Karlsruhe Institute of Technology

Verification of Data-Value-Aware Process Models

—

Zur Erlangung des akademischen Grades einer
Doktorin der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

angenommene Dissertation

von

M.Sc. Elaheh Ordoni

Referent:	Prof. Dr.-Ing. Klemens Böhm
Korreferent:	Prof. Dr. Andreas Oberweis

Tag der mündlichen Prüfung: 19.01.2024

Acknowledgement

I would like to take this opportunity to express my deep appreciation and gratitude to the individuals who have played a significant role in the completion of my PhD dissertation.

First and foremost, I am immensely grateful to my family for their unwavering support and encouragement, despite the physical distance between us. Their belief in my abilities and their constant motivation have been instrumental in helping me navigate through the challenges of this journey. A special thank you goes to my husband, Dr. Aso Rahimzadegan, whose unwavering support, both academically and emotionally, has been my rock throughout this arduous process.

I am indebted to my supervisor, Prof. Klemens Böhm, for his invaluable guidance, expertise, and unwavering commitment to my academic growth. His constructive feedback and relentless pursuit of excellence have shaped my research and pushed me to bring out my best. I am truly fortunate to have had the privilege to work under his mentorship.

I would also like to express my gratitude to Prof. Andreas Oberweis, who served as the second reviewer of my dissertation. His valuable insights, suggestions, and fruitful discussions have significantly contributed to the refinement and improvement of my work.

My heartfelt thanks also go to Jutta Mülle for her continuous support, availability, and willingness to engage in insightful discussions. Her encouragement and belief in my research have been truly motivating.

I am grateful to my dear friends, Abel Elekes, Holger Trittenbach, and Pawel Bielski, for their unwavering support, meaningful discussions, and friendship. Their presence throughout this journey has made the challenges more manageable and the successes more meaningful.

To all those mentioned above and the countless others who have contributed to my academic and personal growth, I offer my heartfelt thanks. Your support and encouragement have been invaluable, and I am deeply grateful for the role each of you has played in the successful completion of this dissertation.

Abstract

Verification methods detect unexpected behaviour of business process models before their execution. To verify many applications, *data values* play an essential role. A data value is a value in the domain of a data object, e. g., \$1000 as the price of a product. Data values may influence the execution behaviour of the process models in two ways: *data-value conditions* and *data-value functions*. A data-value condition compares a value of a data object, e. g., *price* $> 100,000$. A data-value function modifies the values of data objects during process execution, e.g., increasing the price of a product.

A process model in which the elements are associated with data-value conditions or data-value functions is a so-called *data-value-aware process model*. To verify a data-value-aware process model, one must include the execution semantics of data-value conditions and functions in the state space of the process model for correct verification. In addition, the resulting state space must remain small enough for a model checker to explore. This is challenging because the size of state space may increase exponentially with the number of elements associated with data-value conditions or functions, resulting in state-space explosion. Furthermore, data-value conditions and functions in which data objects have large domains may also lead to the same problem.

Our first step in this thesis is to provide a generic approach to verify data-value-aware process models, that is, to include the semantics of data-value conditions and functions in the state space of the process model. To achieve this, we enhance process models with detailed information on which elements use data values. Then, we propose a novel algorithm to transform the data-value-aware process models to Petri nets, taking the semantics of data-value conditions and functions into account. This way of generating a Petri-Net-based representation is generic, i. e., does not take any application-specific characteristic into account. In the second step, we approach the issue of state-space explosion in data-value-aware process models from two perspectives: First, we reduce the size of the process models in which many elements are associated with data-value functions or conditions. We propose novel relevance functions to detect the necessary elements of a process model to verify a given property, so-called *relevant elements*. Once the relevant elements have been detected, we provide reduction algorithms that prune the irrelevant elements accordingly. Second, we reduce the state space caused by data-value conditions and functions in which data objects with large domains are involved. To achieve this, we propose a novel binary encoding technique, that is able to map Data Object d with the domain of $[0..n-1]$ to at most $2 \cdot (\lfloor \log_2 n \rfloor + 1)$ new places in the Petri net. With our binary encoding approach, we achieve a logarithmic reduction of the state space caused by data-value conditions while supporting the entire domain of

data objects. The binary encoding technique also reduces the state space caused by data-value functions. It allows us to employ *Binary Decision Diagrams* (BDD) to map the semantics of data-value functions in a Petri net. Thus, we can use the existing BDD reduction techniques to reduce the number of edges and nodes in BDDs, causing the reduction of places and transitions in the Petri net. As a result, we can map data-value functions into much smaller Petri nets and state space.

The running example of the thesis and the use case for the evaluation is a real-world application: Simultaneous Multi-Round (SMR) spectrum auction. In particular, we verify the German 4G spectrum auction to sell one of the most valuable bandwidths, the 800 MHz band. We derive the values of three essential measures in the worst case, for the first time: *auctioneer's revenue*, i. e., the sum of the final prices of the products, *bidder's profit*, i. e., the sum of the unused budgets of bidders who have won a product, and *auction's efficiency*, i. e., the share of products sold to bidders with the highest budgets.

Zusammenfassung

Verifizierungsmethoden erkennen unerwartetes Verhalten von Geschäftsprozessmodellen vor ihrer Ausführung [1]. Für die Verifikation vieler Anwendungen spielen *data values* eine wesentliche Rolle. Ein Datenwert ist ein Wert in der Domäne eines Datenobjekts, z.B. $1000 als Preis für ein Produkt. Datenwerte können das Ausführungsverhalten der Prozessmodelle auf zwei Arten beeinflussen: *data-value conditions* und *data-value functions*. Eine Datenwert-Bedingung vergleicht einen Wert eines Datenobjekts, z. B. $Preis > 100.000$. Eine Datenwertfunktion ändert die Werte von Datenobjekten während der Prozessausführung, z. B. Erhöhung des Preises eines Produkts.

Ein Prozessmodell, in dem die Elemente mit data-value conditions oder data-value functions verknüpft sind, wird als *data-value-aware process model* bezeichnet. Die Verifikation von *data-value-aware process model* ist eine Herausforderung. Um solche Modelle zu verifizieren, muss man einen allgemeinen Ansatz zur Behandlung aller Arten von Datenbedingungen und -funktionen bereitstellen. Darüber hinaus sollte die Größe des Zustandsraums in einem Bereich gehalten werden, den ein Modellprüfprogramm noch erkunden kann. Die Größe des Zustandsraums von datenwertbasierten Prozessmodellen steigt jedoch exponentiell mit der Anzahl der Elemente, die mit data-value conditions oder -funktionen assoziiert sind, was zu einer Explosion des Zustandsraums führt [2]. Nicht nur die Elemente, die mit Datenwerten verbunden sind, sondern auch data-value conditions und -funktionen, bei denen Datenobjekte große Domänen haben, können zur Zustandsraumexplosion führen.

In dieser Arbeit schlagen wir einen generischen Ansatz zur Verifikation von datenwertbasierten Prozessmodellen vor, der alle Arten von data-value conditions und data-value functions berücksichtigt. Anschließend gehen wir das Problem der Zustandsraumexplosion in solchen Prozessmodellen aus zwei Perspektiven an: Erstens reduzieren wir die Größe von Prozessmodellen, in denen viele Elemente mit data-value functions oder Bedingungen verbunden sind. Wir schlagen neuartige Relevanzfunktionen vor, um die notwendigen Elemente eines Prozessmodells zu erkennen, um eine bestimmte Eigenschaft zu verifizieren, sogenannte *relevante Elemente*. Sobald die relevanten Elemente erkannt wurden, stellen wir Reduktionsalgorithmen bereit, die die irrelevanten Elemente entsprechend beschneiden. Zweitens reduzieren wir den Zustandsraum eines datenbasierten Prozessmodells, der durch data-value conditions und Funktionen mit Datenobjekten mit großen Domänen entsteht. Um dies zu erreichen, schlagen wir eine neuartige binäre Kodierungstechnik vor, die in der Lage ist, das Datenobjekt d mit der Domäne $[0..n-1]$ auf höchstens $2 \cdot (\lfloor \log_2 n \rfloor + 1)$ neue Stellen im Petrinetz abzubilden. Mit unserem binären Kodierungsansatz erreichen wir eine logarithmische Reduktion des Zustandsraums, der durch data-value conditions verursacht wird, und unterstützen gleichzeitig den gesamten Bereich der

Datenobjekte. Die binäre Kodierungstechnik reduziert nicht nur die Größe des Zustandsraums, der durch Bedingungen, sondern auch durch data-value functions verursacht wird. Sie erlaubt es uns, *Binäre Entscheidungsdiagramme* [3] (BDD), um die Semantik von Daten-Wert-Funktionen in einem Petrinetz abzubilden. So können wir die bestehenden BDD-Reduktionstechniken [4] nutzen, um die Anzahl der Kanten und Knoten in BDDs, d.h. die Anzahl der Stellen und Übergänge in unserem Petrinetz, zu reduzieren. Infolgedessen können wir Daten-Wert-Funktionen in viel kleineren Petri-Netzen und somit in einem kleineren Zustandsraum abbilden.

Die Beiträge in unserer Dissertation erlauben es uns, zum ersten Mal eine reale Anwendung zu verifizieren: die deutsche 4G-Spektrum-Auktion, um eine der wertvollsten Bandbreiten, das 800-MHz-Band, zu verkaufen. Wir leiten die Werte von drei wesentlichen Maßen im schlechtesten Fall ab citebrunner2010experimental: *auctioneer's revenue*, d.h. die Summe der Endpreise der Produkte, *bidder's profit*, d.h. die Summe der ungenutzten Budgets der Bieter, die ein Produkt gewonnen haben, und *auction's efficiency*, d.h. der Anteil der Produkte, die an Bieter mit den höchsten Budgets verkauft wurden.

Contents

Part I

Introduction

1. Motivation

In business process models, the absence of errors is one of the most critical quality indicators. It annoys us when our smartphones stop working or when software behaves in unexpected ways. We may not consider these errors so crucial in our lives. Nevertheless, there are some examples that have a substantial financial impact for manufacturers. Denver's airport was closed for nine months because of a software error in a baggage handling system resulting in a loss of 1.1 million dollars per day. As another example, the wrong policy of increasing bids resulted in the loss of a 30 MHz bandwidth during the US spectrum auction. Due to the flaws in the policies, the consequences cost around 70 billion dollars [5]. Detecting errors at an earlier stage does not only save money but can also save lives: Six cancer patients died from an overdose of radiation due to a faulty software system in the Therac-25 radiation therapy machine.

Verification techniques are vital to improve the reliability of business process models, i.e., to ensure that a given process model fulfills all the necessary properties. Such techniques explore all the possible states in which a process model can be in a brute-force manner. In the same way that a computer can check possible moves in a chess game, a verification technique like model checking examines all possible scenarios for a process model. By intensively exploring all possible states, a verification technique can reveal errors that might remain undetected with other techniques, e.g., simulation and testing.

In many applications, verification techniques hing on *data values* associated with process model elements. A data value is a value in the domain of a data object, e. g., \$1000 as the price of a product. To highlight the role of data values in verification, think of spectrum auctions. Data values, such as the price of products and the bids, determine the outcome of the auction and other properties of the process, e. g., the revenue of the auctioneer or whether one bidder can obtain all available goods (spectra in this case). Data values may influence the execution behaviour of the process models in two ways: *data-value conditions* and *data-value functions*. A data-value condition compares a value of a data object, e. g., $price > 100,000$. A data-value function modifies the values of data objects during process execution, e.g., increasing the price of a product. A process model in which the elements are

associated with data-value conditions or data-value functions is a so-called *data-value-aware process model*. To verify such process models, one must include the execution semantics of data-value conditions and functions in the state space of the process model for correct verification.

Example 1. *Figure 1.1 shows parts of an auction process model in BPMN notation. Here, the gateway with data-value condition "bid > 5" determines which branch the process will follow. The data value used in this condition, the value of "bid", may be modified by a preceding activity, in this case "place new bid". This modification may affect the decision of the gateway and thus, affect the execution behaviour of the process model.*

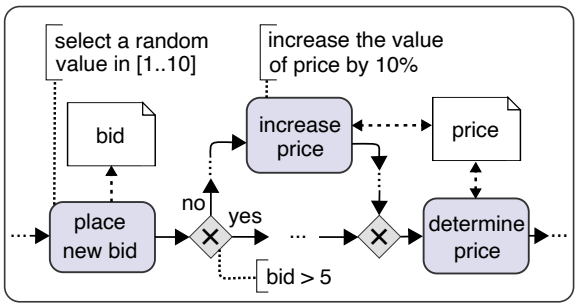


Figure 1.1: Parts of an auction process modeled in BPMN

1.1 Challenges

Having the semantics of data-value conditions and functions in the state space of the process models is challenging. To do so, one requires a generic approach capable of handling different types of data conditions and functions. Furthermore, the resulting state space must remain small enough for a model checker to explore. In the following, we list three challenges to verify data-value-aware process models.

(C1) Generic Approach. It is possible to manually include the semantics of a data-value condition or function in the state space of a process model as presented in [6]. However, the manual support for different types of conditions and functions is not a practical solution to verify data-value-aware process models. Process elements, such as activities and events, may modify the value of data objects in different ways. For example, Activity *increase price* in Figure 1.1 increases the price of a product by %10, another activity may increase the price differently, or even decreases the price. The same applies for data-value conditions comparing the value of different data objects with different constants. So the challenge is how to provide a generic verification technique that works with different types of data-value conditions and data-value function.

(C2) Large Process Models. Verification of data-value-aware process models in which many elements are associated to data-value conditions or data-value modifications often cause state-space explosion [2], i.e., the size of state space may increase exponentially with the number of elements associated with data-value conditions or functions. To illustrate, think of an auction with six products, each one associated with a price and three bids with a value ranging from 1 to 100. A single gateway

which takes decision based on the value of *bids* amounts to 100^{6^3} (bids) new states. Modifications of data values increase the state space even more, by yielding parallel branches in the process model, reflecting the values of a data object that may result. Thus, a naive verification scheme, i. e., one that explores the entire state space, is infeasible in large process models, i.e., processes in which many elements are associated with data-value conditions or functions.

(C3) Large Data Domains. Not only large process models but also large domains of data objects can cause state-space explosion. Suppose that the domain of Data Object *price* ranges from 1 to 500,000. The state space of a process model with a data condition on *price* is 500,000 times the one without. So the challenge is how to prevent the state explosion in the presence of data-value conditions and function with large domains, while supporting the entire domain of each data object, i.e., without abstracting from individual values.

1.2 Contributions

In this thesis, we address the question of how to verify data-value-aware process models by automatically including the semantics of data-value conditions and functions in the state space of the process models. Then, we approach the issue of state-space explosion in such processes from two perspectives: First, we reduce the data-value-aware process models to the relevant parts. Second, we provide a novel technique to reduce the state space of data-value-aware process models with large data domains. The three significant contributions presented in this thesis allow the verification of real-world applications such as spectrum auctions for the first time.

Generic data-value-aware process verification technique. To start, we aim to provide a generic approach to verify data-value-aware process models, that is, to include the semantics of data-value conditions and functions in the state space of the process model. To do so, we formally define data-value-aware process models. Then, we propose a novel algorithm to transform the data-value-aware process models to Petri nets, a formal modeling language, for which a comprehensive set of analysis tools is available [7]. This way of generating a Petri-net-based representation is generic, i. e., does not take any application-specific characteristic into account. Based on the generated Petri nets, resulting from our data-value-aware transformation, one can specify data-value centered properties of the process model as CTL formulas [8]. By deploying an off-the-shelf model checker [2], one is now able to verify data-value-aware process models.

Reduction of data-value-aware processes based on relevance. Next, we deal with state-space explosion problem in large data-value-aware process models. To do so, we propose a property-specific approach to reduce the size of data-value-aware process models, and thus, the state space. To reduce a process model, we identify the control-flow nodes and data elements that influence the verification of a given Property ϕ, so-called *relevant elements*. To identify such elements, we propose novel relevance functions that output whether a control-flow node or data element is necessary to verify ϕ. We prove the correctness of our relevance functions, i. e., that elements labeled as irrelevant by our functions do not influence the verification of ϕ. Once the relevant elements have been detected, we provide reduction algorithms that prune the irrelevant elements accordingly. Our reduction technique also targets process

models with many parallel branches resulting from the modification of data values during the process.

Binary encoding approach to deal with large data domains. Finally, we deal with the state-space explosion problem in data-aware process models caused by large data domains. To do so, we propose a *binary encoding* techniques, completely mapping Data Object d with the domain of $[0..n-1]$ to at most $2 \cdot (\lfloor \log_2 n \rfloor + 1)$ new places in the Petri net. This is in contrast to approaches which represent each value with a separate place or token. The reduction of places by binary encoding technique alone does not yet reduce the state space in data-aware process models. Thus, we take further steps to tackle the state-space explosion problem as follows.

Reduction of the state-space caused by data-value conditions. We propose a novel algorithm based on the theory of Boolean functions [9] to reduce the number of transitions needed to represent the data conditions. This algorithm maps Data Condition *con* in a Petri net using at most $\lfloor \log_2 n \rfloor + 1$ new transitions. This means that one can evaluate *con* in the state space by exploring $\lfloor \log_2 n \rfloor + 1$ instead of n states. A distinctive feature of the state-space-reduction approach presented here is to support the entire domain of each data object, i.e., without abstracting from individual values. Our binary encoding technique reduces the state space of the process model with Data Condition *con* from $n \cdot |s|$ states to $\log_2 n \cdot |s|$, where $|s|$ is the size of the state space without data conditions.

Reduction of state-space caused by data-value functions. The binary encoding allows us to employ the state-of-the-art data structure *Binary Decision Diagrams* [3] (BDD) to map the semantics of data-value functions in a Petri net. With the data-value functions in the form of BDD in a Petri net, it is possible to use the existing BDD reduction techniques [4]. Such techniques reduce the number of edges and nodes in BDDs, which means the number of places and transitions in our Petri net. As a result of this reduction, we can map data-value functions into much smaller Petri nets.

1.3 Thesis outline

The thesis consists of three parts. In the first part, we explain the fundamentals of model checking and verification of process models (Chapter 2). We explain the basics of spectrum auctions, in particular Simultaneous Multi-Round Spectrum (SMR) auction in Chapter 3. In Chapter 4 we review related works. The second part is the core of this thesis and describes our three contributions. In Chapter 5 we propose a generic approach to verify data-value-aware process models. In Chapter 6 we provide a novel technique to reduce data-value-aware process models based on relevance. In Chapter 7 and 8, we propose our binary encoding technique to reduce the state space of process models caused by data-value conditions and data-value functions, respectively. Chapter 9 demonstrates the usefulness of our contribution by verifying a real-world application, the German 4G spectrum auction. In the last part of this thesis, we conclude with a summary and discuss open questions for future research (Chapter 10).

2. Fundamentals

This chapter introduces the formal representations of process models and notations as well as preliminaries for model checking.

2.1 Business Process Languages

The standard Business Process Model and Notation (BPMN) gives the execution semantics and graphical elements to model business processes. It provides a common language so that users can easily read and understand the flow of activities.

Definition 1. *(Business Process Model and Notation)[10]. The elements of Business Process Model and Notation belongs to the following categories:*

- *Activities. They represent the tasks to be performed within a business process. A single activity is a task and a composition of activities is a sub process.*
- *Events. A process can catch or trigger events while it is running. There are different events based on specific conditions. For example, the start message event triggers on receive of a message.*
- *Gateways. They are used to split or merge the sequence of flows. Exclusive gateway activates only one of the outgoing sequence flows and parallel gateway activates all outgoing branches simultaneously.*
- *Swimlanes. They separate different organization units in a process model. Pools represent the whole unit process and lanes show its hierarchical sub divisions.*
- *Data Objects. The most important element to show the data in the BPMN model is data object, which represents the flow of information through the process. A data object can be read or written by an activity defining data need and data result, respectively. The collection of data needs and data results for a specific activity describe its input set and output set, respectively. A data object as an external input of the whole process is called data input, and the data created by the process is called the data output.*

- *Artifacts. They help to understand the semantics of processes easier by adding the complementary information. A text annotation associates with elements to provide additional information. A group describes the set of elements which are logically related.*

Figure 2.1 represents an overview on the BPMN 2.0 elements. BPMN 2.0 is the last published version of this standard. We will use BPMN instead of BPMN 2.0 in the following, if it is unambiguous.

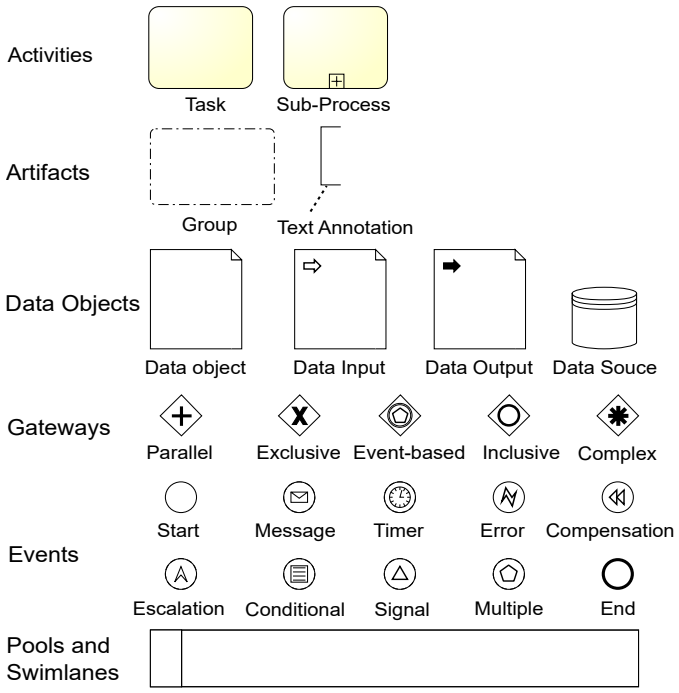


Figure 2.1: Overview of BPMN

The domain of a process model consists of process artifacts, e. g., activities and events. Definition 2 is the basis for both process models and properties referring to data values that we use in this thesis.

Definition 2. *(Process Domain). A Process Domain* $\mathcal{D}$ *is a tuple* $\mathcal{D} = (\mathbb{T}, \mathbb{E}, \mathbb{O}, \mathbb{D}, \mathcal{O}, \mathcal{A}, dom, att, type)$ *where:*

- $\mathbb{T}$ *is a set of activity types,*
- $\mathbb{E}$ *is a set of event types,*
- $\mathbb{O}$ *is a set of data-object types,*
- $\mathbb{D}$ *is a finite set of discrete data domains,*
- $\mathcal{O}$ *is a universe of data objects,*
- $\mathcal{A}$ *is a set of attributes, and*
- $dom : \mathcal{A} \rightarrow \mathbb{D}$ *is a function assigning a data domain to each attribute.*

- $att : \mathbb{O} \rightarrow 2^{\mathcal{A}}$ *is a function assigning a set of attributes to each data-object type.*
- $type : \mathcal{O} \rightarrow \mathbb{O}$ *is a function assigning a data-object type to each data object.*

We further define:

- *Each data object has a key that is confined to its type. A key is a set of attributes that identifies a data object in the set of object instances of its type.*
- *A data value is a value in the domain of an attribute.*
- *Data Element is a general name to refer to a data object or a data-object type.*

In the following, we refer to a data object as *type.key* in order to directly relate the data object to its type and key.

Example 2. *Data Object "bidder.1" refers to Data-object type "bidder" with key* $= 1$.

Definition 3. *(Process Graph). A Process Graph is a tuple* $P = (T, E, AG, XG, N, F, O, I, atype, etype)$ *over* $\mathcal{D} = (\mathbb{T}, \mathbb{E}, \mathbb{O}, \mathbb{D}, \mathcal{O}, \mathcal{A}, dom, att, type)$ *where:*

- T *is a finite set of activities,*
- E *is a finite set of events,*
- AG *is a finite set of AND gateways,*
- XG *is a finite set of XOR gateways,*
- $N = T \cup E \cup AG \cup XG$ *is a finite set of nodes,*
- $F \subseteq N \times N$ *is a finite set of edges,*
- $O \subseteq \mathcal{O}$ *is a finite set of data objects, and*
- $I \subseteq N \times O \cup O \times N$ *is a set of data associations from (to) a node to (from) a data object.*
- $atype : T \rightarrow \mathbb{T}$ *is a function assigning an activity type to an activity in* P.
- $etype : E \rightarrow \mathbb{E}$ *is a function assigning an event type to an event in* P.

Further, we use the following notations:

- *Input(n) is the set of data objects having a data association to node* $n \in N$,
- *Output(n) is the set of data objects having a data association from node* $n \in T \cup E$.

Example 3. *Activity Node* n *reads Data Object* d. *The corresponding process graph has a data association from* d *to* n, *and* $d \in Input(n)$.

In the following, we give the definition of Petri nets, as are one of the best known techniques for specifying business processes in a formal and abstract way, so they serve as a basis for process languages. "Formal" means that the semantics of process instances resulting from process models specified in Petri nets is well defined and not ambiguous. Petri nets are "abstract", because they disregard the execution environment of a business process, so that all aspects other than the functional and process perspectives are not covered. Petri nets can be used to model dynamic systems with a static structure. The static structure is represented by a Petri net, and the dynamic behaviour is captured by the token play of the Petri net.

Definition 4. *(Petri Nets [7]). A Petri Net is a triple* (P, T, F) *where:*

- *a finite set* P *of places,*
- *a finite set* T *of transitions such that* $T \cap P = \varnothing$*, and*
- *a flow relation* $F \subseteq (P \times T) \cup (T \times P)$*.*
- *A place* $p \in P$ *is an input place of a transition* $t \in T$ *if and only if there exists a directed arc from* p *to* t*, that is, if and only if* $(p, t) \in F$*. The set of input places for a transition* t *is denoted* $\bullet t$*.*
- *A place* p *is an output place of a transition* t *if and only if there exists a directed arc from* t *to* p*, that is, if and only if* $(t, p) \in F$*. The set of output places for a transition* t *to* p *is denoted* $t\bullet$*.*
- $p\bullet$ *and* $\bullet p$ *denote the sets of transitions that share* p *as input places and output places, respectively.*

Petri nets consist of places, transitions, and directed arcs connecting places and transitions. They are bipartite graphs, so that arcs never connect two places or two transitions. In graphical notations, places are represented by circles, transitions by rectangles, and connectors by directed arcs. Transitions have input and output places. The input places of a transition are the places at the sources of its incoming arcs. Accordingly, a transition's output places are located at the end of its outgoing arcs.

Example 4. *Figure 2.2 shows the graphical representation of a Petri* (P, T, F) *with:*

- $P = \{p1, p2, p3, p4, p5, p6, p7\}$,
- $T = \{t1, t2, t3, t4, t5, t6\}$,
- $F = \{(p1, t1), (t1, p2), (t1, p3), (p3, t2), (p3, t3), (t2, p4), (t3, p5), (p4, t4), (p5, t5), (t4, p6), (t5, p6), (p6, t6), (t6, p7)\}$.

The state of the Petri net is characterized by the distribution of tokens on the places of the net. The dynamic behaviour of a system is represented by state changes, which are subject to firing rules.

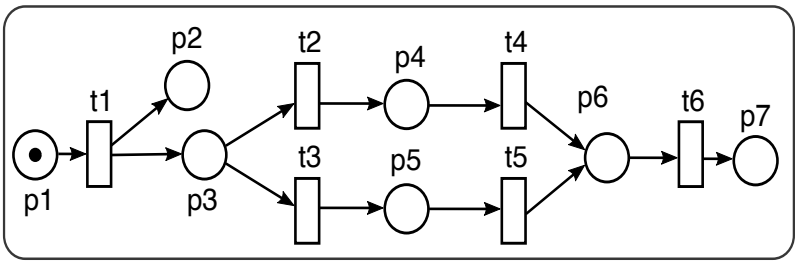


Figure 2.2: The graphical representation of a Petri Net (P, T, F)

Definition 5. *(Marking). The marking (or state) of a Petri Net (P, T, F) is defined by a function $M : P \to \mathbb{N}$ mapping the set of places into the natural numbers, where $\mathbb{N}$ is the et of natural numbers including 0.*

Definition 6. *Let (P, T, F) be a Petri net and M a marking. The firing of a transition is represented by a state change of the Petri net.*

- *$M \overset{t}{\mapsto} M'$ indicates that by firing t, the state of the Petri net changes from M to M'.*
- *$M \to M'$ indicates that there is a transition t such that $M \overset{t}{\mapsto} M'$.*
- *$M_1 \overset{*}{\mapsto} M_n$ indicates that there is a sequence of transitions $t_1, t_2, ..., t_{n-1}$ such that $M_i \overset{t_i}{\mapsto} M_{i+1}$, for $1 \leqslant i < n$.*
- *A state M' is reachable from a state M if and only if $M \overset{*}{\mapsto} M'$.*

Example 5. *The state of the Petri net shown in Figure 2.2 is represented by $M(p1) = 1$ and $M(p2) = M(p3) = M(p4) = M(p5) = M(p6) = M(p7) = 1$. Here, once the t_1 transition fires, a token is removed from $p1$ and tokens are put on both p_2 and $p3$, representing the execution of the corresponding activity in the process model. The execution time of this activity instance is not represented; in Petri nets, transitions fire instantly without consuming time.*

Formal verification entails checking if a model representing a system conforms to a set of specifications. One approach to formal verification is model checking, where systems are represented as labeled transition systems. In order to use model checking techniques over a Petri Net, we use state-based labeled transition systems called *Kripke structure* [2] to represent the execution state space of a Petri Net.

Definition 7. *(Kripke Structure). Let AP be a set of atomic propositions. A Kripke structure K over AP is a quadruple $K = (S, S_0, R, L)$, where:*

- *S is a finite set of states,*
- *$S_0 \subseteq S$ is a set of initial states,*
- *$R \subseteq S \times S$ is a transition relation such that it is left-total, meaning that for each $s \in S$ there exist a state $s' \in S$ such that $(s, s') \in R$, and*
- *$L : S \to 2^{AP}$ is a labeling function over the set of atomic propositions.*

2.2 Predicate Properties

In the following, we define the semantics of Computation Tree Logic (CTL) [11] on a Kripke structure K using the minimal set of CTL operators $\{\neg, \vee, EX, EG, EU\}$.

Definition 8. *(Computation Tree Logic (Syntax)). A Computation Tree Logic (CTL) formula has the following syntax:*

$$\varphi ::= true \mid p \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid EX\varphi \mid EG\varphi \mid E[\varphi U \varphi]$$

where p ranges over a set of atomic propositions, $\neg$, $\wedge$ are the classical logic connective, and the remaining elements are temporal operators.

