Efficient Verification of Process Models Supporting Modifications of Data Values

Elaheh Ordoni*, Jutta Mülle†, Kuan Yang‡, Klemens Böhm§
Institute for Program Structures and Data Organization
Karlsruhe Institute of Technology
76131 Karlsruhe, Germany
Email: {*elaheh.ordoni, †jutta.muelle, ‡kuan.yang, §klemens.boehm}@kit.edu

***Abstract*—Verification techniques detect undesirable behaviour of process models before their execution. In many use cases, *data-value functions* are essential. A data-value function modifies the values of data objects in a process model, e.g., increases the price of a product. Supporting such functions when verifying process models is challenging. This is because data objects with large domains often lead to state-space explosion. In this paper, to address this issue, we propose a novel approach using a binary encoding technique. We make use of *Binary Decision Diagrams* (BDD) to map the semantics of data-value functions into a Petri Net. This allows using the existing BDD reduction techniques to reduce the number of edges and nodes in BDDs and, ultimately, of places and transitions in Petri Nets. One can now map process models with data-value functions into much smaller Petri Nets, whose verification is feasible. We show that this is indeed the case, by verifying properties of an important real-world application, the German 4G spectrum auction.**

I. Introduction

Motivation. Verification methods detect unexpected behaviour of business process models before their execution [1]. In many cases, verification hinges on *data values* associated with a process. A data value is a value in a respective domain, e. g., \$1000 as the price of a product. To highlight the role of data values in verification, think of spectrum auctions [2]. Data values, such as the price of a product or a bid, determine the outcome of the auction and other properties of it, like the revenue of the auctioneer or whether one bidder can obtain all available goods (spectra in this case). 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 1. *Figure 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. Verification requires a so-called *data-aware process model*. To verify such a model, 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

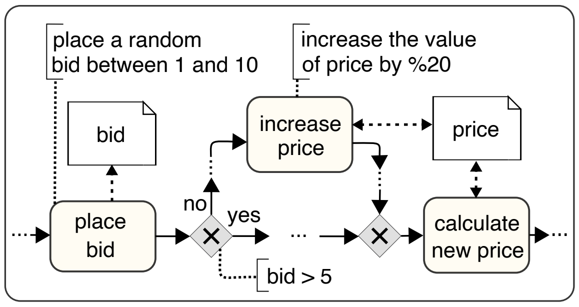
place a random bid between 1 and 10
increase the value of price by %20
bid
increase price
price
no
yes
place bid
...
calculate new price
bid > 5

Fig. 1. Parts of an auction process modeled in BPMN

modifies the value of a data object during process execution, e.g., increases the price of a product.

There exist techniques to verify so-called *data-aware process models* [3], [4]. However, they suffer from state-space explosion [5], 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 question is how to reduce the state space of data-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 [6], [7]. 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 [8]. 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 2. *Consider the process model shown in Figure 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 paper 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-aware process model into Petri Nets using the rules in [9]. It then maps each data-value function into Petri Nets – and this mapping is the core contribution of our paper. 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 [3], [10]. 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 [11]. 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 [3]. The reduced state space allows deriving important values, like *auctioneer's revenue*, i. e., the sum of the final prices of the products [12], in the worst case. Auctions such as the Dutch UMTS auction have had disastrous outcomes for the auctioneers [13]. Being able to detect such outcomes in advance is a significant contribution to auction design.

a) Paper outline: Section II explains spectrum auctions. Section III provides definitions. Section IV features our approach. Section V is our evaluation. Section VI covers related work, and Section VII concludes.

II. Simultaneous Multi-Round (SMR) Auction

Simultaneous Multi-Round (SMR) auctions have been the standard format to allocate spectrum licenses to bidders for more than two decades [14]. This auction type allows selling several products, e.g., spectrum licenses, after several rounds of bidding. At the beginning of an auction, the auctioneer specifies a *reserve price* for each product, i.e., its lowest acceptable price. In each round, each bidder may choose to bid on zero, one, or multiple products simultaneously. In the type of auction we analyse, each bidder has a budget for each product. The budget reflects the valuation of the bidder for this product. Bidders cannot use leftover budget from one product for another product. In addition, bidders make separate bids for each product, i.e., they cannot bid on bundles of products. This is different from combinatorial auctions. Next, there is a so-called *capacity rule* [15]: Each bidder has a *capacity*, the maximum number of products he or she may win. This rule prevents bidders from winning too many items. In spectrum auctions, this guarantees a certain number of bidders awarded and prevents bidders from forming a monopoly. After bidding finishes in a certain round, the highest bid for each product will be its reserve price in the following round. This bid is announced to all bidders, while other bids are not disclosed. Bidders also do not know the bids of their competitors from the current round. The auction ends when there is no new bid for any product in a certain round. For each product, the bidder with the highest standing bid is the winner. The BPMN model of the SMR auction is represented in Appendix A.

