order*, *assess risks*, *check competencies*, *check extended competencies*, and *execute order*, which are part of the excerpt of the process, are available in the library. In the initial state, an order is entered by a customer with the parameters *orderAmount* that may reach from 0 to 250,000, *orderState* with the value *entered*, and *orderType* with the feasible values *buyOrder* and *sellOrder.* In the goal state, the order must be executed (*executed* is the required value for the parameter *orderState*).

Applying a version of the well-known forward search algorithm of Bertoli et al. [4], which is implemented in our SBPM process planning software (cf. Section 6.2), results in the Search Graph depicted in Figure 1. This graph was chosen to ensure a sound and established general basis, which is also used by other approaches, for instance in the field of web service composition. The Search Graph is an acyclic, bipartite directed graph with a set of nodes and a set of labeled arcs. The set of nodes consists of actions (e.g., *check competencies*) and belief states (e.g., bs_1). The arcs are labeled with the preconditions (e.g., *precond*(*check competencies*)) and effects (e.g., *effects*(*check competencies*)) of the corresponding actions. Containing all sequences of actions leading from the initial state to the goal state(s), this graph provides a sound basis for constructing exclusive choices.

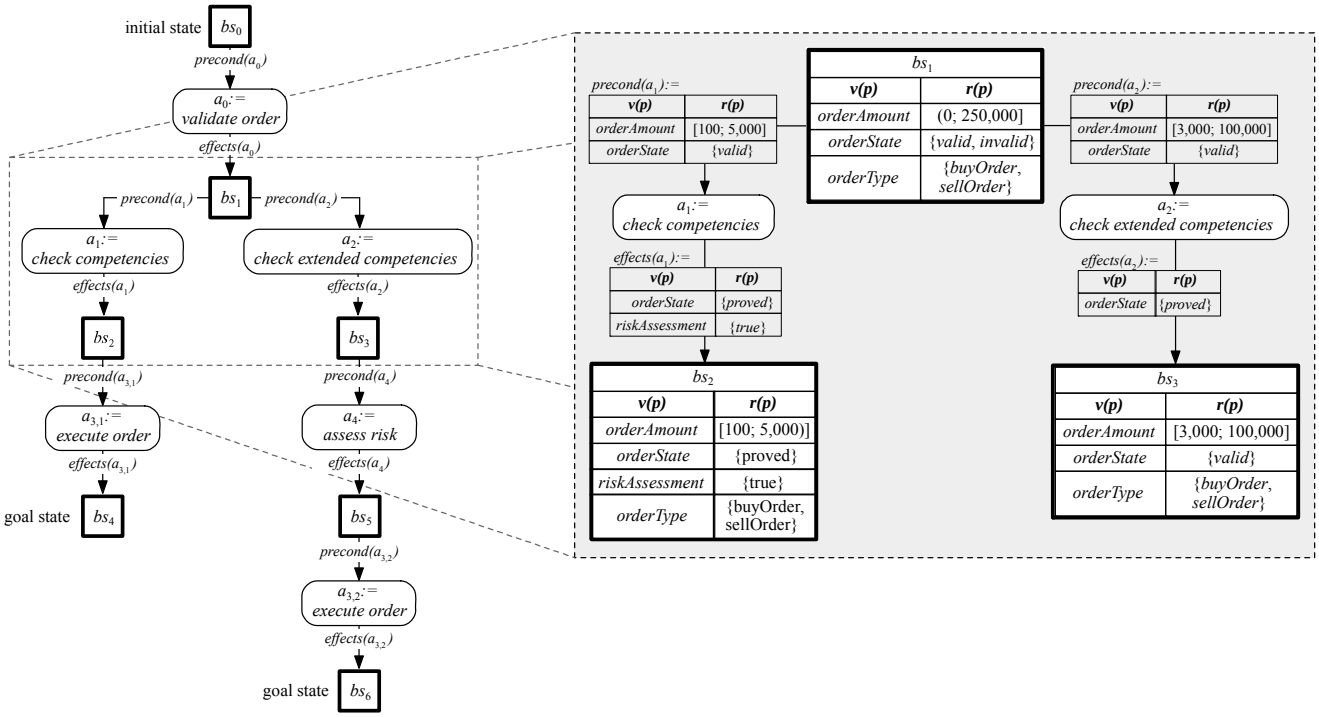

Fig. 1. Search Graph for constructing exclusive choices

For brevity, we only present in detail the part of the Search Graph in the dashed rectangle. In belief state bs_1, the action *validate order* has already been applied. Therefore, the parameter *orderState* is assigned either of the values *valid* or *invalid* (in contrast to *entered* in the initial state). In bs_1, it is necessary to decide which check routine to apply. In this context, the preconditions specified for the actions must be considered. The actions *check competencies* and *check extended competencies* are applicable to orders with an *orderAmount* between 100 and 5,000 and between 3,000 and 100,000, respectively. Both actions require *orderState valid* and result in *orderState proved*. Because the action *check competencies* comprises a lean risk assessment that is sufficient for orders with an *orderAmount* between 100 and 5,000, the effects of this action additionally contain the parameter *riskAssessment* with the value *true.* Thus, planning the action *check competencies* after belief state bs_1 leads to belief state bs_2. In contrast, the action *check extended competencies* does not comprise a simple risk assessment. Rather it is necessary to conduct a more comprehensive risk assessment by a separate action *assess risk*. Thus, planning the action *check extended competencies* after belief state bs_1 leads to belief state bs_3. This excerpt of the entire order execution process serves as a running example to illustrate our design steps.

5. NOVEL APPROACH TO CONSTRUCT EXCLUSIVE CHOICES

To design our approach, we initially specify an abstract representation language. Using this language, we define our planning problem and design an algorithm to construct exclusive choices.

5.1 Specifying an Abstract Representation Language

In this subsection, we present an abstract representation language for planning process models. Classical planners, which use other representation languages such as set-theoretic representation or state-variable representation [16], usually consider a finite set of world states and enumerate, for instance, all world states that may occur after applying an action explicitly [7]. This is not possible in the context of planning process models. Here, we must consider large data types and possibly infinite sets of world states. However, if we restrict all of the atoms and belief state variables to be ground, then from a theoretical point of view, our abstract representation language has equivalent expressiveness compared to the other languages mentioned. Our abstract representation language does not depend on any concrete process modeling language but provides a general and formal basis and serves as vocabulary to describe our planning problem and design our artifact. When talking about process models, the annotation of their actions includes a specification of the preconditions and the effects. We define a parameter of the preconditions and effects as *belief state tuple* that consists of the parameter's name and a subset of its predefined domain containing all values that can be assigned to the parameter in a specific world state (corresponding to an individual process execution). Thus, the name of a parameter is also understood as a variable that can take all values in the specified subset. The data type of a parameter is the predefined domain of a belief state tuple.

Definition 1 (Belief state tuple). A *belief state tuple p* is a tuple of a *belief state variable* $v(p)$ and a subset $r(p)$ of its predefined domain $dom(p)$, which we will write as $p := (v(p), r(p))$. It is $v(p) \in r(p)$ in a specific world state. When talking about belief states, $v(p)$ is the symbol of the belief state variable. The set $r(p) \subseteq dom(p)$ defined by logical and set-theoretic expressions is called the *belief-state-variable restriction* (abbr.: restriction) of $v(p)$, which contains all of the values that can be assigned to $v(p)$ in a specific world state. If $r(p) = \emptyset$ then the belief state variable does not exist (anymore), allowing its deletion.

According to this definition, each belief state variable $v(p)$ has a predefined data type specifying the domain $dom(p)$ (e.g., $dom(orderAmount)$ = double). Additionally, the restriction $r(p)$ can be defined for each belief state variable $v(p)$. A restriction can either be described by logical expressions (e.g., $r(orderAmount) = \{x \mid x \in dom(orderAmount) \wedge 100 < x \leq 5{,}000\} = (100; 5{,}000]$) defining a set of values or an explicit enumeration of discrete values (e.g., $r(orderState) = \{valid, invalid\}$) for a specific belief state variable. Example 1 illustrates the definition of belief state tuples in bs_1 of our running example.

Example 1. $bs_1 = \{(orderAmount, (0; 250{,}000])$, $(orderState, \{valid, invalid\})$, $(orderType, \{buyOrder,$

sellOrder})} represents a set of belief state tuples. The restriction of *orderAmount* is an interval of the double data type, whereas 0 is not part of the interval but 250,000 is. The domains of the belief state variables are predefined as $dom(orderAmount)$ = double, $dom(orderState) = \{entered, valid, invalid, proved, executed\}$, and $dom(orderType) = \{buyOrder, sellOrder\}$.

By using belief state tuples, we can represent belief states and describe in what sense we understand them as possibly infinite sets of world states with the help of Definition 3 in combination with Definition 2.

Definition 2 (⊑). Let $A = \{u_1, ..., u_k\}$ and $B = \{w_1, ..., w_m\}$ be two finite sets of belief state tuples. Then

$$A \sqsubseteq B :\Leftrightarrow \forall w \in B\ \exists u \in A: v(w) = v(u) \wedge r(u) \subseteq r(w) \wedge |r(u)|=1.$$

Definition 3 (Belief state and world state). Let $BST = \{p_1, \ldots, p_n\}$ be a finite set of *belief state tuples*. A *belief state bs* is a subset of *BST*, containing every belief state variable one time at most (i.e., $\forall p_i, p_j \in bs$, $p_i \neq p_j$: $v(p_i) \neq v(p_j)$). A world state *s* is a member of a belief state *bs*, in the context that $s \sqsubseteq bs$.

According to Definition 3, a belief state *bs* is characterized by a finite set of pairwise distinct belief state variables and their restrictions (i.e., $bs \subseteq BST = \{p_1, \ldots, p_n\}$ $\forall p_i, p_j \in bs, p_i \neq p_j: v(p_i) \neq v(p_j)$). Similar to Petrick and Bacchus [41], set *bs* can be interpreted as a type of knowledge base capturing the knowledge about available belief state variables $v(p_i)$ including the values $r(p_i)$ that can be assigned to these variables in a specific world state. In contrast to a world state – which generally refers to an individual process execution – *bs* describes different conceivable world states. Indeed, *bs* is a set of world states whereby in a world state *s* of belief state *bs* (i.e., $s \sqsubseteq bs$), each belief state variable $v(p_i)$ in *bs* is available and is assigned a specific concrete value which satisfies the corresponding belief state variable restriction $r(p_i)$ (cf. Definition 2). According to the literature [4, 7, 28, 29], a set of world states is called a belief state. Definition 3 follows this wording.

This means of representing a set of world states is the starting point to define our planning problem. Furthermore, we can explicitly represent belief states intuitively from a process modeling perspective. Additionally, with this representation language, a belief state can be a possibly infinite set of world states.

5.2 Specifying the Planning Problem

Our planning problem can be considered as a nondeterministic planning problem with initial state uncertainty. The theoretical basis for our approach is the framework given in Bertoli et al. [4]. However, we will describe our domain using the abstract representation language specified in Subsection 5.1 to allow the consideration of data types accompanied by possibly infinite sets of world states and to handle nondeterminism differently.

Furthermore, Bertoli et al. [4] explicitly assume given observations for which the control flow might branch on conditions as part of the problem domain. Thus, the observations and the corresponding conditions must be known in advance. Because this is not realistic in the context of process modeling, we must develop our own approach to make it possible to identify these points in the plan and to create conditions under which the plan branches (i.e., exclusive choice) in an automated manner.

In this subsection, we describe our design process, beginning with a nondeterministic state-transition system and its definition (Starting Point). To be able to cope with possibly infinite sets of world states, we modify and extend this definition. Instead of states, we use belief states in the transition system, which makes it possible to change the transition function to be deterministic concerning belief states. This approach results in the definition of a *deterministic belief-state-transition system* (First Step). To be able to build branches in the plan, we further extend the transition function by *conditions* and define our planning domain as a *conditional deterministic belief-state-transition system* (Second Step). Finally, the planning domain, in combination with the initial state and the goal states, represents our planning problem (Third Step).

5.3 Starting Point: Nondeterministic State-Transition System by Bertoli et al. [4]

When confronted with a nondeterministic planning problem, it is common to use a nondeterministic planning domain. In general, "a nondeterministic state-transition system is defined in terms of its states, its actions, and a transition function that describes how (the application of) an action leads from one state to possibly many states" [4]. We use this as a working definition of a nondeterministic state-transition system. More precisely, a state-transition system and (non)determinism in state space are defined as follows [4].

Definition 4 (Nondeterministic state-transition system). "A nondeterministic state-transition system is a tuple $\Sigma = (S, A, R)$, where

- S is a finite set of *states*,
- A is a finite set of *actions*, and
- $\gamma: S \times A \rightarrow 2^S$ is the transition function. The transition function associates to each state $s \in S$ and to each action $a \in A$ the set $\gamma(s, a) \subseteq S$ of next states."

Definition 5 ((Non-)determinism in state space). "An action a is *applicable* in a state s ([…]) iff $|\gamma(s, a)| > 0$; it is *deterministic* (*nondeterministic*) in s iff $|\gamma(s, a)| = 1$ ($|\gamma(s, a)| > 1$). If a is applicable in s, then $\gamma(s, a)$ is the set of states that can be reached from s by applying a."

As mentioned, we take this nondeterministic state-transition system as a starting point and modify and extend it to be able to cope with possibly infinite sets of world states.

5.3.1 First Step: Definition of a Deterministic Belief-State-Transition System

We defined a nondeterministic state-transition system and what we understand as a belief state. On this basis, we define a deterministic belief-state-transition system and determinism in belief space. Our transition system is called a belief-state-transition system because it is not based on states but on belief states. Similar to the nondeterministic state-transition system, we formulate a working definition of a *deterministic belief-state-transition system* that is defined in terms of its belief states (sets of world states), its actions, and a transition function that describes how (applying) an action leads from one belief state to one and only one belief state.

Definition 6 (Deterministic belief-state-transition system). A deterministic belief-state-transition system is a tuple $\Sigma_d = (BS, A, \gamma_d)$, where

- BS is a finite set of belief states,
- A is a finite set of actions, and
- $\gamma_d : BS \times A \rightarrow BS$ is the transition function. The transition function associates to each belief state $bs \in BS$ and to each action $a \in A$ the next belief state $\gamma_d(bs, a) \in BS$.

Definition 7 (Determinism in belief state space). An action a is *deterministic* in a belief state bs iff $|\gamma_d(bs, a)| = 1$. If a is applicable in bs, then $\gamma_d(bs, a)$ is the set of belief states that can be reached from bs by applying a.