The other temporal operators can be derived from EX, EG and EU: AXϕ = $\neg$EX $\neg\phi$; AGϕ = $\neg$ EF$\neg\phi$; AFϕ = $\neg$ EG$\neg\phi$; EFϕ = E [true U ϕ]; A[ϕ U β] = $\neg E[\neg\beta$ U $\neg\phi \wedge \neg\beta] \wedge \neg$EG $\neg\beta$.
Each temporal operator consists of a path quantifier (E, "there is a path", or A, "for all paths") followed by a state operator (X, "next state", U, "until", G, "globally in states" or F, "some future state").

Definition 9. *(Computation Tree Logic (Semantic)). Let $PN = (P, T, W)$ be a Petri Net, $s \in S$ a state of PN and ϕ a CTL formula. We define $(PN, s) \models \phi$ inductively as follows:*

1. $(PN, s) \models true$.
2. $(PN, s) \models p$ *iff* $p \in L(s)$.
3. $(PN, s) \models \neg\phi$ *iff* $(PN, s) \not\models \phi$.
4. $(PN, s) \models \phi_1 \wedge \phi_2$ *iff* $(PN, s) \models \phi_1$ *and* $(PN, s) \models \phi_2$.
5. $(PN, s) \models EX\phi$ *iff* $\exists s' \in S$ *such that* $(s, s') \in R$ *and* $(PN, s') \models \phi$.
6. $(PN, s) \models EG\phi$ *iff there is a path* $p = [s_0, s_1, ...]$ *in the states of* PN *such that* $s_0 = s$ *and* $(PN, s_i) \models \phi$ *for all* $i \geqslant 0$.
7. $(PN, s) \models E[\phi_1 U \phi_2]$ *iff there is a path* $p = [s_0, s_1, ...]$ *in the states of* PN *such that* $s_0 = s$, $\exists i \geqslant 0$, $(PN, s_i) \models \phi_2$ *and* $\forall j < i$, $(PN, s_j) \models \phi_1$.

Given a Petri Net $PN = (P, T, W)$ and its set of initial states $Init(S) \subseteq S$, we say that $PN \models \phi$, if and only if $(PN, s) \models \phi$ for some $s \in Init(S)$.

Definition 10. *(Data Property). Let P be a process graph. A Flow Condition FC_ϕ is a set of Activities or Events in P. A Data Condition DC_ϕ is a set of data values in P. A Data-Value Centered Property is a CTL formula whose atomic formulas each refer to the marking of a corresponding place of an element in FC_ϕ or DC_ϕ.*

Example 6. *The data property for the question: "Can product.1 have a price of* 10 *at the end of the auction?" is:*

$$EF(product.1.price.10 \wedge e.end)$$

In this formula, "product.1.price.10" shows the "price" of 10 for "product.1". The atomic formula "e.end" is an end event, i. e., represents the end of the process.

3. Simultaneous Multi-Round Spectrum Auction

There is substantial agreement among economists that auctions are the best way to assign spectrum resources [12]. There are several advantages of using auctions to assign the spectrum [13]. The primary advantage is that the auctions assign the spectrum to those with the highest values. The companies with the highest value for a spectrum are the most likely to have higher bids and win the licenses. A second advantage of an auction is that the competition resulted from the auction increases the revenue of an auctioneer. Finally, an auction procedure to assign licenses is transparent, i.e., the way that the parties win licenses is clear, and all parties can see who won the auction and at which price.

Literature features two alternatives to study auctions, namely (a) experimental analyses with human subjects performed in laboratories [14], or (b) theoretical analyses. However, finding undesirable outcomes continues to be a challenge with both categories: Regarding (a), laboratories tend to perform only relatively few experiments to arrive at possible outcomes. To check all possibilities of an experimental design in [15] would require more than 13 million experiments, to give an example. Such a setup is beyond the capacity of any laboratory. Regarding (b), researchers use auction theory to predict equilibrium results, based on assumptions on bidding behavior [16]. A standard assumption is rationality of bidders [17]. However, this assumption does not always hold [18, 19]. Even the development of frameworks for truthful bidding under interference constraints [20] cannot preclude irrational bidders. Overall, catastrophic outcomes of auctions can result from design errors going unnoticed.

In the Netherlands UMTS Auction in 2000, for example, the low revenue caused a fiasco in public policy [21]. A decade-long loss of 30 MHz in the U.S. mobile market happened because of a policy to increase bid prices. As a result, the flaws in policies cost around 70 billion dollar[5]. In a Switzerland auction in 2011 one bidder paid 485 million Swiss Francs, and another one paid 360 million Swiss Francs for nearly the same package[22]. In another example mentioned in [23], about fifty percent of products remained undersold at the end of the auction.

In this thesis, we show how our proposed verification techniques can be used to analyze and improve a real-world application, the German 4G spectrum auction to sell one of the most valuable bandwidths, the 800 MHz band [24]. We derive the values of three important measures in the worst case [15]: *auctioneer's revenue*, i. e., the sum of the final prices of the products, *bidder's profit*, i. e., the sum of the unused budgets of bidders who have won a product, and *auction's efficiency*, i. e., share of products sold to bidders with the highest budgets. Thus, our proposed techniques can help an auction designer to compare and improve different auction designs.

In summery, the reasons to choose SMR auctions to evaluate our techniques are as follows:

- It is a real-world and an important application with a significant impact on the economy.
- Despite much testing with human subjects ("experiments" in what follows), auctions with unexpected outputs have happened. The unexpected outputs might relate to possible courses of auctions which have remained unobserved in experiments. Verification methods in contrast explore each possible course.
- Verification of auctions allows to detect undesirable outcomes upfront. For instance, one can detect the lowest possible revenue of an auction or its lowest efficiency.
- Spectrum auctions have a clear structure that does not change when altering their parameters, e.g., the domain of bids. This allows to study the impact of our approach systematically.

3.1 SMR Auction Design

The main goal of designing an auction is two-fold [13]. The first is getting the spectrum in the hands of those best able to use it, so-called *efficiency* of the auction. The second is maximizing the auctioneer's revenue. The revenue of an spectrum auction has a significant impact on the economy. To illustrate, they earned 50.8 billion Euros in Germany in 2000 [25].

In line with the goals of an auction design, the Simultaneous Multi-Round (SMR) auctions have been emerged as the standard approach to spectrum auctions. SMR auctions have been the standard format to allocate spectrum licenses to bidders for more than two decades [26]. Many countries such as United States, Australia, Canada, Mexico, Germany, and the United Kingdom have used this design to assign the licenses as well.

Simultaneous Multi-Round auction works as follows. At the beginning of an auction, the auctioneer specifies a *reserve price* for each product, i.e., its lowest acceptable price. So bidders cannot start with a bid smaller than the reserve price. Before the auction starts, each bidder has a particular *eligibility point* based on their deposit. It determines how many bids a bidder can have in each round. The bidder's *eligibility point* decrease if their number of bids are lower than their current *eligibility point*. A bidder can not bid any more, if their *eligibility point* is zero. Each bidder may choose to bid on zero, one, or multiple products simultaneously in each round,

depending on their eligibility points. In the type of auction we analyze, each bidder has an individual budget for each product. In this case, the budget closely reflects a bidder's valuation for this individual product. Thus, bidders cannot use the leftover budget from one product to acquire a different product. In addition, bidders make separate bids for each product, i.e., they cannot bid on bundles of products, different from combinatorial auctions.

To promote competition, there is a so-called *capacity rule* [27]: Each bidder has a *capacity*, the maximum number of products they may win. This rule prevents bidders from winning too many items. In spectrum auctions, this guarantees a certain number of awarded bidders and prevents bidders from forming a monopoly. For example in US auctions, firms can hold no more than 45 MHz of broadband spectrum in any area, assuring that there are at least five broadband wireless competitors in each market.

After bidding finishes in a round, the highest bid for each product will be its reserve price in the following round. Thus, the bidders cannot bid for a product with a price lower than the highest bid of the previous round. This bid is announced to all bidders, while other bids are not disclosed. In addition, bidders do not know their competitors' bids from the currently ongoing bidding round. The auction ends when there is no new bid for any product in a bidding round. The bidder with the highest standing bid for each product is its winner.

Payments of the auction are normally received at three times: (1) An upfront payment before the bidding begins assures that the bidder is serious. Any withdrawal or default penalties are taken from the bidder's upfront payment. (2) A down payment of 20 percent is paid within five business days after the close of the auction. (3) A final payment of the remaining 80 percent is paid within five business days of the award of the license. Licenses are awarded one to three months after the auction.

3.2 SMR Auction in BPMN

Figure. 9.2 shows a simplified version of Simultaneous Multi-Round auction in BPMN notation. This model consists of three subprocesses: *availability of bidders*, *bidding of each bidder*, and *winner determination*.

The first subprocess represented in Figure 3.2 checks whether there is a bidder who can afford a product (*Availability of bidders*). The product that they can afford must not be a product that they have already won. There are two gateways to check these conditions, i.e., *bidder = current winner* and *budget > product's price*. The auction continues if there is at least one qualified bidder who can place a bid in Subprocess *bidding of each bidder*.

The subprocess in Figure 3.3 places bids. Here, Activity *place bid* issues a random bid between the current price of the product and the budget of the qualified bidder. The bid is randomly selected to simulate all the possible behaviours of the bidders in a real auction. In our process model, bidders will always bid if they have both budget and capacity left to acquire a product. Activity *decrease capacity* decreases the capacity of the bidder who just won a product. This capacity point will be increase again, in case another bidder wins the product in the next round. If a bidder has no capacity left, Activity *remove bid* removes their bids. This is because their bids are not valid.

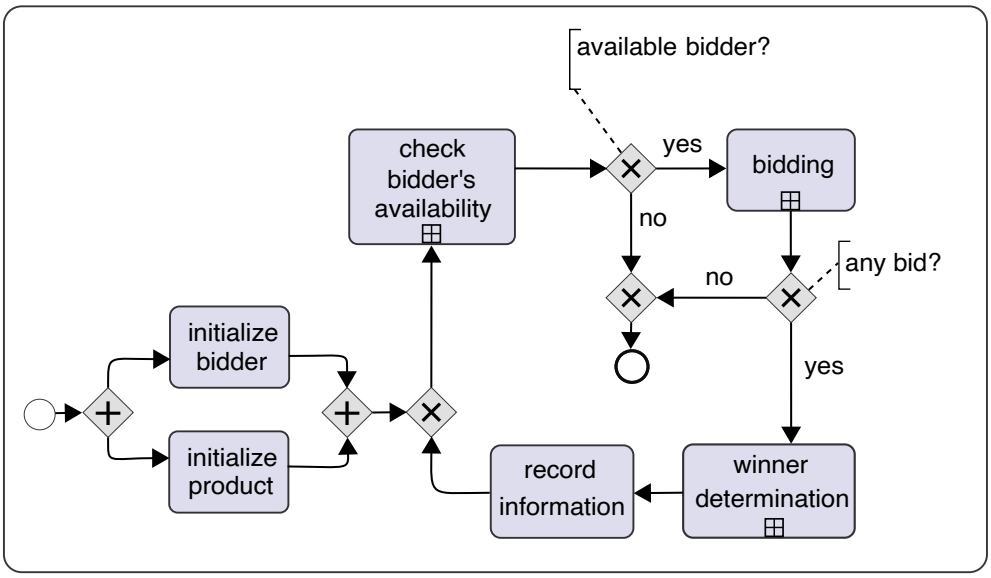


Figure 3.1: Simplified process model of an SMR auction in BPMN notation.

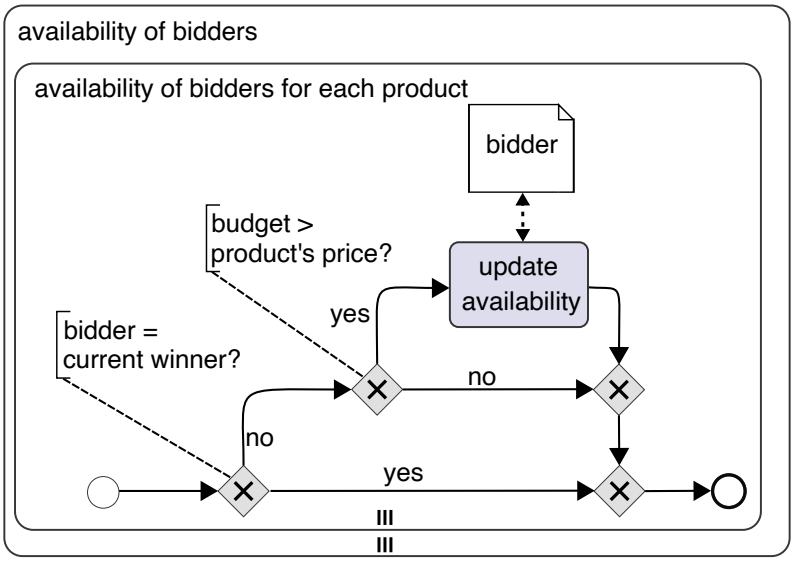


Figure 3.2: Subprocess checking the availability of bidders

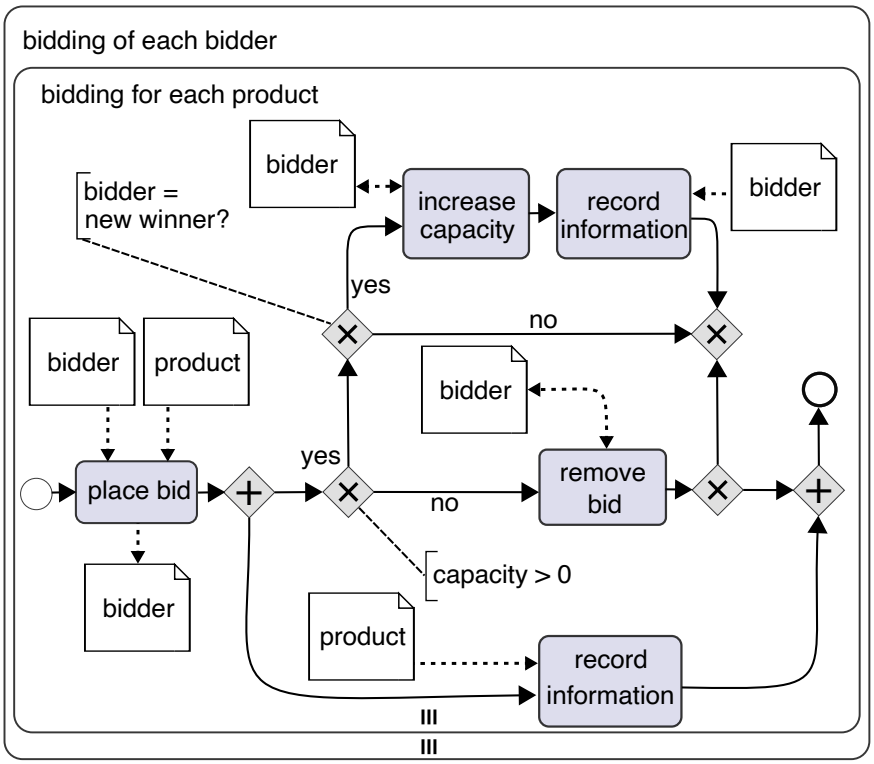


Figure 3.3: Subprocess placing bids

The subprocess in Figure 3.4 outputs the new reserve prices and the winners based on the existing bids. The bidder with the highest bid will be the winner and its bid indicates the reserve price of the product for the next round. Activity *remove price* removes the previous price of the product and updates it with the new one. These three subprocesses are repeated until no more bids are placed.

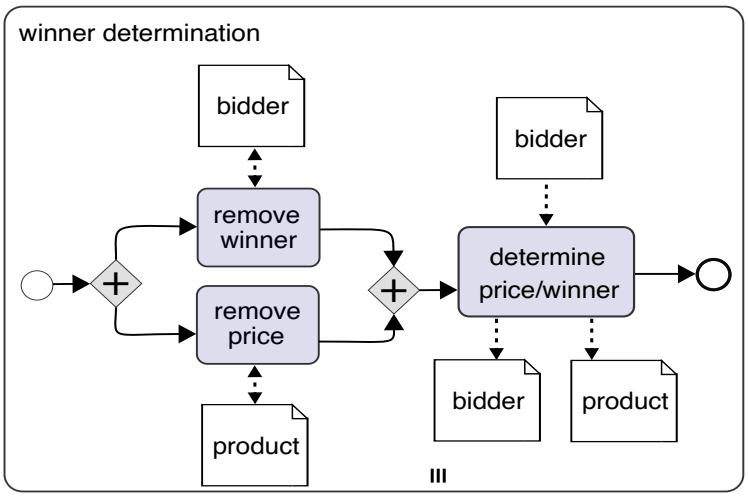


Figure 3.4: Subprocess determining the winner and price

4. Related Works

4.1 Verification of Process Models

Many approaches exist for verification of process models with a focus on the control flow. In the following, we only mention work that deals with data in process models. [28] is an early approach based on Petri Nets to deal with data conditions in process models. They map a data object to a Petri Net by creating a new place for each state of a data object. They can detect data anomalies, e. g., missing data.

Another contribution [29] features a technique to verify decision-aware process models. A *decision-aware process model* is a process model integrated with decision tables. A decision table comprises a set of rules, which consist of conditions for the inputs and expressions. To verify such process models, the authors formalize decision-aware processes as colored Petri nets, extract the state space, and check compliance rules using temporal logic model checking. The authors of [30] and [31] propose an approach to verify decision-aware process models. Their purpose, however, is to verify the soundness of process models, i.e., to ensure that there are no deadlocks and that termination is possible.

The authors of [32] propose and evaluate an extension of coloured Petri nets, called catalog-nets to deal with verification of safety properties. Safety properties ensure that an undesirable behaviour will never occur. The approach of [32] has two key features: First, net transitions feature guards and, at the same time, query facts in a read-only, persistent database and inspect the content of tokens. Second, the transitions can inject data into tokens. These transitions extract values from the database or generate new ones to do so. [33] considers safety checking of systems with read-only database. It provides a parameterized verification of data-aware BPMN called DAB to verify safety properties. DAB supports case data, as well as persistent relational data partitioned into a read-only catalog and a read-write repository. Case and persistent data are used to express conditions in the process as well as task preconditions; tasks, in turn, change the values of the case variables and insert/update/delete tuples into/from the repository. By encoding DABs in an array-based artifact system framework, SMT-based model checking of array-based systems with read-only database [34] can be used to verify properties of DABs. [35]

proposes a technique to verify the structural and behavioural properties of process models with data-value conditions. They abstract conditions from a model and argue that the full data perspective is not required to verify such properties.

In [36] authors present an approach to apply data-centric verification to activity-centric processes. However, they focus on reachability analysis, and do not support checking of arbitrary logic formulas as in our approach. A so-called artifact system [37] features a restricted class of artifact systems and LTL-FO properties to make the satisfaction decidable. There, artifacts carry values of an attribute which external services can update. These services work with an underlying database. They do not support arithmetic operations, which play a significant role in applications such as spectrum auctions. [38] formalizes data-aware properties by Linear Temporal Logic with Past Operators and employs a model checker to verify properties. When detecting an error, they apply Temporal Logic Querying to explain the violation. An additional recent study [39] presents an approach to verifying the soundness of data-aware processes with arithmetic conditions.

The authors of [40] introduce the property of "finite summary" and study the verification problem for the linear-time, finite-trace temporal logic in Data-Aware Dynamic Systems with Arithmetic (DDSAs). Finite summary guarantees the existence of a faithful, finite- state abstraction for DDSAs. The authors extended the work in [41] by proposing a technique to compute witness maps for a given DDSA and CTL* property. A witness map specifies conditions on the initial variable assignment such that the property holds. The addressed problem is thus a slight generalization of the common verification problem.

4.2 Reduction on the level of Process Models

There exist various techniques to deal with state-space explosion in data-aware process models. The authors of [42] propose an approach that abstracts from the data domains and finds intervals using the conditions of gateways. They specify conditions of XOR gateways based on data values, but they do not specify how the process elements modify the data values used in conditions. So this information is lost, and they cannot verify properties in process models which modify data values. [43] features a restricted view on the data objects used in a process model to simplify the process view. They propose relevance functions to detect the relevant data objects, independently from the control-flow aspect. Their definition of relevance does not cover the case of control-flow nodes manipulating data objects. Next, their relevance-based reduction is restricted to the data aspect and does not deal with the control flow.

The approach in [44] reduces data-aware process models as well. The idea is that activities associated with the same data elements are semantically related and therefore are candidates to be aggregated into the same activity. However, two activities might use the same data object, but modify it differently, i. e., one might increase and the other one decrease its value.

A range of abstraction techniques has been developed aiming to reduce the size of the process models. In abstraction techniques, a process element is called significant when it fulfills a certain requirement, e. g., high execution cost. Insignificant elements will be either eliminated or aggregated, while preserving the execution logic of the

process [45]. In [46], the authors first carry out an abstraction of the process model and then evaluate all data objects of each abstracted process fragment with respect to three sets of rules. Each rule decides to keep or delete a data object in a process model. The rules in [46] do not preserve the property of processes when data values are modified. The complexity of process models evaluated also is low, with fewer than 10 data objects. [47] reduces data objects and attributes used in a process model by abstraction as well. Their reduction aims to simplify the visualization of process models.

The approach in [48] annotates activities with supplementary information, e. g., cost and time. Then various functions map the process model to one where an insignificant process fragment is replaced with a generalization. This approach is able to estimate properties of the process such as effort and cost.

The approach described in [49] provides rules to simplify complex BPMN models by reducing control flow and data. But this does not include a mechanism to carry out and implement these reductions. Next, their simplification rules are not valid in our contexts. For example, they eliminate a data object which is connected to only one basic flow object. In our applications however, this data object might be relevant, as it can be used to modify the value of another data object. [50] reduces the complexity of BPMN process models by detecting redundant or repetitive paths that do not provide any new information. The authors identify five patterns to accomplish this. Among these patterns, only two are usable in our applications. The other ones do not preserve verification results of process models with data-based gateways.

A range of program slicing techniques exists to detect the parts of a program without any effect on the semantics of interest. See [51] for an overview. Since our technique applies to process models, and program slicing is for software, the methods of detecting relevant elements and of removing irrelevant elements are different. To illustrate, in our context, one cannot remove an irrelevant activity from the process model when it has an XOR ancestor, while program slicing might do this.

There exist techniques based on predicate abstraction to reduce the size of the state space [52, 53]. Our approach is similar to such techniques in the sense that they both make the state space smaller. Predicate abstraction enables reasoning over an abstract state space that is much smaller than the original one.

4.3 Reduction on the level of State Space

The problem of state-space explosion has been dealt with by proposing a range of reduction techniques on the level of the state space. Stubborn set reduction [54], a partial order reduction technique, is one such technique. It selects a subset of the operations enabled in a state s (stubborn set for s) and only explores these subsets. The approach of Gerth et al. [55] is a reduction approach for CTL-formulas without the Next operator. One can find smaller stubborn sets if the CTL formula is limited to the EF/AG-Operators [56]. But with these operators, it is not possible to express all possible CTL-formulas. Identification of stubborn sets is not feasible for branching time model checking, e. g., CTL or CTL* on complex processes, like the ones we want to verify. One reason is that a dependent operation can activate another operation that then affects the property analyzed. This results in large

stubborn sets, i. e., will not yield a noticeable reduction in various realistic data-value-aware process models, including the ones we work with in our evaluation. In our evaluation, we have used stubborn set reduction and other approaches on this level. There, stubborn set reduction has worked only in combination with our approach. With stubborn set reduction in isolation, the state space remains too large.

Part II

Verification of Data-Value-Aware Process Models

5. Verification of Data-Value-Aware Process Models

In this chapter, we propose a new approach to verify process models including data values, so-called *data-value aware process models*. The content of this chapter is based on [57].

5.1 Introduction

Verification techniques are fundamental to improve the reliability of process designs in practice. Current verification approaches tend to be confined to control flows, even though data values play a significant role. We address this issue by proposing a new data-value-aware verification approach: We enhance the process model to facilitate specifying the usage of data values and their modifications during the process flow, using so-called Data-Value Enhancement Functions (EF). Next, we propose a new algorithm to transform this data-value-aware process model to Petri Nets, a formal modeling language, for which a comprehensive set of analysis tools is available [7]. This way of generating a Petri-Net-based representation is generic, i. e., does not take any application-specific characteristic into account. Our approach is analogous to [58]. However, they only address the optional and alternative usage of data objects by activities as a whole and not at the level of data values. To our knowledge, we are the first to consider data values and their modifications in the transformation of process models to Petri Nets. Based on the generated Petri Nets, one can specify data-value centered properties of the process model as CTL formulas [8]. By deploying an off-the-shelf model checker [2], one is now able to find, say, undesirable outcomes. We evaluate our approach against the use case of Simultaneous Multi-Round (SMR) spectrum auctions. Our evaluation shows that our approach allows to check important data-value centered properties of SMR auctions. For example, one can now find the worst possible values of three relevant measures of auction designs [15]: *auctioneer's revenue*, i. e., the sum of the final price of products, *bidder's profit*, i. e., the sum of the unused budgets of bidders who have won a product, and *auction's efficiency*, i. e., selling the products to bidders with the highest budgets. Put differently, we cannot only prove the presence of flaws, but

also their absence. Last but not least, when we detect an undesirable outcome, we can provide a counterexample. On the downside, the auctions whose verification has been possible without state-space explosion are smaller than the ones having taken place in reality. Thus, the reduction of state space is required, as we will explain in the following chapters.

5.2 Our approach

Figure 5.1 is an overview of our approach. The gray boxes indicate the contributions of this chapter. Task *enhance process model* enriches the designed process model by specifying the usage of data values and their modifications. To this end, a specification will make use of our new Data-Value Enhancement Functions (EF), see Section 5.2.1. Task *data-value-aware transformation* transforms the enhanced process model to a Petri Net. To do so, our approach generates subnets reflecting the semantics of both the preconditions and the effects on data values, see Section 5.2.2. Task *specify data-value centered properties* specifies properties as CTL formulas on the resulting Petri Net, Section 5.2.3. In the last step, Task *model checking* checks the properties with an off-the-shelf model checker.

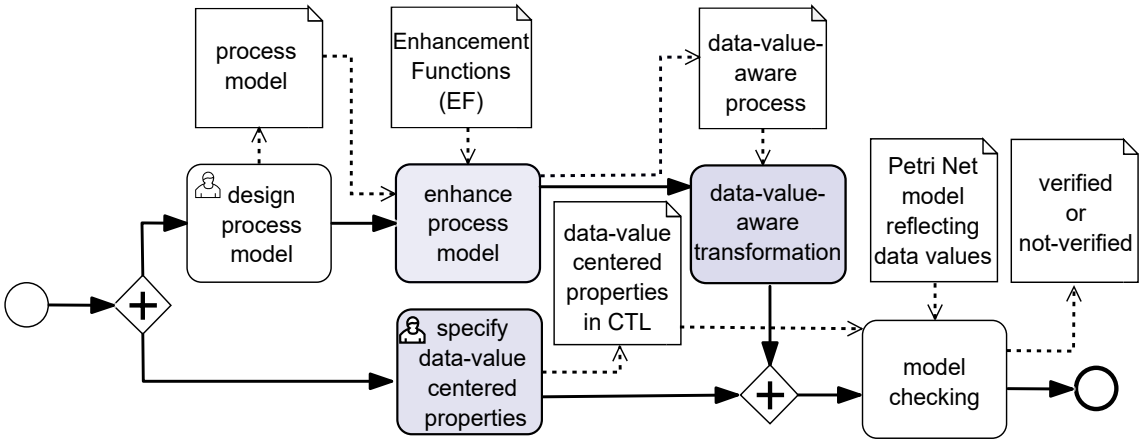


Figure 5.1: Overview of our approach on verification of data-value-aware process models

5.2.1 Process Enhancement with Data Values

Data values used in a process influence the process execution. In particular, the conditions of gateways determine the branch the process will follow. These conditions use the values of data objects which preceding activities have assigned.

Example 7. *In the SMR auction process model, a gateway with condition "bidder's capacity points > 0" checks the "capacity rule" of the SMR auction. The gateway takes the decision based on the value of "bidder's capacity points". However, Activity "decrease capacity" can modify this value during the process. This modification may affect the execution of the process.*

To take the usage of data values into account, we allow to specify data values as *preconditions* as well as *effects* of a process element, as follows. We refer to the preconditions as *data-value conditions*. We deal with the effects of process elements on data values with so-called *data-value functions*.

Definition 11 (Data-Value Condition)**.** *Let P be a process graph from Data Domain $\mathcal{D}$. A Data-Value Condition is an expression of the form $(a \otimes dv)$ where a is an attribute in P, dv is a value in the domain of Attribute a, and $\otimes := \{=, \neq, <, >, \leqslant, ...\}$. We define $DVConds$ as the set of all data-value conditions with the elements of Process Graph P.*

Definition 12 (Data-Value Function)**.** *Let P be a process graph from Data Domain $\mathcal{D}$. A Data-Value Function is a function with attributes of P as input and output parameters. The types of the parameters are from a domain in $\mathbb{D}$. We define $DVFuncs$ as the set of all data-value functions with elements of P, i.e.:*
$DVFuncs : \times_{i=1}^{n} D_i \rightarrow \times_{j=1}^{m} D_j$, where $D_i, D_j \in \mathbb{D}$.

Example 8. *Suppose that data value dv in function f_1 belongs to Attribute price of Data object product.1. Then function f_1 reduces the price of this product by 1.*
$DVConds := \{(capacity \geqslant 1), ...\}$,
$$DVFuncs := \left\{ f_1(dv) = \begin{cases} dv - 1 & dv \geqslant 1 \\ 0 & dv < 1 \end{cases}, ... \right\}$$

So far, process models do not allow to specify conditions on data values or modifications. So we propose so-called *Data-Value Enhancement Functions* (EF) to annotate process graph elements of P with the usage of data values.