III. Preliminaries

This section introduces the notation used in this paper. We define data-aware process models supporting modification of data values, cf. Definition 3, and use Petri Nets as a language for verification, cf. Definition 4.

Definition 1 (Process Domain [16]). *A Process Domain $\mathcal{D}$ is a tuple $\mathcal{D} = (\mathbb{A}, \mathbb{E}, \mathcal{O}, \mathbb{D}, dom)$ where*

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

We also define that a data value is a value in the domain of a data object.

Definition 2 (Data-Value Function). *Let $\mathcal{D}$ be a process domain. A Data-Value Function is a function with data domains of $\mathcal{D}$ as input and output parameters. We define DF as the set of all data-value functions over elements of $\mathcal{D}$, i.e.: $DF : \times_{i=1}^{n} D_i \to \times_{j=1}^{m} D_j$, where $D_i, D_j \in \mathbb{D}$.*

Definition 3 (Process Graph [16]). *Let $\mathcal{D}$ be a process domain. Then a process graph is a tuple $P = (N, F, O, I, type, EnhancementA, A_{enh})$ where*

- *$N = A_P \cup E_P \cup G_P$ is a finite set of nodes that is partitioned into the set of activities A_P, the set of events E_P and the set of gateways G_P.*
- *$F \subseteq N \times N$ is the sequence flow relation between nodes.*
- *$O \subseteq \mathcal{O}$ is a finite set of data objects.*
- *$I \subseteq O \times N \cup N \times O$ is the data flow relation between nodes and data objects.*
- *$type : A_P \cup E_P \to \mathbb{A} \cup \mathbb{E}$ is a function assigning an activity type to each activity in P and an event type to each event in P such that $a \in A_P \Rightarrow type(a) \in \mathbb{A}$ and $e \in E_P \Rightarrow type(e) \in \mathbb{E}$.*
- *$EnhancementA : T_P \cup E_P \to DVFuncs$ is a function assigning a data-value function $f : I \to O$ to an Activity/Event t, where I is the domains of the data*

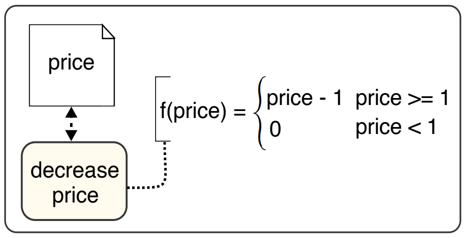

Fig. 2. An example of activity enhanced with *Enhancement*A

objects in $InputSet(t)$ and O is the domains of the data objects in $OutputSet(t)$.

- *A_{enh} is the set of activities and events in P enhanced with EnhancementA.*

Example 3. *Figure 2 shows the enhancement of Activity "decrease price" to reduce the price of a product. The activity reads the price, reduces its value by one, and writes the modified value to the same data object.*

Definition 4. *(Petri Nets [17]) A Petri Net is a triple (S, T, W) where:*

- *S is a finite set of places, represented as circles*
- *T is a finite set of transitions, represented as rectangles. $S \cap T = \varnothing$*
- *$W \subseteq (S \times T) \cup (T \times S)$ is a set of directed arcs (flow relation).*

A marking of a Petri Net is a mapping: $M : P \to N$ which assigns a non-negative number of tokens to each place. M_0 is the initial marking. For each transition there are directly preceding places called input places: $\bullet t = \{s \in S \mid W(s,t) > 0\}$ and subsequent places called output places: $t\bullet = \{s \in S | W(t,s) > 0\}$. A transition t can fire when all its input places have enough tokens; firing leads to a new state M. M results from M by t consuming a token from each of its input places and producing a token in each of its output places. The set of all states reachable from the initial state M_0 is the state space of the Petri Net.

IV. Our Approach

This paper proposes an efficient verification scheme for data-aware process models supporting data-value functions. We start to map the control flow of a process model to Petri Nets using the rules in [9]. However, this mapping only covers the "control flow". Thus, our approach extends this mapping with the semantics of data-value functions, a core contribution of this paper, as follows:

1) We map each data object to a set of places. Any distribution of tokens in these places represents a data value (Section IV-A).
2) Given these places, we map each data-value function to a Petri Net. Each of these mappings represents how a data-value function modifies values of a data object in the Petri Net (Section IV-B).
3) We merge the "control flow" Petri Net and the mappings of data-value functions (Section IV-C). The result is the transformation of a data-aware process model to a Petri Net that includes the semantics of modifications of data values.
4) We define properties referring to data values that will be verified against the resulting Petri Net (Section IV-D).

Given the properties and the final Petri Net, an off-the-shelf model checker, e.g., Lola [18], can then verify properties of the data-aware process model. We use plain Petri Nets as a target for the mapping because of the availability of respective efficient verification techniques [17].