According to Definition 7, the belief-state-transition system in Definition 6 is called deterministic because its transition function associates to each belief state $bs \in BS$ and to each action $a \in A$ the next belief state $\gamma_d(bs, a) \in BS$. Therefore, $|\gamma_d(bs, a)| = 1$ holds for each action a that is applicable in bs. This deterministic belief-state-transition system allows transitions from a set of world states to a set of next world states, which is made possible through using belief states. Hence, the nondeterministic state-transition system is extended to allow transitions from one world state to a set of next world states. In addition, belief states may represent possibly infinite sets of world states. Consequently, the deterministic belief-state-transition system extends the nondeterministic state-transition system, which can only cope with finite sets of world states.

Based on Definition 6, we define our planning domain as a deterministic belief-state-transition system $\Sigma_d = (BS, A, \gamma_d)$. To do so, we must specify the finite set of belief states BS, the finite set of actions A, and the

transition function $\gamma_d : BS \times A \rightarrow BS$, which associates to each belief state $bs \in BS$ and to each action $a \in A$ the next belief state $\gamma_d(bs, a) \in BS$. First, we provide some basic definitions.

Definition 8 (Action). Let $BST = \{p_1, \ldots, p_n\}$ be a finite set of belief state tuples. An *action* a is a triple consisting of the action name and two subsets of BST, which we will write as $a := (name(a), precond(a), effects(a))$. Here, $precond(a) \subseteq BST$ denotes the preconditions, and $effects(a) \subseteq BST$ denotes the effects of a, both containing each belief state variable one time at most (i.e., $\forall p_i, p_j \in precond(a)$, $p_i \neq p_j$: $v(p_i) \neq v(p_j)$ and $\forall p_i, p_j \in effects(a)$, $p_i \neq p_j$: $v(p_i) \neq v(p_j)$, respectively).

Definition 9 (μ). Let $A = \{u_1, \ldots, u_k\}$ and $B = \{w_1, \ldots, w_m\}$ be two finite sets of belief state tuples, both containing each belief state variable one time at most (i.e., $\forall u_i, u_j \in A$, $u_i \neq u_j$: $v(u_i) \neq v(u_j)$ and $\forall w_i, w_j \in B$, $w_i \neq w_j$: $v(w_i) \neq v(w_j)$, respectively). Then

$$A \mu B :\Leftrightarrow \forall w \in B \; \exists u \in A: v(w) = v(u) \wedge r(u) \subseteq r(w).$$

Definition 10 (Applicable). Let bs be a belief state and a an action. Then, action a is *applicable* in belief state bs (denoted with $applicable(a, bs)$) iff $bs \; \mu \; precond(a)$.

Definition 10 phrases a sufficient condition that must be met so that an action a can actually be applied in all possible world states of a belief state bs. All belief state variables in $precond(a)$ are available in bs. At the same time, the restriction of each belief state variable in bs is a subset of the restriction required by a belief state variable in $precond(a)$. In other words, an action is applicable iff the action can be applied in each world state $s \sqsubseteq bs$. On this basis, we define a deterministic belief-state-transition system.

Definition 11 (Deterministic belief-state-transition system). Let $BST = \{p_1, \ldots, p_n\}$ be a finite set of belief state tuples. We define our planning domain as a deterministic belief-state-transition system $\Sigma_d = (BS, A, \gamma_d)$, where

- $BS \subseteq 2^{BST}$ is a finite set of belief states (i.e., each belief state $bs \in BS$ is a subset of BST),
- A is a finite set of actions, where $precond(a) \subseteq BST$ and $effects(a) \subseteq BST$ for each $a \in A$, and
- $\gamma_d : BS \times A \rightarrow BS$ is the transition function that associates to each belief state $bs \in BS$ and to each action $a \in A$ the next belief state $\gamma_d(bs, a) \in BS$ as follows: $\gamma_d(bs, a) = \{(bs \setminus \{(v_{bs}, r_{bs}) \mid (v_{bs}, r_{bs}) \in bs \wedge (v_{effects}, r_{effects}) \in effects(a) \wedge v_{bs} = v_{effects}\}) \cup effects(a)\}$ if $applicable(a, bs)$, and $\varnothing$ otherwise.

BS is closed under γ_d, i.e., if $bs \in BS$, then for each belief state, $bs \in BS$, and each action $a \in A$, $\gamma_d(bs, a) \in BS$ holds (Definition 6).

Example 2 illustrates the *deterministic belief-state-transition system* for the application of the action *check*

competencies in bs_1 of our running example.

Example 2. Let (a_1 := *check competencies,* *precond*(a_1) := {(*orderAmount*, [100; 5,000]), (*orderState*, {*valid*})}, *effects*(a_1) := {(*orderState*, {*proved*}), (*riskAssessment*, {*true*})}) be an action and let bs_1 = {(*orderAmount*, [1,000; 3,000]), (*orderState*, {*valid*}), (*orderType*, {*buyOrder*, *sellOrder*})} be a belief state. a_1 is applicable in bs_1, since bs_1 μ *precond*(a_1) because for each variable among the preconditions of a_1 there is a variable in bs_1 and it is $[1{,}000; 3{,}000]_{orderAmount\_bs1} \subseteq [100; 5{,}000]_{orderAmount\_a1}$, $\{valid\}_{orderState\_bs1} \subseteq \{valid\}_{orderState\_a1}$. As a result it is $\gamma_d(bs_1, a_1)$ = {{(*orderAmount*, [1,000; 3,000]), (*orderState*, {*proved*}), (*riskAssessment*, {*true*}), (*orderType*, {*buyOrder*, *sellOrder*})}}.

5.3.2 Second Step: Definition of a Conditional Deterministic Belief-State-Transition System

In the deterministic belief-state-transition system of Definition 11, a transition can only occur if an action is applicable for all world states of a belief state. Therefore, the transition function γ_d might leave out transitions that are typical in the process modeling context. To illustrate this problem, compared with Example 2 we change the restrictions of belief state variables *orderAmount* and *orderState* of bs_1 (cf. also our running example illustrated in Figure 1).

Example 3. Let (a_1 := *check competencies*, *precond*(a_1) := {(*orderAmount*, [100; 5,000]), (*orderState*, {*valid*})}, *effects*(a_1) := {(*orderState*, {*proved*}), (*riskAssessment*, {*true*})}) be an action and let bs_1 = {(*orderAmount*, (0; 250,000]), (*orderState*, {*valid*, *invalid*}), (*orderType*, {*buyOrder*, *sellOrder*})} be a belief state. a_1 is not applicable in bs_1 because bs_1 o *precond*(a_1) (this is due to $(0; 250{,}000]_{orderAmount\_bs1} \not\subseteq [100; 5{,}000]_{orderAmount\_a1}$ and $\{valid, invalid\}_{orderState\_bs1} \not\subseteq \{valid\}_{orderState\_a1}$). The transition function $\gamma_d(bs_1, a_1)$ would be considered to result in a value of ∅, although it would be possible to apply a_1 if, for example, (*orderAmount*, {3,000}) and (*orderState*, {*valid*}) hold in an individual process execution (world state of bs_1).

To account for the fact that an action *a* may be applied in a certain world state of a belief state *bs*, although *a* is not applicable in *bs*, it is necessary to construct branches with conditions (which are the base to construct exclusive choices) in a process model. Hence, we extend the deterministic belief-state-transition system (Definition 6) by so-called *conditions*, which are comparable to the routing constraints in Sun et al. [47] and define a conditional deterministic belief-state-transition system.

Definition 12 (Conditional deterministic belief-state-transition system). A conditional deterministic belief-state-transition system is a tuple Σ_{cd} = (*BS*, *A*, *C*, γ_{cd}) where

- BS is a finite set of belief states,
- A is a finite set of actions,
- C is a finite set of conditions, and
- $\gamma_{cd} : BS \times 2^C \times A \rightarrow BS$ is the transition function. The transition function associates to each belief state $bs \in BS$, to each set of conditions $c \in 2^C$, and to each action $a \in A$ the next belief state $\gamma_{cd}(bs, c, a) \in BS$.

Based on Definition 12, we define our problem domain as a conditional deterministic belief-state-transition system $\Sigma_{cd} = (BS, A, C, \gamma_{cd})$. Before doing so, we provide some basic definitions.

Using a conditional deterministic belief-state-transition system, it is possible to consider an action that is not applicable in a belief state but is applicable in at least one specific world state s of this belief state bs ($s \sqsubseteq bs$).

Definition 13 (Partly applicable). Let bs be a belief state and a an action. Then, action a is *partly applicable* in belief state bs (denoted with *partly_applicable*(a, bs)) iff

$$\forall u \in precond(a)\ \exists w \in bs\colon v(u) = v(w) \wedge (r(u) \cap r(w) \neq \emptyset).$$

All belief state variables required by an action a are available in belief state bs. At the same time, for each belief state variable in *precond*(a), there is at least one value in bs that fulfills the corresponding restriction for that belief state variable. However, there may also be situations (certain world states of bs) in which applying a is not possible due to the restrictions of the belief state variables in *precond*(a). Example 4 illustrates the partial applicability of the action *check competencies* in bs_1.

Example 4 Let (a_1 := *check competencies*, *precond*(a_1) := {(*orderAmount*, [100; 5,000]), (*orderState*, {*valid*})}, *effects*(a_1) := {(*orderState*, {*proved*}), (*riskAssessment*, {*true*})}) be an action and let bs_1 = {(*orderAmount*, (0; 250,000]), (*orderState*, {*valid*, *invalid*}), (*orderType*, {*buyOrder*, *sellOrder*})} be a belief state. a_1 is partly applicable in bs_1 because it is $(0; 250{,}000]_{orderAmount\_bs1} \cap [100; 5{,}000]_{orderAmount\_a1} = [100; 5{,}000] \neq \emptyset$ and $\{valid, invalid\}_{orderState\_bs1} \cap \{valid\}_{orderState\_a1} = \{valid\} \neq \emptyset$.

In a belief state bs, there might be a nonempty set $A_{p\_a} \subseteq A$ of actions that are partly applicable in bs according to Definition 13. In an individual process execution, a specific value for every belief state variable in bs can be observed for a certain world state $s \sqsubseteq bs$. In this world state s, it may be possible to apply all actions in A_{p_a} or only the actions in a subset of A_{p_a}. In other words, for every belief state variable of a belief state, we must decide for which observations an action can actually be applied. Therefore, we must find a *set of conditions* under which an action can always be applied, that is, under which set of conditions an action is applicable in bs.

On this basis, it is possible to construct exclusive choices using these sets of conditions and the corresponding actions to construct branches in a process model. Former works [4] do not consider sets of conditions (sets of sets of observations).

Definition 14 (Condition). A *condition q* is a tuple of a *condition variable v(q)* and a subset $r(q) \neq \emptyset$ of its predefined domain $dom(q)$, which we will write as $q := (v(q), r(q))$ where $v(q) \in r(q)$ holds in a world state. When talking about conditions, $v(q)$ is the symbol of the condition variable. The set $r(q) \subseteq dom(q)$ is called the *condition restriction* of $v(q)$, which contains all of the values that can be assigned to $v(q)$ in a world state.

A set of conditions c must be built for all actions that are partly applicable in a belief state bs, which are the actions in $A_{p\_a}(bs) := \{a \in A \mid partly\_applicable(a, bs)\}$. This set of conditions c (observations) may be different for every action in $A_{p\_a}(bs)$, and is – in contrast to other planning problems – not provided prior to the construction of a process model. In particular, it is not predefined by the planning domain. Thus, it must be determined in an automated manner when constructing exclusive choices. In a world state $s \sqsubseteq bs$, we can then apply these actions iff all conditions are *fulfilled*, which means, for every condition in c, there is an observed value in the world state, and the observed value of the condition variable is a member of the corresponding condition restriction. Thus, it is known in each world state which actions can be applied. An action can be applied in a world state of a belief state if the action either is applicable in the belief state (i.e., $c = \emptyset$) or is partly applicable (but not applicable) in the belief state and at the same time all of its conditions are fulfilled.

As mentioned, the set of conditions may be different for every action that is partly applicable. To assign a set of conditions c to a belief state bs and an action a, we define a *condition function*. It associates to each belief state bs and each (partly applicable) action a a set of conditions under which a can be applied in bs.

Definition 15 (Condition function). Let $\Sigma_d = (BS, A, \gamma_d)$ be a deterministic belief-state-transition system. Let C be a set of conditions. A condition function over BS and A is a function $\theta : BS \times A \rightarrow 2^C$ that associates to each belief state $bs \in BS$ and to each partly applicable action $a \in A$ in bs the set of conditions $\theta(bs, a) \subseteq C$ under which this action is applicable.

Therefore, not only an action influences the transition from one belief state to another but also the conditions under which this action can be applied. When constructing a process model, the conditions are not predefined by the planning domain but must be determined when planning exclusive choices. Before we describe the algorithm, we formulate our planning domain as a conditional deterministic belief-state-transition system.

Definition 16 (Conditional deterministic belief-state-transition system). Let $BST = \{p_1, ..., p_n\}$ be a finite set of belief state tuples. We define our planning domain as a conditional deterministic belief-state-transition system $\Sigma_{cd} = (BS, A, \theta, \gamma_{cd})$, where

- $BS \subseteq 2^{BST}$ is a finite set of belief states,
- A is a finite set of actions, where $precond(a) \subseteq BST$ and $effects(a) \subseteq BST$ for each $a \in A$, and
- $\theta : BS \times A \rightarrow 2^C$ is a condition function over BS and A (Definition 15), with the set of conditions $C = \bigcup_{i=1}^{n} \left\{ (v(p_i), r) \middle| r \in 2^{r(p_i)} \setminus \varnothing \right\}$.
- $\gamma_{cd} : BS \times 2^C \times A \rightarrow BS$ is the transition function that associates to each belief state $bs \in BS$, to each set of conditions $c \in 2^C$, and to each action $a \in A$ the next belief state $\gamma_{cd}(bs, c, a) \in BS$ as follows: $\gamma_{cd}(bs, c, a) = \{((((bs \setminus \{(v_{bs}, r_{bs}) \mid (v_{bs}, r_{bs}) \in bs \wedge (v_c, r_c) \in c \wedge v_{bs} = v_c\}) \cup c) \setminus \{(v_c, r_c) \mid (v_c, r_c) \in c \wedge (v_{effects}, r_{effects}) \in effects(a) \wedge v_c = v_{effects}\}) \setminus \{(v_{bs}, r_{bs}) \mid (v_{bs}, r_{bs}) \in bs \wedge (v_{effects}, r_{effects}) \in effects(a) \wedge v_{bs} = v_{effects}\}) \cup effects(a)\}$ if $partly\_applicable(a, bs)$ and $c = \theta(bs, a)$, and $\varnothing$ otherwise.