Definition 13. *(Data-Value Enhancement Functions) Let P be a process graph from Data Domain $\mathcal{D}$. Data-Value Enhancement Functions are a set of functions to enhance process graph elements of P as follows:*

EnhancementO : $O \rightarrow (\mathbb{A}, pk)$ *is a function to enhance a data object $o \in InputSet(t) \cup OutputSet(t) \cup InputSet(g)$, where $t \in T_P \cup E_P$, $g \in G_P$ with attributes $A \subseteq att(o) \subseteq \mathbb{A}$. In this, pk is the key value of Data object o. It allows to deal with several data objects from the same type. O_{enh} is the set of data objects in P enhanced with EnhancementO.*

EnhancementA : $T_P \cup E_P \rightarrow DVFuncs$ *is a function to enrich activities and events in P. It assigns a data-value function $f : I \rightarrow O$ to an Activity/Event t with $InputSet(t) \subseteq O_{enh}$ and $OutputSet(t) \subseteq O_{enh}$. I is the domains of the enhanced data objects in $InputSet(t)$ and O is the domains of the enhanced data objects in $OutputSet(t)$. Activity/Event t reads $InputSet(t)$ and applies Function f. It writes the output parameters of Function f to $OutputSet(t)$. A_{enh} is the set of activities and events in P enhanced with EnhancementA.*

EnhancementG : $G_P \rightarrow DVConds$ *is a function which assigns a data-value condition $\in DVConds$ to a gateway g with $InputSet(g) \subseteq O_{enh}$. The data-value condition specifies a decision formula based on the attributes belonging to an enhanced data object in $InputSet(g)$. G_{enh} is the set of gateways in P enhanced with EnhancementG.*

Example 9. *Figure 5.2 shows the enhancement of Data object product with EnhancementO. The enhancement specifies Attributes price and winner of Data object product.1. Figure 5.3 shows the enhancement of Activity decrease to reduce the price*

of product.1. Activity decrease reads the price of product.1 *and writes the modified value to the same data object. Figure 5.4 illustrates an enhanced gateway to check if the capacity of bidder*.1 *is greater than zero.*

Process-graph elements form the core of many process specification languages such as BPMN or BPEL. As we only enhance the process-graph elements, one has to add a certain number of workflow elements to the process graph in order to model iterative or multi-instance subprocesses of high-level process modeling languages. To illustrate, all bidders in an SMR auction have the same *bidding* subprocess, consisting of process-graph elements. To model this subprocess without iterative subprocesses, one needs to add the subprocess separately for each single bidder into the process model. This leads to more BPMN elements. For a discussion of the number of additional elements see Section 5.3.

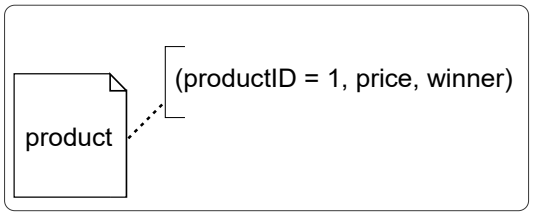


Figure 5.2: Example an enhanced data object

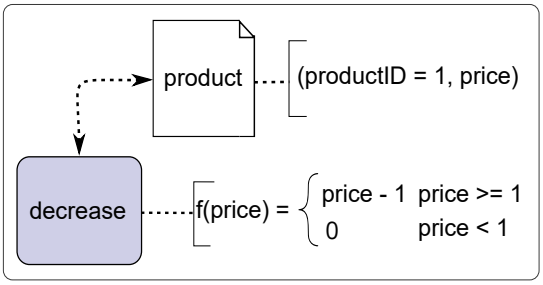


Figure 5.3: Example of an enhanced activity

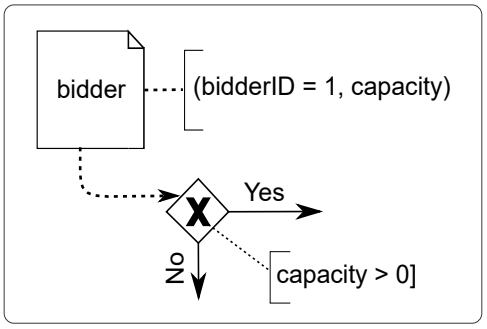


Figure 5.4: Example of an enhanced gateway

5.2.2 Transformation of data-value-aware processes to Petri Nets

To facilitate rigorous analyses, various approaches transform process models to Petri Nets. [59] is a survey of transformations from process models to Petri Nets.

However, all these approaches do not transform data-value-aware processes, let alone ones that modify data values. So we propose a new approach to transform a data-value-aware process to a Petri Net, see Algorithm 1.

First we transform the control flow, regardless of the data objects, with the rules in [60] (Line 2). However, the resulting "control flow" Petri Net does not reflect the semantics of the data value usages. To overcome this problem, our algorithm now works in two parts. The first part, the so-called *mapping*, generates new places corresponding to data values used in the process (Lines 3, 4). The second part creates subnets for the process elements which use data values that extend the already generated Petri Net. This is called *unfolding* (Lines 5, 6).

Algorithm 1 Transformation: From a Data-Value-Aware Process to a Petri Net

```
1: procedure ProcessModelPetriNet(P)
2:     Petri Net pn ← transform control flow of P
3:     for all data value dv in P do
4:         pn.place ← new place p.dv                    ▷ Mapping
5:     for each element ∈ {A_enh ∪ G_enh} do
6:         subnet ← CreateSubNet(element, pn)
7:         extend pn with subnet                        ▷ Unfolding
8:     return pn
```

1. Mapping. The mapping generates a new place for each data value which belongs to a data object in Process Graph P (Lines 3, 4). The number of data objects for each data-object type in P depends on the domain of *key*s. For example, if the key of Data-object type *bidder* has a domain of $\{1, 2\}$, it generates two sets of places, one set for *bidder*.1, another set for *bidder*.2. These new places are called *data-value place*s. A data-value place has a name with the following structure: *p*.[*object*].*pk*.[*attribute*].*dv*. Here, [*object*]/[*attribute*] is the name of the data-object type/attribute the data value belongs to. *pk* is the *key value* of the data object, and *dv* is its data value. The naming allows to identify places relevant for the definition of properties.

Example 10. *Suppose that data-object type bidder has Attributes capacity with domain* $\{0, 1, 2\}$ *and bidderID as the key with domain* $\{1, 2\}$*. The mapping generates 6 places:*

$\{$*p.bidder.1.capacity.0, p.bidder.1.capacity.1, p.bidder.1.capacity.2, p.bidder.2.capacity.0, p.bidder.2.capacity.1, p.bidder.2.capacity.2*$\}$.

2. Unfolding. For every enhanced element of the control flow, i. e., the elements of $A_{enh} \cup G_{enh}$, we create a new subnet, to reflect the semantics of the use of data values (Lines 5-7 of Algorithm 1). To do so, we propose Algorithm 2 described next. It creates different subnets depending on whether the enhanced element is an activity/event or a gateway.

a) Creating a subnet for an enhanced activity/event *element*: Observe that the element is associated with a data-value function. The algorithm creates a new transition for every combination of data values belonging to the enhanced data objects

of *InputSet*(*element*) (Lines 4-6). Each of these transitions has the corresponding data-value places as *input places* (Line 7). The algorithm computes the output parameters of the data-value function (Line 8). The transition created has the data-value places of the output parameters as its *output places* (Line 9). All transitions have an outgoing arc to Place *p.end* (Line 10).

Example 11. *Figure 5.5 shows the subnet created for an enhanced activity with Data Object "product.1" as its input and the following data-value function:*

$$f(price) = \begin{cases} price - 1 & price \geqslant 1 \\ 0 & price < 1 \end{cases}$$

The activity decreases the price of "product.1" by 1. The gray places and transitions already exist as a result of the transformation of the control flow (without data aspect) into a Petri Net. The algorithm creates a separate transition for each data value dv in the domain of Attribute price of Data object "product.1", $[1; 10] \cap \mathbb{N}$*. Each of these transitions gets the corresponding data-value places as "input place"s, for instance "p.product.1.price.10". The transitions also get the data-value places of the output parameter of Function f as "output place"s, e. g., "p.product.1.price.9". In the subnet generated, we handle all places representing the input parameter values (possible values are 1 to 10) by a transition and yield a resulting value decreased by 1.*

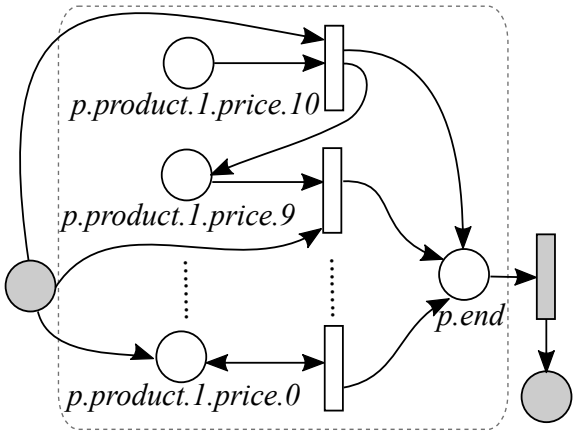


Figure 5.5: Example of a created subnet for an enhanced activity

b) Creating a subnet for an enhanced gateway *element*: Observe that the element is associated with a data-value condition. We create a single transition for each data value specified in a data-value condition for an enhanced data object in the *InputSet*(*element*) (Lines 11, 13). Each of these transitions has the corresponding data-value places as its *input place* (Line 14). Since the result of the data-value condition is either *True* or *False*, we create two new places for the result: *p.True* and *p.False*. Depending on whether the data value represented by the *input place* of a transition satisfies the data-value condition, the *output place* of the transition is Place *p.True* or *p.False* (Lines 15, 16).

Example 12. *Figure 5.6 shows the subnet created for an enhanced gateway with Data Object bidder*.1 *as its input and Data-Value Condition capacity* > 0. *Suppose capacity has a domain of* [0..2]. *Then, for all data values of Attribute "capacity" of Data object bidder*.1, *the algorithm creates a new transition "t". Each data-value place is at the same time an input and output place of transition "t", because it has to keep its data value.*

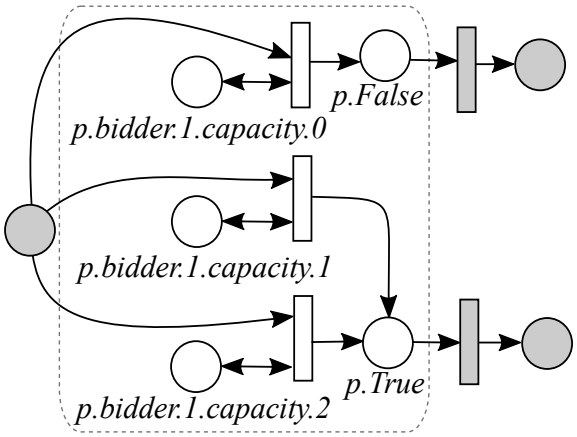


Figure 5.6: Example of a created subnet for an enhanced gateway

Algorithm 2 Create Subnet

```
1:  procedure CreateSubNet(element, pn)
2:      create Petri Net subnet
3:      if element ∈ A_en then
4:          domElement ← dom(InputSet(element))
5:          for all combination of dv⃗ ∈ domElement do
6:              subnet.transition ← new transition t
7:              t.inputPlace = pn.getPlace(dv⃗)
8:              dv⃗_result ← element.dvFunction(dv⃗)
9:              t.outputPlace = pn.getPlace(dv⃗_result),
10:                             subnet.getPlace(p.end)
11:     else element ∈ G_enh
12:         for all dv ∈ dom(InputSet(element)) do
13:             subnet.transition ← new transition t
14:             t.inputPlace = pn.getPlace(dv)
15:             element.dvCondition(dv) == true ?
16:             t.outputPlace = p.True : p.False
17:     return subnet
```

5.2.3 Specification of properties

The result of the transformation is a Petri Net of the process reflecting the semantics of the use of data values. To verify the Petri Net created, one must specify properties in a formal language such as CTL. For example, to verify the values of

three characteristics of an SMR auction (as we explained in Chapter 3), we have to specify the properties referring to data values in a formal language like CTL. In this section, we demonstrate how to specify SMR properties. For the definition of such properties we use Definition 10.

Example 13. *Consider the following question: "Can bidder.*1 *win product.*2 *at a price of 8 at the end of the auction?" – The respective data-value property is:*

$$EF(p.product.2.price.8 \wedge p.product.2.winner.1 \wedge e.end)$$

In this formula, "$p.product.2.winner.1 \in DC_\phi$*" shows bidder.1 wins product.2.* $p.product.2.price.8 \in DC_\phi$ *shows the "price of 8 for product.2" and "*$e.end \in FC_\phi$*" is the result of the transformation of an end event, i. e., it represents the end of the process.*

Example 14. *"Can all bidders win at least one product?" translates to:*

$$EFAG((p.product.1.winner.1 \vee p.product.2.winner.1) \wedge (p.product.1.winner.2 \vee p.product.2.winner.2))$$

Example 15. *"Can product.*1 *be sold for the price of 2?" translates to:*

$$EF(p.product.1.price.2 \wedge e.end)$$

Verifying the property of Example 15 allows to find the lowest revenue of the auctioneer (R_{lowest}). To do so, we first find the lowest final *price* of *products*, starting with the reserve prices as in Example 15. In case the property is not satisfied, we now verify a property with an increased price, until there is a state that fulfills the property. In the next step, we find the lowest combination of prices in a similar way, starting with the lowest final price of products. Suppose that the lowest price for *product.1* and *product.2* is 4 and 6, respectively. Then the first step is to verify:

$$EF(p.product.1.price.4 \wedge p.product.2.price.6 \wedge e.end)$$

If this verification fails, we perform the verification for other combinations of increased prices. For this, we do not have to look at all prices in combination, but only at those between a maximum and a minimum possible price. The maximum price is known by the maximum budget of a bidder for a certain product, and the minimum is the result of a verification as explained in Example 15. More sophisticated search strategies can restrict combinations to be verified further. This is future work. By verifying the combination of prices, we find the lowest auctioneer revenue possible.

Next, one can find the minimum profit of the bidder (P_{lowest}) associated with R_{lowest}. We check if bidders with the lowest budget can be winners. Suppose that *bidder.1* and *bidder.2* have the lowest budget for *product.1* and *product.2*, respectively. The first formula to check is:

$$EF((p.product.1.price.4 \wedge p.product.2.price.6) \wedge (p.product.1.winner.1 \wedge p.product.2.winner.2) \wedge e.end)$$

If the model checker cannot find a state, i. e., an execution path fulfilling the property, we change the winners. This results in new formulas to be checked. This is done until the formula is satisfied.

One can also find the lowest market *efficiency* (E_{lowest}) while the auctioneer's revenue is minimal. To define efficiency, we use a measure described in [15]. It quantifies how the surplus resulting from the worst allocation (S_{lowest}) can deviate from the surplus with a random allocation S_{random}:

$$E_{lowest} = \frac{S_{lowest} - S_{random}}{S_{optimal} - S_{random}} \times 100 \text{ percent}$$

Here, the surplus resulting from an optimal allocation ($S_{optimal}$) is computed by giving the products to bidders with the highest budget. S_{lowest} is the sum of the lowest revenue and profit.

5.3 Implementation and Evaluation

We have implemented a framework to verify data-value centered properties in SMR auctions. Its input of our framework is a process model in BPMN format. To transform the process model into a Petri Net, we have implemented Algorithm 1. The generated Petri Net is verified against properties specified as CTL formulas. To do this, we use the state-space generation and model checking algorithm of the LoLA framework [61].

Evaluation setting. To evaluate our framework, we first model and enhance the SMR auction process including eligibility and capacity rules. We compute the auction measures for three different settings.

- *Process 1*: Three bidders compete to win two products.
- *Process 2*: Two bidders compete to win three products.
- *Process 3*: Four bidders compete to win six products – the size of the German 4G spectrum auction in 2010 [24].

In all settings, we have assigned a random budget to each bidder for a certain product in the range of $[2 - 10]$, similarly to [15]. We also have defined a reserve price of 1 for all products, so all bidders can afford all products at the beginning of the auction. All bidders have a *capacity point* and an *eligibility point* of 2. In this evaluation, we first detect the lowest *revenue*, *bidder's profit* and *efficiency of auction*. Then we analyze the run time of the verification process.

Enhancement of SMR process model. We have modeled and enhanced the three processes with BPMN. Note that the BPMN activities (without subprocesses), events and gateways directly refer to the process graph representation. Table 5.1 lists

the numbers of BPMN elements after the enhancement. As explained, we have to add a certain number of BPMN elements when increasing the number of bidders or of products. For example, to model the *bidding* subprocess not as a multi-instance subprocess, one must add the same subprocess for each single bidder separately. This leads to more BPMN elements of the model, by a constant factor. To illustrate, adding a new bidder increases the number of activities and gateways of the BPMN model by 15 and 27, respectively.

Table 5.1: Number of enriched BPMN elements for SMR auctions with different size

process	activities	gateways	annotations	data associations
2 bidders and 2 products	50	55	145	94
3 bidders and 2 products	65	82	189	105
2 bidders and 3 products	69	81	206	119
4 bidders and 6 products	156	213	744	216

Transformation of SMR process model into Petri Net. We have used the enhanced SMR BPMN models with the characteristics listed in Table 5.1 as input to our transformation into Petri Nets. With this transformation, a new bidder increases the number of bidder objects by 1, and the possible values for bids and winners increase as well. So the result of the transformation is a larger Petri Net because of the unfolding of subnets due to the usage of data values. This can lead to higher verification times and state-space explosion. We will observe the limits of our current approach due to this effect with Process 3.

Table 5.2: The results of transformation and verification of different SMR auction sizes

process	places	transitions	transformation	properties	verification(longest)
Process 1	589	715	0.3 sec	20	2 min and 09 sec
Process 2	869	1123	0.4 sec	24	3 min and 20 sec
Process 3	4979	4876	1.6 sec	-	-

Verification of SMR process model. We have verified the data-value centered properties to compute the auction measures. This has been possible for Processes 1 and 2, but not for Process 3, because of (the expected) state-space explosion, so the remaining discussion focuses on those two processes. To detect the lowest possible revenue, one must verify all possible combinations of product prices. *product.1* and *product.2* have 8 respectively 9 different possible prices, based on the budgets of the bidders. This requires to check 8×9 properties in Process 1. Using the same procedure for Process 2 requires to check 432 properties. However, if we limit the range of product prices by a minimum and maximum value (see Section 5.2.3), the number

of combinations of prices/properties is restricted to only 4 × 5 and 4 × 6 × 1 for Processes 1 and 2 respectively. See Table 5.3. Our verification of Process 1 reveals that the auctioneer's revenue is minimal when assigning *product.1* and *product.2* to *bidder.1* for the price of 5. In this case, the efficiency of the auction and bidder's profit are 100% and 7, respectively. In Process 2, *product.1* has been sold to *bidder.2* for the price of 5, although both bidders would have had more budget. Here, the auctioneer's revenue is 15. So the efficiency is 82.60%. This is interesting: Even for small auctions, the conventional SMR design has inefficiencies, and our approach can reveal them. So an auction designer can now take this format as a starting point for improvement, by also applying our method to new designs and compare results. Put differently, there is a value in verifying small auctions.

Performance of verification. We have studied the run time of verifying properties and the ones of transforming enhanced process models into Petri Nets. We have verified 24 respectively 20 properties – the combinations of possible prices limited by a minimum and maximum value, for Processes 1 and 2, respectively. Verification of a single property takes less than 4 minutes in Processes 1 and 2, using a computer with 16 GB main memory and 2 Intel 3 GHz CPUs. Transformation of Processes 1 and 2 into Petri Nets has been within less than 1 second, see Table 8.3. However, our evaluation attempts of Process 3 have revealed the limitations of a generic approach to Petri Nets transformation. This calls for further work, which is studied in the next chapter of this thesis.

Table 5.3: Verification of properties in Processes 1 and 2

process	product	winner	price	revenue	profit	efficiency
Process 1	product.1	bidder.1	5	10	7	100%
	product.2	bidder.1	5			
Process 2	product.1	bidder.1	5	15	4	82.60%
	product.2	bidder.2	4			
	product.3	bidder.1	6			

5.4 Summery

This chapter has featured a new generic method to verify data-value-aware processes. – as is the case with SMR spectrum auctions and with other settings. To leverage existing model checking approaches and tools, we then have transformed the enhanced process model into Petri Nets. To this end, we have proposed a new algorithm considering data values and their modifications. So one can freely define data-value centered properties as CTL formulas for verification. – we have showcased this for important measures of the quality of auction design. Our approach also generates execution traces that may serve as an explanation of unexpected results. In summary, we have developed a new data-value-aware verification approach, which is generic and not confined to a specific domain, and we have evaluated it with the use

case of SMR auctions. Our verification efforts for various settings of SMR auctions have been successful. They have pointed to inefficiencies of existing auction designs. This holds even though we have only been able to verify slightly smaller settings than ones having taken place in reality. The restriction just mentioned is due to a well-known severe limitation of any verification approach based on model checking, namely state-space explosion. This calls for research on reducing Petri Nets, using domain knowledge of data-value centered processes. In the next chapter, we target at such a reduction, taking inspiration from [62]. The reduction proposed in [62] is however confined to data flows of objects in processes and not on data values, i. e., it does not solve this current problem. All in all, our research not only is an advancement in the field of process verification, but also a valuable contribution on auction research.

6. Reduction of Data-Value-Aware Processes by Relevance

In the previous chapter, we have introduced a new generic approach to verify a data-value-aware process model. In this chapter, we introduce a novel technique to elude the state-space explosion problem by reducing data-value-aware process models to the relevant parts.

6.1 Introduction

There exist techniques to verify data-value-aware process models [6, 29]. However, existing work suffers from state-space explosion [2], i. e., the state space increases exponentially with the size of the domain. To verify a property, one must inspect the state space of the process model. With a naive verification technique, the graph representing the state space must be complete in the sense that it contains all possible states which a property might satisfy, i.e., every possible trace has to be represented. Therefore, each state that a data object can take during process execution must be included in the state space, and this leads to the state-space explosion problem. Think of an auction with six products, each one associated with a price and three bids with a value ranging from 1 to 100. This amounts to 100^6 (price) $\times$ 100^{6^3} (bids) states. Modifications of data values increase the state space even more, by yielding parallel branches in the process model, reflecting the values of a data object that may result. Thus, a naive verification scheme, i. e., one that explores the entire state space, is infeasible. Hence, the question is how to reduce the state space of data-value-aware process models to facilitate verification.

To overcome the problem, reduction techniques have been proposed, either on the level of (a) the state space or (b) the process model. Approaches such as stubborn set reduction [54] fall into the first category. However, many realistic data-value-aware process models, including the ones we work with in our evaluation, are too large to be verified using only reduction techniques on the level of the state space. With this kind of reduction, we could not even construct the state space in our evaluation. Regarding (b), only few proposals exist. Existing work uses data-value

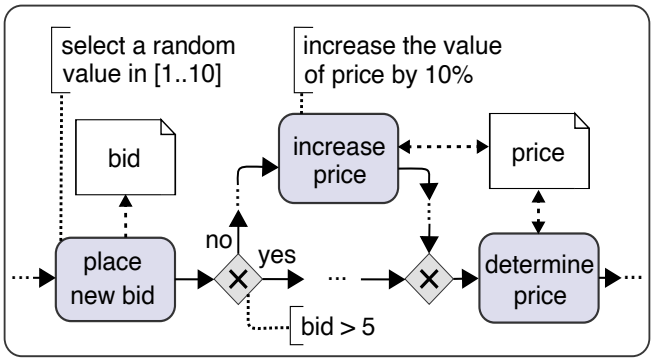


Figure 6.1: Parts of a simplified auction process modeled in BPMN

abstractions [42] and property-specific reduction techniques [63], [62]. The idea is that an element of a process model is relevant if it influences the verification in question. These relevant elements need to be identified. Non-relevant elements are candidates for reduction. However, existing techniques on the level of process models do not solve the current problem: Their definition of relevance is not valid in the presence of modifications of data values. To illustrate, with these proposals, an activity is relevant if it is referred to in ϕ, the property to be verified, or if it requires a data object referred to in ϕ [63]. But in a data-value-aware process model, an activity may not meet these requirements, while it still is relevant for the verification of ϕ. This is because the activity may modify the value of a data object that influences the verification of ϕ. The following is an example of an activity that is not referred to in ϕ but is relevant.

Example 16. *Consider the process model in Figure 6.2 and the property "Can the price be higher than \$10?". With existing approaches, only Activities "increase price" and "determine price" are relevant because they require a data object referred to in the property, i. e., "price". However, Activity "place new bid" also influences the verification of that property. This is because it modifies the value of Data Object "bid". This modification is important, because the new value of "bid" may or may not alter the value of "price" and the verification result. None of the existing approaches deems "place new bid" relevant.*

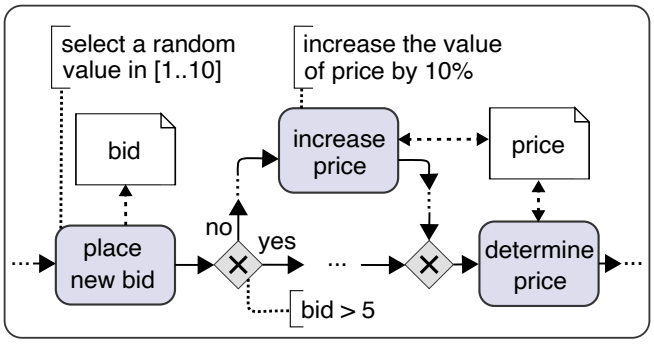


Figure 6.2: Parts of a simplified auction process modeled in BPMN

Detecting the relevant activities and other control-flow nodes of a data-value-aware process model is challenging. Detecting the relevant data elements, e. g., data objects and data-object types, yields further challenges. In general, a data object is relevant

to verify a property ϕ when it is in the input or output of a control-flow node referred to in ϕ [43]. However, it can happen that a data object may not be used by any of the control-flow nodes referred to in ϕ, but still is relevant to verify ϕ. This could be, say, the input data object of a gateway that leads to the execution of a relevant activity. The following example illustrates this.

Example 17. *Consider the process model and the property mentioned in Example 16. The gateway makes its decision based on the value of "bid", which may lead to the execution of a relevant activity, in this case "increase price". Although Data Object "bid" is not used by any control-flow node referred to in the given property, it influences the verification result.*

Contributions. In this chapter, we propose a property-specific reduction approach to address the state-space explosion problem with data-value-aware process models. It enables the first-ever verification of properties in an important real-world application, namely spectrum auctions, with their true size (number of goods and bidders). Our core contribution is an approach that identifies the control-flow nodes and data elements that influence the verification of a given Property ϕ against a data-value-aware process model. To do so, we propose novel relevance functions that output whether a control-flow node or data element is relevant for the specific verification. We prove the correctness of our relevance functions, i. e., that elements labeled as irrelevant by our functions do not influence the verification of ϕ. A distinctive feature of our relevance functions is that they can detect the relevant elements in the presence of modifications of data values. We provide a reduction algorithm for the process model that, once the relevant elements have been detected, reduces the state space to facilitate the verification envisioned. This reduction targets at process models with many parallel branches that result from the modification of data values during the process. Our reduction technique is fairly universal, i.e., different verification methods such as [6], [29] and [32] can be used to verify behavioral properties of the reduced process model.

To validate our approach, we focus on a real-world application, the German 4G spectrum auction to sell one of the most valuable bandwidths, the 800 MHz band [24]. We derive the values of three important measures in the worst case [15]: *auctioneer's revenue*, i. e., the sum of the final prices of the products, *bidder's profit*, i. e., the sum of the unused budgets of bidders who have won a product, and *auction's efficiency*, i. e., share of products sold to bidders with the highest budgets. To derive extreme values of these measures, we repeatedly verify properties stating that the respective measure has a certain value, e.g., a property stating that a certain product can be sold at Price p. In these repeated verification steps, we vary the value of p so that the interval where the extreme value sought falls into is narrowed down with each step.

Our approach reduces the size of the Petri Net by 81% compared to the original one. The reduction of the state space is even much larger. While verifying those essential characteristics of the auction model has been infeasible so far [6], our approach accomplishes this in less than 15 seconds.

6.2 Preliminaries

Our intention is to detect the control-flow and data elements that are necessary to verify a certain property. The property in question might contain conditions

on activities, events, and data values. The definition of data property is given in Chapter 2. Given a data property, we can now evaluate which control-flow nodes and data elements of the process model are relevant to verify it. For the definition of relevance of control-flow nodes and data elements, we rely on the notions of witness and influence on the verification of properties introduced in [62].

Definition 14. *(Execution Path, Witness for Property [62]). Let P be a process graph. An execution path of P is a sequence of nodes $N = \langle i, n_1, ..., n_z, o \rangle$ from the start node i to the end node o, where the following holds:*

- $\{(i, n_1), ..., (n_k, n_{k+1}), ..., (n_z, o)\} \subseteq F$ *for* $1 \leqslant k \leqslant z - 1$.
- *If* $n_k \in XG_P$, *for* $k \leqslant z$, *then exactly one node of* $\{n|(n_k, n) \in F\}$ *is in* N.
- *If* $n_k \in AG_P (k \leqslant z)$, *then all nodes of* $\{n|(n_k, n) \in F\}$ *are in* N.

An execution path in P that makes ϕ true is a witness for the statement that ϕ is true.

Note than the definition of witness for a property relies on a single execution path. However, we support universal properties as well, and are not limited to a particular set of CTL formulas. All CTL formulas can be written in Existential Normal Forms (ENF), i.e., for every CTL formula there exists an equivalent CTL formula in ENF [64]. In other words, all CTL formulas can be represented by EX, EU, EG, and EF. For example, $AG\phi$ is equivalent to $\neg EF(\neg\phi)$. This implies that the existential properties are sufficient to represent all the CTL formulas, including the universal ones.

We refer to the number of activities between Activity a and Activity b in an execution path as the *distance* of Activities a and b. Note that our approach works with properties specifying the order of activities/events and not their exact distances, which one could express using the Next operator in CTL. We will discuss this further in Section 8.2 and explain there why we do not see this as much of a restriction. Observe that earlier work on reduction of process models [63] has left aside the Next operator as well.