A. Mapping of Data Objects to Petri Nets

In any state, a data object takes one value from its domain, but not more. So one can represent different values using the same places in a Petri Net. 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 describe how to map a bit into a Petri Net. Then we explain the mapping of other values.
Mapping of a bit b_i. We represent a bit b_i in a Petri Net with two places $p.\bar{b}_i$ and $p.b_i$. Place $p.b_i$ ($p.\bar{b}_i$) takes a token when $b_i = 1$ ($b_i = 0$). We call these places 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 4. *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* $= 262,144 = 2^{18}$.

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

B. Mapping of Data-Value Functions to Petri Nets

To verify a data-aware process model, it is essential to include the semantics of data-value functions in the Petri Net, i. e., capturing it in the state space of the process model. Consider Data-Value Function $f : D \to D'$ where D is domain $[0 \mathrel{..} n-1]$ and D' is domain $[0 \mathrel{..} m-1]$. 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 [3], [10]. This results in n new transitions to map Data-Value Function f into a Petri Net. But our intention is to reduce the number of transitions required.

a) Data-Value Functions to Truth Tables.: The first step to map a data-value function into Petri Nets is to build its truth table, i.e., a table that assigns each input value of the function to its corresponding output value.

Example 5. *Consider Data-Value Function $f(dv) : D \to D'$, where D has Domain $[0 \mathrel{..} 3]$, D' has Domain $[0 \mathrel{..} 1]$, and*
$$f(dv) = \begin{cases} 0 & dv \leqslant 0 \\ 1 & dv > 0 \end{cases}.$$
One needs two bits (b_1 and b_0) to represent the input values

TABLE I
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

and one bit (q_0) to represent the output values. Table I is the truth table of Function f.

b) Truth Tables to Ordered Decision Diagrams: Given the truth table of a data-value function, we derive its *Binary Decision Diagram* (BDD) [19]. A Binary decision diagram is a data structure representing a truth table as a directed acyclic graph, a compressed form of a decision tree. The occurrences of decision variables in the graph fulfill an ordering constraint, yielding an *Ordered Binary Decision Diagram* (OBDD). The reason behind transforming a truth table to its OBDD is that: (1) OBDDs are a good basis for effective symbolic model checking, and (2) one can apply an existing reduction algorithm [20] to reduce any OBDD. The reduced OBDD allows reducing the Petri Net and the state space of the process models, as we will explain.
Suppose $f : D \to D'$ is a data-value function, where D and D' have Domain $[0 .. n]$ and $[0 .. m]$, respectively. The truth table of f contains bits b_i, where $0 \leqslant i \leqslant \log_2 n$, to represent the input values and bits q_j, where $0 \leqslant j \leqslant \log_2 m$, to represent the output 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 6. *Let $f : D \to D'$ be a data-value function as in Example 5 and its truth table. The input of f has Domain $[0 .. 3]$, which is represented by b_0 and b_1. Figure 3 shows the OBDD of f. The dashed edges represent zero values of the truth table.*

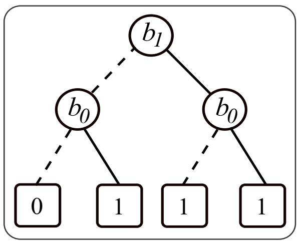

Fig. 3. Example of an OBDD

c) Reduction of Ordered Binary Decision Diagrams: To reduce the ordered binary decision diagrams, we use Algorithm 1 provided by [20]. The algorithm takes two steps: (1) It eliminates all duplicate leafs, i.e., for a duplicate zero-leaf (or one-leaf), redirects all incoming edges to only one of them. (2) It eliminates 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 redirects all incoming edges to w to v.