BS is closed under γ_{cd}. In other words, $\gamma_{cd}(bs, c, a) \in BS$ holds for every action a that is partly applicable in $bs \in BS$ and for the corresponding conditions $c = \theta(bs, a) \in 2^C$ that must be considered.

In contrast to former approaches, our sets of conditions are not hard coded in the planning domain and are thus not given in advance because this is not realistic in the context of process modeling. Rather, they are derived from the domain applying the condition function $\theta : BS \times A \rightarrow 2^C$ to make it possible to construct exclusive choices in an automated manner.

5.3.3 Third Step: Definition of the Planning Problem

Based on the planning domain defined above (cf. Definition 16) in combination with the initial state and the goal states, our planning problem is defined as follows.

Definition 17 (Planning problem). Our *planning problem* is a triple $P = (\Sigma_{cd}, bs, BS_g)$, where

- $\Sigma_{cd} = (BS, A, \theta, \gamma_{cd})$ is a planning domain (Definition 16),
- $bs \in BS$ is a belief state ($bs \neq \varnothing$),
- $A_{p\_a}(bs) \subseteq A$ is the set of actions that are partly applicable in bs, and
- $BS_g \subseteq 2^{BS}$ is the set of belief states which can be reached after applying an action $a_i \in A_{p\_a}(bs)$. This set of belief states is defined as $BS_g = \bigcup_{i=1}^{m} \left\{ \gamma_{cd}(bs, \theta(bs, a_i), a_i) \in 2^{BS} \middle| a_i \in A_{p\_a}(bs) \right\}$, $m = \left| A_{p\_a}(bs) \right|$.

The planning problem states that, given the planning domain and belief state *bs*, the set BS_g of belief states, which can be reached after applying an action $a_i \in A_{p\_a}$, must be constructed.

5.4 Designing the Planning Algorithm

This subsection focuses on the construction of exclusive choices based on the defined planning problem (cf. Definition 17) and the Search Graph (cf. Figure 1). We must construct the required branches and conditions to construct exclusive choices and make partly applicable actions applicable in a belief state. Applying our algorithm results in an Extended Search Graph, which is an acyclic, bipartite directed graph consisting of a set of nodes *Nodes*, which contains the actions and belief states and a set of labeled arcs *Arcs*. However, in contrast to the Search Graph, the labels of the arcs, which lead from a belief state to an action that is partly applicable in this belief state, are extended by the conditions required to construct exclusive choices.

To generate the Extended Search Graph, we first introduce the EXTENDGRAPH procedure (Procedure 1), which creates nodes and arcs for the Extended Search Graph. Second, we present the CONDITIONS procedure (Procedure 2), which builds a set of conditions serving as the basis for constructing the branches in a given belief state. In this context, the PARTITION procedure (Procedure 3) is required to create disjoint partitions of the restrictions of belief state tuples. This procedure is provided in a third step. Finally, we introduce the CONDITIONFUNCTION procedure (Procedure 4), which builds the sets of conditions required to branch the Search Graph and label its arcs. How these parts of the algorithm work and how they interact with one another to construct the Extended Search Graph is described in the following and illustrated by our running example.

The EXTENDGRAPH procedure (Procedure 1) receives Search Graph *SG* and node *bs*. Clearly, the procedure is invoked for a belief state *bs* iff there are partly applicable actions in *bs* and an exclusive choice must be constructed accordingly. In a first step, by means of the CONDITIONS procedure, set *C* is built and contains the conditions required to construct the branches in node *bs*. Because these conditions depend on the preconditions of all partly applicable actions in *bs* and to avoid redundant calculations within the CONDITIONFUNCTION procedure, which must be executed for each partly applicable action in *bs* thereafter, we decided to conduct this step beforehand. For belief state bs_1 in our running example, the CONDITIONS procedure builds the set of conditions $C := \{$(*orderAmount*, $(0; 100) \cup (100{,}000; 250{,}000]$), (*orderAmount,* $[100; 3{,}000)$), (*orderAmount,* $[3{,}000; 5{,}000]$), (*orderAmount,* $(5{,}000; 100{,}000]$), (*orderState*, {*valid*}), (*orderState*, {*invalid*})} (details on how this set is built are provided in the CONDITIONS procedure).

For every action that is partly applicable in belief state *bs*, a set of sets of conditions C_{arc} is built by executing

the CONDITIONFUNCTION procedure. For the action *check competencies* and belief state bs_1 in our running example, the CONDITIONFUNCTION procedure returns the set of sets of conditions C_{arc} := {{(*orderAmount,* [100; 3,000)), (*orderState*, {*valid*})}, {(*orderAmount,* [3,000; 5,000]), (*orderState*, {*valid*})}} (details on how this set is built are provided introducing the CONDITIONFUNCTION procedure).

In a next step, based on the set C_{arc}, it is possible to construct the nodes and arcs that must be added to Search Graph *SG* to account for partly applicable action *a* in belief state *bs* (lines 5 to 9). Here, for each element c_{arc} of C_{arc}, a new node representing new belief state *bs'* (which can be reached by applying the action *a* in belief state *bs*) is built by applying the transition function $\gamma_{cd}(bs, c_{arc}, a)$ (Definition 16) and added to the set of nodes *Nodes*. Furthermore, new arcs $\langle bs, \{precond(a), c_{arc}\}, a\rangle$ and $\langle a, effects(a), bs'\rangle$ are created between *bs*, *a*, and *bs'* and added to the set of arcs *Arcs*. The outgoing arc of *bs* is thereby labeled with both the preconditions of action *a* and the corresponding set of conditions c_{arc}. For the action *check competencies* and belief state bs_1, this results in the two nodes $bs_{2,1}$ and $bs_{2,2}$ (Figure 2) and the corresponding arcs (in Figure 2, we omitted depicting preconditions *precond*(*a*) and effects *effects*(*a*)).

Having added the nodes and arcs for all partly applicable actions in $a \in A_{p\_a}(bs)$, we finally add the arc $\langle bs$, else, termination$\rangle$ to the Search Graph. This is because belief state *bs* might involve world states in which no action can be applied. In our running example, this condition is the case if belief state variable *orderState* equals *invalid*, for example.

Procedure 1. EXTENDGRAPH

```
1   procedure EXTENDGRAPH(bs, SG)
2       C := CONDITIONS(bs, SG)
3       forall a ∈ A_p_a(bs)
4           C_arc := CONDITIONFUNCTION(bs, C, a)
5           forall c_arc ∈ C_arc
6                   bs' := γ_cd(bs, c_arc, a)
7                   Nodes := Nodes ∪ {bs'}
8                   Arcs := Arcs ∪ {⟨bs, {precond(a), c_arc}, a⟩, ⟨a, effects(a), bs'⟩}
9           endfor
10      endfor
11      Arcs := Arcs ∪ ⟨bs, else, termination⟩
12  end
```

Within the EXTENDGRAPH procedure, the CONDITIONS procedure (Procedure 2) must be executed to derive the set of conditions *C*, which serves as a basis for constructing the branches of the Extended Search Graph in belief state *bs*. To this end, we initialize *C* with an empty set (line 2). Furthermore, we derive set bs_{pre} which contains all belief state tuples (v_{bs}, r_{bs}) of *bs* for which belief state variable v_{bs} is in the precondition, but restriction r_{bs} is not a subset of the corresponding restriction r_{pre} for at least one action a_{p_a}, which is partly

applicable in *bs* (line 3). In other words, for all belief state tuples (v_{bs}, r_{bs}) in bs_{pre}, there is at least one action which is partly applicable (i.e., this action cannot be applied in all possible world states of belief state *bs* concerning restriction r_{bs} of belief state variable v_{bs}). For belief state bs_1 in our running example, considering the partly applicable actions *check competencies* and *check extended competencies*, set bs_{pre} equals {(*orderAmount*, (0; 250,000]), (*orderState*, {*valid, invalid*})}.

Then, for each belief state tuple (v_{bs}, r_{bs}) of bs_{pre}, we build the set of conditions C_{part} to account for the preconditions of all partly applicable actions in *bs* and add them to the set of conditions *C* (lines 4 to 9). To do this, we first build set *R* consisting of all restrictions r_{pre} that refer to belief state variable v_{bs} and are part of the preconditions of the partly applicable actions in belief state *bs*. Next, to partition restriction r_{bs} of belief state variable v_{bs} in pairwise disjoint sets considering the restrictions $r_{pre} \in R$, we execute the PARTITION procedure with r_{bs} and *R*. Thus, restriction r_{bs} of belief state variable v_{bs} is partitioned such that it is clearly defined for each pairwise disjoint set and each action which is partly applicable in *bs,* whether the action can be applied in all possible world states or no world state at all with respect to this belief state variable v_{bs}. The PARTITION procedure is described in detail later on in this subsection. When deriving the set *Partition* for the belief state tuple (*orderAmount*, (0; 250,000]), for example, the PARTITION procedure is executed with r_{bs} := (0; 250,000] and *R* := {[100; 5,000], [3,000; 100,000]}. Here, the set of restrictions *R* consists of the restrictions concerning belief state variable *orderAmount*, which are part of the preconditions of the partly applicable actions *check competencies* and *check extended competencies*. In our example, executing the PARTITION procedure with r_{bs} and *R*, results in the set *Partition* := {(0; 100) ∪ (100,000; 250,000], [100; 3,000), [3,000; 5,000], (5,000; 100,000]}. In the next step, based on set *Partition*, set C_{part} is built, which contains the set of belief state tuples (v_{bs}, r_p) containing a restriction r_p that is an element of set *Partition*. For the belief state tuple (*orderAmount*, (0; 250,000]), set C_{part} equals {(*orderAmount*, (0; 100) ∪ (100,000; 250,000]), (*orderAmount*, [100; 3,000)), (*orderAmount*, [3,000; 5,000]), (*orderAmount*, (5,000; 100,000])}.

Finally, having executed lines 4 to 9 for each belief state tuple of bs_{pre} and having extended the set of conditions *C*, the resulting set *C* is returned by the CONDITIONS procedure. In our running example for bs_1 the set {(*orderAmount*, (0; 100) ∪ (100,000; 250,000]), (*orderAmount,* [100; 3,000)), (*orderAmount,* [3,000; 5,000]), (*orderAmount*, (5,000; 100,000]), (*orderState*, {*valid*}), (*orderState*, {*invalid*})} is returned.

Procedure 2. CONDITIONS

```
1   function CONDITIONS(bs, ST)
2       C := ∅
3       bs_pre := {(v_bs, r_bs) ∈ bs | (v_bs, r_pre) ∈ precond(a_p_a), r_bs ⊄ r_pre, a_p_a ∈ A_p_a(bs)}
4       forall (v_bs, r_bs) ∈ bs_pre
5           R := {r_pre | (v_bs, r_pre) ∈ precond(a_p_a), a_p_a ∈ A_p_a(bs)}
6           Partition := PARTITION(r_bs, R)
7           C_part := {(v_bs, r_p) | r_p ∈ Partition}
8           C := C ∪ C_part
9       endfor
10      return C
11  end
```

Within the CONDITIONS procedure, we execute the PARTITION procedure (Procedure 3) to partition the restriction r_{bs} of a belief state variable v_{bs} of a belief state *bs* in pairwise disjoint sets to account for the set of restrictions *R* for v_{bs} of all partly applicable actions in *bs*. We will examine this procedure more closely.

First, we nondeterministically take one element of *R* as *rest* and build the difference *diff* and the intersection *inter* of r_{bs} and *rest.* In doing so, we partition r_{bs} into two subsets. In our example with r_{bs} := (0; 250,000] and *R* := {[100; 5,000], [3,000; 100,000]} and taking the first element of *R* as *rest* (i.e., [100; 5,000]), we obtain the two subsets *diff* := (0; 100) ∪ (5,000; 250,000] and *inter* := [100; 5,000]. Furthermore, we initialize *solution* with an empty set.

If the number of elements in *R* is greater than 1 – i.e., there are, in addition to *rest,* further restrictions remaining in *R* that must be considered – we continue as follows (lines 6 to 12):

(1) If set *diff* is not empty as in our running example, we further partition *diff* by executing the PARTITION procedure for set *diff* and the difference of *R* and *rest* (i.e., all restrictions in *R* besides *rest*) and store the result in *solution*. In our running example, the recursive execution of the PARTITION procedure finally results in *solution* := {(0; 100) ∪ (100,000; 250,000], (5,000; 100,000]} (line 8). (2) If the set *inter* is not empty as in our running example, we further partition *inter* by executing the PARTITION procedure for set *inter* and all restrictions in *R* in addition to *rest.* In our running example, this recursive execution results in the set {[100; 3,000), [3,000; 5,000]}.

The union of both sets determined in (1) and (2) (line 11) finally results in the set *solution* := {(0; 100) ∪ (100,000; 250,000], [100; 3,000), [3,000; 5,000], (5,000; 100,000]}.

Else, for $|R| \leq 1$, solution equals the difference of the union of the two subsets *diff* and *inter* and the empty set (lines 13 to 15). Finally, the procedure returns the set *solution*.

Procedure 3. PARTITION

```
1   function PARTITION(r_bs, R)
2       nondeterministically choose rest ∈ R
3       diff := r_bs \ rest
4       inter := r_bs ∩ rest
5       solution := ∅
6       if |R| > 1 then
7           if diff ≠ ∅ then
8               solution := PARTITION(diff, R \ rest)
9           endif
10          if inter ≠ ∅ then
11              solution := solution ∪ PARTITION(inter, R \ rest)
12          endif
13      else
14          solution := {diff, inter} \ {∅}
15      endif
16      return solution
17  end
```

The CONDITIONFUNCTION procedure (Procedure 4) is central to the EXTENDGRAPH procedure and must be executed to branch the Search Graph and label its arcs. Based on the set of conditions C, previously derived by means of the CONDITIONS procedure, CONDITIONFUNCTION builds a set of sets of conditions C_{arc} under which a partly applicable action a can be applied in belief state bs. In the following, we will examine this procedure more closely.

If a is applicable in bs, there is no need for conditions. Therefore, the empty set, which was chosen in our algorithm to represent this fact (lines 2 and 3), is returned. This makes it possible to avoid redundant branches within the Extended Search Graph without any loss of feasible solutions.