Example 18. *Consider Data Property "ϕ: After an occurrence of initialize price there is an occurrence of remove bids at some time in the future". In CTL this is "initialize price → EF(remove bids)". There exists an execution path p = <start, gateway, initialize budget, ..., remove bid, parallel gateway, ..., end> where ϕ is true. So p is a witness for Data Property ϕ. 11Here, the distance of Activities "initialize deposit" and "place bid" is one, because Activity "initialize price" is in between.*

Definition 15. *(Influence of Control-Flow Nodes). Let ϕ be a data property, n a node in the Process Graph P and w a witness for ϕ in P. Then n influences the verification of ϕ*

1. *if n lies on w, and*
2. *if we remove n from w and P, then w is not a witness for ϕ any more.*

Definition 16. *(Influence of Data Elements). Let P be a process graph, N the control-flow nodes that influence the verification of Data Property ϕ in P, and w a witness for ϕ. Then Data Element d influences the verification of ϕ*

1. *if d is needed for the execution of a node $n \in N$, and*
2. *if we remove d from P, then n cannot be executed, and w is not a witness for ϕ any more.*

Example 19. *Consider the property mentioned in Example 18. Here, Activity "remove bids" influences the verification of ϕ. This is because it lies on the execution path p, and if we remove it from p, then p is not a witness for ϕ any more. In addition, Data Object "bid" has an influence on the verification of ϕ. Namely, Activity "remove bid" requires this data object for its execution, and if we remove this data object, then Activity "remove bid" cannot be executed, and p cannot be a witness for ϕ any more.*

One possibility to check the influence of control-flow nodes and data elements on verification of a data property is by 11looking at the witness that makes ϕ true, i.e., 11to check whether the elements lying on the witness are necessary to verify ϕ and therefore influence the verification. However, finding witnesses for a property is the task of a model checker, i. e., the task that we want to speed up. Hence, with the conventional chain of tools for verification, we cannot directly detect the influence of elements. So we propose an approach to identify and remove the nodes from the process model that do not, with certainty, influence the verification of ϕ. The result is a process graph with nodes that *may influence* the verification. The meaning of *may influence* in concrete terms depends on the property ϕ to be verified.

Definition 17. *(Relevant Control-Flow/Data Elements). Let relevant be a function for a property ϕ and a control-flow/data element e as follows:*

$$relevant(\phi, e) = \begin{cases} true & \text{if } e \text{ may influence the verification of } \phi \\ false & \text{otherwise} \end{cases}$$

A control-flow/data element e is relevant to verify ϕ if $relevant(\phi, e) = true$.

Example 20. *Consider a process model with Activity "increase price" that increases the "price" of "product1" and data property: "Can the price of product1 be higher than 10?" To verify the property, our approach identifies Activity "increase price" as relevant. However, Activity "increase price" may not influence the verification of the given property if the value of "price" is higher than 10 before the execution of this activity.*

We consider a process graph as a model for the control flow of a business process. For our reduction technique, it is important to find a decomposition of a process graph into a hierarchy of sub-processes that are sub-graphs with a single entry and a single exit of control. Following is a definition of such decomposition taken from [65].

Definition 18. *(Boundary Node, Entry, Exit, Fragment [65]). Let $P = (N, F, O, I, type, enh_A, enh_G)$ be a process graph and $Fr \subset F$ a subset of the edges of P such that P_{Fr} is a connected subgraph of P.*

- *A node $n \in N_{Fr}$ is a boundary node of Fr if it is the start or end node of P, or if P has edges $e \in Fr$ and $e' \notin Fr$ such that n is incident to e and e'.*
- *A boundary node n is an entry of Fr if no incoming edge of n is in Fr, or if all outgoing edges of n are in Fr.*
- *A boundary node n is an exit of Fr if all incoming edges of n are in Fr, or if no outgoing edge of n is in Fr.*
- *Fr is called a fragment if it has exactly two boundary nodes, entry and exit.*

Definition 19. *(Refined Process Structure Tree (RPST) [65]). Let P be a process graph. A fragment of P is canonical if it does not overlap with any other fragment of P. The set of all canonical fragments of P is the RPST decomposition of P. The corresponding parse tree, i. e., the tree of the canonical fragments such that the parent node of a canonical fragment Fr properly contains Fr, is the Refined Process Structure Tree (RPST) of P.*

6.3 Our approach

The objective of our approach is to identify the relevant elements of a data-value-aware process model, i. e., the control-flow nodes and data elements required to verify a given property. The non-relevant elements are candidates for reduction, potentially yielding a much smaller process model and state space.

Figure 6.3 is an overview of our approach. Our main contributions appear as gray boxes. The approach starts with a transformation of the process model to a data-aware Refined Normal Process Structure Tree (`d_NPST`), Section 6.3.1. The hierarchical structure of the `d_NPST` allows to analyze how a process element and its parent node use data values and to prune the tree to relevant parts. Identifying the relevant control-flow nodes and date elements is the next step (Section 6.3.2). To this end, one must take the role of modification of data values into account. Thus, we first detect the relevant activity/event nodes and the relevant data objects, dubbed *relevant prime elements*. This is because the relevant prime elements are directly related to the modification of data values in a process model, i. e., an activity/event node may modify the value of a data object. Second, given the relevant prime elements, we propose novel relevance functions that yield the relevance of control-flow nodes or data elements of all types, including those not handled so far, e. g., XOR, SEQ, and data-object types. Given the relevant control-flow nodes and data elements, we prune the irrelevant elements and reduce the hierarchically structured `d_NPST` accordingly (Section 6.3.3). Then we transform the reduced `d_NPST` back into a data-value-aware process model (Section 6.3.4). Finally, by transforming the process model into a Petri Net following the rules in [6], we employ an off-the-shelf model checker for verification.

6.3.1 Transformation of Data-Value-Aware Process Models to `d_NPST`

Our assumption is that the process model is well-structured, that is, for every node with multiple incoming arcs (a join), there is a node with multiple outgoing arcs (a split), and vice versa [66]. In this case, the nodes between the split and the join form

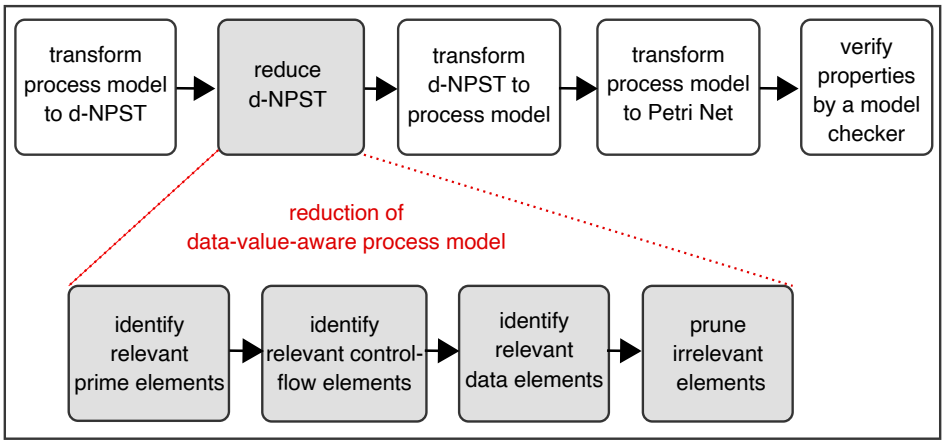


Figure 6.3: Overview of our approach

a single-entry, single-exit region; otherwise, the process model is unstructured. There exist techniques to transform the unstructured process models into the structured ones, if the model is not inherently unstructured, i.e., process models that have no equivalent well-structured representation [67–69].

We employ the parsing technique from [70] to transform a well-structured process model to a *Refined Process Structure Tree* (RPST) and then to the hierarchical structure of the process model where reduction is possible by pruning subtrees that are not relevant. The parsing technique is applicable to any *two-terminal graph*. A graph is two-terminal (TTG) if it has exactly one start and exactly one end node. However, a process graph according to Definition 3 is a *multi-terminal graph* (MTG), i. e., can have several start and/or end nodes. If the graph is a MTG, it must be transformed into a TTG, see [70] before decomposing the RPST. The algorithm from [70] computes the RPST of a TTG in linear time.

To support data values and their modifications, one must analyse how RPST nodes use data values. To do so, we transform the unique modular decomposition in the form of an RPST into a more comprehensive format that represents activities as nodes. So, in what follows, we use the so-called *Data-Aware Normal Process Structure Tree* (`d_NPST`) that is based on [62]. Here, for each process element, there is a node in the `d_NPST`, and this mapping is 1-1 based on the definition. Instead of RPST and NPST one could use an annotated RPST, the so-called WF-Tree [71]. However, it builds on workflow nets instead of process graphs, and process graphs are the core of many high-level process model languages like BPMN. Next, WF-Trees do not include the semantics of data values either. So using WF-Trees would be an alternative to, but not an improvement over our current solution. On the other side, using WF-Trees would require adapting the reduction approach described here so that it is applicable to WF-Trees.

Definition 20. *(Data-Aware Normal Process Structure Tree (`d_NPST`)). A Data-Aware Normal Process Structure Tree is a tree* (N, r, IN, L, E, l) *where* N *is the set of nodes with* $N = \{r\} \cup IN \cup L$*,* r *is the root node,* IN *are the inner nodes and* L *are the leaf nodes,* $E \subset N \times N$ *the edges, and* $l : IN \cup \{r\} \rightarrow \{AND, XOR, SEQ\}$ *a partial mapping together with annotations of the nodes in* L*, so that*

Table 6.1: Data elements of the process model in Figure 6.4

data-object type	data object	attribute	domain
bidder	bidder1	budget	[0..10]
	bidder1	eligibility	[0..3]
	bidder1	bid	[0..10]
	bidder1	deposit	[2..10]
	bidder2	bid	[0..10]
product	product1	price	[0..10]
	product1	winner	[0..2]

- *its root node r and its inner nodes IN correspond to the fragments of the RPST, labeled with the respective fragment type,*
- *its leaf nodes L are the activities and events,*
- *its edges E denote the containment relationship of activities or fragments of activities,*
- *the annotation of a leaf node consists of its input data objects, output data objects and its data-value function,*
- *the annotation of an XOR node consists of its input data objects and its data-value condition.*

The inner nodes are labeled with the type of the fragment, which we determine according to its structure. As fragment types we support sequence and gateways, i. e., AND and XOR. The data elements are the same as the ones in the process graph the RPST is derived from.

Example 21. *Consider process model P from Figure 6.4. This model is a simple auction process consisting of two bidders and one product. Table 6.1 lists the data elements used. The process starts with activities to initialize auction parameters, e. g., Activity "initialize price". Then Activity "place bid" sets a random value for "bid", to mimic a bidding. A bid is a value lower than the bidder's budget and larger than the price of the product. Each bidder makes an upfront payment before the auction starts, called "deposit". Each bidder receives a certain "number of eligibility points" based on his deposit. This number specifies how many bids he can make in each round. If the number of eligibility points of a bidder is zero, his bid is not valid. If the bid is higher than 10, bidders have to increase their deposit, i. e., execute Activity "increase deposit". Next, Activity "price determination" finds the highest bid of the bidders and assigns it to the "price" of "product1". The auction ends when Event end is triggered. Figure 6.5 is the* `d_NPST` *of the data-value-aware process model represented in Figure 6.4. The data elements used in the* `d_NPST` *are the same as in Table 6.1.*

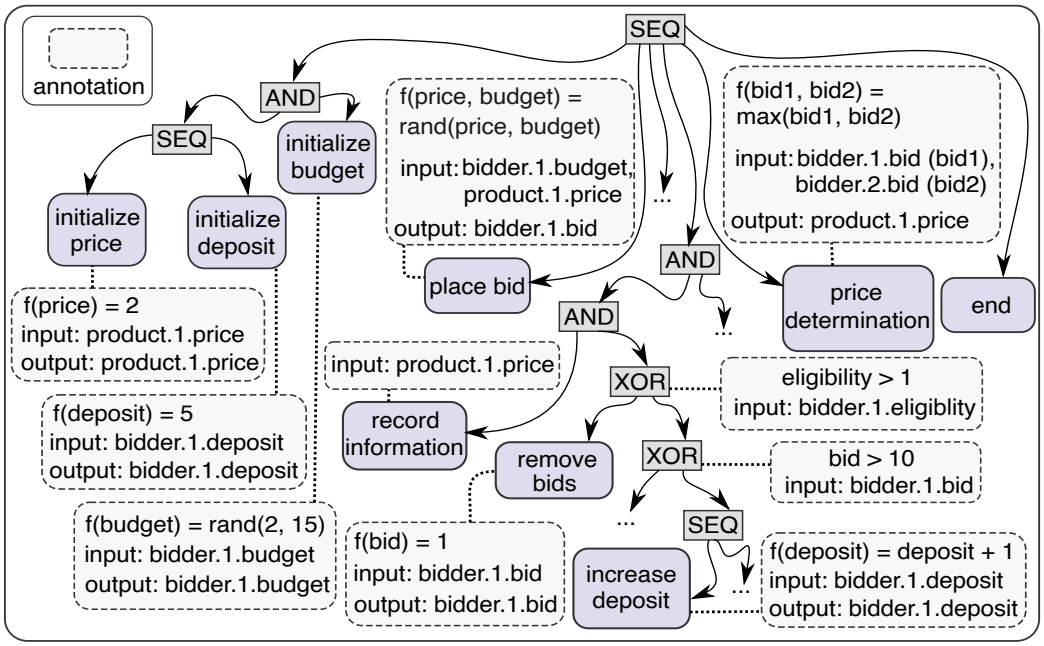


Figure 6.4: Data-value-aware process model in BPMN notation

6.3.2 Relevance Functions

A relevance function states whether a control-flow node or data element is relevant to verify Property ϕ. To derive the relevance functions, we first detect the *relevant prime elements*, i. e., the relevant activity/event nodes and the relevant data objects. Then, given the relevant prime elements, we derive the relevance functions to identify the relevant control-flow nodes and data elements of all types.

Algorithm 3 takes a process model P in form of a `d_NPST` and ϕ as input parameters. It returns the relevant prime elements, i. e., the relevant activity/event nodes A_{rel} and the relevant data objects O_{rel}. First, it initializes A_{rel} and O_{rel} with the activity/event nodes and data objects referred to in ϕ, respectively (Line 6). Then it loops over the entire process model to detect the activity/event nodes which modify a relevant data object and the data objects which are modified by a relevant activity/event node. To this end, each iteration consists of two steps, which we describe in the following. The algorithm ends when A_{rel} and O_{rel} do not change any more, i. e., no new relevant prime elements have been detected (Line 17).

Step 1: The algorithm identifies new relevant data objects and adds them to O_{rel}. In particular, it checks all the detected relevant activity/event nodes and finds the data objects needed for their execution. The algorithm starts by checking every activity/event node n referred to in Property ϕ. In subsequent iterations, this step also takes the relevant activity/event nodes found in previous iterations of the algorithm into account, i. e., $n \in \{A_{rel} \cap A_{enh}\}$ (Line 7). The input/output data objects of n are relevant as well (Lines 8, 9) because n has a data-value function which may modify its input/output data objects. Further, the algorithm checks each ancestor m of Node n (Line 10). If m is an annotated XOR-node, the algorithm marks its input data object as relevant (Line 11). To be specific, the decision of gateway m might affect the verification result, since it determines whether n will be executed or not. Hence, the data object of this gateway is relevant, as the decision based on this data object affects the execution of the process model.

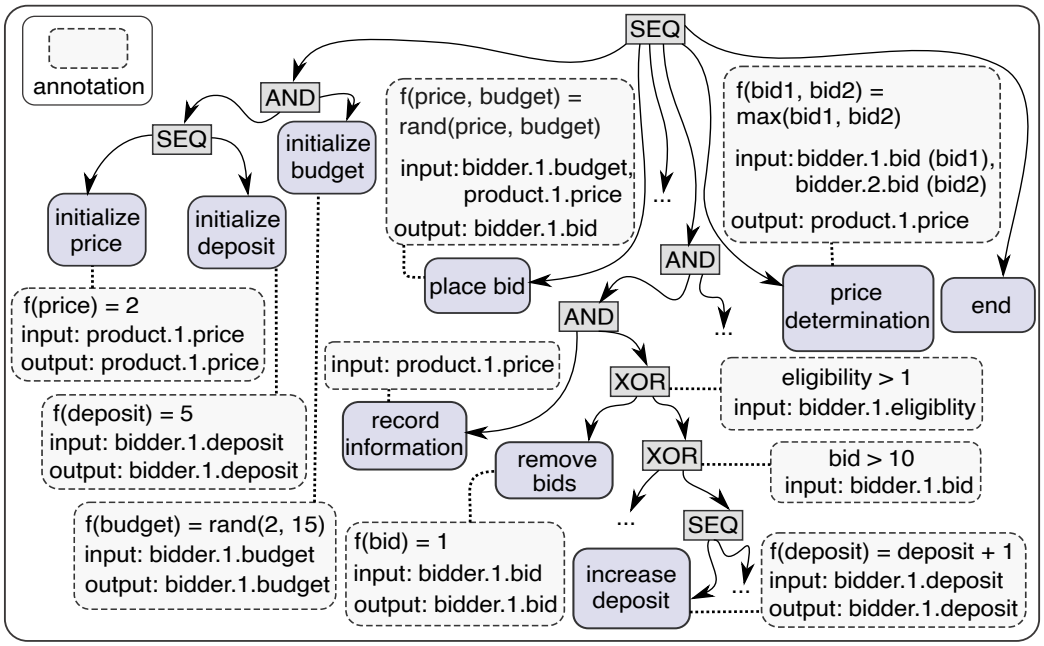


Figure 6.5: The `d_NPST` of the data-value-aware process model in Figure 6.4

Step 2: The algorithm identifies new relevant activity/event nodes and adds them to A_{rel}. To this end, it checks all the activity/event nodes in A_{enh}, finds those that modify a relevant data object and labels them as relevant (Lines 14, 15).

Lemma 1. *Let P be a process graph and ϕ a data property. If it holds that Activity/Event Node $n \notin A_{rel}$, then n is not relevant to verify ϕ.*

Proof. Let ϕ be a data property referring to Data Objects O_ϕ and Activity/Event Nodes A_ϕ, and let w be a witness for the statement that ϕ is true. We assume that P contains Activity/Event Nodes $\{n_k, n_{k-1}, n_{k-2}, ..., n_1\} \subseteq \{A \cup E\}$ and Data Objects $\{d_k, d_{k-1}, d_{k-2}, ..., d_1, d_0\} \subseteq O$, where:

- $d_k \in O_\phi$ is the output of Activity/Event Node n_k which needs d_{k-1} for its execution.
- d_{k-1} is the output of Activity/Event Node n_{k-1} which needs d_{k-2} for its execution etc..
- d_1 is the output of Activity/Event Node n_1 which needs d_0 for its execution.

We have to prove that an Activity/Event Node $n \notin A_{rel}$ does not influence the verification of ϕ. If n does not lie on w, then we can remove n from P, and w remains the same (by Definition 15). If n lies on w, then $n \notin A_{rel}$ implies that:

1. $n \notin \{n_k, n_{k-1}, ..., n_1\}$ (due to Line 15 in Algorithm 3). Then
   - n does not modify any data object in O_ϕ, and
   - n does not modify any data object needed for the activities which modify a data object in O_ϕ.

Algorithm 3 Detect relevant prime elements

Input: d_NPST P, Data Property ϕ
Output: A_{rel}, O_{rel}

1: $N = \{$activity/event nodes referred to in $\phi\}$
2: $O = \{$data objects referred to in $\phi\}$
3: $A_{enh} = \{$annotated activity/event nodes of $P\}$
4: $G_{enh} = \{$annotated XOR nodes of $P\}$.
5: **do**
6: $A_{rel} \leftarrow N$, $O_{rel} \leftarrow O$
7: **for** all nodes $n \in \{A_{rel} \cap A_{enh}\}$ **do** ▷ Step 1
8: $O.add(Input(n))$
9: $O.add(Output(n))$
10: **for** all nodes $m \in n.ancestors$
11: **if** $(m \in G_{enh})$ **then** $O.add(Input(m))$
12: **for** all data objects $o \in O_{rel}$ **do** ▷ Step 2
13: **for** all nodes $n \in A_{enh}$
14: **if** $(o \in \{Input(n) \cup Output(n)\})$ **then** $N.add(n)$
15: **while** $(A_{rel} \neq N) \vee (O_{rel} \neq O)$
16: **return** A_{rel}, O_{rel}

2. $n \notin A_\phi$.

Hence, we can remove n from P, and w remains the same, i. e., n does not influence verification of ϕ. □

Lemma 2. *Let P be a process graph and ϕ be the data property to be verified. If Data Object $d \notin O_{rel}$, then d is not relevant.*

Proof. Let N be the relevant control-flow nodes to verify ϕ, and let w be a witness for the statement that ϕ is true. We have to prove that a Data Object $d \notin O_{rel}$ does not influence the verification of ϕ. Suppose that $d \in \{Input(n) \cup Output(n)\}$.
If n is an XOR node in P, then n does not have any relevant child $m \in N$ (according to Algorithm 3). Then n cannot influence the verification of ϕ, and its input data object does not influence the verification either. If n is an activity/event node in P, then:

1. If n does not lie on w, then n does not influence the verification of ϕ (by Definition 16), and its input or output data objects do not influence it either.
2. If n lies on w and $n \notin N$, then n does not influence the verification of ϕ (by Definition 16), and its input or output data objects do not influence it either.
3. If n lies on w and $n \in N$, this contradicts the fact that n is not relevant. This is because $d \notin O_{rel}$ implies that $d \notin \{Input(n) \cup Output(n)\}$, where $n \in N$ (according to Algorithm 3).

Hence, we can remove d from P, and w remains the same, i. e., d does not influence verification of ϕ. □

Table 6.2: Iterations of Algorithm 3 in Example 22

	iteration #1	iteration #2	iteration #3	iteration #4
O_{rel}	{product1-price}	{product1-price, bidder1-bid, bidder2-bid, bidder1-budget}	{product1-price, bidder1-bid, bidder2-bid, bidder1-budget}	{product1-price, bidder1-bid, bidder2-bid, bidder1-budget, bidder1-eligibility}
A_{rel}	{end, price determination, initialize price, place bid}	{end, price determination, initialize price, place bid}	{end, price determination, initialize price, place bid, remove bid, initialize budget}	{end, price determination, initialize price, place bid, remove bid initialize budget}

Example 22. *Table 6.2 illustrates the iterations of Algorithm 3 for the* `d_NPST` *in Figure 6.5 and the data property in Example 40, i. e., "Can product*1 *have a price of* 10 *at the end of the auction?". The algorithm initializes O_{rel} and A_{rel} with the Data Object "product1-price" and Event "end", respectively. In the first iteration, the set of relevant data objects remains unchanged, because Event end $\notin A_{enh}$. However, the algorithm updates A_{rel} with activities which read or write Data Object "product1-price", e. g., Activities "initialize price" and "place bid". In the last iteration, Data Object "bidder1-eligibility" is detected as relevant, because the XOR ancestor of Activity "remove bid" reads this data object. Note that Activity "record information" does not belong to A_{enh}, i.e., does not modify the value of any data object. As such, it does not affect the price of product*1 *and is therefore not relevant.*

Algorithm 3 outputs the relevant prime elements. In the following, we propose the relevance functions to detect the relevant elements of all element types, using the output of Algorithm 3.

Definition 21. *(Relevance Function for Control-Flow Nodes) Let P be a process graph in the form of a* `d_NPST`*, n a node of its* `d_NPST` *and ϕ a data property. Then, $relevant(\phi, n)$ is as follows:*

$$relevant(\phi, n) = \begin{cases} true & n \in A_{rel} \\ true & n \notin A_{enh} \wedge \exists n' \in n.ancestors \mid n' \in A_{rel} \\ false & otherwise \end{cases}$$

Lemma 3. *The irrelevant nodes according to the relevance function for control-flow nodes do not influence the verification of ϕ.*

Proof. We have to prove that $relevant(\phi, n) = False$ implies that n cannot influence the verification of ϕ. Let w and n be as in Definition 15, and suppose that n lies on

w. If n is an activity/event node, then the lemma holds because of Lemma 1. If n is a gateway and $n \in XG_P \cup AG_P$ then n cannot influence the verification of ϕ. This is because n does not have a relevant child, i. e., its execution does not influence the execution of any relevant control-flow node. □

Definition 22. *(Relevance Function for Data Elements) Let P be a process graph in the form of a* `d_NPST`*, d be a data element of its* `d_NPST` *and ϕ be a data property. Then, $relevant(\phi, d)$ is as follows:*

$$relevant(\phi, d) = \begin{cases} true & d \in O_{rel} \\ true & type(o) = d \wedge o \in O_{rel} \\ false & otherwise \end{cases}$$

Lemma 4. *The irrelevant elements according to the relevance function for data elements cannot influence the verification of ϕ.*

Proof. We have to prove that $relevant(\phi, d) = False$ implies that d cannot influence the verification of ϕ. If d is a data object, then this holds because of Lemma 2. If d is a data-object type, then there does not exist any Data Object o of type d which influences ϕ. Thus, d cannot influence the verification of ϕ either. □

6.3.3 `d_NPST` reduction

Our procedure to remove the control-flow nodes of a `d_NPST` is different from that for data elements. So one cannot readily combine these procedures to reduce a `d_NPST`. To illustrate, when a data element is irrelevant, one can simply remove it from the `d_NPST` wherever it is used in the control flow of the `d_NPST`. However, in case a control-flow node, e. g., an activity node, is irrelevant, one cannot simply remove it, i. e., pruning an irrelevant activity node may lead to a reduced process model that does not preserve the verification result. We will explain this in Section 6.3.4.
In the following, we reduce the control flow of a `d_NPST` using Algorithms 4 and 5, borrowed from [63]. Then we propose Algorithm 6 to reduce the data elements of a `d_NPST`, using Definition 22.

6.3.3.1 Control-Flow Reduction

Algorithms 4 and 5 reduce the control flow of a `d_NPST`, using our relevance function from Definition 21. Algorithm 4 recursively checks whether each node of the tree is relevant (Line 1) using our function in Definition 21. If the function yields true, the algorithm keeps the node. Otherwise, it deletes the node unless the type of the parent of the node is XOR (Line 6). In this case, the node is replaced by λ, an empty node (Line 7). We will explain this in Section 6.3.4. Algorithm 5 trims the pruned `d_NPST`. In case an inner node has only one child, it replaces the node with its child (Lines 2-4).

Algorithm 4 pruneNode(node, ϕ)

```
1: if relevant(φ, node) then
2:     for all c ∈ node.children
3:       pruneNode(c, φ)
4: else
5:     if type of node.parent is XOR then
6:         replace node with λ
7:     else
8:         delete node
```

Algorithm 5 trimTree(node, ϕ)

```
1: if |node.children| == 1 then
2:     c ← first(node.children)
3:     connect c to node.parent
4:     delete node
5:     trimTree(c, φ)
6: else
7:     for all c ∈ node.children
8:       trimTree(c, φ)
```

6.3.3.2 Data Reduction

We propose Algorithm 6 to reduce the data elements of a `d_NPST`. It loops over all data elements including data objects and data-object types and removes the data elements that are not relevant to verify ϕ, using Definition 22.

Algorithm 6 removeData(Data element d, ϕ)

```
1: for each Data-Object Type d in d_NPST do
2:     if relevant(φ, d) then
3:         for each Data Object o of d do
4:             if (!relevant(φ, o)) then remove o
5:     else
6:         remove d
```

Example 23. *Figure 6.6 shows the reduced* `d_NPST` *of Example 21. Table 6.3 lists its data elements. The* `d_NPST` *in Example 21 consists of 17 nodes. So Algorithm 4 requires 17 recursive invocations to check the relevance of each of its elements. To this end, it performs a depth-first search starting from the root, i. e., the SEQ node. If the algorithm finds an activity node that is irrelevant and does not have an XOR ancestor, e. g., "initialize deposit" and "record information", it removes the node. The algorithm also replaces "increase deposit", an irrelevant activity with an XOR ancestor, with* λ.

6.3.4 Transformation of `d_NPST` to Data-Value-Aware Process Model

To verify the `d_NPST`, we transform it back to a process graph according to the rules in [62]. This transformation is straightforward except for the handling of XOR-

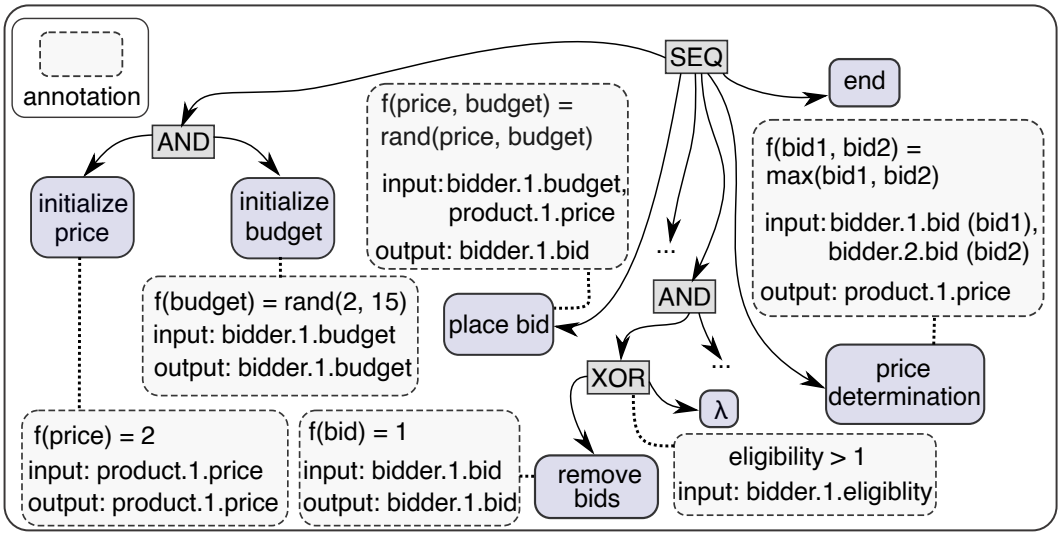


Figure 6.6: Reduced `d_NPST` in Example 21

Table 6.3: Data elements of the Reduced `d_NPST`

data-object type	data object	attribute	domain
bidder	bidder1	budget	[0..10]
	bidder1	bid	[0..10]
	bidder1	eligibility	[0..3]
	bidder2	bid	[0..10]
product	product1	price	[0..10]
	product1	winner	[0..2]

nodes. To preserve the verification result, λ, the empty node for the XOR-node must remain.

Example 24. *Consider the formula "After an occurrence of initialize price there always is an occurrence of remove bids at some time in the future". In CTL this is AG(initialize price → AF(remove bids)). This evaluates to false for the process in Figure 6.6. But it would be true if we had removed the other child of the XOR-node. 11This is because after the XOR-node, there would be only one path to continue, and it includes "remove bids". So the formula evaluates to true.*

Let n be an XOR-node in the reduced `d_NPST` with two child nodes n_1 and n_2. We discern between three cases to transform the `d_NPST` to a data-value-aware process model: If $n_1 \neq \lambda$ and $n_2 \neq \lambda$, then the transformation is straightforward. If $n_1 \neq \lambda$ and $n_2 = \lambda$, then the second branch can be reduced to a simple activity with no data usage. If $n_1 = n_2 = \lambda$, then the complete XOR-node is deleted.

Lemma 5. *Given a Data Property ϕ, s.t. ϕ does not contain the next operator X, and a Process Graph P, $P \models \phi$ iff $P' \models \phi$, where P' is the reduced process model.*

Proof. All data properties can be written in terms of $EG\phi$, $E(\phi_1 U \phi_2)$, $EF\phi$, and $EX\phi$ [64]. Let S be the set of states and $EG(\phi)$ the data property, i.e., there exists a path $p = (s_1, s_2, ..., s_n)$ where ϕ is always true. As a result of our reduction, the paths are limited to the ones leading to states that are able to influence ϕ, called S_ϕ, $p' = (s'_1, s'_2, ..., s'_n)$, $s'_i \in S_\phi \subseteq S$. If a path p where ϕ is always true exists in the original Petri Net, a path p' exists in the reduced Petri Net, too. If no path p exists in the original Petri Net, there cannot exist a path p' in the reduced Petri Net. □

We analogously prove the correctness of our approach for $E[\phi_1 \cup \phi_2]$ and $EF\phi$.

Lemma 6. *Let P be a process graph and $\phi : EF(\phi)$ be a data property to be verified. Then, $P \models \phi$ iff $P' \models \phi$, where P' is the reduced process model.*

Proof. Let S be the set of states of P. If $P \models \phi$, then, there exists a path $p = (s_1, s_2, ..., s_n)$ where ϕ is eventually true. As a result of our reduction, the paths are limited to the ones leading to states that are able to influence ϕ, called S_ϕ, $p' = (s'_1, s'_2, ..., s'_n)$, $s'_i \in S_\phi \subseteq S$. If a path p where ϕ is at some point true exists in the original Petri Net, a path p' exists in the reduced Petri Net, too. If $P \not\models \phi$, then no path p exists in the original Petri Net, there cannot exist a path p' in the reduced Petri Net. □

Lemma 7. *Let P be a process graph and $\phi : E[\phi_1 \cup \phi_2]$ be a data property to be verified. Then, $P \models \phi$ iff $P' \models \phi$, where P' is the reduced process model.*

Proof. Let S be the set of states of P. If $P \models \phi$, then, there exists a path $p = (s_1, s_2, ..., s_n)$ where ϕ_1 is true until ϕ_2 is true. As a result of our reduction, the paths are limited to the ones leading to states that are able to influence ϕ, called S_ϕ, $p' = (s'_1, s'_2, ..., s'_n)$, $s'_i \in S_\phi \subseteq S$. If a path p where ϕ_1 is true until ϕ_2 is true exists in the original Petri Net, a path p' exists in the reduced Petri Net, too. If $P \not\models \phi$, then no path p exists in the original Petri Net, there cannot exist a path p' in the reduced Petri Net. □

We now explain why our approach cannot handle the Next operator X. Our proposed reduction technique does not change the sequence of the activities, but it may change the distance of activities in the traces of the executions of a process schema. For example, trace $a \rightarrow b \rightarrow c$ in the original process P can be reduced to trace $a \rightarrow c$ by pruning b. The distance between a and c changes from one to zero. The CTL formula $AG(a \rightarrow EXc)$ evaluates to $false$ for the original trace, but not for the reduced trace. Thus, our specification language allows to refer to the order of tasks, but not to their exact distance. On the other side, a respective feature might be less practical in our context than it might appear at first sight: Namely, the state distance may not be intuitive here. This is because it depends on the specific transformation of the model to the Petri Net how many states are between the tasks. A solution would be extending the notion of explicit time to tasks, see [72]. This would require a different formal model (e.g., timed Petri Nets) and a different specification language (e.g., RTCTL). We leave aside the Next Operator (X) of CTL to refer to the distance of states in properties.

6.4 Evaluation

In this section, we show that our reduction technique enables the verification of properties of spectrum auctions of realistic size, i. e., the German 4G spectrum auction to sell 800 MHz band [24] in 2017. Our approach allows to detect the worst possible values of three important auction measures [15]: auctioneer's revenue, bidders' profit, and auction efficiency. We demonstrate the impact of our reduction approach on the size of process models and of state spaces and on the runtimes of the model checker when verifying the properties needed to calculate the measures.

6.4.1 Evaluation Setting

Our evaluation focuses on four settings, i. e., auction process models with different numbers of bidders and products:

1. *Process 1*: Two bidders compete to win two products.
2. *Process 2*: Two bidders compete to win three products.
3. *Process 3*: Three bidders compete to win two products.
4. *Process 4*: Four bidders compete to win six products – the size of the German 4G spectrum auction to sell the 800 MHz band [24].

The German 4G spectrum auction also handles other bandwidths, which are not subject to the capacity rule, i. e., a bidder can win as many products (blocks) of such bandwidths as he or she can afford. In order to find the auction measures of bandwidths without the capacity rule, one can assign the products to the bidders with the highest budgets, i. e., no verification tool is needed. Hence, we evaluate our approach by verifying the auction to sell only the 800 MHz band, subject to the capacity rule. Nevertheless, as we will show in Section 6.4.3, the number of products has a minor effect on the size of the process model and the one of the state space after reduction. Put differently, our approach can also verify auctions with more products.

We have modeled all auction process models with information on how they use data values. We have added a certain number of BPMN elements when increasing the number of bidders or of products. This is necessary to specify the respective multi-instance subprocesses as process graphs. For example, to model the Subprocess *bidding of each bidder* in a process graph without multi-instance subprocesses, one must add the same subprocess for each bidder, separately from the subprocesses for the other bidders. This leads to more BPMN elements of the model, by a constant factor. To illustrate, adding a new bidder increases the number of activities and gateways of the BPMN model by 13 and 20, respectively.
We examine the size of the auction processes modeled in BPMN notation in Section 6.4.3. In all process models, we have assigned a random budget to each bidder for a certain product in the range of $[2-10]$, similarly to the auction experiments in [15]. We also have defined a reserve price of 1 for all products, so all bidders can afford all products at the beginning of the auction. Two bidders have a *capacity point* of 2, the others have a *capacity point* of 3, according to the 4G German spectrum auction [24].

6.4.2 Data Properties Required for Auction Measures

We detect the lowest values of three relevant measures of all auction process models. To do this, we have to verify the lowest final price of each product, i. e., to check whether the *price* of a *product* can take a certain value when the process ends. Observe that, due to the capacity rule, the order of products in which one verifies the lowest final prices matters.

Example 25. *In order to detect the lowest revenue of Process 3, we verify the lowest final prices of "product1", "product2", and "product3", respectively. However, "product1" might have a lower final price if we verify its final price last, i. e., a bidder with a lower budget might win this product. This is because the bidders with higher budgets might have already won the other products and have no capacity points left for "product1".*

Thus, we have to check all the possible combinations of products in all our auction processes in our evaluation. In Process 4, for example, we have calculated the *auctioneer's revenue* in $6! = 720$ combinations. In each combination, first we verify the lowest final price of each product, starting with the reserve prices. To illustrate, the following property checks whether *product*1 can be sold for the *price* of 2 at the end of the process.

$$\text{Property (I): } EF(product1\_price\_2 \wedge e\_end).$$

If the property is not satisfied, we verify a new one with an increased price, until there is a state that fulfills the property. In each combination, the sum of the lowest prices found for each product is the *auctioneer's revenue*. The lowest revenue among all combinations is the lowest *auctioneer's revenue* (R_{lowest}).

Next, we detect the profit of the winners, which is the winner's budget minus the final price of the product. Since we are interested in finding the auction measures in the worst possible scenario, we detect the profit of the winners when the final prices

of products are minimal, i. e., P_{lowest}. To calculate the minimum profit P_{lowest}, first we check whether the bidder with the lowest budget can be the winner. Suppose that $bidder1$ has the lowest budget for $product1$, and the lowest final price of $product1$ is 4. Then, the formula to check is:

$$\text{Property (II): } EF(p.product1\_price\_4 \wedge product1\_winner\_1 \wedge e\_end).$$

If the model checker cannot find a state, i. e., an execution path fulfilling the property, we change the winner in the formula to the one with the next higher budget. This results in new formulas to be checked. This is done until the formula is satisfied. 11The formula will be satisfied eventually, since the budgets of bidders are higher than the reserve prices of products, and bidders do not have fewer capacity points in total than the number of products.

One can also find the lowest market *efficiency* (E_{lowest}) while the auctioneer's revenue is minimal. To quantify efficiency, we use a measure from [15]. It quantifies how the surplus resulting from the worst allocation (S_{lowest}) can deviate from the surplus with a random allocation S_{random}:

$$E_{lowest} = \frac{S_{lowest} - S_{random}}{S_{opt} - S_{random}} \times 100\ percent$$

Here, the surplus resulting from an optimal allocation (S_{opt}) is computed by giving the products to bidders with the highest budget. S_{lowest} is the sum of the lowest revenue and profit.

6.4.3 Reduction of Process Models

Given the data properties, we reduce the process models accordingly. We have two categories of properties needed to detect measures of SMR auctions: (a) properties to check the lowest price of products, and (b) properties to check the winner of products. Interestingly, with our reduction technique, we have observed that both categories result in the same reduced process models. Properties (I) and (II) in Section 6.4.2, for example, lead to the same relevant elements and, thus, the same reduced process models. However, the property to verify the lowest price of Product p_1 results in a set of relevant elements different from that of Product p_2. With n products, this calls for n different reductions of the process model. In Process 4, for example, we reduce the process 6 times, each time for one product. In what follows, we focus on the average number of elements among n reductions when analyzing the reduced process models.

Table 6.4 lists the number of control-flow nodes and data elements of process models with and without reduction. Process 4 is the largest process model among the four processes introduced in Section 6.4.1. As expected, the number of control-flow nodes and data elements in all processes decreases with reduction. The reduction of Process 4 is most significant. The number of activities and gateways, for example, dropped from 172 and 251 to 42 and 60, respectively. Regarding the data elements, the number of data objects and values also decreased considerably, from 150 and 444 to 26 and 120, respectively. The size of Process 4 after reduction is almost 23% of the original one.

Table 6.4: The number of control-flow nodes and data elements with and without reduction

process		no.activities & events	no.gateways	no.annotations	no.data references	no.data-object types	no. data objects	data values
Process1 (2/2)	original	42	59	123	74	4	20	98
	reduced	28	36	78	51	4	14	66
Process2 (2/3)	original	56	81	169	97	4	27	138
	reduced	28	36	78	51	4	14	66
Process3 (3/2)	original	55	79	162	92	4	27	139
	reduced	35	49	98	61	4	20	93
Process4 (4/6)	original	172	251	529	266	4	150	444
	reduced	42	60	119	71	4	26	120

6.4.4 Reduction Effects on Petri Nets

Given the reduced process models, we have transformed them to Petri Nets. These Petri Nets have been verified against properties specified in Section 6.4.2. To do this, we have used the state-space generation and model checking algorithm of the LoLA framework [61].

Table 6.5 lists the sizes of the Petri Nets created and of the state spaces and the runtimes of LoLA to generate the state spaces. All processes have been shrunk to a Petri Net smaller than the original one. The biggest process has the largest relative reduction. In Process 4, the size of the Petri Net after reduction is almost 19% of the original size. The reduction of the state spaces is even larger than the one of the Petri Nets. This is a significant improvement since it has not been possible so far to construct the state space of Process 4. With our reduction in turn, this has taken place in reasonable time, namely 10 seconds. Note that the state space does not need to be entirely constructed to verify the properties formalized in Section 6.4.2. We have just shown how the state space has been pruned using our reduction technique.

6.4.5 Verification of Properties Required for Auction Measures

In Table 6.6, we present the runtime of the model checker to verify the properties described in Section 6.4.3, using a computer with 16 GB main memory and 2 Intel 3 GHz CPUs. See Table 6.6. Note that, for the reduced process models, we have a larger number of verification runs than with the original ones. Particularly, in processes without reduction, one can verify the prices of all products with one property. This is not possible for the reduced process models. So we had to reduce the process model for each product separately in order to verify its price. This requires verifying more properties, resulting from different orders of products to be verified, as explained in Section 6.4.2. However, the average verification time of all reduced process models is much lower than that of the original model, as well as it is for the

Table 6.5: The size of Petri Nets with and without reduction

process		no.places	no.transitions	no.states	time(sec)
Process1 (2/2)	original	609	637	> 2M	5
	reduced	338	356	> 54K	< 1
Process2 (2/3)	original	881	918	> 54M	600
	reduced	339	358	> 51K	< 1
Process3 (3/2)	original	822	854	> 58M	790
	reduced	456	473	> 442K	< 1
Process4 (4/6)	original	2,714	2,532	not possible	not possible
	reduced	562	568	> 3M	10

verification time of all reduced process models in total. In Process 4, the average verification time is less than 1 second, while the verification has not been possible before, due to state-space explosion. Table 6.6 shows how our reduction approach decreases the verification time for the other three process models as well.

Table 6.6: Verification Time

process		no.verification runs	avg.time	max.time (sec)
Process1 (2/2)	original	15	< 1	< 1
	reduced	26	< 1	< 1
Process2 (2/3)	original	20	30	30
	reduced	108	< 1	< 1
Process3 (3/2)	original	24	15	15
	reduced	26	< 1	< 1
Process4 (4/6)	original	not possible	not possible	not possible
	reduced	29,201	< 1	< 15

6.4.6 Verification Results

Table 6.7 lists the results of our verification runs for each process model. Our verification reveals which assignment of products to bidders yields the lowest revenue of the auctioneer. In Process 1 for example, this revenue is minimal when assigning $product1$ to $bidder1$ and $product2$ to $bidder2$ for the price of 8 and 4, respectively. In this case, the lowest bidder profit is 5, i. e., bidders would have had 5 more currency

units to pay. Adding another product in Process 2 decreases the efficiency of the auction to 83.3%. This might be because *bidder*2 could win the same product as in Process 1, but at a lower price, 3 units. In Process 4, the lowest revenue and efficiency are 32 and 91.6%, respectively. This means that winners are different from the bidders with the highest budgets.
These observations are interesting: The conventional SMR design which has been used for more than two decades to allocate spectrum licenses has inefficiencies, and our approach can reveal under which circumstances these inefficiencies may happen. An auction designer can now take this design as a starting point for improvement, by applying our method to new designs as well and comparing the results.

Table 6.7: Verification results

process	product	winner	price	R_{lowest}	P_{lowest}	E_{lowest}
Process1 (2/2)	product1	bidder1	8	12	5	100%
	product2	bidder2	4			
Process2 (2/3)	product1	bidder1	8	16	4	83.3%
	product2	bidder2	3			
	product3	bidder1	5			
Process3 (3/2)	product1	bidder1	8	12	5	100%
	product2	bidder2	4			
Process4 (4/6)	product1	bidder1	8	32	12	91.6%
	product2	bidder2	3			
	product3	bidder3	5			
	product4	bidder3	6			
	product5	bidder1	6			
	product6	bidder3	4			

6.5 Summery

Data values and their modifications play a major role in many business processes. Verification of such processes must take them into account. Verifying process models with data values, i. e., *data-value-aware process models*, often leads to state-space explosion. In this article, we have proposed a new property-specific reduction approach. It facilitates verification of data-value-aware process models checking the relevance of control-flow nodes and data elements of a data-aware process model for the verification of a given possibly data-aware property. A distinctive feature of our approach is that it detects the relevant elements with respect to the modification of data values. Given these relevant elements, our approach prunes the process model significantly so that verification becomes possible in much more cases and for larger process models than without reduction. We have evaluated our approach by verifying an important real-world application of realistic size, the 4G German

spectrum auction to sell the 800 MHz band. Our reduction facilitates, for the first time, the verification of these auctions. One can now detect the extreme values of three important measures for auctions: auctioneer's revenue, bidder's profit, and the efficiency of the auction.

Our reduction technique may output a data object with a large domain as relevant, depending on the property of interest. In such cases, verification may still lead to state-space explosion due to the large number of values the relevant data object can take during process execution. We plan to continue our work on reducing the state space of data-value-aware process models with large data domains. To this end, we are investigating how to apply abstraction methods for data domains when modification of data values takes place. The abstraction of data domains together with our proposed reduction technique will target to enable verification of data-value-aware process models when process elements modify data objects with large domains.

7. Dealing with Data-Value Conditions with Large Domains

Not only large process models but also large domains can cause a state-space explosion. In this chapter, we provide a novel technique to reduce the state space of process models in which data conditions with large domains exist.

7.1 Introduction

Legal and organizational regulations and policies that processes must fulfill are numerous. Verification techniques analyze whether the process models meet such requirements. This chapter focuses on requirements in the form of data conditions. To illustrate, think of conditional branching gateways in process models. A branching gateway takes a decision based on a data condition and determines which branch of the process follows, i. e., the data condition influences the execution of the process model. A verification technique must be able to deal with data conditions. For verification, these conditions with their execution semantics must be represented in the state space of the process models.

Example 26. *Figure 7.1 shows an order-to-delivery Process Model P in BPMN notation. Consider property c: "Products are not shipped for free if the bonus is less than 50 or the price is less than 200,000." To verify c, not only control-flow but data conditions such as "bonus* < 50*" and "price* $< 200,000$*" have to be represented in the state space of the process model. Otherwise, one cannot detect a violation of c, since Activity "offer free shipping" could be executed in any case.*

Although handling data conditions is essential for verification, literature does not adequately address this. The scientific literature features two alternatives to do so, namely (a) to represent values not used in data conditions as one abstracted value, or (b) to explicitly represent each value in the respective domain. Existing approaches in both categories have issues: The idea behind the first category is to determine the values of data objects needed for verification, so-called *relevant values*. The problem is that such techniques might yield an incorrect result when process elements modify values used in data conditions.

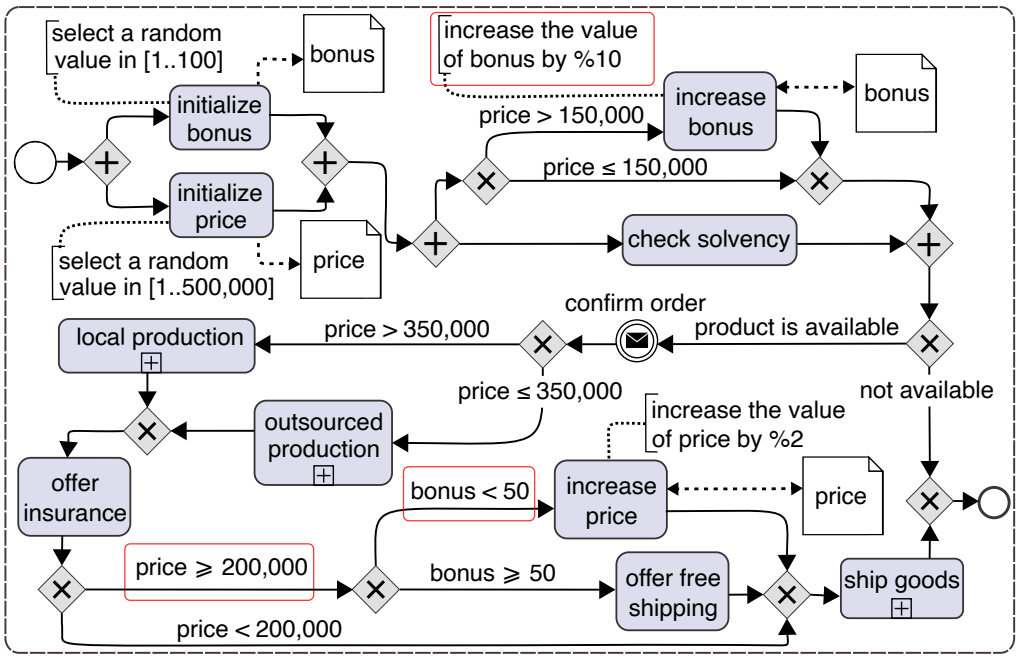


Figure 7.1: An order-to-delivery process model

Example 27. *Existing abstraction techniques argue that, to verify Property c of Example 26, the value of bonus is not relevant if "bonus >= 50" when executing the process model [42]. Thus, they treat all values of bonus equal to or higher than 50 as one "abstracted" value. However, Activity "increase bonus" may modify the value of "bonus". Values less than 50 may fall into the interval ">= 50" afterwards, and this changes the process execution and may lead to an incorrect diagnosis. It is essential to support the entire set of values that bonus can take.*

Approaches in Category (b) exist as well [6, 28, 32]. Data conditions with large domains (such as integers) can quickly cause state-space explosion. For example, approaches based on Petri Nets create n new transitions to map a Data Condition *con* using a Data Object d with domain $[0..n-1]$. The firing of each such transition leads to a new state in the Petri Net. The state space of the process model with a data condition on d is n times the one without.

Example 28. *Consider Process Model P from Figure 7.1. Suppose that the domain of price ranges from 1 to 500,000. Existing techniques create* $500,000$ *new transitions for each data condition of P in which price is used, e. g., "price* $< 200,000$*".*

This chapter takes up the challenge to prevent state-space explosion in data-value-aware process models with large domains. It reduces the state space of the process model with Data Condition *con* from $n \cdot |s|$ states to $\log_2 n \cdot |s|$, where $|s|$ is the size of the state space without data conditions. A distinctive feature of the state-space-reduction approach presented here is to support the entire domain of each data object, i.e., without abstracting from individual values.
Our approach is based on Petri Nets and starts by transforming the control flow of the data-value-aware process model into Petri Nets using the rules in [60]. Then it extends this transformation with the semantics of data conditions in two steps, which are the core contributions of this chapter. The first step is a *binary encoding*,

completely mapping Data Object d with the domain of $[0..n-1]$ to at most $2 \cdot (\lfloor \log_2 n \rfloor + 1)$ new places in the Petri Net. This is in contrast to approaches which represent each value with a separate place or token. However, the reduction of places alone does not yet reduce the state space in data-value-aware process models. Thus, as a second step, we reduce the number of transitions needed to represent the data conditions. To do so, we propose a novel algorithm based on the theory of Boolean functions [9]. This algorithm maps Data Condition *con* in a Petri Net using at most $\lfloor \log_2 n \rfloor + 1$ new transitions. This means that one can evaluate *con* in the state space by exploring $\lfloor \log_2 n \rfloor + 1$ instead of n states; with n places for d used in the condition, n is the number with existing approaches.

We evaluate our approach with a real-world process, the German 4G spectrum auction. It has sold one of the most valuable bandwidths, the 800 MHz band [24]. Our approach reduces the size of the Petri Net by 95.5% compared to the unreduced one. Given the reduced Petri Net, we are able to generate the state space of this auction model with real domains which has been infeasible so far. The reduced state space then allows to verify important properties of such auctions like the lowest possible revenue.

7.2 Preliminaries

In the following, we introduce the notation behind the theory of Boolean functions, used for binary encoding of large data domains.

Definition 23 (Boolean Function [9])**.** *A Boolean Function of n variables is a function of type $B^n \to B$, where B is the set $\{true, false\}$, n is a positive integer, and B^n is the n-fold Cartesian product of B with itself. A point $X^* = (x_1, x_2, ..., x_n) \in B^n$ is a true point (false point) of the Boolean function f if $f(X^*) = 1$ ($f(X^*) = 0$).*

Definition 24 (Literal, Elementary Conjunction [9])**.** *A literal is an expression of the form x or $\overline{x}$, where x is a Boolean variable and $\overline{x}$ its negation. An elementary conjunction is an expression of the form $C = \bigwedge_{i \in A} x_i \bigwedge_{j \in B} \overline{x}_j$ where A, B are disjoint subsets of indices, i. e., $A \cap B = \varnothing$.*

Definition 25 (Implicant, Prime Implicant [9])**.** *Let f be a boolean function and C be an elementary conjunction. C is an implicant of f if it implies f. A prime implicant of f is an implicant that is not included in any other implicant of the function, i.e., no other implicant of f absorbs the prime implicant.*

Example 29. *Let $f = xy \vee x\overline{y}z$. Then xy, $x\overline{y}z$, and xz are implicants of f. xy and xz are prime implicants of f. $x\overline{y}z$ in turn is not prime since xz absorbs $x\overline{y}z$, i. e., xz covers $x\overline{y}z$, but not vice versa.*

7.3 Our approach

This section proposes an efficient verification scheme for data-value-aware process models using data objects with large domains. Figure 7.2 gives an overview of our approach. The colored boxes stand for the contributions of this chapter.

1. We start to map the control flow of a process model to Petri Nets (Activity *map control flow*).

2. We map each data object to a set of places. Any distribution of tokens in these places represents a data value (Activity *map data objects*).

3. We use these places and map each data condition to a Petri Net. Each of these mappings represents the semantics of the condition (Activity *map data conditions*).

4. We merge the "control flow" Petri Net obtained from Step 1 with the results of Step 3 (Activity *merge*). The result is a Petri Net that includes the semantics of data conditions and the complete domain of the data objects.

5. Given the final Petri Net, an off-the-shelf model checker then allows verifying properties of the data-value-aware process model (Activity *model checking*).

We use plain Petri Nets as a target for the mapping because of the availability of efficient analysis techniques [59].

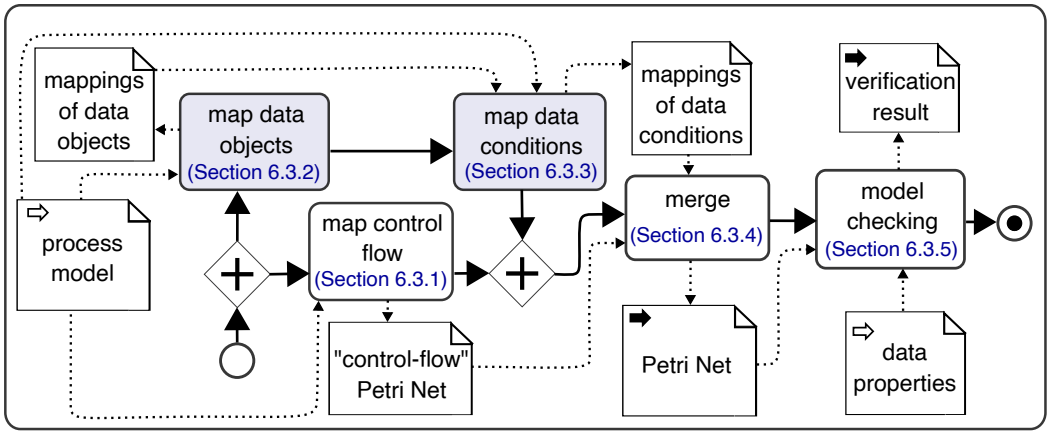


Figure 7.2: Overview of our approach

7.3.1 Mapping control flows to Petri Nets

We use plain Petri Nets as a target for the mapping because of the availability of efficient analysis techniques [59]. To map the control flow of a process model to Petri Nets, we use the rules in [60]. However, the mapping introduced in [60] only covers the "control flow". Thus, our approach extends this mapping with the semantics of data-value conditions.

7.3.2 Mapping data objects to Petri Nets

In any state, a data object takes one value from its domain. So one can represent different values of a data object using the same places in a Petri Net. To facilitate this, we map a data object to a set of places. These places receive a token according to the *binary encoding* of the respective value, i. e., its binary representation. In the following, we first describe how to map a single bit into a Petri Net. Then we explain the mapping of other values.

Mapping of a bit b_i. We represent a single bit b_i in a Petri Net using two places $p.\overline{b}_i$ and $p.b_i$. Place $p.b_i$ ($p.\overline{b}_i$) takes a token when $b_i = 1$ ($b_i = 0$). We call the places to represent bit b_i, *b_i-places*.

Mapping of Data Object d. Suppose that Data Object d has domain $[0..n-1]$, i. e., n values. We generate $\lfloor \log_2 n \rfloor + 1$ times b_i-*places* where $0 \leqslant i \leqslant \lfloor \log_2 n \rfloor$. The distribution of tokens in b_i-*places* represents the value of d in a certain state.

Example 30. *Consider Data Object price with domain* $[0..500,000]$. *We create* $\lfloor \log_2 500,000 \rfloor + 1 = 19$ b_i-*places, where* $0 \leqslant i < 19$, *as follows. The highlighted places take a token iff price* $= 262,144 = 2^{18}$.

$\{\ p.\bar{b}_{18},$	$\boldsymbol{p.\bar{b}_{17}},$	$\boldsymbol{p.\bar{b}_{16}},$	$\boldsymbol{...},$	$\boldsymbol{p.\bar{b}_{3}},$	$\boldsymbol{p.\bar{b}_{2}},$	$\boldsymbol{p.\bar{b}_{1}},$	$\boldsymbol{p.\bar{b}_{0}},$
$\boldsymbol{p.b_{18}},$	$p.b_{17},$	$p.b_{16},$	$...,$	$p.b_{3},$	$p.b_{2},$	$p.b_{1},$	$p.b_{0}\}$

7.3.3 Mapping data-value conditions to Petri Nets

To verify a data-value-aware process model, it is essential to include the semantics of data-value conditions in the Petri Net, i. e., capture it in the state space of the process model. Consider Data-Value Condition $con : (d\ \Theta\ c)$ where d has domain $[0..n-1]$. To map con to a Petri Net, one possibility is to create a new transition for each value in the domain of d, i. e., n new transitions [28, 57]. But our intention is to reduce the number of transitions required to map a data-value condition in order to reduce the state space.
Data-value conditions to Boolean functions. The idea behind our mapping is to transform Data-Value Condition $con : (d\ \Theta\ c)$ to a Boolean function as follows:

$$f_{(d\ \Theta\ c)}(b_0, b_1, ...b_m) = \{v_1, v_2, ..., v_{n'-1}, v_{n'}\}, \quad n' \leqslant n,\ m = \lfloor \log_2 n \rfloor$$

where v is a value in the domain of d, and n' is the number of values of d for which Data-Value Condition con is *True*.

Example 31. *The Boolean function of Data-Value Condition* $bonus < 50$, *where bonus has a domain of* $[0..100]$ *is:* $f_{(bonus<50)}(b_0, b_1, ..., b_5) = \{0, 1, 2, ..., 49\}$.

The number of Prime Implicants (PI) evaluating Boolean function $f_{(d\ \Theta\ c)}$ is equal to the number of transitions required to map Data-Value Condition $con : (d\ \Theta\ c)$. This is because one can evaluate a prime implicant using only one transition in the Petri Net. To do this, one can connect the literals of the prime implicant, which are in the form of places, to a single transition. The transition fires when all its incoming places are marked, i. e., when all the literals of the prime implicant are true. Note that we have already mapped all literals to places using our binary encoding, see Section 7.3.2. In Lemma 8, we prove that Boolean function $f_{(d\ \Theta\ c)}$ requires at most $\lfloor \log_2 n \rfloor + 1$ prime implicants. This implies that the mapping of Data-Value Condition $con : (d\ \Theta\ c)$ needs at most $\lfloor \log_2 n \rfloor + 1$ new transitions.