If a is partly applicable but not applicable in bs, sets of conditions c_{arc} are created under which this action is applicable in bs (lines 4 and 7). Thus, in a first step, subset C_a of all belief state tuples (v_c, r_c) containing a restriction r_c, which is a subset of the restriction r_a of the corresponding belief state tuple (v_c, r_a) in the preconditions of a, is derived from C (lines 5). For the action *check competencies* and the belief state bs_1 in the running example, the set C_a equals {(*orderAmount*, [100; 3,000)), (*orderAmount*, [3,000; 5,000]), (*orderState*, {*valid*})}. In a next step, the required set of sets of conditions C_{arc} is built. C_{arc} includes all subsets c_{arc} of C_a, containing every condition variable that is part of the conditions in C_a exactly once (line 6). Here, the first term ensures that the number of conditions in c_{arc} (i.e., $|c_{arc}|$) equals the number of different condition variables in C_a and the second term relates to the fact that all condition variables which are part of the conditions in c_{arc} are pairwise distinct. In our running example the set of sets C_{arc} which is returned in line 7 equals {{(*orderAmount,* [100; 3,000)), (*orderState*, {*valid*})}, {(*orderAmount,* [3,000; 5,000]), (*orderState*, {*valid*})}}.

Procedure 4. CONDITIONFUNCTION

```
1  function CONDITIONFUNCTION(bs, C, a)
2      if applicable(a, bs) then
3          return {∅}
4      else
5          C_a := {(v_c, r_c) ∈ C | r_c ⊆ r_a, (v_c, r_a) ∈ precond(a)}
6          C_arc := {c_arc ⊆ C_a | |c_arc| = |{v_c | (v_c, r_c) ∈ C_a}| ∧ ∀i = (v_i, r_i) ∈ c_arc, j = (v_j, r_j) ∈ c_arc, i≠j : v_i≠v_j}
7          return C_arc
8  end
```

The application of our algorithm to the Search Graph of the running example results in an Extended Search Graph (i.e., Figure 2, wherein we omitted to depict the preconditions *precond*(*a*) and effects *effects*(*a*) on the arcs). The upper part of Figure 2 illustrates a part of this Extended Search Graph. Here, we focus on belief state bs_1 and the partly applicable actions *check competencies* and *check extended competencies*. Because belief states $bs_{2,2}$ and $bs_{3,1}$ are equal, the set *Nodes* which results from applying the algorithm on our running example contains only three elements, and not four as depicted for illustration purposes in Figure 2. This benefits further planning steps because the Search Graph must be further analyzed and extended only once for $bs_{2,2}$ and $bs_{3,1}$.

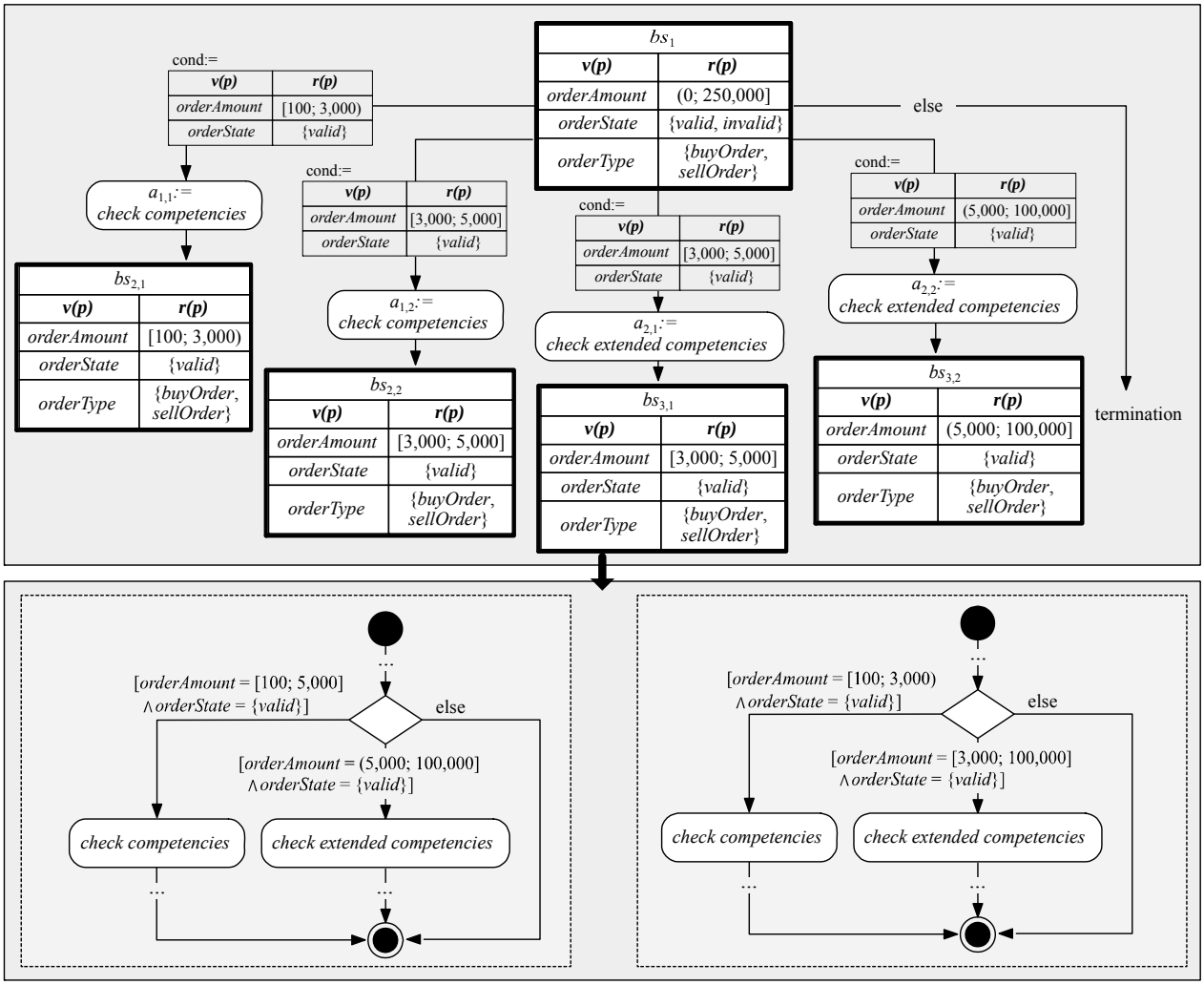

Fig. 2. Deriving UML Activity Diagrams from the Extended Search Graph

Based on the generated Extended Search Graph, we are able to construct exclusive choices. For example, considering the arc labels in the Extended Search Graph of Figure 2, an exclusive choice must be designed after bs_1 because the constructed restrictions concerning the condition variable *orderAmount* of outgoing arcs <bs_1, {*precond*(*check competencies*), {(*orderAmount*, [100; 3,000)), (*orderState*, {*valid*})}}, *check competencies*> and <bs_1, {*precond*(*check extended competencies*), {(*orderAmount*, (5,000; 100,000]), (*orderState*, {*valid*})}}, *check extended competencies*>, for example, are mutually exclusive. In contrast, the conditions of arcs <bs_1, {*precond*(*check competencies*), {(*orderAmount*, [3,000; 5,000]), (*orderState*, {*valid*})}}, *check competencies*> and <bs_1, {*precond*(*check extended competencies*), {(*orderAmount*, [3,000; 5,000]), (*orderState*, {*valid*})}}, *check extended competencies*> are equal. Thus, both the actions *check competencies* and *check extended competencies* can be alternatively executed if *orderAmount* ∈ [3,000; 5,000] and *orderState* ∈ {*valid*} hold. This fact indicates that two different process models are feasible. In one process model, the action *check competencies* is executed for *orderAmount* ∈ [3,000; 5,000]; in the other one, the action *check extended competencies* is executed. Both process models contain the control flow pattern exclusive choice, which is constructed using the conditions denoted on the arc labels. Two feasible solutions are illustrated in the bottom part of Figure 2 as UML Activity Diagrams.

6. EVALUATION

We first evaluate our design artifact in terms of whether it always creates correct exclusive choices for a given planning problem by mathematically proving its functional key properties termination and correctness. Second, we implement the approach in SBPM process planning software and analyze its feasibility and applicability by means of several real-world processes. We finally discuss the practical utility of our approach compared with manual process modeling by means of the criteria flexibility by definition (cf. [49]), modeling costs and modeling time.

6.1 Functional Properties of the Algorithm

One major functional property of the algorithm is its termination. We state the following theorem:

THEOREM 1 (TERMINATION). Given our planning problem, the execution of the algorithm terminates.

We prove Theorem 1 by showing that, given any instance of our planning problem, (1) each statement of the algorithm terminates and (2) the number of iterations of every loop is finite (cf. Appendix A).

To show that the algorithm provides correct results (branches and conditions for exclusive choices), we state

Theorem 2 with respect to the generated Extended Search Graph:

THEOREM 2 (CORRECTNESS). Iff an action can be applied in a world state *s* of belief state *bs*, there is a branch in the Extended Search Graph constructed by the algorithm that represents this fact.

We prove this theorem by showing that the following two statements hold (cf. Appendix B):

(1) If an action *a* cannot be applied in a world state *s* of belief state *bs*, there is no branch in the Extended Search Graph constructed by the algorithm representing that action *a* is applied in this world state *s*.

(2) If an action *a* can be applied in a world state *s* of belief state *bs*, there is a branch in the Extended Search Graph constructed by the algorithm that represents this fact.

Consequently, based on a given planning problem, our algorithm is not subject to potential modeling failures, as manual process modeling potentially is.

6.2 Feasibility and Applicability of the Algorithm

To analyze the feasibility and applicability of our algorithm, we examine the following evaluation questions:

E.1 Can the algorithm be instantiated in the form of a software implementation to demonstrate its feasibility ("proof by construction"; cf. [27, 39])?

E.2 Can the algorithm be applied in a practical setting and what inputs are needed (cf. [27])?

E.3 What is the output resulting from the practical application of the algorithm ("analysis of the results")?

To address E.1, the proposed algorithm was implemented as part of our planning approach in SBPM process planning software (a demo version of the web interface of the software can be accessed at *http://www-sempa.uni-regensburg.de/*). Indeed, the exclusive choice pattern is of one of the most important control flow patterns, and its consideration in the planning software constitutes an essential step toward planning real-world processes. To attain a user-friendly interface, we implemented a web application that guides the user during each step of the approach, providing a description of inputs, tasks and outputs. The application allows a step-by-step specification of action libraries (i.e., actions including their preconditions and effects) and the initial state and the goal states of the planning problem. Moreover, the web application supports the reuse of already defined action libraries in different planning problems, for instance, to analyze the changes in a planned process model in case the initial state and/or the goal states are modified. To test the implementation, persons other than the programmers performed an analysis of the source code, and we made a series of tests using the JUnit Framework, including runs with extreme values, JUnit regression tests, and unit tests. The implementation did not show any errors at the end of the test phase.

In a second step, we analyzed whether the planning approach can be applied in real-use situations and what initial effort must be taken (cf. E.2). To do this analysis, we focused on several process (re)designs of a major European bank in which core business processes had manually been (re)designed several times due to new products, new regulations, or changing organizational requirements (e.g., outsourcing parts of processes to external service providers). For illustration purposes, we consider the order execution process (the running example used within this paper is an excerpt of this process), which refers to the security order management of the bank. To apply our algorithm in this context, as a major initial effort, we had to extract all actions included in former process models of the area of security order management to our planning software. This was done in an automated manner via the bank's process modeling tool (ARIS toolset) which provides an XML interface. In doing so, we imported approximately 200 actions including their preconditions and effects. Thereafter, we reviewed all actions, for example with respect to the completeness of their preconditions and effects. Such a review, which is also necessary during manual process (re)design, must be performed more accurately for process planning. Based on new requirements resulting from new products or new regulations, for example, we could easily adapt some extracted actions and add further actions. Moreover, together with the responsible department of the bank and based on the project requirements, we specified and entered the initial state and the intended goal states. Finally, we planned the feasible process models using our planning software in an automated manner.

Concerning E.3, we analyzed the output resulting from the practical application of the algorithm and compared it to the manually built process models. Figure 3 shows such an output in terms of a feasible process model for the order execution process planned by the algorithm. Within this solution, several exclusive choices were constructed. The first one differentiates three paths concerning the parameter *securityType* (data type *string*) with the values *stock*, *fund*, or *certificate*. Thus, the actions *prove stock*, *prove fund*, and *prove certificate* can be applied in the respective path. The following three exclusive choices are necessary to consider different check routines (*prove plausibility*, *prove competency*, and *prove coverage*). If one of these checks fails, the order execution process terminates due to existing risk regulations. The last exclusive choice refers to our running example (cf. Figure 2). Here, the parameter *orderAmount* (data type *double*), which reaches values from *0* to *250,000* Euro, is used. The feasible solution presented in Figure 3 is the one corresponding to the exclusive choices resulting from the manual redesign of this process. The feasible solutions resulting from the application of our algorithm were discussed with the responsible department of the bank to assess their feasibility (e.g.,

considering the resource constraints when conducting the process; cf. also Subsection 6.3) and their respective advantages and disadvantages. Moreover, it was also possible to determine the changes of the planned process models in case the initial state and/or the goal states were modified and/or additional actions were considered. This gave us the chance to analyze the consequences when regulation requirements (e.g., concerning the order size or the counterparty risk) required changes.

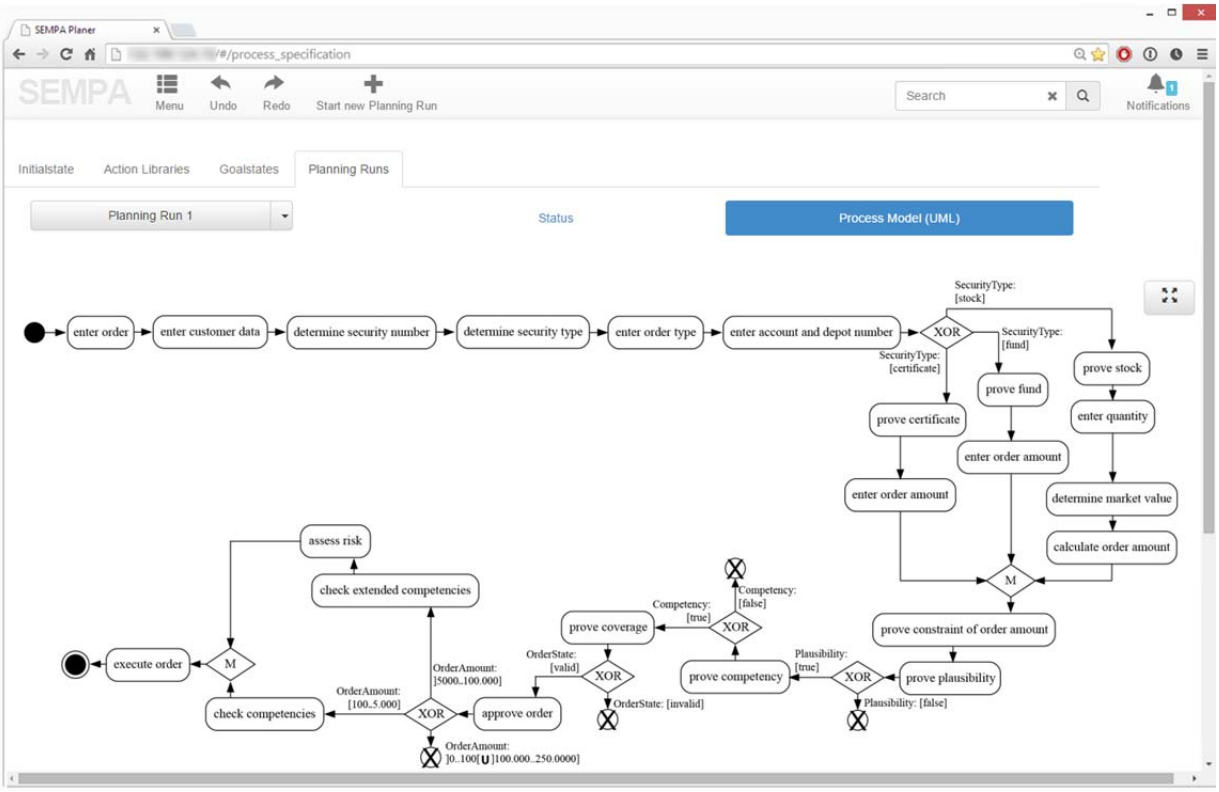

Fig. 3. Example of a Planned Feasible Model for the Order Execution Process

The evaluation questions E.1 to E.3 were also analyzed in eleven additional real-use situations: four from a bank (application contexts “Loan Management” and “Private Banking”), two from an insurance company (application context “Insurance Management”), four from a mechanical engineering company (application context “Project Management”), and one from a university (application context “Human Resources Management”). Table 1 provides an overview of the key figures (the planning software ran on an Intel Core i7-2600 3.40 GHz, Windows 7 64 Bit, Kernel Version 7601.22616, Java 8):

Table 1. Application of our approach in additional real-use situations

Application Context	Process description	Number of actions / states in the initial search graph	Number of constructed exclusive choices (all/only nested)	Number of actions / states included by exclusive choices	Data types incorporated in the conditions	Number of feasible solutions	Modeling time in seconds
Project Management	Preparing and coordinating project profile	17/15	3/2	4	String, Class	2	<0.001

Project Management	Specifying project resources	25/18	1/0	1	String, Class	2	<0.001
Project Management	Allocating project resources	26/22	2/1	8	String, Class	2	<0.001
Project Management	Preparing the project report for the board	38/25	2/1	6	String, Class	2	<0.001
Insurance Management	Administrating customer and product database	43/38	8/7	29	String, Class	5	0.003
Insurance Management	Handling insured events	54/44	13/12	33	String, Class	7	0.002
Loan Management	Analyzing credit rating	40/31	5/4	25	String; Class	2	0.001
Loan Management	Selling mortgage loans	57/43	5/0	14	Double, String, Class	2	0.002
Loan Management	Settling mortgage loans	122/69	3/0	54	Double, String, Class	2	0.001
Private Banking	Contracting wealth manage-ment customer	278/189	33/11	82	Boolean, String, Class	3	0.820
Human Resources Management	Engaging new staff	83/75	16/15	76	String, Class	29	0.004

The process models are of different sizes, containing between 17 and 278 actions in the initial search graph. The largest model (Contracting wealth management customer), for instance, contains actions conducted by external service providers and multiple departments of the bank. The process models include simple and nested exclusive choices. Nested exclusive choices are located on branches of preceding exclusive choices that have not yet been merged. The constructed exclusive choices include a significant number of actions and states following on the respective branches. The conditions of the constructed exclusive choices contain primitive data types, such as Boolean, String, and Double, and list union data types such as Classes. Overall, this evaluation step demonstrates the feasibility and applicability of our approach. Table 2 summarizes the results.

Table 2. Results concerning evaluation questions E.1 to E.3

Evaluation Question	Result
E.1 Can the algorithm be instantiated in the form of a software implementation to demonstrate its feasibility (“proof by construction”; cf. [27, 39])?	The algorithm was successfully implemented and tested as part of our planning approach in a SBPM process planning software.
E.2 Can the algorithm be applied in a practical setting and which input is needed accordingly (cf. [27])?	The algorithm was applied in several real-use situations from different application contexts and companies. The required input could be determined in each case.
E.3 What is the output resulting from the practical application of the algorithm (“analysis of the results”)?	The output of the algorithm does not only comprise the exclusive choices resulting from the manual design of the processes but multiple feasible solutions.

6.3 Practical Utility of the Algorithm

In addition to the applications of our approach (cf. Subsection 6.2), we evaluated its practical utility.

Specifically, we illustrate its advantages concerning the criteria *flexibility by definition* (cf. [49]), *modeling costs*, and *modeling time* compared with a manual (re)design of process models.

Flexibility by Definition: As defined by van der Aalst [49] in his taxonomy of process flexibility, "*flexibility by definition* is the ability to incorporate alternative execution paths within a process definition at design time such that selection of the most appropriate execution path can be made at runtime for each process instance." Applying our algorithm does not only result in the manually designed exclusive choices and one feasible process model: indeed, it also provides multiple feasible solutions (cf. column 7 of Table 1) and consequently increases flexibility by definition. This capability is beneficial because the resulting feasible solutions may be compared to one another based on economic criteria (e.g., process costs per execution and cycle time) and resource constraints (e.g., staff utilization). Hence, automated process planning enables the flexibility to select the most appropriate feasible process model or path for execution.

Modeling Costs: To analyze the costs of automated process planning, Krause et al. [35] developed an economic model to compare the modeling costs that result from the process planning approach (cf. Section 3) with the costs resulting from modeling processes manually. They analyzed the cost-efficiency of the planning approach applied in the context of several process (re)design projects at a major bank. They found that the planning approach results in higher initial setup costs (particularly for analyzing and semantically annotating actions), but the direct modeling costs for (re)designing process models are much lower compared with the comparable costs of manual process modeling. As Krause et al. [35] analyzed, the contribution margin of the considered process (re)design projects was increased by approximately 20% through cost savings in the direct modeling costs that are supposed to cover the higher initial setup costs. Because Krause et al. [35] only considered a short period (approximately two years), the cost-efficiency of the planning approach should even increase in the long term because the cost savings in modeling costs increase over time for frequently (re)designed processes.

Modeling Time: Based on the considered process (re)designs, the direct modeling time of our planning approach is lower compared with that of manual process modeling. For instance, the three feasible solutions of the order execution process including multiple exclusive choices (one solution is illustrated in Figure 3) were created in approximately 0.178 seconds. Indeed, for each process listed in Table 1, all feasible process models were created in less than one second, which is obviously much less time than is required for manual process modeling. Similarly to the initial setup costs discussed in Krause et al. [35], automated process planning requires adding some initial setup time for analyzing and semantically annotating actions. However,

particularly for processes that must be frequently (re)designed, the savings in modeling time outweigh the additional setup time.

Overall, based on the analyzed real-use situations, our approach has advantages compared with manual process modeling concerning the criteria flexibility by definition, modeling costs, and modeling time.

7. DISCUSSION

We have demonstrated the strength of our algorithm; however, some limitations remain that must be discussed in more detail. First, our algorithm is limited to the construction of exclusive choices. To plan entire process models in an automated manner, we must answer the challenging question of whether other control flow patterns such as parallel splits, synchronizations, simple merges, and arbitrary cycles can be constructed. Based on the abstract representation language and the planning domain defined in this paper, we already have begun both developing algorithms for further control flow patterns and integrating them into our approach for an automated planning of process models presented in Section 3.

Second, we did not answer the question of whether an appropriate process model can be selected out of the provided set of feasible solutions in practice (cf. discussion of *flexibility by definition* in Subsection 6.3). This may be done manually, for instance, by a process modeler or in an (semi)automated manner. In the case of manual selection, it seems to be appropriate to determine clusters of similar feasible solutions (e.g., using similarity measures for process models; cf. [10, 12]). Thus, the modeler can evaluate these clusters by their main differences and exclude inadequate clusters (e.g., concerning a specific application context). This procedure may considerably reduce the number of feasible solutions that must be further evaluated. The clustering approach can be applied again on the remaining feasible solutions until the number of process models is manageable. Such a procedure can support a two- or more-step selection process. How to cluster and illustrate different feasible solutions is part of our present and future research. For selecting in a (semi)automated manner, there are process simulation techniques that typically aim to compare different process models based on non-functional criteria (e.g., process costs and cycle time determined in a specific application context) and resource constraints. When the process models are realized by services, the research field of Quality-of-Service (QoS)-aware Service Selection provides a large number of approaches (e.g., [2, 9, 22, 60]), each aiming to determine an optimal QoS-aware service composition in an automated manner based on a given number of process models. Thus, the set of feasible process models could serve as input to this research.

A third limitation refers to workflow management and process mining tool interfaces that our SBPM process planning software currently lacks. To integrate our software into a company's application landscape, such interfaces are required. Their realization represents a future task in our research.

Further directions for future research include extensions of our planning approach in terms of considering static and non-static context information about the environment in which the planned process is executed (context-aware planning; e.g., [24]) and multiple users that must be coordinated when the planned process is executed.

8. CONCLUSION

Both the increasing frequency and the complexity of required (re)designs of companies' processes constitute a significant degree of manual work, making the (re)designs cost-intensive and time-consuming. To alleviate this drawback, we envision the automated planning of process models. In this paper, we present an abstract representation language that enables the representation of possibly infinite sets of world states (belief states). After a formal definition of the planning problem, we propose a novel algorithm for the automated construction of exclusive choices in to-be process models. This is the first algorithm that explicitly addresses the construction of conditions for each branch of an exclusive choice considering multiple belief state variables and large data types. This is essential for the automated construction of exclusive choices in the context of process planning. Moreover, our approach constructs not only one model, as manual process modeling usually does, but multiple feasible process models differing with respect to their exclusive choices. By means of our evaluation, we showed that our algorithm terminates and creates correct solutions. We also applied our algorithm to several real-world processes in different application contexts and companies and found support that the algorithm is fully applicable in practice. Furthermore, it has advantages compared with manual process modeling concerning the criteria flexibility by definition, modeling costs, and modeling time.

We conclude with the generalizability and the breadth of applicability of our approach. In the course of the discussion of related literature, we already identified promising links to other research streams. To build the Search Graph, many approaches coping with nondeterminism and initial state uncertainty can be used [4, 8, 28]. Such approaches are the underlying technique for several domains such as, for instance, (web) service composition [1, 5, 19, 30, 45]. Hence, our approach also seems to be applicable to other domains.

ACKNOWLEDGMENT

The research was funded by the Austrian Science Fund (FWF) [P 23546-G11].

REFERENCES

[1] V. Agarwal, G. Chafle, K. Dasgupta, N. Karnik, A. Kumar, S. Mittal, B. Srivastava, Synthy: A system for end to end composition of web services, Journal of Web Semantics 3 (2005) 311–339.

[2] D. Ardagna, B. Pernici, Adaptive Service Composition in Flexible Processes, IEEE Transactions on Software Engineering 33 (6) (2007) 369–384.

[3] R. Bergenthum, J. Desel, R. Lorenz, S. Mauser, Process mining based on regions of languages, in: Business Process Management, Springer, Berlin Heidelberg, Germany, 2007, pp. 375-383.

[4] P. Bertoli, A. Cimatti, M. Roveri, P. Traverso, Strong planning under partial observability, Artificial Intelligence 170 (2006) 337-384.

[5] P. Bertoli, R. Kazhamiakin, M. Paolucci, M. Pistore, H. Raik, M. Wagner, Control Flow Requirements for Automated Service Composition, *Proceedings of the IEEE International Conference on Web Services*, Los Angeles, USA, 2009, pp. 17-24.

[6] P. Bertoli, M. Pistore, P. Traverso, Automated composition of Web services via planning in asynchronous domains, Artificial Intelligence 174 (2010) 316–361.

[7] B. Bonet, H. Geffner, Planning with Incomplete Information as Heuristic Search in Belief Space. *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems*, Breckenridge, CO, 2000, pp. 52-61.

[8] B. Bonet, H. Geffner, GPT: A Tool for Planning with Uncertainty and Partial Information, *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, Seattle, WA, 2001, pp. 82-87.

[9] G. Canfora, M. Di Penta, R. Esposito, M. L Villani, A Framework for QoS-aware Binding and Re-binding of Composite Web Services, Journal of Systems and Software 81 (10) (2008) 1754–1769.

[10] L. Chen, M. Reichert, A. Wombacher, On Measuring Process Model Similarity Based on High-Level Change Operations, Proceedings of the 27th International Conference on Conceptual Modeling, Barcelona, 2008, pp. 248–264.

[11] I. Constantinescu, B. Faltings, W. Binder, Large scale, type-compatible service composition. *Proceedings of the IEEE International Conference on Web Services*, 2004, pp. 506-513.

[12] R. Dijkman, M. Dumas, B. van Dongen, R. Käärik, J. Mendling, Similarity of business process models: Metrics and evaluation, Information Systems 36 (2) (2011) 498-516.

[13] R. Dijkman, B. Gfeller, J. Küster, H. Völzer, Identifying refactoring opportunities in process model repositories, Information and Software Technology 53 (9) (2011) 937-948.

[14] W. Gaaloul, K. Baïna, C. Godart, Towards mining structural workflow patterns, Database and Expert Systems Applications, Springer, Berlin Heidelberg, Germany, 2005.

[15] H. Geffner, Perspectives on Artificial Intelligence Planning, *Proceedings of the 18th National Conference on Artificial Intelligence*, Edmonton, Canada, 2002, pp. 1013-1023.