Lemma 8. *Let a Boolean function* $f(b_0, b_1, ...b_m) > c$ *when* $m \geqslant 1$ *be given. Then f requires at most* $m + 1$ *prime implicants.*

Proof. We prove this lemma by induction on the number of variables and that every Boolean function with $m + 1$ variables requires less than $m + 1$ prime implicants.
Base case: The statement holds for $m = 1$, i. e., Boolean function f with 2 variables requires $m + 1 = 2$ prime implicants at most, because: $f(b_0, b_1) > 0$ requires 2

prime implicants: b_0 and b_1. $f(b_0, b_1) > 1$ requires a single prime implicant: b_1, and $f(b_0, b_1) > 2$ requires a single prime implicant: $b_0 b_1$.
Induction hypothesis: Assuming that the statement holds for all functions of up to m variables, i. e., Boolean function f with m variables requires m prime implicants at most.
Induction step: By the induction hypothesis, we prove that Boolean function f with $m + 1$ variables requires $m + 1$ prime implicants at most. First, we consider any function $f(b_0, b_1, ...b_{m-1}, b_m) > c$ with $c \geqslant 1$. To cover this function, we can use a single prime implicant b_m to cover all values from 2^{m-1} to $2^m - 1$. Now, we define relative to f, the function $g(b_0, b_1, ...b_{m-1}) = f(b_0, b_1, ...b_{m-1}, 0)$. The function g has m variables, and so by inductive hypothesis we need less than m prime implicants to cover it. This means that those prime implicants cover the values $c + 1$ through $2^{(m-1)} - 1$ for the f function. Thus, we can cover the original f using those less than m prime implicants, plus the single prime implicant b_m. This means that we can cover f with less than $m + 1$ prime implicants.
Second, consider function $f(b_0, b_1, ...b_{m-1}, b_m) > 0$. This function can be covered by $b_0 + b_1 + ... + b_m$, i. e., by $m + 1$ prime implicants. □

We can limit the number of required prime implicants for Boolean function $f_{(d \,\Theta\, c)}$ where $\Theta \in \{=, \neq, >, \leqslant, ...\}$ analogously to Lemma 8.

Finding prime implicants of Boolean functions. We have proven that Boolean function $f_{(d \,\Theta\, c)}$ requires $\lfloor \log_2 n \rfloor + 1$ prime implicants. To find the prime implicants, one possibility is to use existing algorithms [73–75]. However, these algorithms either require implicants to compute the prime implicants, or they are NP-complete. This is because they find the prime implicants of any Boolean functions. However, if the domain of d is ordered, e. g., $\mathbb{N}$, $\mathbb{Z}$ or $\mathbb{R}$, we can efficiently find the prime implicants of a Boolean function $f_{(d \,\Theta\, c)}$: The ordered domain gives way to binary search to find the intervals of d which fulfill Data-Value Condition $con : (d\ \Theta\ c)$. Each interval corresponds to a prime implicant of $f_{(d \,\Theta\, c)}$. Algorithm 7 detects the prime implicants of Boolean functions $f_{(d < c)}$. Algorithms for $f_{(d \,\Theta\, c)}$, where $\Theta \in \{=, \neq, > , \leqslant, ...\}$, are analogous. Algorithm 7 takes Data Object d and Constant c as input parameters, where $dom(d) = [0..n-1]$, and c is a value in $dom(d)$. The idea behind the algorithm is to find the subsets of $dom(d)$ smaller than c, i. e., intervals that satisfy the condition $d < c$. Each interval is used to either update a prime implicant or to create a new one. To find the intervals, the algorithm performs a binary search on the domain of d until it finds c. To this end, it makes at most $\lfloor \log_2 n \rfloor + 1$ iterations (Lines 3-21). In each iteration, it evaluates an interval of $dom(d)$, i. e., it determines whether the values in the interval of $dom(d)$ are equal, less, or greater than c (Lines 7, 11, and 15). It then updates the implicants (Lines 9, 13, and 17) and/or creates a new one (Line 18) as follows:

1. If the middle value of the domain under evaluation (*mid*) is greater than c, all values greater than *mid* cannot make $d < c$ *true*. So the algorithm updates the latest implicant found to remove those values (Lines 10-13).

2. If *mid* is smaller than c, then all values smaller than *mid* can make $d < c$ *true*. So the algorithm updates the latest implicant to cover these values (Line 18). However, there might be some values greater than *mid* which evaluate $d < c$ to

Algorithm 7 Find Prime Implicants of $f_{(d < c)}$

Input: $dom(d) = [0..n-1]$, $c \in dom(d)$
Output: Prime Implicants of $f_{(d < c)}$: $implicants$

```
1: Stack<PImplicant> implicants;
2: PImplicant imp, imp';
3: high = 2^⌈log2 n⌉, low = 0;
4: do
5:     mid = low + (high − low)/2; k = ⌈log2 mid⌉
6:     if (implicants is not empty) then imp = implicants.pop()
7:     if (mid = c) then
8:         found = true
9:         if (imp is Null) then imp = b̄_k else imp = conjunction of imp and b̄_k
10:        implicants.push(imp)
11:    if (mid > c) then
12:        high = mid
13:        if (imp is Null) then imp = b̄_k else imp = conjunction of imp and b̄_k
14:        implicants.push(imp)
15:    if (mid < c) then
16:        low = mid
17:        if (imp is Null) then imp' = b_k else imp' = conjunction of imp and b̄_k
18:        if (imp is Null) then imp = b̄_k else imp = conjunction of imp and b̄_k
19:        implicants.push(imp); implicants.push(imp')
20: while (!found)
21: return implicants
```

true and are not considered yet. To cover these values, the algorithm generates a new implicant according to the interval (Lines 14-19).

3. If *mid* is equal to c, then *mid* cannot make $d < c$ *true*. So the algorithm updates the latest implicant found (Lines 7-9) and returns *implicants* as the set of prime implicants.

Example 32. *Figure 7.3 shows the steps of Algorithm 7 to find the prime implicants of Boolean function* $f_{(d < 3)}$*, where* $dom(d) = [0..7]$*. The prime implicants have to represent the highlighted values in Figure 7.3. In the first step, mid equals to 4 and is higher than* $c = 3$*. The algorithms adds implicant* $= \overline{b}_2$ *to the stack. In the second step, mid equals to 2 and is lower than c. The algorithm adds literal* $\overline{b}_1$ *to the latest implicant in the stack, i. e., implicant* $= \overline{b}_2.\overline{b}_1$*. It also creates a new implicant* $\overline{b}_2.b_1$ *and adds it to the stack. In the last step, mid* $= c$*, and the algorithm adds* $\overline{b}_0$ *to the latest implicant in the stack. So the prime implicants of Boolean function* $f_{(d < 3)}$ *are* $\overline{b}_2.\overline{b}_1$ *and* $\overline{b}_2.b_1.\overline{b}_0$*.*

Mapping Boolean Function $f_{(d \ominus c)}$. We map Data Condition $con : (d \ominus c)$ in a Petri Net using the prime implicants of its Boolean function $f_{(d \ominus c)}$. To this end, we propose Algorithm 8. It takes Boolean function $f_{(d \ominus c)}$ consisting of Prime Implicants $C_1, C_2, ..., C_k$ as input. It returns a Petri-Net mapping of $con : (d \ominus c)$. The idea behind Algorithm 8 is to create a transition t_{C_i} for each Prime implicant C_i (Line 3) which fires only when all literals of C_i are true, i. e., when all places that correspond

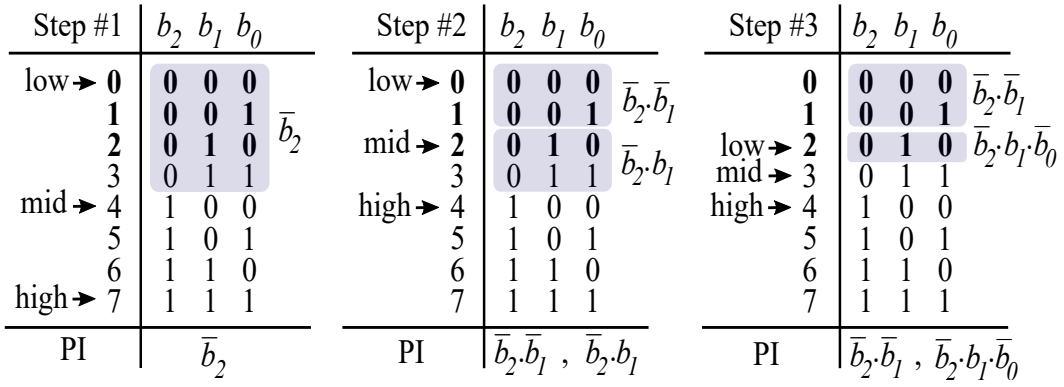


Figure 7.3: Steps of Algorithm 7 to find the prime implicants of Boolean function $f_{(d < 3)}$

to its literals are marked (Lines 4 - 6). Note that we have already mapped all the literals to places using our binary encoding, see Section 7.3.2. In the end, all these transitions are connected with Place $p.f$ (Line 7). This place can be marked only when at least one of the prime implicants of $f_{(d \ominus c)}$ evaluates to *true*. Put differently, this happens when $con : (d \ominus c)$ is satisfied.

Algorithm 8 Mapping Boolean function $f_{(d \ominus c)}$ to Petri Net

Input: Prime Implicants of $f_{(d \ominus c)}$: *implicants*
Output: Mapping of Boolean function $f_{(d \ominus c)}$: *pn*

1: $pn.places \leftarrow$ new Places $p.f$,
2: **for** each implicant C in *implicants* **do**
3: $pn.transitions \leftarrow$ new Transition t_C
4: **for** each literal l in C **do**
5: t_C.addIncomingPlace($p.l$)
6: t_C.addOutgoingPlace($p.l$)
7: t_C.addOutgoingPlace($p.f$)
8: **return** *pn*

Example 33. *Figure 7.4 and Figure 7.5 show the mappings of two Boolean functions,* $f_{(d < 3)}$ *and* $f'_{(d \geqslant 3)}$ *where* $dom(d) = [0..7]$.

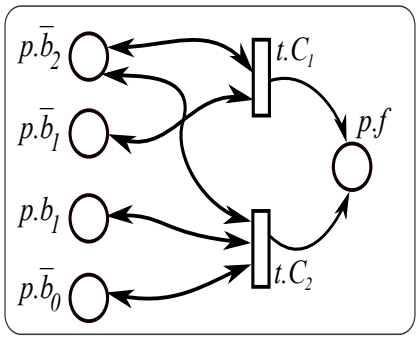


Figure 7.4: Mapping of Boolean function $f_{(d < 3)}$ into Petri Net

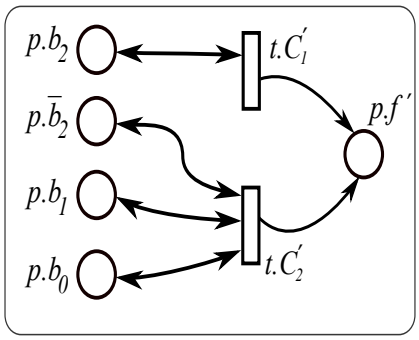


Figure 7.5: Mapping of Boolean function $f'_{(d \geq 3)}$ into Petri Net

7.3.4 Merge mappings of data conditions and control flow

In the next step, we connect the "control flow" Petri Net and the mappings of data conditions. The composition is straightforward. The "control flow" Petri Net has a single transition t for each sequence flow of an XOR-gateway [60]. In case the sequence flow is assigned to a Data Condition *con*, this transition can fire if and only if *con* is satisfied. To impose this condition to the execution of t, we connect Place $p.f$ from mapping of *con* as input of t. Thus, the token of Place $p.f$ is a precondition for the execution of t, i. e., t can fire only when the corresponding data condition is satisfied. So far, it can happen that more than one prime implicant of *con* is evaluated to *true* and, thus, more than one of the Transitions $t.C_i$ can fire at the same time. To avoid this, we connect the single incoming place of the gateway to all Transitions t_{C_i}. So all these transitions compete for a single token, and only one of them can fire.

Example 34. *Figure 7.6 shows the composition of the control flow and data conditions of an enhanced gateway. The mappings of data conditions are as in Example 33. The grey places and transitions belong to the control-flow Petri Net.*

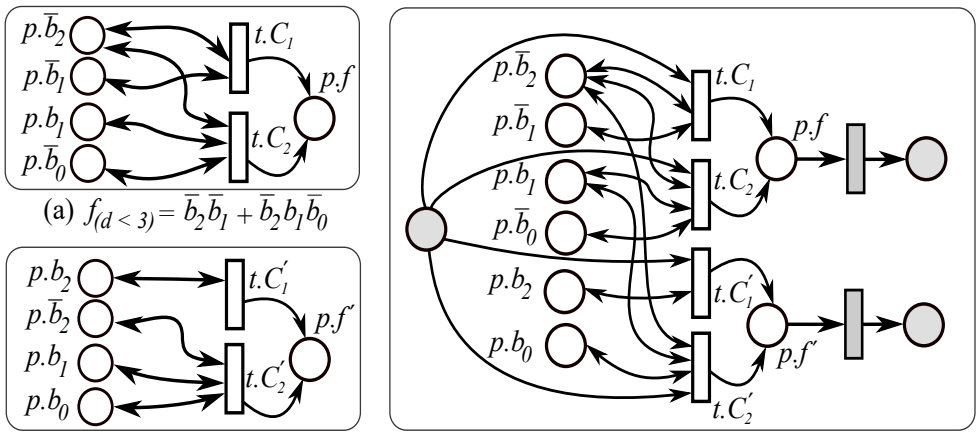


Figure 7.6: Composition of an XOR-gateway with two mappings of data conditions

7.3.5 Model checking

The result of the transformation is a Petri Net of the process representing the semantics of data-value conditions. After the transformation, we generate the state

space of the Petri Net using the LoLA-Toolkit [61]. Given the state space in which data-value conditions are represented, one can specify data properties of the process model using existing work such as [42, 57]. An off-the-shelf model checker then allows to verify properties of data-value-aware process models.

7.4 Evaluation

We want to find out how our approach affects the size of Petri Nets resulting from process models with different data-value conditions. We also want to study the real-world impact of our approach. So our evaluation consists of two parts: The first one computes the size of mappings for data-value conditions with different ordering relations $\Theta \in \{=, \neq, <, >\}$ and different domain sizes. The second one studies the size of the Petri Nets and of the state spaces of a real-world process, the 4G German spectrum auction. In both parts, we compare our work with the existing Petri-Net-based approaches which support the entire domain of data objects used in data-value conditions ("competitors" in what follows) [29, 57].

7.4.1 Comparison of data-condition mappings

Table 7.1: Size of Petri Nets and transformation times

	reduced Petri Net						unreduced Petri Net		
$dom(d)$	places	transitions $(d > c)$	$(d < c)$	$(d = c)$	$(d \neq c)$	time (s)	places	transitions	time (s)
$[0..2^{10}]$	10	10	10	10	1	≈ 1	2^{10}	2^{10}	≈ 1
$[0..2^{12}]$	12	12	12	12	1	≈ 1	2^{12}	2^{12}	≈ 3
$[0..2^{14}]$	14	14	14	14	1	≈ 1	2^{14}	2^{14}	≈ 67
$[0..2^{16}]$	16	16	16	16	1	≈ 1	2^{16}	2^{16}	≈ 995
$[0..2^{18}]$	18	18	18	18	1	≈ 1	2^{18}	2^{18}	$\approx 14,212$
$[0..2^{20}]$	20	20	20	20	1	≈ 1	2^{20}	2^{20}	$\approx 275,762$

To evaluate the scalability of our approach, we look at different domains for Data Object d, i. e., $dom(d) = [0..2^{10}]$, $dom(d) = [0..2^{11}]$, ..., $dom(d) = [1..2^{20}]$. Table 7.1 lists the number of places and transitions for each mapping of a data-value condition and compares them with the competitors. The reduction in the number of places and transitions for the mapping of data-value conditions is significant, for large domains of d in particular. Observe that Table 7.1 lists the numbers for our approach in the worst-case scenario, i. e., they can be smaller. This is the case when our algorithm finds the prime implicants with less than $\lfloor \log_2 n \rfloor + 1$ comparisons. Moreover, one can represent a data-value condition $d = c$ using only 1 transition, regardless of the size of $dom(d)$, since $f_{(d = c)}$ has one prime implicant.
This reduction also curbs the runtime of the mapping by much. To illustrate, to map Data-Value Condition $(d \; \Theta \; c)$, where $dom(d) = [1..2^{20}]$, our approach requires less than a second, using a computer with 16 GB main memory and 2 Intel 3 GHz CPUs. Competitors need more than 3 days for the same mapping.

7.4.2 Simultaneous Multi-Round Auction

We also evaluate our approach against a real-world process, the German 4G spectrum auction that sells the 800 MHz band [24]. There are four reasons for this use case for our evaluation: (1) Spectrum auctions are an important process with a significant impact on the economy. To illustrate, they earned 50.8 billion Euros in Germany in 2000 [25]. (2) Verification of auctions allows to detect undesirable outcomes upfront. For instance, one can detect the lowest possible revenue of an auction or its lowest efficiency. However, existing verification techniques are unable to verify spectrum auctions with domains of larger than [0..10], due to state-space explosion [57]. (3) Data conditions over data objects with large domains are essential to verify properties of auction models, as we will see momentarily. (4) Spectrum auctions have a clear structure that does not change when altering their parameters, e.g., the domain of bids. This allows to study the impact of our approach systematically.

7.4.2.1 Evaluation setting

To evaluate our approach, we focus on three auction settings, i. e., auctions with different numbers of bidders and products:

- *Process 1*: Two bidders compete to win two products.
- *Process 2*: Three bidders compete to win three products.
- *Process 3*: Four bidders compete to win six products – this has been exactly the size of the German 4G spectrum auction to sell 800 MHz band [24].

The models consist of two data objects with large domains, namely Data Objects *bid* and *price*. To evaluate the scalability of our approach, we experiment with different domain sizes, ranging from [0..10] to [0..10^5]. This amounts to 3 (process models) $\times$ 5 (alignment of domains) = 15 process models used in our evaluation. We have selected 10^5 as the highest value of *bid* and *price*, in line with the actual values. The smallest jump in this auction was 10^4, the next one 20^4, etc. The highest bid was $\approx 10^9$ placed by Telefonica O2 [24]. This allows to confine the domain of *bid* and *price* from [0..10^9] to [0..10^5].

7.4.2.2 Comparisons of Petri Nets

We have transformed the auction process models to Petri Nets using our approach. Table 7.2 lists the number of places and transitions for each process model. As shown, our approach transforms all process models to smaller Petri Nets than competitors ("unreduced" in the table). This reduction is more significant when the domain of the data objects is the largest one. In Process 3, for example, the size of the Petri Net with our encoding is almost 0.5% of the unreduced size. Moreover, our approach outperforms the existing works, even when the size of data domains is small.

Table 7.2: The size of Petri Nets

		number of places			number of transitions		
		Process1	Process2	Process3	Process1	Process2	Process3
$[1..10^1]$	reduced	170	244	308	130	192	234
	unreduced	206	290	362	146	214	265
$[1..10^2]$	reduced	188	268	338	146	219	264
	unreduced	746	1,010	1,262	506	754	985
$[1..10^3]$	reduced	206	292	368	248	246	294
	unreduced	6,146	8,210	10,262	4,106	6,154	8,185
$[1..10^4]$	reduced	230	324	408	190	282	334
	unreduced	60,146	80,210	100,262	40,106	60,154	80,185
$[1..10^5]$	reduced	248	348	438	208	309	364
	unreduced	600,146	800,210	1,000,262	400,106	600,154	800,185

7.4.2.3 Comparison of state spaces

Table 7.3 lists the size of the state spaces of auction process models generated by the LoLA framework [61]. The reduction of the state spaces is even larger than for Petri Nets. Given the unreduced Petri Nets, it has not been possible to generate the state space of Process 3 when the domain is larger than [0..10]. However, the Petri Nets resulting from our approach have $\approx 21 \times 10^6$ states when the domain size is the maximal one. This reduction enables verification of auction models using existing model checkers like LoLa.

Table 7.3: The size of the state spaces

	reduced state space			unreduced state space		
	Process1	Process2	Process3	Process1	Process2	Process3
$[1..10^1]$	$\approx 10 \times 10^2$	$\approx 10 \times 10^3$	$\approx 15 \times 10^4$	$\approx 22 \times 10^3$	$\approx 32 \times 10^4$	$\approx 11 \times 10^7$
$[1..10^2]$	$\approx 20 \times 10^2$	$\approx 55 \times 10^3$	$\approx 13 \times 10^5$	$\approx 20 \times 10^5$	np*	np
$[1..10^3]$	$\approx 30 \times 10^2$	$\approx 15 \times 10^4$	$\approx 50 \times 10^5$	np	np	np
$[1..10^4]$	$\approx 60 \times 10^2$	$\approx 43 \times 10^4$	$\approx 21 \times 10^6$	np	np	np
$[1..10^5]$	$\approx 80 \times 10^2$	$\approx 77 \times 10^4$	$\approx 46 \times 10^6$	np	np	np

*np: not possible due to state-space explosion.

7.4.2.4 Verification of properties.

Given the reduced state space, we have been able to verify properties of Process 3. We have modeled these properties as CTL formulas [8] using the rules in [57]. Then we have run the LoLA model checker. We have verified the lowest final price of each product, i.e., whether a product can be sold for a certain price. The average time for this verification has been less than 6 seconds, while this simply has not been possible before. Our verification also reveals which assignment of products to bidders leads to the lowest final prices. An auction designer can now take this as a starting point for improvement by also applying our method to new designs and comparing results.

7.5 Summery

Verification of process models with data-value conditions over large domains is essential for many process models. However, existing techniques often suffer from state-space explosion or even lead to an incorrect diagnosis. In this chapter, we have introduced a novel approach to overcome these challenges by representing data objects in Petri Nets using *binary encoding*. Then, we have proposed a new algorithm to reduce the number of transitions, based on the theory of Boolean functions. Our evaluation shows that our approach is able to reduce the size of Petri Nets and of state spaces by several orders of magnitude. To our knowledge, we are the first to achieve a logarithmic reduction of the state space while supporting the entire domain of data objects. This allows to verify properties of an important process model in its true size, namely the German 4G spectrum auction.

8. Dealing with Data-Value Functions with Large Domains

In the previous chapter, we have provided a new technique based on binary encoding to handle data-value conditions with large domains. In this chapter, we use the binary encoding technique to tackle state-space explosion in process models in which data-value functions with large domains exist.

8.1 Introduction

Data-value conditions play a critical role in process models, but so do data-value functions. Process elements like activities can modify data values during execution of the process. Modified values can influence the behaviour of the process and, thus, the verification result.

Example 35. *Figure 8.1 shows parts of a simplified auction process model in BPMN notation. The gateway with condition "bid > 5" determines the branch the process will follow. The data value used in this condition, the value of "bid", may be modified by a preceding activity, in this case "place new bid". This modification may affect the decision of the gateway and. thus, the execution of the process. In consequence, properties of the process may change, like the final price of the good.*

Challenges. To verify data-value-aware process models, the execution semantics of data-value functions must be represented in the state space of the process model. A data-value function is one that modifies the value of a data object during process execution, e.g., increases the price of a product.

There exist techniques to verify data-value-aware process models [6, 29]. However, they suffer from state-space explosion [2], i. e., the state space increases exponentially with the number of data values modified during process execution. Think of an auction with six products, each associated with a price and three bids with a value ranging from 1 to 100. This amounts to 100^6 (price) $\times$ 100^{6^3} (bids) states. A verification scheme that explores the entire state space is infeasible. Hence, the

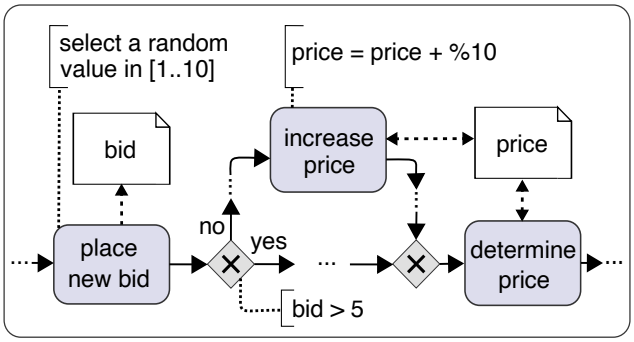


Figure 8.1: Parts of an auction process modeled in BPMN

question is how to reduce the state space of data-value-aware process models when modification of data values takes place.

To solve this problem, (a) reduction techniques based on relevance and (b) abstraction techniques to enable symbolic execution have been proposed. Reduction techniques detect the elements of the process model that influence the verification of a certain property, so-called *relevant element*, and prune the irrelevant ones [62, 63]. However, state-space explosion prevails when a data object with a large domain is relevant and is kept in the reduced model. Abstraction techniques in turn represent the domain of a data object with a symbol, instead of considering each possible data value separately [29]. Whenever a model checker reaches a point in the state space where a data value is modified, the modification is applied to the abstract symbol. However, the state space branches for each data value used in a data-value function. So abstraction techniques are ineffective when a function modifies the values of a data object with a large domain.

Example 36. *Consider the process model shown in Figure 8.1. Assume that the domain of price ranges from 1 to 500,000. Abstraction techniques require branching the state space for each value of price to represent function price = price+%10. This amounts to 500,000 branches in the state space, each for one value in the domain of price.*

This current chapter proposes a Petri-Net-based approach to avoid the state-space explosion problem caused by data-value functions which modify data objects with large domains. Our approach starts by transforming the control flow of the data-value-aware process model into Petri Nets using the rules in [60]. It then maps each data-value function into Petri Nets – and this mapping is the core contribution of this chapter. We propose a *binary encoding*, mapping each data object with n possible values to at most $2{\cdot}(\lfloor \log_2 n \rfloor{+}1)$ new places in the Petri net. This is in strong contrast to approaches that represent each value with a different place or token. Our binary encoding brings two important advantages. First, it significantly reduces the number of places in the Petri net, compared to existing work [6, 28]. Second, it allows deploying the state-of-the-art data structure *Binary Decision Diagrams* (BDD) to map the semantics of data-value functions to a Petri net. Given the mappings of functions in the form of BDD, we one can now apply existing BDD reduction techniques to reduce the size of mappings and, thus, the state space of the process models.

We evaluate our approach with a real-world process, the German 4G spectrum auction. It has sold a very valuable bandwidth, the 800 MHz band [24]. Our approach reduces the number of places and transitions of the Petri Net by 80.3% and 95%, respectively. Given the reduced Petri Net, one can now generate the state space of this auction model with large domains. This has been infeasible so far [6]. The reduced state space allows deriving important values, like *auctioneer's revenue*, i. e., the sum of the final prices of the products [15], in the worst case. Auctions such as the Dutch UMTS auction have had disastrous outcomes for the auctioneers [21]. Being able to detect such outcomes in advance is a significant contribution to auction design.

8.2 Our approach

This chapter proposes an efficient verification scheme for data-value-aware process models supporting modification of data values. We start to map the control flow of a process model to Petri Nets using the rules in [60]. However, this mapping only covers the "control flow". Thus, our approach extends this mapping with the semantics of data modifications, which stand for the contributions of this chapter:

1. We map each data object to a set of places using the rules in Section 7.3.1. Any distribution of tokens in these places represents a data value.

2. We use these places and map each data-value function to a Petri Net. Each of these mappings represents how a data-value function modifies data values in the Petri Net (Section 3.2).

3. We merge the "control flow" Petri Net and the mappings of data-value functions (Section 3.3). The result is the transformation of a data-value-aware process model to a Petri Net that includes the semantics of modification of data values.

Given the resulting Petri Net, an off-the-shelf model checker then allows verifying properties of the data-value-aware process model. We use plain Petri Nets as a target for the mapping because of the availability of efficient analysis techniques [7].

8.2.1 Mapping of data-value functions to Petri Nets

To verify a data-value-aware process model, it is essential to include the semantics of data modifications in the Petri Net, i. e., capture it in the state space of the process model. Consider Data-Value Function $f : D \to D'$ where D has domain $[0..n-1]$ and D' has domain $[0..m-1]$. To map f to a Petri Net, one possibility is to create a new transition for each value in the domain of D and add the incoming places and outgoing places to the transitions according to the function [6, 28]. This results to n new transitions to map Data-Value Function f into a Petri Net. But our intention is to reduce the number of transitions required to map a data-value function and to reduce the state space. *Data-Value Functions to Truth Tables.* The first step to map a data-value function into Petri Nets is to build its the truth table, i.e., a table that assigns each input data-value to the corresponding output data-value.

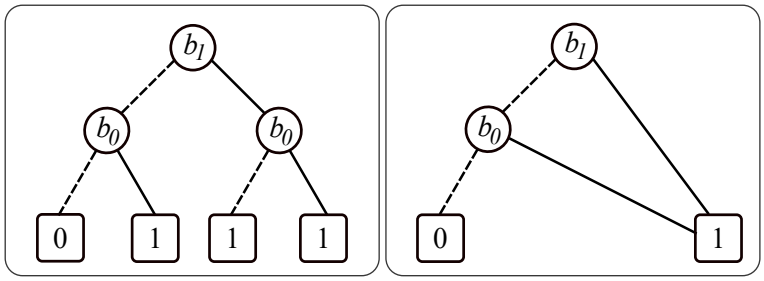
Table 8.1: Truth table of Data-Value Function $f(dv)$

	input		output
	b_1	b_0	q_0
0:	0	0	0
1:	0	1	1
2:	1	0	1
3:	1	1	1

Example 37. *Consider Data-Value Function* $f(dv) : D \rightarrow D'$*, where* D *has a domain of* $[0..3]$*,* D' *has a domain of* $[0..1]$*, and* $f(dv) = \begin{cases} 0 & dv \leqslant 0 \\ 1 & dv > 0 \end{cases}$*. One requires two bits (*b_1 *and* b_0*) to represent the input data-values and one bit (*q_0*) to represent the output data-values. Table 8.1 is the truth table of Function* f*.*

Truth Tables to Ordered Decision Diagrams. Given the truth table of a data-value function, we derive its *Binary Decision Diagram* (BDD) [3]. A Binary decision diagram is a data structure for representing a truth table as a directed acyclic graph, corresponding to a compressed form of decision tree. An ordering constraint is imposed among the occurrences of decision variables in the graph, yielding *Ordered Binary Decision Diagrams* (OBDD). The reason behind transforming a truth table to its OBDD is that: (1) OBDDs are especially effective as the algorithmic basis for symbolic model checkers, and (2) one can apply the existing reduction algorithm to reduce any OBDD. The reduced form of OBDD gives way to reduce the size of Petri Net and the state space of the process models as we will explain.
Suppose $f : D \rightarrow D'$ is a data-value function, where D and D' have a domain of $[0..n]$ and $[0..m]$, respectively. Thus, the truth table of f contains bits b_i, where $0 \leqslant i \leqslant \log_2 n$ to represent the input data-values and bits q_j, where $0 \leqslant j \leqslant \log_2 m$ to represent the output data-values. For each q_j, we generate a single OBDD with the height of $\log_2 n$. Each path through the OBDD corresponds to exactly one row in the truth table.