[16] M. Ghallab, D. Nau, P. Traverso, Automated Planning - Theory & Practice, Morgan Kaufmann, San Francisco, CA, 2004.

[17] G. Greco, A. Guzzo, L. Ponieri, D. Sacca, Discovering expressive process models by clustering log traces, IEEE Transactions on Knowledge and Data Engineering 18 (8) (2006) 1010-1027.

[18] S. Gregor, A. R. Hevner, Positioning and presenting design science research for maximum impact, MIS Quarterly 37 (2) (2013) 337-356.

[19] O. Hatzi, D. Vrakas, N. Bassiliades, D. Anagnostopoulos, I. Vlahavas, The PORSCE II Framework: Using AI Planning for Automated Semantic Web Service Composition, The Knowledge Engineering Review, Cambridge University Press, 2010.

[20] B. Heinrich, M.-A. Bewernik, M. Henneberger, A. Krammer, F. Lautenbacher, SEMPA - A Semantic Business Process Management Approach for the Planning of Process Models, Business & Information Systems Engineering (formerly Wirtschaftsinformatik) 50 (6) (2008) 445-460 (in German).

[21] B. Heinrich, M. Bolsinger, M.-A. Bewernik, Automated Planning of Process Models: The Construction of Exclusive Choices, *Proceedings of the 30th International Conference on Information Systems*, Phoenix,

AZ, 2009, Paper 184.

[22] B. Heinrich, M. Klier, L. Lewerenz, S. Zimmermann, Quality of Service-aware Service Selection: A Novel Approach Considering Potential Service Failures and Non-Deterministic Service Values. INFORMS Service Science, 7 (1) (2015) 1-22.

[23] B. Heinrich, M. Klier, S. Zimmermann, Automated Planning of Process Models - Towards a Semantic-based Approach, in: S. Smolnik, F. Teuteberg, O. Thomas (Eds.), Semantic Technologies for Business and Information Systems Engineering: Concepts and Applications, IGI Global, Hershey, PA, 2011, pp. 169-194.

[24] B. Heinrich, D. Schön, Automated Planning of context-aware Process Models. *Proceedings of the European Conference on Information Systems*, Münster, Germany, 2015.

[25] M. Henneberger, B. Heinrich, B. Bauer, F. Lautenbacher, Semantic-Based Planning of Process Models. *Proceedings of the Multikonferenz Wirtschaftsinformatik*, Munich, Germany, 2008, pp. 1677-1689.

[26] M. Hepp, F. Leymann, J. Domingue, A. Wahler, D. Fensel, Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management. *Proceedings of the IEEE International Conference on e-Business Engineering*, Beijing, China, 2005, pp. 535-540.

[27] A. R. Hevner, S. T. March, J. Park, S. Ram, Design Science in Information Systems Research. MIS Quarterly 28 (1) (2004) 75-105.

[28] J. Hoffmann, R. I. Brafman, Contingent Planning via Heuristic Forward Search with Implicit Belief States, *Proceedings of the 15th International Conference on Automated Planning and Scheduling*, Monterey, CA, 2005, pp. 71-80.

[29] J. Hoffmann, R. I. Brafman, Conformant planning via heuristic forward search: A new approach. Artificial Intelligence 170 (2006) 507-541.

[30] J. Hoffmann, P. Bertoli, M. Helmert, M. Pistore, Message-Based Web Service Composition, Integrity Constraints, and Planning under Uncertainty: A New Connection, Journal of Artificial Intelligence Research 35 (2009) 49-117.

[31] J. Hoffmann, I. Weber, F. M. Kraft, SAP Speaks PDDL: Exploiting a Software-Engineering Model for Planning in Business Process Management, Journal of Artificial Intelligence Research 44 (2012) 587-632.

[32] T. Hornung, A. Koschmider, A. Oberweis, Rule-based Autocompletion of Business Process Models, *Proceedings of the 19th Conference on Advanced Information Systems Engineering*, Trondheim, Norway, 2007.

[33] E. Kindler, Model-based Software Engineering and Process-aware Information Systems, in: K. Jensen (Ed.), Transactions on Petri Nets and Other Models of Concurrency II, Springer, Berlin, Germany, 2009, pp. 27-45.

[34] G. Koliadis, A. Ghose, Verifying semantic business process models in inter-operation, *Proceedings of the IEEE International Conference on Services Computing*, Salt Lake City, UT, 2007.

[35] F. Krause, M.-A. Bewernik, G. Fridgen, Valuation of Manual and Automated Process Redesign from a Business Perspective. Business Process Management Journal 19 (1) (2013) 95-110.

[36] Q. A. Lang, S. Y. W. Su, AND/OR Graph and Search Algorithm for Discovering Composite Web Services, International Journal of Web Services Research 2 (4) (2005) 46-64.

[37] F. Lecue, A. Delteil, A. Leger, O. Boissier, Web Service Composition as a Composition of Valid and Robust Semantic Links, International Journal of Cooperative Information Systems 17 (10) (2008) 1-63.

[38] H. Meyer, M. Weske, Automated Service Composition using Heuristic Search, *Proceedings of the 4th International Conference on Business Process Management*, Vienna, Austria, 2006, pp. 81-96.

[39] J. F. Nunamaker, M. Chen, T. D. M. Purdin, Systems Development in Information Systems Research, Journal of Management Information Systems 7 (3) (1991) 89-106.

[40] J. Pathak, S. Basu, V. Honavar, Modeling Web Service Composition using Symbolic Transition Systems, *Papers from the AAAI Workshop on AI-Driven Technologies for Service-Oriented Computing 2006*, Boston, MA, 2006.

[41] R. P. A. Petrick, F. Bacchus, A Knowledge-Based Approach to Planning with Incomplete Information and Sensing, *Proceedings of the 6th International Conference on AI Planning Systems*, Toulouse, France, 2002, pp. 212-221.

[42] M. Pistore, P. Traverso, P. Bertoli, A. Marconi, Automated Synthesis of Composite BPEL4WS Web Services, *Proceedings of the 3rd International Conference on Web Services*, 2005, pp. 293-301.

[43] N. Russell, A. H. M. Ter Hofstede, W. M. P. Van der Aalst, N. Mulyar, Workflow Control-Flow Patterns: A Revised View, BPM Center Report BPM-06-22, http://bpmcenter.org/reports 2006, (June 25, 2013).

[44] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, X. Xu, Web services composition: A decade's overview, Information Sciences (280) (2014) 218-238.

[45] A. Sirbu, A. Marconi, M. Pistore, H. Eberle, F. Leymann, T. Unger, Dynamic Composition of Pervasive Process Fragments, *Proceedings of the IEEE International Conference on Web Services*, Washington, USA, 2011, pp. 73-80.

[46] E. Sirin, B. Parsia, D. Wu, J. Hendler, D. Nau, HTN Planning for Web Service Composition Using SHOP2, Journal of Web Semantics 1(4) (2004) 377-396.

[47] S. X. Sun, J. L. Zhao, J. F. Nunamaker, O. R. L. Sheng, Formulating the Data-Flow Perspective for Business Process Management, Information Systems Research 17 (2006) 374-391.

[48] P. Traverso, M. Pistore, Automated composition of semantic web services into executable processes, in: S. A. McIlraith, D. Plexousakis, F. van Harmelen (Eds.), The Semantic Web-ISWC 2004, Springer, Berlin/Heidelberg, Germany, 2004, pp. 380-394.

[49] W. M. P. van der Aalst, Business Process Management: A Comprehensive Survey, ISRN Software Engineering (2013), Article ID 507984.

[50] W. M. van der Aalst, T. Weijters, L. Maruster, Workflow Mining: Discovering Process Models from Event Logs, IEEE Transactions on Knowledge and Data Engineering 16 (9) (2004) 1128-1142.

[51] W. M. van der Aalst, V. Rubin, H. M. W. Verbeek, B. F. Van Dongen, E. Kindler, C. W. Günther, Process mining: a two-step approach to balance between underfitting and overfitting, Software & Systems Modeling 9(1) (2010) 87-111.

[52] W. M. P. van der Aalst, AK Alves de Medeiros, A. J. M. M. Weijters, Genetic process mining, Applications and Theory of Petri Nets, Springer, Berlin Heidelberg, 2005, pp. 48-69.

[53] W. M. P. van der Aalst et al. Process Mining Manifesto, Business process management workshops, Springer, Berlin Heidelberg, 2012.

[54] P. Wang, Z. Ding, C. Jiang, M. Zhou, Automated web service composition supporting conditional branch structures, Enterprise Information Systems 8 (1) (2014) 121-146.

[55] I. Weber, J. Hoffmann, J. Mendling, Beyond Soundness: On the Verification of Semantic Business Process Models, Distributed and Parallel Databases 27(3) (2010) 271-343.

[56] A. J. M. M. Weijters, W. M. van der Aalst, A. A. De Medeiros, Process mining with the heuristics miner-algorithm, Technische Universiteit Eindhoven, Tech. Rep. WP, 2006.

[57] L. Wen, W. M. P. van der Aalst, J. Wang, J. Sun, Mining process models with non-free-choice constructs, Data Mining and Knowledge Discovery 15(2) (2007) 145-180.

[58] B. Wetzstein, Z. Ma, A. Flilipowska, M. Kacmarek, S. Bhiri, S. Losada, J.-M. Lopez-Cobo, L. Cicurel, Semantic Business Process Management: A Lifecycle Based Requirements Analysis, *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management at the 3rd European Semantic Web Conference*, Innsbruck, Austria, 2007, pp. 1-11.

[59] D. Wu, B. Parsia, E. Sirin, J. Hendler, D. Nau, Automating DAML-S web services composition using SHOP2, in: D. Fensel, K. Sycara, J. Mylopoulos (Eds.), The Semantic Web - ISWC 2003, Springer, Berlin/Heidelberg, Germany, 2003, pp. 195-210.

[60] T. Yu, Y. Zhang, K. Lin, Efficient Algorithms for Web Services Selection with end-to-end QoS Constraints, ACM Transaction on the Web 1 (1) (2007) Article 6.

Supplement for the Manuscript "Automated Planning of Process Models: Design of a Novel Approach to Construct Exclusive Choices"

In this supplement, we provide the proofs for Theorem 1 (cf. Appendix A) and Theorem 2 (cf. Appendix B).

A. PROOF OF THEOREM 1

THEOREM 1 (TERMINATION). Given our planning problem, the execution of the proposed algorithm terminates.

PROOF. Termination is proved by showing that each statement of the algorithm terminates and that the number of iterations of every loop is finite. This is done in the following:

In the first statement of the EXTENDGRAPH procedure (line 2 in Procedure 1), the CONDITIONS procedure is invoked for the belief state *bs* and the Search Graph *SG*. To prove that the CONDITIONS procedure terminates, it is sufficient to show that the for-loop (lines 4 to 9 in Procedure 2) terminates. This is due to the fact that the remaining statements (lines 2, 3, and 10) are simple (set) operations which terminate obviously. Our line of argument that the for-loop terminates is as follows:

The statements within the for-loop are executed for each element of the set bs_{pre} which is a subset of *BST* ($bs_{pre} \subseteq bs \subseteq BST$). Due to the fact that *BST* is a finite set of belief state tuples (cf. definition of the planning domain in Definition 16), the subset bs_{pre} is finite as well ($bs_{pre} \subseteq BST \Rightarrow |bs_{pre}| \leq |BST| < \infty$). Thus, the number of iterations of the for-loop is finite. Further it has to be proven that each iteration of the for-loop terminates: In the first statement within the loop, a set *R* of restrictions is built (line 5). Thereby, the number of elements within the set *R* is finite (i.e. $|R| < \infty$) due to two reasons: (1) the set $A_{p\_a}(bs)$ of actions which are partly applicable in *bs* is finite ($A_{p\_a}(bs) \subseteq A \Rightarrow |A_{p\_a}(bs)| \leq |A| < \infty$; cf. definition of the planning domain); (2) the set of preconditions $precond(a_{p\_a})$ is finite for each partly applicable action a_{p_a} in *bs* ($precond(a_{p\_a}) \subseteq BST \Rightarrow |precond(a_{p\_a})| \leq |BST| < \infty$; cf. definition of the planning domain). Afterwards, the PARTITION procedure is invoked with the restriction of the belief state r_{bs} and the finite set of restrictions *R*. For a finite set of restrictions *R,* the PARTITION procedure itself terminates and returns a finite set of restrictions (cf. Lemmata 1 and 2 which are presented und proven subsequent to this proof). The remaining statements within the for-loop of the CONDITIONS procedure are simple (set) operations which obviously terminate and build a finite set of conditions *C*. Thus, each iteration of the for-loop terminates, and so does the CONDITIONS procedure.

In the next step, the nested loops within the EXTENDGRAPH procedure are executed (lines 3 to 10 in Procedure 1). To prove their termination, our line of argument is as follows:

The statements within the outer loop (lines 3 to 10) are executed for all actions $a \in A_{p\_a}(bs)$ which are partly applicable in *bs*. As already shown above, the set $A_{p\_a}(bs)$ and thus the number of iterations of the outer loop are finite. Within the outer loop (line 4), the CONDITIONSFUNCTION procedure (Procedure 4) is invoked for the belief state *bs*, the previously built finite set of conditions *C*, and action *a*. The CONDITIONSFUNCTION procedure creates the set C_{arc} by means of simple (set) operations which terminate, respectively. Thus, the CONDITIONSFUNCTION procedure terminates as well. The returned set of sets of conditions C_{arc} is finite due to the facts that C_a is finite ($C_a \subseteq C \Rightarrow |C_a| \leq |C| < \infty$) and C_{arc} is a subset of the power set of C_a ($C_{arc} \subseteq 2^{C_a} \Rightarrow |C_{arc}| \leq |2^{C_a}| = 2^{|C_a|} \leq 2^{|C|} < \infty$). Afterwards, the statements within the inner loop of the EXTENDGRAPH procedure (lines 5 to 9) are executed for each element c_{arc} of the set C_{arc}. As C_{arc} is finite (see above) and the statements within the inner loop are simple (set) operations which terminate, respectively, the inner loop terminates as well, and so does the outer loop. Finally, the arc ⟨*bs*, else, termination⟩ is added to the set of arcs *Arcs* of the Search Graph and the algorithm terminates. □

In the following, we provide and prove the two lemmata referenced in the proof of Theorem 1.

LEMMA 1. Given a finite set of restrictions *R*, the execution of the PARTITION procedure terminates.