Example 38. *Let* $f : D \rightarrow D'$ *be a data-value function as in Example 37 and its truth table. The input of this function has a domain of* $[0..3]$ *which is represented by* b_0 *and* b_1*. Figure 8.2(a) shows the corresponding OBDD of Data-Value Function* f*. The dash edges represent zero values of the truth table.*



(a) OBDD of Table 1 (b) Reduced OBDD of Table 1

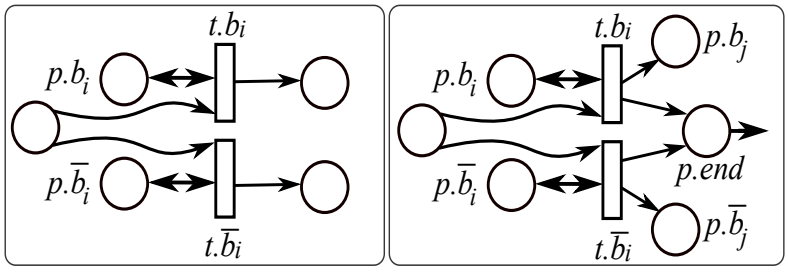
Figure 8.2: Example of an OBDD and its reduced form

Reduction of Ordered Binary Decision Diagrams To reduce the ordered binary decision diagrams, we use Algorithm 9 provided by [4]. The algorithm takes two steps: (1) It eliminates all the duplicate leafs, i.e., for a duplicate zero-leaf (or one-leaf), redirects all the incoming edges to only one of them. (2) It eliminates the isomorphic subtrees. Two subtrees are isomorphic when they are connected in the same way, i.e., they have the same nodes and edges. When $v \neq w$ are roots of two isomorphic subtrees, the algorithm removes w and redirect all the incoming edges to w to v.

Algorithm 9 OBDD reduction

```
1: procedure OBDD REDUCTION(OBDD)
2:     for node in OBDD do
3:         if node has other isomorphic node(s) then
4:             eliminate and re-direct all its isomorphic node(s).
5:     if OBDD contains double edges then
6:         eliminate and re-direct all nodes which have double edges.
```

Reduced Ordered Binary Decision Diagrams to Petri Nets The transformation of reduced OBDD into Petri Nets is straight forward. For each inner node b_i in the OBDD, we create two transitions $t.b_i$ and $t.\overline{b}_i$ connected to Place $p.b_i$ and Place $\overline{b}_i$, respectively. Note that we have already created these places using our binary encoding, see Section 7.3.2. Each transition fires if the corresponding place is marked, i.e., transition $t.b_i$ ($t.\overline{b}_i$) fires when $b_i = 1$ ($\overline{b}_i = 1$). In the end, both transitions are connected with a single place. This place helps to connect Petri Nets resulted from other inner-nodes. Figure 8.3(a) shows the transformation of an OBDD inner-node into Petri Nets. The transformation of leaf nodes are the same as the inner-nodes, but Transitions $t.b_i$ and $t.\overline{b}_i$ are connected to the b_j-*places*, according to the truth table. Figure 8.3(b) represents the transformation of a leaf-node of a reduced OBDD.



(a) OBDD inner-node in Petri Net (b) OBDD leaf-node in Petri Net

Figure 8.3: Transformation of OBDD nodes to Petri Nets

8.2.2 Merge mappings of data functions and control flow

In the next step, we connect the "control flow" Petri Net and the mappings of data-value functions. The composition is straightforward. The "control flow" Petri Net has a single transition t for each activity/event of the process model [60]. In case the activity/event is assigned to a Data-Value Function f, this transition fires only after imposing the effects of Function f on data values. To do so, we connect Place $p.end$ from mapping of f as the incoming place of t. Thus, the token of Place $p.f$ is a precondition for the execution of t, i. e., t can fire only when the modification of data values are in place.

Example 39. *Figure 8.4 shows the mapping of the reduced OBDD in Example 38 into Petri Nets. The OBDD has two inner nodes, b_1 and b_0. For Node b_1, we create two transitions $t.b_1$ and $t.\overline{b_1}$, each connected to the corresponding place. Transition $t.\overline{b_1}$ fires only when $p.\overline{b_1}$ is marked, i.e., $\overline{b_1} = 1$. This is how our proposed binary-encoding approach facilitates the model checker: In case $t.\overline{b_1}$ fires, the model checker does not explore other states resulted from the blue part in Figure 8.4, because the output is already determined and there is no need to investigate the other states. When $t.b_1$ fires, the model checker needs to explore bit b_2 to determine the correct output. The grey places and transitions belong to the control-flow Petri Net.*

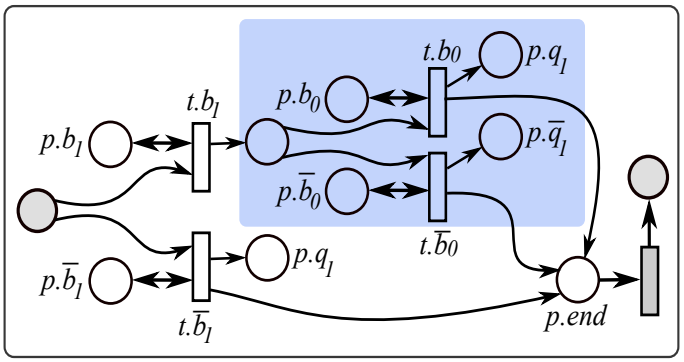


Figure 8.4: Example of OBDD into Petri Nets

8.3 Evaluation

We want to find out how our approach affects the size of Petri Nets resulting from process models with different data-value functions. We also want to study the real-world impact of our approach. So our evaluation consists of two parts: First, we compute the number of places and transitions required to map different data-value functions with different sizes of domains. Second, we study how our approach to handling data-value functions enables verification of a real-world application, i.e., the 4G German spectrum auction to sell one of the most valuable bandwidths, the 800 MHz band [24].

8.3.1 Comparison of mappings of different data-value functions

We use four data-value functions for our evaluation and calculate the number of places and transitions in the Petri net. These four functions are: $f(x) = x+2$, $f(x) = x+y$, $f(x) = x \times 1.2$, and $f(x) = max(x, y)$. To study the scalability of our approach, we vary the size of each function's input domain from $[0..2^{10}]$ to $[0..2^{20}]$. Table **??** shows how many places and transitions we need to transform each function to a Petri net with varying sizes of the domain. Function of multiplication is more expensive than the other functions, i.e., it requires more number of places and transitions in the Petri net. In general, the number of places and transitions grows logarithmically with the size of the data domain using our approach. This is a improvement over the existing works [6], where the number of places and transitions has a linear relationship with the size of the data domain. For example, one requires 2^{20} places and transitions to transform Data-Value Function $y = x + 2$, when $domain(x) = [0..2^{20}]$, using the existing works.

Table 8.2: Number of places and transitions for different functions

domain	size of net	$y = x + 2$	$z = x + y$	$y = x \times 1.2$	$z = max(x, y)$
$[0..2^{10}]$	places	77	158	676	141
	transitions	65	147	987	151
$[0..2^{12}]$	places	91	188	806	167
	transitions	77	175	1183	179
$[0..2^{14}]$	places	105	218	936	193
	transitions	89	203	1379	207
$[0..2^{16}]$	places	119	248	1066	219
	transitions	101	231	1575	235
$[0..2^{18}]$	places	133	278	1196	245
	transitions	113	259	1771	263
$[0..2^{20}]$	places	147	308	1326	271
	transitions	125	287	1967	291

8.3.2 Evaluation setting

To evaluate our approach, we model the SMR auction in BPMN. The process model consists of four bidders which compete to win six products – this has been exactly the size of the German 4G spectrum auction to sell 800 MHz band [24]. We assign a random budget to each bidder for a certain product in the range of $[2 - 100]$, similarly to [15]. We also have defined a reserve price of 1 for all products, so all bidders can afford all products at the beginning of the auction. Bidders 1 and 3 have capacity point of 2, Bidders 2 and 4 have capacity point of 3 and 1, respectively.

8.3.3 Transformation of SMR process model to Petri Nets

We have implemented a framework to verify SMR auction process models using our binary approach. The input of our framework is a process model in BPMN format. The SMR process model in form of BPMN consists of data-value functions such as *increase price*, *place bids*, *decrease capacity*, and *remove bids*. Our framework takes this process model and transforms it to a Petri Net. Table 8.3 lists the number of places and transitions using our approach and compares that with the naive transformation approach [6]. As shown, our approach transforms the SMR process model to a smaller Petri Net and state space than the naive approach. We are able to generate the Petri Net including the semantics of modification of data values using only 576 places and 786 transitions, while these numbers are 2912 and 15421, respectively, using the existing techniques.

Table 8.3: The results of transformation of SMR auction models

approach	places	transitions	transformation time	state space
our approach	576	786	$< 1sec$	324,647
naive approach	2912	15421	$< 1sec$	not possible

8.3.4 Verification of SMR process models

Given the reduced state space, we have been able to verify properties of SMR auctions with larger domains. To do this, we use the state-space generation and model checking algorithm of the LoLA framework [61]. We have modeled these properties as CTL formulas [8] using the rules in [6]. We have verified the lowest final price of each product, i.e., whether a product can be sold for a certain price. To do so, we have verified 130,292 properties. The average time for this verification has been less than 7 seconds, while this simply has not been possible before. Our verification also reveals which assignment of products to bidders leads to the lowest final prices. Table 8.4 shows one of the assignments out of 13, which leads to the lowest possible revenue for the auctioneer. An auction designer can now take this as a starting point for improvement by also applying our method to new designs and comparing results.

Table 8.4: Verification Result

	product 1	product 2	product 3	product 4	product 5	product 6
final price	70	80	80	70	60	70
winner	bidder 2	bidder 1	bidder 1	bidder 2	bidder 2	bidder 4

8.4 Summery

Verification of process models with data modifications over large domains is essential for many process models. However, existing techniques often suffer from state-space explosion or even lead to an incorrect diagnosis. In this chapter, we have introduced a novel approach to overcome these challenges by representing modification of data values as Ordered Binary Decision Diagrams (OBDDs). This has enabled us to apply the state-of-the-art reduction techniques to reduce OBDDs and thus, the size of Petri Nets and of state space of the process models. Our approach allows to verify properties of a real-world application, the German 4G spectrum auction. We have derived the lowest possible revenue of the auction for the first time while increasing the domain of data objects by ten times.

9. Results from the Verification of Models of Spectrum Auctions

9.1 Introduction

Spectrum auction revenue is a significant source of governmental income. Germany and the U.K., for example, have earned respectively 50.8 and 37.5 billion Euros in 2000 [25]. Although auctions should be designed so that undesirable outcomes do not occur, catastrophic results have happened in several cases. In the Netherlands, a political fiasco occurred in 2000 because of the low revenue of the Dutch UMTS auction [21]. In another example in the U.S., an auction policy that raised bid prices caused a loss of 30 MHz for a decade. This flaw cost around 70 billion dollars[5]. In yet another case [23], about half of the products were left out.

Literature offers two possible ways of studying auctions: (a) experimental analyses performed in laboratories with human subjects [14], and (b) theoretical analyses. Finding undesirable outcomes with either technique continues to be challenging: When it comes to (a), laboratories perform relatively few experiments. To illustrate, to investigate all experimental designs in [15], over 13 million experiments would be necessary. No institution would be able to accommodate such a setup. In (b), researchers use auction theory to predict equilibria, relying on assumptions regarding bidding behavior [16]. In general, rational behavior of bidders is part of the assumptions [17]. But this assumption is not always valid [19]. In spite of developing frameworks for truthful bidding under interference constraints [20], bidders can still be irrational. In consequence, design errors that go unnoticed can lead to catastrophic auction results.

To detect such cases, verification techniques can be applied before executing the auctions, i.e., one can detect unexpected outcomes before they actually happen. Authors in [6], for example, have proposed a Petri-Net-based approach to verify data-value-aware process models. In such processes, values of data objects such as the price of products play an important role, and process elements can modify these values while the process is executed. Our approach allows verifying certain properties of spectrum auctions. For example, one can derive the value of the lowest

auctioneer's revenue, i. e., the sum of the final prices of the products. The evaluation in [6] only covers one setting, i.e., selling two products to three bidders with fixed auction parameters. So this study alone does not provide much information to auction designers.

In this chapter, we report on the results of a systematic analysis of a real-world application, the German 4G spectrum auction, to sell one of the most valuable bandwidths, the 800 MHz band [24]. This auction has had four bidders and has sold six products. More specifically, we study the effect of so-called *capacity point*s in this auction. A capacity point is the maximum number of licenses that a bidder can win. This parameter prevents bidders from winning too many items. With this feature, an auction designer can guarantee a certain number of bidders being awarded a good and prevent bidders from forming a monopoly. Our study focuses on capacity points for two reasons: (1) The capacity points assigned to each bidder influence the revenue of the auctioneer. (2) In contrast to other parameters like the budgets of the bidders, the auctioneer controls the value of capacity points. In other words, he or she can change their number. We study the impact of capacity points by systematically distributing different capacities among the bidders and assigning a random budget according to [15]. Studying all such distributions in combination with all different budgets that are possible is future work. In particular, we consider the following research questions:

1. Which assignments of capacity points lead to the lowest and the highest revenue of the auctioneer?

2. With a certain allocation of the capacity points, how often the lowest revenue can happen for the auctioneer?

3. Does increasing capacity points always increase the revenue?

4. What is the best assignment of capacity points to the bidders, i.e., making the worst revenue possible not too low?

5. Does changing the capacity point of a single bidder always change the auction's outcome?

To verify properties of the process model of a spectrum auction, we make use of a Petri-Net based verification technique developed in [6]. We come up with a rigorous formulation of the above questions, referred to as properties in what follows, as CTL formulas [11]. To do so, we have verified more than 2 million properties. Our findings are interesting: Varying the capacity points does not always affect the revenue. More specifically, varying the capacity points of some bidders does not have any impact on the lowest revenue, whereas varying the points of other bidders dramatically changes the results. Our method allows identifying bidders whose capacity points have a significant impact on the lowest revenue, as well as the corresponding allocation of capacity points that leads to this outcome. Next, we have observed a trade-off between the 'extent of monopolism' vs. the 'expected revenue' which the allocation of capacity points may influence. This might help the auctioneer to assign capacities in line with his/her objectives.

9.2 Our Approach

We collect a dataset describing the different outcomes of a spectrum auction in order to answer the research questions. We obtain this dataset by verifying the respective model of the spectrum auction. In the following, we describe the verification approach used. We do so because the specifics of what we can verify (both on a functional and on a non-functional level) rely on it. Figure 9.1 serves as an overview. (1) We model the SMR auction in BPMN notation [10]. BPMN is a suitable language for the description of spectrum auction models in a visual way and allows the subsequent analysis using the existing verification techniques. (2) We transform the SMR process model into Petri Nets. We use plain Petri Nets as a target for the mapping because of the availability of efficient analysis techniques [7]. (3) Given the resulting Petri Nets, one must specify properties of SMR auction in a formal language such as Computation Tree Logic (CTL) for verification. (4) In the final step, Activity *model checking* verifies the properties against the resulting Petri Nets and outputs the verification results.

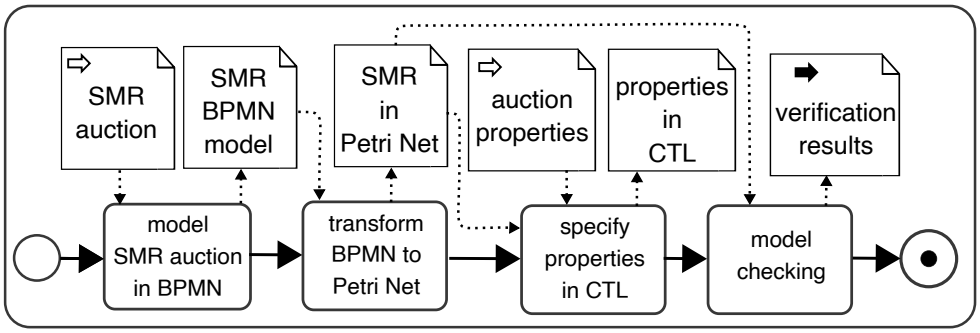


Figure 9.1: Overview of verification procedure

9.2.1 SMR auction in BPMN model

In Fig. 9.2, we show a simplified version of an SMR auction in BPMN notation. The complete BPMN model is also available[1]. Observe that the model does not specify a certain bidding behavior. To do so, we issue a random bid that falls between the current price of a product and the budget of the qualified bidders. Doing without such an assumption is in some contrast to auction theory, which tends to focus on rational bidders, even though this kind of behavior is not guaranteed. We consider all possible valid bids to derive extreme outcomes of an auction, including the lowest prices that are possible. The BPMN model has three subprocesses. (1) The first subprocess examines whether a bidder can afford further products he or she has not won yet (*availability of bidders*). The auction continues with Subprocess *bidding of each bidder*. (2) Activity *place bid* issues a random bid. If a bidder has both budget and capacity left to acquire the product, he or she will always submit a bid. Activity *decrease capacity* reduces the bidder's capacity right after having won a product. Activity *remove bid* removes bids from bidders who have no capacity left. (3) Subprocess *winner determination* determines the new reserve prices and the winners. Until no more bids are submitted, these three subprocesses are repeated. Note that changing the parameter values (e.g., capacity points), but

[1] https://doi.org/10.5445/IR/1000143697

leaving the BPMN structure unchanged, gives us a different auction process model in our terminology.

9.2.2 SMR Process Model to Petri Nets

We use an existing verification framework [57], to verify properties of the SMR auction. It transforms the BPMN model of the auction into a Petri net. We use plain Petri Nets, in contrast to, say, colored Petri Nets, as the target of the mapping. This is because efficient analysis techniques are available [59]. Another reason is that plain Petri nets provide counterexamples when they verify a property. This makes it relatively easy for the designer of the model to detect where the unintended behavior of the process occurs and to fix it as necessary. To illustrate, think of spectrum auctions. When the model checker finds a path leading to the lowest revenue, this helps the auction designer with that chore.

9.2.3 Specification of Properties

The result of the transformation just described is a Petri Net representing the semantics of the use of data values in the process. To verify a spectrum auction, one must specify properties in a formal language such as Computation Tree Logic (CTL). In the following, we show how such properties referring to data values can be specified using CTL.

Example 40. *The data property for the question: "Can product.*2 *have a price of* 2 *at the end of the auction?" is:*

$$EF(product.1.price.2 \wedge e.end).$$

*In this formula, "product.*2*.price.*2*" is the "price" of 2 for "product.2". The atomic formula "e.end" is an end event, i. e., represents the end of the process.*

To detect the lowest revenue of the auctioneer, we first find the lowest final *price* of *products*, starting with the reserve prices, using the property in Example 40. In case this property is not satisfied, we now verify a new property with an increased price. We continue increasing prices until there is a state that fulfills the property. Next, we detect the winner who won a product at a certain price. To do so, for each bidder who has a budget equal to or higher than the final price of the product, we check whether they can be the winner.

Example 41. *Suppose that the lowest price for product.*1 *is 4, and the budget of bidder.*1 *is higher than 4. Then the property to check is:*

$$EF(p.product.1.price.4 \wedge product.1.bidder.1 \wedge e.end).$$

*In this property, "p.product.1.price.4" fixes the price for product.*1 *to 4, and "product.1.bidder.1" expresses that product.*1 *belongs to bidder.*1*, i.e., bidder.*1 *wins this product.*

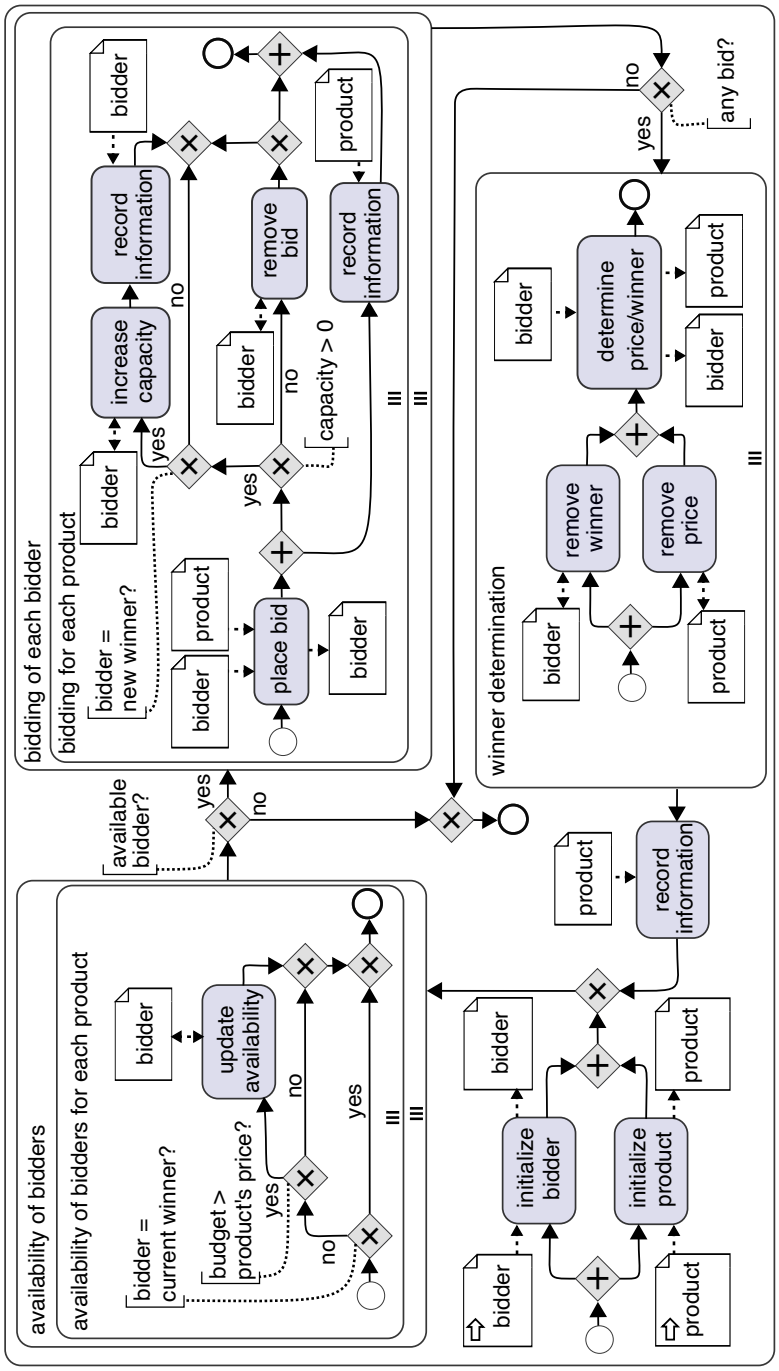


Figure 9.2: Simplified process model of an SMR auction in BPMN notation.

If the model checker can find a state, i. e., an execution path fulfilling the property, we record the bidder whose budget is sufficient as the winner of the product and continue to check whether other bidders can be the winner as well. In case two bidders (e.g., *bidder*.1 and *bidder*.2) are the potential winners for a certain product, we continue with both cases. The first case is to fix *bidder*.1 as the winner, decrease his/her capacity points and verify the price of other products, i.e., to check whether the other products can be sold for a certain price. The second case is to identify *bidder*.2 as the winner, decrease her/his capacity point, and continue verifying the other products.

9.3 Evaluation

In the following, we describe the auction parameters used in the process models to be verified, Section 9.3.1. In Section 9.3.2, we first report on characteristics of the verification procedure itself and then describe the dataset obtained that we will then use to answer those research questions. Using the obtained dataset results, we address the research questions listed in Section 9.1; see Section 9.3.3.

9.3.1 Evaluation Setting

The SMR auction model evaluated here consists of 4 bidders and 6 products. This is exactly in line with the German 4G spectrum auction to allocate licenses belonging to the 800 MHz band. Having four different bidders in the auction results in $3^4 = 81$ different assignments of capacity points. Each assignment results in a different process model. However, assignments where the sum of capacity points is less than 6 cannot happen. This is because the number of products to be auctioned off is 6, so the sum of capacity points must be at least 6 to sell all of them. After reducing such assignments, we get 76 pairwise different process models, i.e., bidders have different capacity points. Table 9.1 lists the capacity points of the bidders in each process model. At first, we keep the capacity points of *Bidder*.1 at 1 and vary the capacity points of the other bidders (Processes 1 to 22). In order to do this, we keep the capacity points of *bidder*.2 at 1, and we vary the capacities of *bidder*.3 and *bidder*.4 (Processes 1 to 6). Therefore, we keep the 's capacity points of *bidder*.3 at 1 and change the ones of bidder.4 from 1 to 3. The capacity points of 2 and 3 are distributed the same among the bidders. To each bidder we have assigned a random budget for a certain product in the range of [2..10], similarly to [15].

9.3.2 Verification of Properties

The number of properties to be verified in each process model varies between 24,844 and 29,880. Namely, verifying the price of a certain product might be fast, i.e., without having to verify many properties. For example, when the lowest final price is 2, the higher prices do not need to be verified. However, when the bidders have higher budgets and enough capacity points left, the competition is harder between bidders, i.e., the product is not sold for a low price, and one must verify more properties to identify the final lowest prices. Figure 9.3 graphs the number of properties verified in each process model. In Process 76, for example, we have verified 29,880 properties.

In the process models on the right side of the figure, the sum of capacity points tends to be higher than on the left side. This means that more properties on process models

Table 9.1: The distribution of capacity points in each process model

process model	capacities	process model	capacities	process model	capacities
Process 1	[1, 1, 1, 3]	Process 27	[2, 1, 2, 2]	Process 53	[3, 1, 2, 1]
Process 2	[1, 1, 2, 2]	Process 28	[2, 1, 2, 3]	Process 54	[3, 1, 2, 2]
Process 3	[1, 1, 2, 3]	Process 29	[2, 1, 3, 1]	Process 55	[3, 1, 2, 3]
Process 4	[1, 1, 3, 1]	Process 30	[2, 1, 3, 2]	Process 56	[3, 1, 3, 1]
Process 5	[1, 1, 3, 2]	Process 31	[2, 1, 3, 3]	Process 57	[3, 1, 3, 2]
Process 6	[1, 1, 3, 3]	Process 32	[2, 2, 1, 1]	Process 58	[3, 1, 3, 3]
Process 7	[1, 2, 1, 2]	Process 33	[2, 2, 1, 2]	Process 59	[3, 2, 1, 1]
Process 8	[1, 2, 1, 3]	Process 34	[2, 2, 1, 3]	Process 60	[3, 2, 1, 2]
Process 9	[1, 2, 2, 1]	Process 35	[2, 2, 2, 1]	Process 61	[3, 2, 1, 3]
Process 10	[1, 2, 2, 2]	Process 36	[2, 2, 2, 2]	Process 62	[3, 2, 2, 1]
Process 11	[1, 2, 2, 3]	Process 37	[2, 2, 2, 3]	Process 63	[3, 2, 2, 2]
Process 12	[1, 2, 3, 1]	Process 38	[2, 2, 3, 1]	Process 64	[3, 2, 2, 3]
Process 13	[1, 2, 3, 2]	Process 39	[2, 2, 3, 2]	Process 65	[3, 2, 3, 1]
Process 14	[1, 2, 3, 3]	Process 40	[2, 2, 3, 3]	Process 66	[3, 2, 3, 2]
Process 15	[1, 3, 1, 1]	Process 41	[2, 3, 1, 1]	Process 67	[3, 2, 3, 3]
Process 16	[1, 3, 2, 1]	Process 42	[2, 3, 1, 2]	Process 68	[3, 3, 1, 1]
Process 17	[1, 3, 1, 3]	Process 43	[2, 3, 1, 3]	Process 69	[3, 3, 1, 2]
Process 18	[1, 3, 2, 1]	Process 44	[2, 3, 2, 1]	Process 70	[3, 3, 1, 3]
Process 19	[1, 3, 2, 2]	Process 45	[2, 3, 2, 2]	Process 71	[3, 3, 2, 1]
Process 20	[1, 3, 2, 3]	Process 46	[2, 3, 2, 3]	Process 72	[3, 3, 2, 2]
Process 21	[1, 3, 3, 1]	Process 47	[2, 3, 3, 1]	Process 73	[3, 3, 2, 3]
Process 22	[1, 3, 3, 2]	Process 48	[2, 3, 3, 2]	Process 74	[3, 3, 3, 1]
Process 23	[2, 1, 1, 2]	Process 49	[2, 3, 3, 3]	Process 75	[3, 3, 3, 2]
Process 24	[2, 1, 2, 1]	Process 50	[3, 1, 1, 1]	Process 76	[3, 3, 3, 3]
Process 25	[2, 1, 2, 2]	Process 51	[3, 1, 1, 2]		
Process 26	[2, 1, 2, 3]	Process 52	[3, 1, 1, 3]		

in which the sum of capacity points is higher need to be verified, See Figure 9.4. Observe that the distribution of capacity points among bidders matters as well. In Process 50, for example, Bidder 1 has a capacity of 3, and the other bidders each have a capacity of 1. Since the first bidder has a high budget for the products and the other bidders have few capacity points left, this process model requires fewer properties to be verified. In total, we have verified 2,129,637 properties. The longest verification time required to verify properties in each process model varies between 6 to 14 seconds. In particular, it makes a big difference for verification time if a bidder still has capacity left or not. In the second case, verification turns out to be false very quickly. The average time to verify properties is about 1 second in all process models. The maximum time to do so is 14 seconds and occurs with Process Models 39, 40, and 49. Each model has a standard deviation of one to two seconds for verification times. Figure 9.5 represents the time required to verify all properties

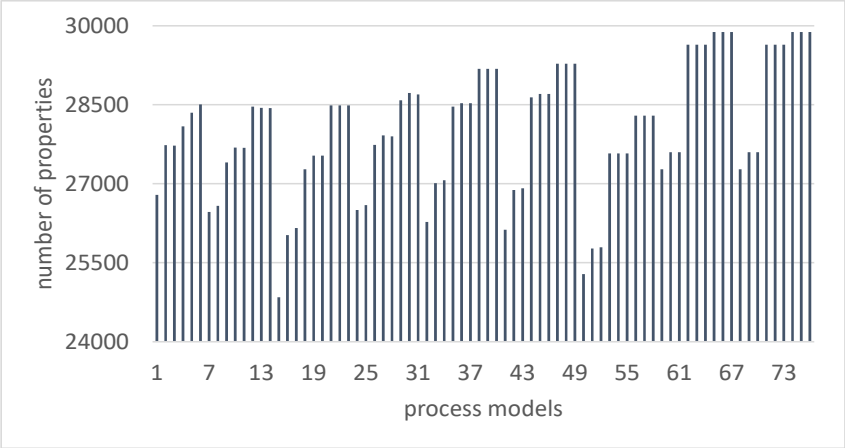


Figure 9.3: Number of properties verified in each process model

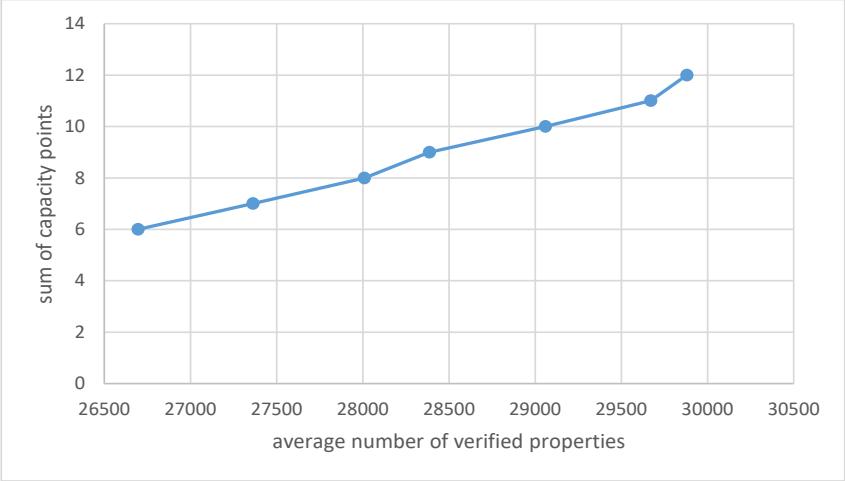


Figure 9.4: The average number of properties per total number of capacity points

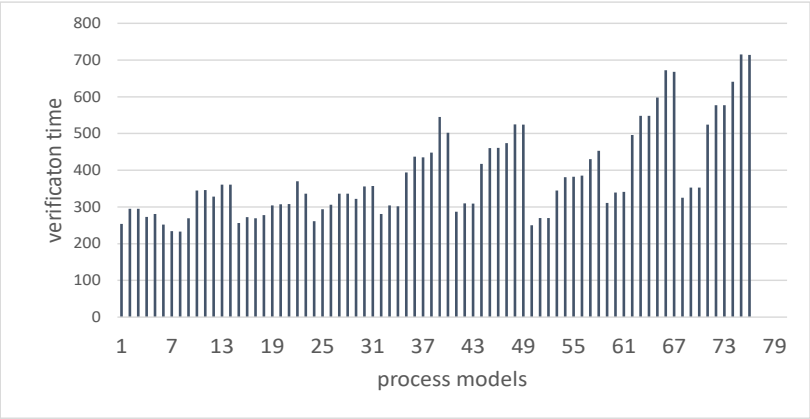


Figure 9.5: Time required to verify all properties in each process model

in each process model. This number tends to grow with the total number of capacity points, as does the total verification time. Verifying each process model has required almost 6 hours and 41 minutes on average to verify all properties, i.e., 487 hours in total to finish the experiment. Across all process models, the standard deviation of verification time has been around 2 hours.