PROOF. To prove the termination of the PARTITION procedure (Procedure 3) for a finite set of restrictions *R* (i.e. $|R| < \infty$), it is sufficient to show that the recursions terminate (lines 8 and 11). This is due to the fact that besides the recursions the PARTITION procedure only contains simple (set) operations (e.g. intersection, union, and difference) that obviously terminate. As a consequence, we prove the lemma by showing that the number of recursive invocations of the PARTITION procedure is finite. The following facts serve as a basis for our line of argument:

- The PARTITION procedure is only invoked recursively if *R* contains more than one element (i.e. $|R| > 1$; line 6). In this case, the PARTITION procedure is invoked two times at the most (lines 8 and 11). If *R* contains only one element (i.e. $|R| = 1$), the PARTITION procedure is not invoked recursively but returns the set *solution* (lines 13 to 16).

- When invoking the PARTITION procedure recursively, the corresponding set of restrictions R is reduced by the element *rest* (lines 8 and 11). That means that each time the number of elements in R is decreased by one.

Consequently, the number of recursive invocations of the PARTITION procedure is finite, because (1) the number of elements in R is finite, (2) the number of elements in R is decreased by one when recursively invoking the PARTITION procedure, and (3) there is a lower bound of one for the number of elements in R (i.e. $|R| = 1$) which serves as the stop criterion for the recursion. □

LEMMA 2. Given a set r_{bs} and a finite set of sets R, the set *solution* := PARTITION(r_{bs}, R) which is returned by the PARTITION procedure is finite as well.

PROOF. We prove this lemma by induction on the number of elements in R (i.e. $|R|$). The line of argument is as follows:

(1) Base step ($|R| = 1$): It can easily be seen from the CONDITIONS procedure that the PARTITION procedure is only invoked for $|R| \geq 1$ (for all (v_{bs}, r_{bs}) in bs_{pre} there exists at least one element in R due to the definitions of bs_{pre} and R in lines 3 and 5 of the CONDITIONS procedure). As a consequence, the base step is conducted for $|R| = 1$. If R contains only one element (i.e. $|R| = 1$), the PARTITION procedure is not invoked recursively, but returns the set *solution* containing the two elements *diff* and *inter* unless these are empty (lines 13 to 16). Hence, $|solution| = |\{diff, inter\} \setminus \{\varnothing\}| \leq |\{diff, inter\}| = 2 < \infty$ and thus the lemma hold for $|R| = 1$.

(2) Induction step ($|R| = n \rightarrow |R| = n+1$ with $n \geq 1$): Assuming that the set *solution* which is returned by the PARTITION procedure is finite for sets of restrictions containing n elements (i.e. $|R| = n$), it has to be shown that the set *solution* is finite for sets of restrictions containing $n+1$ elements (i.e. $|R| = n+1$) as well. We prove that as follows: When executing the PARTITION procedure for a set R with $|R| = n+1$, one element is nondeterministically taken as *rest* to determine *diff* and *inter* (lines 2 to 4). Afterwards, as $|R| = n+1 > 1$ holds, the PARTITION procedure is invoked recursively for the sets *diff* and *inter* unless these are empty whereby the corresponding set of restrictions R is reduced by the element *rest* (lines 7 to 12). Finally, the PARTITION procedure returns the set *solution* which constitutes the union of the recursive invocations of the PARTITION procedure for *diff* and *inter* (line 16). Thereby, $|solution| =$

$|\text{PARTITION}(diff, R \setminus rest) \cup \text{PARTITION}(inter, R \setminus rest)| \leq |\text{PARTITION}(diff, R \setminus rest)| +$ $|\text{PARTITION}(inter, R \setminus rest)| < \infty$ holds (the last inequality holds due to the fact that the set $R \setminus rest$ contains $|R \setminus rest| = (n+1) - 1 = n$ elements in combination with the induction hypothesis that the theorem holds for sets of restrictions containing n elements).

By means of 1) and 2) the lemma has been proven by induction. □

B. PROOF OF THEOREM 2

THEOREM 2 (CORRECTNESS). Iff an action can be applied in a world state *s* of belief state *bs*, there is a branch in the Extended Search Graph constructed by the algorithm that represents this fact.

PROOF. We prove this theorem by proving the following two statements:

(1) If an action *a* cannot be applied in a world state *s* of belief state *bs*, there is no branch in the Extended Search Graph constructed by the algorithm representing the fact that action *a* is applicable in this world state *s*.

(2) If an action *a* can be applied in a world state *s* of belief state *bs*, there is a branch in the Extended Search Graph constructed by the algorithm that represents this fact.

Ad (1): Given an action *a* that cannot be applied in a world state *s* of belief state *bs* (i.e. $s \sqsubseteq bs$) we distinguish two cases which are treated differently by the algorithm: either action *a* is *not partly applicable* in the belief state *bs* or action *a* is *partly applicable* in belief state *bs*, respectively.

Case 1: Action *a* is not partly applicable in belief state *bs* (i.e. $a \notin A_{p\_a}$).

In this case, the algorithm does not create any arcs $\langle bs, \{precond(a), c_{arc}\}, a\rangle$ with regard to belief state *bs* and action *a* (cf. the criterion of the loop in lines 3 to 10 of the EXTENDGRAPH procedure). Hence, the algorithm does neither create any branch that would represent that action *a* could be applied in world state *s* of belief state *bs* and statement (1) has been proven.

Case 2: Action *a* is partly applicable in belief state *bs* (i.e. $a \in A_{p\_a}$).

In this case, due to the fact that action *a* cannot be applied in at least one world state *s* of belief state *bs* ($s \sqsubseteq bs$) the following proposition holds: $\exists u \in precond(a)\ \exists w \in s: v(w) = v(u) \wedge r(w) \notin r(u)$.

Without loss of generality, u^* and w^* represent one combination of u and w which is characterized by this property.

In the following, we provide a proof by contradiction that statement (1) holds in *case 2*. In this respect, we assume that the algorithm constructs a branch in the Extended Search Graph representing that action a can be applied in world state s of belief state bs ($s \sqsubset bs$), while, actually, action a cannot be applied in this world state s.

As a consequence, an arc $\langle bs, \{precond(a), c_{arc}\}, a\rangle$ (line 8 of the EXTENDGRAPH procedure) with $bs' = \gamma(bs, c_{arc}, a)$ (line 6 of the EXTENDGRAPH procedure) is created by the algorithm that represents that action a can be applied in world state s of belief state bs. In this respect, we have to distinguish two cases:

- *Case 2.1*: The arc $\langle bs, \{precond(a), c_{arc}\}, a\rangle$ represents that action a can be applied in all world states of the belief state bs (i.e. $c_{arc} = \emptyset$)

  In the Extended Search Graph, $c_{arc} = \emptyset$ represents that action a can actually be applied in all possible world states of belief state bs (Definition 10). Hence, action a can particularly be applied in world state $s \sqsubset bs$. In case the algorithm returns $c_{arc} = \emptyset$ for a belief state bs and an action a, it follows that $\emptyset = c_{arc} \in C_{arc} = \text{CONDITIONFUNCTION}(bs, C, a)$ (lines 4 and 5 of the EXTENDGRAPH procedure). As the result of the CONDITIONFUNCTION procedure only contains the empty set if $applicable(a, bs)$ holds true (lines 2 to 4 of the CONDITIONFUNCTION procedure), the action a has to be applicable in the belief state bs. However, this cannot be the case because action a cannot be applied in world state s of belief state bs ($s \sqsubset bs$). ↯

- *Case 2.2*: All conditions in $c_{arc} \neq \emptyset$ are fulfilled in world state s (i.e. $c_{arc} \neq \emptyset \wedge \forall c \in c_{arc}\ \exists w \in s$: $v(c) = v(w) \wedge r(w) \in r(c)$)

  In this case, $c_{arc} \neq \emptyset$ holds but all conditions $c \in c_{arc}$ are fulfilled in world state s. Thereby, with respect to w^* (cf. above) two different cases are possible. Either no condition exists in c_{arc} which refers to the condition variable $v(w^*)$ (i.e. $\nexists c \in c_{arc}$: $v(c) = v(w^*)$), or the corresponding condition is fulfilled (i.e. $\exists c \in c_{arc}$: $v(c) = v(w^*) \wedge r(w^*) \in r(c)$).

  - *Case 2.2.1*: $\nexists c \in c_{arc}$: $v(c) = v(w^*)$

    According to the algorithm $c_{arc,} \in C_{arc} = \text{CONDITIONFUNCTION}(bs, C, a)$ (lines 4 and 5 of the

EXTENDGRAPH procedure) holds. In this context, C_{arc} includes all subsets c_{arc} of C_a containing every condition variable which is part of the conditions in C_a exactly once (line 6 of the CONDITIONFUNCTION procedure). As a consequence, c_{arc} with $\forall c \in c_{arc}$: $v(c) \neq v(w^*)$ implies that $\nexists c_a \in C_a$: $v(c_a) = v(w^*)$. This leads to the fact that $\nexists c_c \in C$: $v(c_c) = v(w^*)$, $r(c_c) \subseteq r_a$, $(v(c_c), r_a) \in precond(a)$ (line 5 of the CONDITIONFUNCTION procedure). Hence, set C which results from the CONDITIONS procedure does not contain any element c_c which is characterized as follows: $v(c_c) = v(w^*)$, $r(c_c) \subseteq r_a$, and $(v(c_c), r_a) \in precond(a)$. This fact has to be analyzed in detail. In this context, we distinguish the two cases depending on whether the for-loop of the CONDITIONS procedure (lines 4 to 9 of the CONDITIONS procedure) *is executed* for (v_{bs}, r_{bs}) with $v_{bs} = v(w^*)$ (*case a)*) or *not* (*case b)*):

Case a): The for-loop of the CONDITIONS procedure (lines 4 to 9) is executed for (v_{bs}, r_{bs}) with $v_{bs} = v(w^*)$. In this context, the set *Partition* contains the result of the PARTITION procedure for r_{bs} and R whereas $r_a \in R$ because of $a \in A_{p\_a}$ (lines 5 and 6 of the CONDITIONS procedure). Thereby, due to Lemma 3 (the lemma is presented und proven subsequent to this proof) $r_{bs} \cap r_a$ is completely covered by the elements of the set *Partition* which are subsets of r_a (i.e. $\bigcup_{\{s \in Partition \mid s \subseteq r_a\}} s = r_{bs} \cap r_a$). Hence, due to $r_{bs} \cap r_a \neq \varnothing$ (this is because of $a \in A_{p\_a}$) at least one element $r_p \in Partition$ (line 6 of the Conditions procedure) exists with $r_p \subseteq r_a$. Hence, it follows that C_{part} and C contain at least one element c_c fulfilling $v(c_c) = v(w^*)$, $r(c_c) \subseteq r_a$ and $(v(c_c), r_a) \in precond(a)$ as well. ↯

Case b): The for-loop of the CONDITIONS procedure (lines 4 to 9) is not executed for (v_{bs}, r_{bs}) with $v_{bs} = v(w^*)$. Hence, the set bs_{pre} does not contain any element (v_{bs}, r_{bs}) with $v_{bs} = v(w^*)$ (lines 2 and 3 of the CONDITIONS procedure). That means $\nexists (v(w^*), r_{bs}) \in bs$: $(v(w^*), r_{pre}) \in precond(a_{p\_a})$, $r_{bs} \not\subset r_{pre}$, $a_{p\_a} \in A_{p\_a}(bs)$. As a consequence and due to $a \in A_{p\_a}(bs)$ (definition of *case 2*), it particularly follows that $\nexists (v(w^*), r_{bs}) \in bs$: $(v(w^*), r_{pre}) \in precond(a)$, $r_{bs} \not\subset r_{pre}$. However, this cannot be the case because of the definition of $w^* \in s \sqsubseteq bs$ and the fact that $\exists u \in precond(a)$: $v(u) = v(w^*) \wedge r(w^*) \notin r(u)$. ↯

- *Case 2.2.2*: $\exists c \in c_{arc}$: $v(c) = v(w^*) \wedge r(w^*) \in r(c)$

  In the following, without loss of generality one tuple c which is characterized by this property is represented by c^*. Due to the fact that the algorithm leads to c_{arc} with $c^* \in c_{arc}$ for belief State bs and

action a, it follows that $c^* \in c_{arc} \in C_{arc} = \text{CONDITIONFUNCTION}(bs, C, a)$ (lines 4 and 5 of the EXTENDGRAPH procedure). The result of the CONDITIONFUNCTION procedure can only contain subsets of the set C_a (line 6 of the CONDITIONFUNCTION procedure). Hence, it follows that $c_{arc} \subseteq C_a$ and so does $c^* \in C_a$. According to the definition of the set C_a (line 5 of the CONDITIONFUNCTION procedure) $c^* \in C_a := \{(v_c, r_c) \in C \mid r_c \subseteq r_a, (v_c, r_a) \in precond(a)\}$ holds. However, due to the facts that $precond(a)$ contains each belief state variable one time at the most (Definition 8) and that $v(c^*) = v(w^*) = v(u^*)$ holds, $r(c^*) \subseteq r(u^*)$ has to be fulfilled. Hence, by means of $r(w^*) \in r(c)$ it follows that $r(w^*) \in r(u^*)$. However, this cannot be the case due to the definition of u^* and w^*. ↯

Consequently, the statement (1) has been shown.

Ad (2): Given an action a that can actually be applied in a world state s of belief state bs ($s \sqsubseteq bs$), it has to be proven that there is a branch in the Extended Search Graph constructed by the algorithm that represents this fact. To do so, we provide a proof by contradiction. In this respect, we assume that the algorithm does not construct any branch in the Extended Search Graph that represents that action a can be applied in world state s of belief state bs ($s \sqsubseteq bs$), although action a can actually be applied in this world state s.

In this context, we have to distinguish two cases. In the first case, the Extended Search Graph does not contain any arc $\langle bs, \{precond(a), c_{arc}\}, a\rangle$. In the second case, for each arc $\langle bs, \{precond(a), c_{arc}\}, a\rangle$ of the Extended Search Graph holds that c_{arc} contains at least one condition which is not fulfilled in world state s:

Case 1: The Extended Search Graph constructed by the algorithm does not contain any arc $\langle bs, \{precond(a), c_{arc}\}, a\rangle$ with regard to belief state bs and action a. In this context, we distinguish two further cases depending on whether the outer loop of the EXTENDGRAPH procedure (lines 3 to 10) *is executed* for action a (*case 1.1*) or *not* (*case 1.2*):

- *Case 1.1*: The outer loop of the EXTENDGRAPH procedure *is executed* for action a.

  Due to the fact that the resulting Extended Search Graph does not contain any arc $\langle bs, \{precond(a), c_{arc}\}, a\rangle$ it follows that the inner loop of the EXTENDGRAPH procedure (lines 5 to 9) is not executed for action a. As a consequence, $\varnothing = C_{arc} = \text{CONDITIONFUNCTION}(bs, C, a)$ (lines 4 and 5) holds. But this cannot be the case because according to the CONDITIONFUNCTION(bs, C, a) procedure it is obvious that

CONDITIONFUNCTION(bs, C, a) $\neq \varnothing$ (the set of sets contains the empty set or at least one subset of C_a, respectively). ↯

- *Case 1.2*: The outer loop of the EXTENDGRAPH procedure is *not* executed for action a. In this case, it follows that $a \notin A_{p\_a}$ (cf. the criterion of the for-loop). Hence, action a is not partly applicable in belief state bs. However, this cannot be the case because action a can be applied in world state s of belief state bs ($s \sqsubseteq bs$). ↯

Case 2: For each arc $\langle bs, \{precond(a), c_{arc}\}, a\rangle$ of the Extended Search Graph holds that c_{arc} contains at least one condition which is not fulfilled in world state s. Therefore, for *each* arc $\langle bs, \{precond(a), c_{arc}\}, a\rangle$ either case a) a condition exists in c_{arc} for which no corresponding belief state variable exists in the world state s (i.e. $\exists c \in c_{arc}\ \forall w \in s$: $v(c) \neq v(w)$ or case b) a condition exists in c_{arc} which is not fulfilled by the corresponding belief state tuple of the world state s (i.e. $\exists c \in c_{arc}\ \exists w \in s$: $v(c) = v(w) \wedge r(w) \notin r(c)$).

In the following, we first focus on case a) $\exists c \in c_{arc}\ \forall w \in s$: $v(c) \neq v(w)$: For all sets of conditions c_{arc} constructed by the algorithm $c_{arc} \in C_{arc}$ = CONDITIONFUNCTION(bs, C, a) holds (lines 4 and 5 of the EXTENDGRAPH procedure). Thereby, the result of the CONDITIONFUNCTION procedure only contains subsets of the set C_a (line 6 of the CONDITIONFUNCTION procedure) which leads to $c_{arc} \subseteq C_a$. Due to the definition of C_a in line 5 of the CONDITIONFUNCTION procedure together with $c_{arc} \subseteq C_a$ it follows that $\forall c \in c_{arc}$ $\exists u \in precond(a)$: $v(c) = v(u)$. Consequently, case a) $\exists c \in c_{arc}\ \forall w \in s$: $v(c) \neq v(w)$ would imply $\exists u \in precond(a)\ \forall w \in s$: $v(u) \neq v(w)$ and that cannot be the case because action a can actually be applied in the world state s (i.e. $\forall u \in precond(a)\ \exists w \in s$: $v(u) = v(w) \wedge r(w) \in r(u)$). Therefore, for each arc $\langle bs, \{precond(a), c_{arc}\}, a\rangle$ of the Extended Search Graph case b) $\exists c \in c_{arc}\ \exists w \in s$: $v(c) = v(w) \wedge r(w) \notin r(c)$ holds.

Within the EXTENDGRAPH procedure, arcs $\langle bs, \{precond(a), c_{arc}\}, a\rangle$ are constructed for *all* $c_{arc} \in C_{arc}$ (lines 5 to 9). Thereby, C_{arc} = CONDITIONFUNCTION(bs, C, a) includes *all* subsets of C_a containing every condition variable which is part of the conditions in C_a exactly once (line 6 of the CONDITIONFUNCTION procedure). Hence, together with the fact that b) $\exists c \in c_{arc}\ \exists w \in s$: $v(c) = v(w) \wedge r(w) \notin r(c)$ for *each* arc $\langle bs, \{precond(a), c_{arc}\}, a\rangle$ of the Extended Search Graph (i.e. for *each* $c_{arc} \in C_{arc}$) it follows that $\exists w \in s$ $\forall (v(w), r(c)) \in C_a$: $r(w) \notin r(c)$ – without loss of generality -.one tuple w which is characterized by this property

is given by w^*. In the following, we analyze the implications of this fact: On the one hand, the set C_a contains at least one condition which refers to the variable $v(w^*)$. Hence, due to the definition of C_a (line 5 of the CONDITIONFUNCTION procedure) it follows that the set C contains at least one element $(v(w^*), r(c))$ which fulfills $r(c) \subseteq r(u)$ with $(v(w^*), r(u)) \in precond(a)$. On the other hand, there is no element in the set C which additionally fulfills $r(w^*) \in r(c)$ (this is because of $\forall (v(w^*), r(c)) \in C_a$: $r(w^*) \notin r(c)$). The set C is derived by the CONDITIONS procedure (line 2 of the EXTENDGRAPH procedure) and all elements in terms of conditions are created based on the results of the PARTITION procedure (line 6 to 8 of the CONDITIONS procedure; the fact that $\exists (v(w^*), r(c)) \in C_a$ shows that the loop in lines 4 to 9 is executed for $(v(w^*), r_{bs}) \in bs$). Thus by means of Lemma 4 ($r(u) \in R$ holds due to $(v(w^*), r(u)) \in precond(a)$, and $a \in A_{p\_a}(bs)$) it follows that $r(w^*) \notin \bigcup_{\{(v(w^*),\, r(c)) \in C \mid r(c) \subseteq r(u)\}} r(c) = r_{bs} \cap r(u)$ with $(v(w^*), r_{bs}) \in bs$ and $(v(w^*), r(u)) \in precond(a)$. But this cannot be the case as action a can be applied in the world state s of belief state bs (i.e. $r(w^*) \in (r_{bs} \cap r(u))$ with $(v(w^*), r_{bs}) \in bs$ and $(v(w^*), r(u)) \in precond(a)$. ↯

Consequently, statement (2) has been shown and Theorem 2 has been proven. □

In the following, we provide and prove the lemmata referenced in the proof of Theorem 2.

LEMMA 3. Given a set r_{bs} and a finite set of sets R with $r_a \in R$, the set $solution$:= PARTITION(r_{bs}, R) which is returned by the PARTITION procedure fulfills $\bigcup_{\{s \in solution \mid s \subseteq r_a\}} s = r_{bs} \cap r_a$ (i.e. $r_{bs} \cap r_a$ is completely covered by the partitions which are subsets of r_a).

PROOF. We prove this lemma by induction on the number of elements in R (i.e. $|R|$). The line of argument is as follows:

1) Base step ($|R| = 1$): Due to the reasons provided in the proof of Lemma 2, the base step is conducted for $|R| = 1$. If r_a is the only element in R (i.e. $|R| = 1$), the PARTITION procedure is not invoked recursively ($|R| = 1 \not> 1$) but returns the set $solution := \{diff, inter\} \setminus \{\varnothing\}$ (lines 13 to 16) with $diff := r_{bs} \setminus r_a$ and $inter := r_{bs} \cap r_a$ (lines 3 and 4). Hence, $\bigcup_{\{s \in solution \mid s \subseteq r_a\}} s = inter = (r_{bs} \cap r_a)$ holds and so does the lemma for $|R| = 1$.

2) Induction step ($|R| = n \rightarrow |R| = n+1$ with $n \geq 1$): Assuming that the lemma holds for sets of sets containing n elements (i.e. $|R| = n$), it has to be shown that it holds for sets of sets containing $n+1$ elements (i.e. $|R| = n+1$) as well. We prove that as follows: When executing the PARTITION procedure for a set R with $|R| = n+1$, one element is nondeterministically taken as *rest* (line 2). In this context, we have to distinguish two cases (*case a)*: $rest = r_a$; *case b)*: $rest \neq r_a$):

- *Case a)*: $rest = r_a$

  In this case the sets $diff := r_{bs} \setminus r_a$ and $inter := r_{bs} \cap r_a$ are built (lines 3 and 4). Afterwards, as $|R| = n+1 > 1$ holds, the PARTITION procedure is invoked recursively for the sets $r_{bs} \setminus r_a$ and $r_{bs} \cap r_a$ unless these are empty (the case that *diff* or *inter*, respectively, are empty is trivial) whereby the corresponding set of sets R is reduced by the element r_a (lines 7 to 12). Thus, by means of Lemma 4 (the lemma is presented and proven subsequent to this proof) it follows that $\bigcup_{s \in PARTITION(r_{bs} \setminus r_a, R \setminus r_a)} s = r_{bs} \setminus r_a$ and $\bigcup_{s \in PARTITION\ (r_{bs} \cap r_a, R \setminus r_a)} s = r_{bs} \cap r_a$. Finally, the PARTITION procedure returns the set *solution* which constitutes the union of the recursive invocations of the PARTITION procedure for $r_{bs} \setminus r_a$ and $r_{bs} \cap r_a$ (line 16). For the set *solution* it holds that

  $$\bigcup_{\{s \in solution \mid s \subseteq r_a\}} s = \bigcup_{\substack{\{s \in (PARTITION\ (r_{bs} \setminus r_a, R \setminus r_a) \cup \\ PARTITION\ (r_{bs} \cap r_a, R \setminus r_a)) \mid s \subseteq r_a\}}} s = \bigcup_{\{s \in PARTITION\ (r_{bs} \setminus r_a, R \setminus r_a) \mid s \subseteq r_a\}} s \cup$$
  $$\bigcup_{\{s \in PARTITION\ (r_{bs} \cap r_a, R \setminus r_a) \mid s \subseteq r_a\}} s = \varnothing \cup (r_{bs} \cap r_a) = (r_{bs} \cap r_a).$$

- *Case b)*: $rest \neq r_a$

  In this case, the sets $diff := r_{bs} \setminus rest$ and $inter := r_{bs} \cap rest$ are built (lines 3 and 4). Afterwards, as $|R| = n+1 > 1$ holds, the PARTITION procedure is invoked recursively for the sets $r_{bs} \setminus rest$ and $r_{bs} \cap rest$ unless these are empty (the case that *diff* or *inter*, respectively, are empty is trivial) whereby the corresponding set of sets R is reduced by the element *rest* (lines 7 to 12). Finally, the PARTITION procedure returns the set *solution* which constitutes the union of the recursive invocations of the PARTITION procedure for $r_{bs} \setminus rest$ and $r_{bs} \cap rest$ (line 16). Thereby the set solution fulfills

  $$\bigcup_{\{s \in solution \mid s \subseteq r_a\}} s = \bigcup_{\substack{\{s \in (PARTITION\ (r_{bs} \setminus rest, R \setminus rest) \cup \\ PARTITION\ (r_{bs} \cap rest, R \setminus rest)) \mid s \subseteq r_a\}}} s = \bigcup_{\{s \in PARTITION\ (r_{bs} \setminus rest, R \setminus rest) \mid s \subseteq r_a\}} s \cup$$
  $$\bigcup_{\{s \in PARTITION\ (r_{bs} \cap rest, R \setminus rest) \mid s \subseteq r_a\}} s = ((r_{bs} \setminus rest) \cap r_a) \cup ((r_{bs} \cap rest) \cap r_a) = r_{bs} \cap r_a$$

(the third equality holds due to the fact that the set ($R \setminus rest$) with $r_a \in (R \setminus rest)$ contains $|R \setminus rest| = (n+1) - 1 = n$ elements in combination with the induction hypothesis that the lemma holds for sets containing n elements).

By means of 1) and 2) the lemma has been proven by induction. □

LEMMA 4. Given a set r_{bs} and a finite set of sets R, the set *solution* := PARTITION(r_{bs}, R) which is returned by the PARTITION procedure fulfills $\bigcup_{s \in solution} s = r_{bs}$ (i.e. r_{bs} is completely covered by the partitions).

PROOF. We prove this lemma by induction on the number of elements in R (i.e. $|R|$). The line of argument is as follows:

1) Base step ($|R| = 1$): Due to the reasons provided in the proof of Lemma 2, the base step is conducted for $|R| = 1$. If R contains one element (i.e. $|R| = 1$), the PARTITION procedure is not invoked recursively ($|R| = 1 \not> 1$) but returns the set *solution*:= {*diff*, *inter*} \ {∅} (lines 13 to 16) with *diff* := $r_{bs} \setminus rest$ and *inter* := $r_{bs} \cap rest$ (lines 3 and 4). Hence,

   $\bigcup_{s \in solution} s = diff \cup inter = (r_{bs} \setminus rest) \cup (r_{bs} \cap rest) = (r_{bs} \cap rest^c) \cup (r_{bs} \cap rest) = r_{bs} \cap (rest^c \cup rest) = r_{bs}$ holds

   and so does the lemma for $|R| = 1$.

2) Induction step ($|R| = n \rightarrow |R| = n+1$ with $n \geq 1$): Assuming that the lemma holds for sets of restrictions containing n elements (i.e. $|R| = n$), it has to be shown that it holds for sets of restrictions containing $n+1$ elements (i.e. $|R| = n+1$) as well. We prove this as follows: When executing the PARTITION procedure for a set R with $|R| = n+1$, one element is nondeterministically taken as *rest* to determine *diff* := $r_{bs} \setminus rest$ and *inter* := $r_{bs} \cap rest$ (lines 2 to 4). Afterwards, as $|R| = n+1 > 1$ holds, the PARTITION procedure is invoked recursively for the sets *diff* and *inter* unless these are empty (the case that *diff* or *inter*, respectively, are empty is trivial) whereby the corresponding set of restrictions R is reduced by the element *rest* (lines 7 to 12). Finally, the PARTITION procedure returns the set *solution* which constitutes the union of the recursive invocations of the PARTITION procedure for *diff* and *inter* (line 16). Due to $|R \setminus rest| = n$ in combination with the induction hypothesis that the lemma holds for sets of restrictions containing n elements it follows

$$\bigcup_{s \in solution} s = \bigcup_{\substack{s \in (PARTITION\ (r_{bs} \setminus rest,\ R \setminus rest) \cup \\ PARTITION\ (r_{bs} \cap rest,\ R \setminus rest))}} s = \bigcup_{s \in PARTITION\ (r_{bs} \setminus rest,\ R \setminus rest)} s \cup \bigcup_{s \in PARTITION\ (r_{bs} \cap rest,\ R \setminus rest)} s =$$

$$= (r_{bs} \setminus rest) \cup (r_{bs} \cap rest) = r_{bs}\ .$$

By means of 1) and 2) the lemma has been proven by induction. □