9.3.3 Research Questions

In this section, we answer the research questions described in Section 9.1.

1. Which assignments of the capacity points lead to the lowest and highest revenue?

Figure 9.6 shows the lowest possible revenue when assigning different capacity points to the bidders. In general, the lowest revenue for higher total numbers of capacity points is relatively high. The lowest revenue is 19 and happens in Process 1, when capacity points are $[1, 1, 1, 3]$. Some of the lowest revenues, i.e., 22, 28, 33, and 34 never occur in any of the process models. The revenue of 26 only occurs once, when the capacity points are $[2, 3, 1, 1]$. The lowest revenue in Process Models 62 to 67 is higher than that of the other process models. Here it is 35. In Process Models 68 to 70, the revenue is 29, probably in line with Bidder 3 having only one capacity point. In Process Models 71 to 76 where this number is again 3, the lowest revenue becomes higher again. In the process models with the maximum lowest revenue, the capacity point of Bidder 1 is always 3, the ones of Bidder 2 and Bidder 3 vary between 2 and 3, and the one of Bidder 4 varies from 1 to 3. We see that the lowest revenue is maximal when Bidder 1 has 3 capacity points. This might be because the budget of Bidder 1 for certain products is higher than that of the other bidders, and assigning a higher capacity point to this bidder increases the revenue.

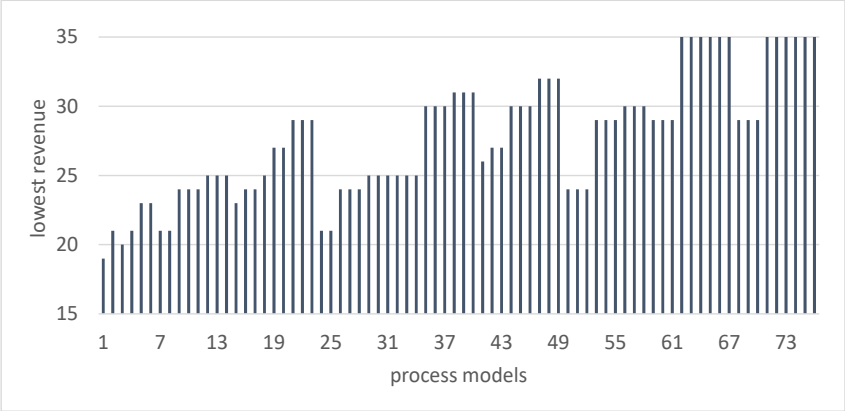


Figure 9.6: The lowest possible revenue of each process model

2. With a certain allocation of the capacity points, how often can the lowest revenue occur?

It is important for the auctioneer to know the lowest possible revenue before the auction. However, how often the lowest revenue can happen matters as well. Figure 9.7 represents the number of scenarios which lead to the lowest possible revenue. In general, when the lowest revenue increases, the number of scenarios yielding the lowest revenue increases as well. The lowest revenue in process 76 is higher than the lowest revenue of the other process models. In other words, the worst possible scenario is not too bad, as the revenue is 35. However, the number of scenarios which lead to the lowest revenue is relatively high. In total, 180 scenarios out of 720 lead to the lowest revenue. In contrast, Process 7, to give an example, has the minimal revenue of 21, but this only happens six times out of 720 scenarios.

3. Does increasing capacity points always increase the revenue?

As shown in Figure 9.6, the lowest revenue increases with the capacity points of bidders. This is because, when bidders have lower capacities, they cannot win a product, even though they can afford it, and a bidder with a lower budget gets the chance to win the product. But when the auctioneer increases the capacities, this becomes less frequent because the bidder with a higher budget has capacity left to win the product. However, increasing the number of capacities may lead to a monopoly. So, when the capacity points increases, the chance of earning a higher revenue is higher; at the same time, the chance of monopoly gets also higher. At this point, an auctioneer can trade off 'extent of monopolism' vs. 'expected revenue' and assign capacities accordingly.

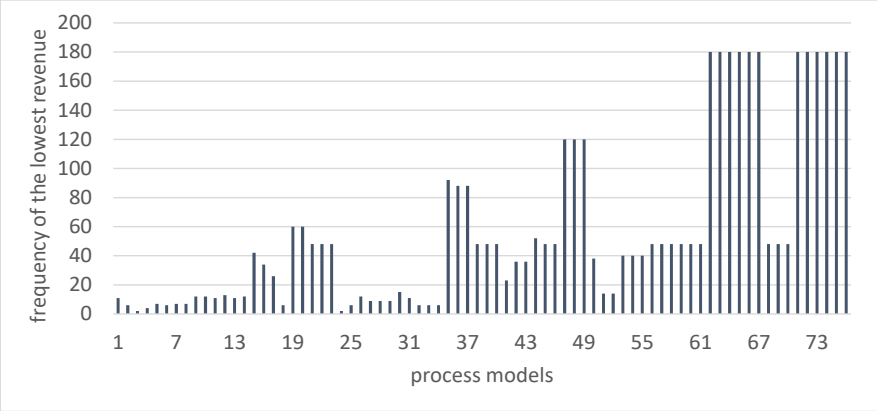


Figure 9.7: The frequency of happening the lowest revenue in each process model

4. What is the best assignment of capacity points to the bidders, i.e., making the worst possible revenue not too low?

As explained, the lowest revenue is maximum in Processes 62 to 67 and 71 to 76. On one side, one can take an allocation of capacity points from there to avoid bad outcomes. However, the process model where bidders have fewer capacity points (Process 69) might be a better choice, since it prevents from any monopoly to some extent. On the other side, the process models in which the lowest revenue is maximal lead to this lowest revenue more often. Put differently, the number of scenarios that lead to the lowest revenue is relatively high.

5. Does changing the capacity point of a single bidder change the outcome of the auction?

Another interesting observation is that increasing capacity points does not always change the revenue of the auctioneer. For example, the lowest revenue of the auction is 24 by assigning capacity points of $[3, 1, 1, 1]$. It is the same when we increase the capacity points of Bidder 4. So, when the auctioneer increases the capacity points to avoid the lowest revenues, it also matters which bidder obtains more points. Another example is that changing the capacity of only Bidder 1 by 1 point changes the lowest revenue of the auction from 23 to 25. This shows that the capacity of this bidder has a significant effect on the final prices.

9.4 Summary

The revenue of spectrum auctions has been an important source of income for governments. In this chapter, we have identified extreme outcomes and the factors leading to them in a systematic manner. We have done this by means of existing

verification techniques. In particular, we have focused on the lowest possible revenue of a spectrum auction, with different *capacity points*. We have compared the outcomes of process models with different capacity points and explained how an auction designer can take our analysis as a starting point to improve existing designs.

Part III

Conclusions

10. Conclusion

In this doctoral thesis, we have embarked on a comprehensive journey to tackle the challenges associated with the verification of data-value-aware process models. By following a systematic and step-by-step approach, we have made novel and significant contributions to the field of process model verification. Our research endeavors have focused on addressing the state-space explosion problem within data-value-aware processes, developing reduction techniques, and demonstrating the practical applicability of our findings through the verification of a real-world application.

Chapter 5 laid the foundation for our thesis by proposing a novel and generic method to verify *data-value-aware process models*. We recognized the importance of considering data values and their modifications in the verification process, especially in domains such as spectrum auctions. To achieve this, we enhanced the process models by incorporating information about data values, enabling the modeling of data-value conditions and functions within the state space. This enhancement facilitated the subsequent verification steps and allowed for the definition of data-value-centered properties as Computational Tree Logic (CTL) formulas.

In Chapter 6, our focus shifted towards mitigating the state-space explosion problem encountered when verifying data-value-aware process models. This problem arises due to the large number of process elements associated with data-value functions or conditions. To address this challenge, we introduced novel relevance functions that detect the elements necessary for verifying specific properties, known as *relevant elements*. By identifying and pruning the irrelevant elements, our reduction algorithms significantly reduced the size of the process models, thereby alleviating the state-space explosion issue. These reduction techniques provided a more scalable and efficient approach to process model verification.

Chapter 7 delved into the complexities arising from data-value conditions involving data objects with large domains. Verification in such scenarios often leads to state-space explosion due to the exponentially increasing number of possible data values. To overcome this challenge, we proposed a novel binary encoding technique that mapped data objects to a reduced number of places in the Petri Net representation. This encoding approach, leveraging the properties of binary numbers, achieved a logarithmic reduction in the state space while still supporting the entire domain

of data objects. By effectively capturing the semantics of data-value conditions, our approach provided a more efficient means of verifying data-value-aware process models with large data domains.

Building upon the reduction techniques introduced in Chapter 7, Chapter 8 extended our research to handle the state-space explosion caused by data-value functions. We employed Binary Decision Diagrams (BDD), a widely-used data structure in symbolic model checking, to represent the semantics of data-value functions within the Petri Net. By leveraging the existing reduction techniques available for BDDs, we were able to reduce the number of edges and nodes in the BDD representation, subsequently reducing the size of the Petri Net and state space. This approach provided an effective means of handling data-value functions and further enhanced the scalability of the verification process.

In Chapter 9, the practical applicability of our contributions was demonstrated through the verification of a real-world application, specifically the German 4G spectrum auction. This case study served as a tangible example to showcase the usefulness and effectiveness of our proposed techniques. By applying our verification methodologies, we were able to identify the lowest possible revenue scenario for the auction by manipulating the input parameter known as "capacity points." The insights gained from this verification process enabled auction designers to evaluate different auction designs and make informed decisions based on the expected outcomes.

The contributions made in this doctoral thesis significantly advance the field of process model verification, particularly in the realm of data-value-aware processes. Our novel techniques and methodologies address critical challenges, such as the state-space explosion problem and the verification of large data domains. By enhancing the efficiency and accuracy of the verification process, our research findings have broader implications for various domains beyond auction research. Industries dealing with data-value-aware processes can benefit from our contributions, enabling more reliable designs and improved decision-making.

11. Outlook

Building upon the achievements of the presented research, there are several promising directions for future work that can further enhance the verification of data-value-aware processes and address the challenges associated with state-space explosion and domain coverage. These future research avenues aim to strengthen the proposed methodologies, expand their applicability to different domains, and overcome limitations to enable more comprehensive and efficient verification of data-value-aware process models.

1. Abstraction Techniques for Data-Value-Aware Process Models with Large Domains: One of the primary challenges in verifying data-value-aware process models is the potential for state-space explosion, especially when dealing with large data domains. While the research has already proposed a reduction approach that prunes the process model based on relevant elements, further investigation is required to handle scenarios where the reduction technique outputs a data object with a large domain. In such cases, the verification may still suffer from state-space explosion due to the large number of values the relevant data object can take during process execution.

To address this challenge, future work should focus on developing advanced abstraction methods specifically tailored to handle data domains with a large number of values. These abstraction techniques would aim to capture the essential characteristics of the data domain while reducing its complexity, enabling more efficient verification. By combining these abstraction methods with the existing reduction technique, it would be possible to mitigate state-space explosion and achieve more scalable verification for data-value-aware process models.

2. Scalable Verification Techniques: As data-value-aware processes grow in complexity and scale, the computational requirements for their verification increase substantially. To overcome the limitations of sequential verification and reduce the overall verification time, future research should explore parallel and distributed verification techniques. These techniques involve partitioning the verification tasks, executing them concurrently, and leveraging the power of parallel and distributed computing resources to accelerate the verification process. Investigating efficient

partitioning strategies, load balancing mechanisms, and synchronization protocols would be essential to achieve effective parallelization of verification tasks.

3. Handling Dynamic Data-Value Changes: In many business processes, data values are subject to frequent modifications during process execution. These dynamic changes in data values can significantly impact the behavior and correctness of the process model. To ensure accurate verification results in such dynamic environments, future work should focus on addressing the challenges posed by dynamic data-value changes.

This research direction involves developing algorithms and methodologies to efficiently capture and represent dynamic data-value changes within the verification framework. It requires extending the existing verification techniques to handle dynamic updates to data values and their impact on the process model. By considering the temporal aspects of data-value changes and incorporating them into the verification process, it would be possible to provide more comprehensive analysis and accurate detection of issues in data-value-aware processes.

4. Real-World Application and Case Studies: While the presented research has demonstrated the effectiveness of the developed techniques in the context of spectrum auctions, future work should include extensive evaluations and case studies on a wider range of real-world applications. This would enable the validation of the proposed methodologies beyond the domain of spectrum auctions and provide insights into their practicality and effectiveness in diverse business processes.

Collaborating with industry partners and stakeholders would be instrumental in applying the verification techniques to different domains and gathering practical insights. By working closely with organizations that deal with complex business processes, researchers can gain a deeper understanding of the challenges and requirements in practical settings, leading to potential improvements and refinements to the proposed methods.

5. Integration with Existing Verification Approaches: To further enhance the applicability and effectiveness of the developed data-value-aware verification approaches, future work should explore the integration of these methodologies with other existing formal verification techniques and tools. This integration would aim to leverage the strengths of different verification methodologies and address the limitations of individual approaches.

Investigating the synergies between data-value-aware verification and other formal verification techniques, such as model checking, theorem proving, or abstract interpretation, can lead to more powerful and comprehensive verification frameworks. By combining different verification methodologies, researchers can leverage their complementary features and capabilities, enabling more accurate and efficient verification of data-value-aware process models.

By pursuing these future research directions, the work presented in the dissertation can make significant contributions to advancing the state of the art in process verification. The proposed methodologies and techniques have the potential to provide more efficient and accurate means of verifying data-value-aware processes across diverse domains, ensuring their correctness, efficiency, and quality in practical settings. Through continued research and exploration, these endeavors can contribute to the

development of robust and scalable verification frameworks that address the challenges of state-space explosion, domain coverage, and dynamic data-value changes in data-value-aware processes.

Bibliography

[1] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[2] E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, and H. Veith, *Model checking*. MIT press, 2018.

[3] S. B. Akers, "Binary decision diagrams", *IEEE Transactions on computers*, vol. 27, no. 06, pp. 509–516, 1978.

[4] R. E. Bryant, "Graph-based algorithms for boolean function manipulation", *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986.

[5] T. W. Hazlett, R. E. Muñoz, and D. B. Avanzini, "What really matters in spectrum allocation design", *Nw. J. Tech. & Intell. Prop.*, vol. 10, p. viii, 2011.

[6] E. Ordoni, J. Mülle, and K. Böhm, "Verifying workflow models with data values-a case study of smr spectrum auctions", 2019.

[7] W. M. Van der Aalst, "The application of petri nets to workflow management", *Journal of circuits, systems, and computers*, vol. 8, no. 01, pp. 21–66, 1998.

[8] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications", *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 8, no. 2, pp. 244–263, 1986.

[9] Y. Crama and P. L. Hammer, *Boolean functions: Theory, algorithms, and applications*. Cambridge University Press, 2011.

[10] OMG, *Business Process Model and Notation (BPMN), Version 2.0*, Object Management Group, 2011.

[11] E. A. Emerson, "Temporal and modal logic", in *Formal Models and Semantics*, Elsevier, 1990, pp. 995–1072.

[12] J. McMillan, "Why auction the spectrum?", *Telecommunications policy*, vol. 19, no. 3, pp. 191–199, 1995.

[13] P. Cramton *et al.*, "Spectrum auctions", *Handbook of telecommunications economics*, vol. 1, pp. 605–639, 2002.

[14] J. H. Kagel and D. Levin, "Behavior in multi-unit demand auctions: Experiments with uniform price and dynamic vickrey auctions", *Econometrica*, vol. 69, no. 2, pp. 413–454, 2001.

[15] C. Brunner, J. K. Goeree, C. A. Holt, and J. O. Ledyard, "An experimental test of flexible combinatorial spectrum auction formats", *American Economic Journal: Microeconomics*, vol. 2, no. 1, pp. 39–57, 2010.

[16] W. Vickrey, "Counterspeculation, auctions, and competitive sealed tenders", *J. Finance*, vol. 16, no. 1, pp. 8–37, 1961.

[17] V. Krishna, *Auction Theory*. Academic Press, 2009.

[18] C. W. Smith, "Auctions: From Walras to the real world", in *Explorations in Economic Sociology*, Russell Sage Foundation, 1993, ch. 7, pp. 176–192.

[19] O. Kirchkamp and J. P. Reiss, "Heterogeneous bids in auctions with rational and markdown bidderstheory and experiment", Friedrich Schiller University Jena and Max Planck Institute of Economics, Tech. Rep., 2008.

[20] S. Gandhi, C. Buragohain, L. Cao, H. Zheng, and S. Suri, "A general framework for wireless spectrum auctions", in *Proc. DySPAN*, 2007, pp. 22–33.

[21] E. Wolfstetter, "The swiss umts spectrum auction flop: Bad luck or bad design?", *Available at SSRN 279683*, 2001.

[22] M. Bichler, J. Goeree, S. Mayer, and P. Shabalin, "Spectrum auction design: Simple auctions for complex sales", *Telecommunications Policy*, vol. 38, no. 7, pp. 613–622, 2014.

[23] L. M. Ausubel, P. Cramton, and P. Milgrom, "The clock-proxy auction: A practical combinatorial auction design", *Handbook of Spectrum Auction Design*, pp. 120–140, 2006.

[24] P. Cramton and A. Ockenfels, *The german 4g spectrum auction: Design and behaviour*, 2017.

[25] D. Engelmann and V. Grimm, "Bidding behaviour in multi-unit auctions–an experimental investigation", *The Economic Journal*, vol. 119, no. 537, pp. 855–882, 2009.

[26] P. Milgrom, *Putting Auction Theory to Work*. Cambridge University Press, 2004.

[27] A. M. Kwasnica and K. Sherstyuk, "Multiunit auctions", *J. Econ. Surv.*, vol. 27, no. 3, pp. 461–490, 2013.

[28] A. Awad, G. Decker, and N. Lohmann, "Diagnosing and repairing data anomalies in process models", in *International Conference on Business Process Management*, Springer, 2009, pp. 5–16.

[29] S. Haarmann, K. Batoulis, and M. Weske, "Compliance checking for decision-aware process models", in *International Conference on Business Process Management*, Springer, 2018, pp. 494–506.

[30] M. d. Leoni, P. Felli, and M. Montali, "A holistic approach for soundness verification of decision-aware process models", in *International Conference on Conceptual Modeling*, Springer, 2018, pp. 219–235.

[31] P. Felli, M. de Leoni, and M. Montali, "Soundness verification of decision-aware process models with variable-to-variable conditions", in *2019 19th International Conference on Application of Concurrency to System Design (ACSD)*, IEEE, 2019, pp. 82–91.

[32] S. Ghilardi, A. Gianola, M. Montali, and A. Rivkin, "Petri nets with parameterised data", in *International Conference on Business Process Management*, Springer, 2020, pp. 55–74.

[33] D. Calvanese, S. Ghilardi, A. Gianola, M. Montali, and A. Rivkin, "Formal modeling and smt-based parameterized verification of data-aware bpmn", in *International Conference on Business Process Management*, Springer, 2019, pp. 157–175.

[34] S. Ranise and S. Ghilardi, "Backward reachability of array-based systems by smt solving: Termination and invariant synthesis", *Logical Methods in Computer Science*, vol. 6, 2010.

[35] H. Groefsema, N. van Beest, and A. A-Cervantes, "Efficient conditional compliance checking of business process models", *Computers in Industry*, vol. 115, p. 103 181, 2020.

[36] R. De Masellis, C. Di Francescomarino, C. Ghidini, M. Montali, and S. Tessaris, "Add data into business process verification: Bridging the gap between theory and practice", in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[37] A. Deutsch, R. Hull, F. Patrizi, and V. Vianu, "Automatic verification of data-centric business processes", in *Proceedings of the 12th international Conference on Database Theory*, 2009, pp. 252–267.

[38] A. Awad, M. Weidlich, and M. Weske, "Specification, verification and explanation of violation for data aware compliance rules", in *Service-oriented computing*, Springer, 2009, pp. 500–515.

[39] P. Felli, M. Montali, and S. Winkler, "Soundness of data-aware processes with arithmetic conditions", in *International Conference on Advanced Information Systems Engineering*, Springer, 2022, pp. 389–406.

[40] ——, "Linear-time verification of data-aware dynamic systems with arithmetic", *Proceedings of 36th AAAI*, 2022.

[41] ——, "Ctl* model checking for data-aware dynamic systems with arithmetic", *arXiv preprint arXiv:2205.08976*, 2022.

[42] D. Knuplesch, L. T. Ly, S. Rinderle-Ma, H. Pfeifer, and P. Dadam, "On enabling data-aware compliance checking of business process models", in *International Conference on Conceptual Modeling*, Springer, 2010, pp. 332–346.

[43] J. Harzmann, A. Meyer, and M. Weske, "Deciding data object relevance for business process model abstraction", in *International Conference on Conceptual Modeling*, Springer, 2013, pp. 121–129.

[44] S. Smirnov, H. A. Reijers, and M. Weske, "A semantic approach for business process model abstraction", in *International Conference on Advanced Information Systems Engineering*, Springer, 2011, pp. 497–511.

[45] S. Smirnov, H. A. Reijers, M. Weske, and T. Nugteren, "Business process model abstraction: A definition, catalog, and survey", *Distributed and Parallel Databases*, vol. 30, no. 1, pp. 63–99, 2012.

[46] A. Meyer and M. Weske, "Data support in process model abstraction", in *International Conference on Conceptual Modeling*, Springer, 2012, pp. 292–306.

[47] R. Bobrik, M. Reichert, and T. Bauer, "View-based process visualization", in *International Conference on Business Process Management*, Springer, 2007, pp. 88–95.

[48] A. Polyvyanyy, S. Smirnov, and M. Weske, "Business process model abstraction", in *Handbook on Business Process Management 1*, Springer, 2015, pp. 147–165.

[49] C. Tsagkani and A. Tsalgatidou, "Abstracting bpmn models", in *Proceedings of the 19th Panhellenic Conference on Informatics*, 2015, pp. 243–244.

[50] M. Ramos-Merino, L. M. Álvarez-Sabucedo, J. M. Santos-Gago, and F. de Arriba-Pérez, "A pattern based method for simplifying a bpmn process model", *Applied Sciences*, vol. 9, no. 11, p. 2322, 2019.

[51] B. Xu, J. Qian, X. Zhang, Z. Wu, and L. Chen, "A brief survey of program slicing", *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 2, pp. 1–36, 2005.

[52] S. Graf and H. Saidi, "Construction of abstract state graphs with pvs", in *International Conference on Computer Aided Verification*, Springer, 1997, pp. 72–83.

[53] W. Visser, S. Park, and J. Penix, "Using predicate abstraction to reduce object-oriented programs for model checking", in *Proceedings of the third workshop on Formal methods in software practice*, 2000, pp. 3–182.

[54] K. Schmidt, "Stubborn sets for standard properties", in *International Conference on Application and Theory of Petri nets*, Springer, 1999, pp. 46–65.

[55] R. Gerth, R. Kuiper, D. Peled, and W. Penczek, "A partial order approach to branching time logic model checking", *Information and Computation*, vol. 150, no. 2, pp. 132–152, 1999.

[56] K. Schmidt, "Stubborn sets for model checking the ef/ag fragment of ctl", *Fundamenta Informaticae*, vol. 43, no. 1-4, pp. 331–341, 2000.

[57] E. Ordoni, J. Mülle, and K. Böhm, "Veryfying workflow models with data values – a case study of smr spectrum auctions", in *IEEE Conference on Business Informatics*, IEEE, 2020, pp. 181–190.

[58] S. Von Stackelberg, S. Putze, J. Mülle, and K. Böhm, "Detecting data-flow errors in bpmn 2.0", *Open Journal of Information Systems (OJIS)*, vol. 1, no. 2, pp. 1–19, 2014.

[59] N. Lohmann, E. Verbeek, and R. Dijkman, "Petri net transformations for business processes–a survey", in *Transactions on petri nets and other models of concurrency II*, Springer, 2009, pp. 46–63.

[60] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in bpmn", *Information and Software technology*, vol. 50, no. 12, pp. 1281–1294, 2008.

[61] K. Schmidt, "Lola a low level analyser", in *International Conference on Application and Theory of Petri Nets*, Springer, 2000, pp. 465–474.

[62] J. Mülle, C. Tex, and K. Böhm, "A practical data-flow verification scheme for business processes", *Information Systems*, vol. 81, pp. 136–151, 2019.

[63] R. Mrasek, J. Mülle, and K. Böhm, "A new verification technique for large processes based on identification of relevant tasks", *Information Systems*, vol. 47, pp. 82–97, 2015.

[64] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press, 2004.

[65] J. Vanhatalo, H. Völzer, and J. Koehler, "The refined process structure tree", *Data & Knowledge Engineering*, vol. 68, no. 9, pp. 793–818, 2009.

[66] B. Kiepuszewski, A. H. M. t. Hofstede, and C. J. Bussler, "On structured workflow modelling", in *International conference on advanced information systems engineering*, Springer, 2000, pp. 431–445.

[67] J. Koehler and R. Hauser, "Untangling unstructured cyclic flows–a solution based on continuations", in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, Springer, 2004, pp. 121–138.

[68] A. Polyvyanyy, L. Garcia-Bañuelos, and M. Dumas, "Structuring acyclic process models", *Information Systems*, vol. 37, no. 6, pp. 518–538, 2012.

[69] R. Eshuis and A. Kumar, "Converting unstructured into semi-structured process models", *Data & Knowledge Engineering*, vol. 101, pp. 43–61, 2016.

[70] A. Polyvyanyy, J. Vanhatalo, and H. Völzer, "Simplified computation and generalization of the refined process structure tree", in *International Workshop on Web Services and Formal Methods*, Springer, 2010, pp. 25–41.

[71] M. Weidlich, A. Polyvyanyy, J. Mendling, and M. Weske, "Causal behavioural profiles–efficient computation, applications, and evaluation", *Fundamenta Informaticae*, vol. 113, no. 3-4, pp. 399–435, 2011.

[72] M. Montali *et al.*, "Verification from declarative specifications using logic programming", in *International Conference on Logic Programming*, Springer, 2008, pp. 440–454.

[73] E. J. McCluskey, "Minimization of boolean functions", *The Bell System Technical Journal*, vol. 35, no. 6, pp. 1417–1444, 1956.

[74] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient sat solver", in *Proceedings of the 38th annual Design Automation Conference*, 2001, pp. 530–535.

[75] D. Déharbe, P. Fontaine, D. Le Berre, and B. Mazure, "Computing prime implicants", in *2013 Formal Methods in Computer-Aided Design*, IEEE, 2013, pp. 46–52.

Affidavit

Ich versichere hiermit wahrheitsgemäß, die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des Karlsruher Instituts für Technologie (KIT) zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, 24.06.2023 _____

Elaheh Ordoni