Algorithm 1 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 isomorphic node(s).
5:     if OBDD contains double edges then
6:         eliminate and re-direct all nodes w/ double edges.
```

Example 7. *Figure 4 illustrates the steps of Algorithm 1 to reduce the OBDD obtained from Example 6. The unreduced OBDD consists of one zero-leaf and four one-leaf nodes, Figure 4 (a). In such a case, the algorithm eliminates two of the one-leaf nodes while keeping only one. Then it redirects the edges from b_0 to this remaining one-leaf node. See Figure 4 (b). The gray node here has a double edge. This means that whether the value of b_0 is one or zero, the output is the same. So the algorithm eliminates this node and redirects the edges accordingly. See Figure 4 (c).*

d) Reduced Ordered Binary Decision Diagrams to Petri Nets: The transformation of reduced OBDD into Petri Nets is straightforward. 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 $p.\overline{b}_i$, respectively. Note that we have already created these places with our binary encoding, see Section 4. 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. We use this place to connect the mappings of different inner nodes together. Figure 5(a) shows the transformation of an OBDD inner node to a Petri Net. The transformation of a leaf node is the same as the one of an inner node, but Transitions $t.b_i$ and $t.\overline{b}_i$ are connected to the b_j*-places*, according to the truth table. Figure 5(b) shows the transformation of a leaf node of a reduced OBDD.

C. Merge Mappings of Data Functions and Control Flow

This step connects the "control flow" Petri Net and the mappings of data-value functions. The "control flow" Petri Net has a single transition t for each activity/event of the process model [9]. If the activity/event is assigned to a Data-Value Function f, Transition t fires only after f has been applied to data values. To modify data values according to f in the Petri Net, we connect Place $p.end$ from the mapping of f to Transition t. Thus, the token of Place $p.f$ is a precondition for the execution of t, i. e., t can fire only when f has modified the data values.

Example 8. *Figure 6 shows the mapping of the reduced OBDD in Example 7 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 one connected to the corresponding place. Transition $t.\overline{b_1}$ fires only when $p.\overline{b_1}$ is marked, i.e., $\overline{b_1} = 1$. Using our binary-encoding technique, model checking becomes*

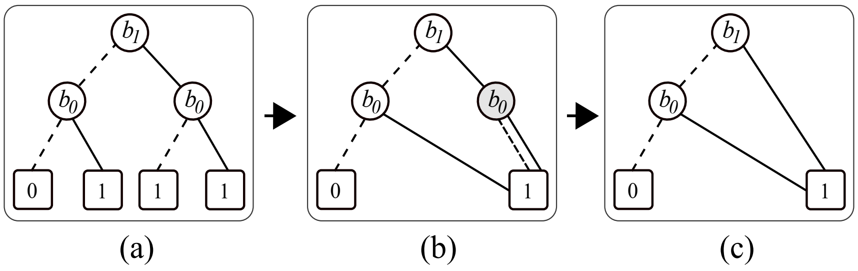

Fig. 4. The steps of Algorithm 1 to reduce OBDD of Example 7

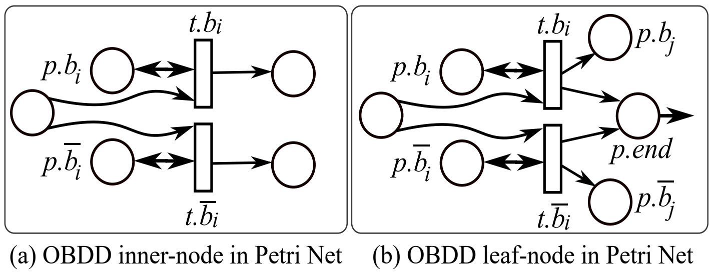
t.b_i
p.b_i
t.b̄_i
p.b̄_i
p.b_j
p.end
p.b̄_j
(a) OBDD inner-node in Petri Net
(b) OBDD leaf-node in Petri Net

Fig. 5. Transformation of OBDD nodes to Petri Nets

practical. If $t.\overline{b_1}$ fires, the model checker does not explore other states resulting from the highlighted part in Figure 6. This is 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.

D. Specification of Properties

The result of the transformation is a Petri Net of the process model representing the semantics of data-value functions. For the verification envisioned, one must specify data-value centered properties in a formal language such as Computation Tree Logic (CTL) [21]. In the following, we show how such properties can look like in CTL.

Definition 5. *(Data Property). Let P be a process graph. A Data Property ϕ is a CTL formula in which an atomic formula refers to either*

- *a node in $A_P \cup E_P$, or*
- *a data value of P.*

Example 9. *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, "$product.1.price.10$" is the "price" of 10 of "product.1". The atomic formula "$e.end$" is an end event, i. e., represents the end of the process.

We use properties analogous to Example 9 to detect the lowest final price of each product, and thus, the lowest revenue of the auctioneer. However, due to the capacity rule, the order of products in which one verifies the lowest final prices matters. This is because when a bidder wins a product at its lowest final price, their capacity point drops, which reduces the possibility to bid for and win the other products.

Example 10. *To detect the lowest revenue, we verify the lowest final prices of "product.1", "product.2", and "product.3", respectively. However, "product.1" might have a lower final price if we verify its final price after verifying the final price of the other products, i. e., a bidder with a lower budget might win this product. This is because the bidders with higher budgets might have already won other products and have no capacity points left for "product.1".*

Each product's final price depends on the order in which the final prices of the other products are verified. Thus, we have to check the lowest revenue for all the possible combinations of products in order to detect the final lowest revenue, i.e., we have calculated the lowest *auctioneer's revenue* in $6! = 720$ combinations.

V. Evaluation

Our evaluation studies how our approach affects the size of Petri Nets resulting from process models with different data-value functions. We also 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 domain sizes. Second, we study whether our approach facilitates the verification of a real-world application, the 4G German spectrum auction of the 800 MHz band [11].

A. Comparison of Data-Value Functions

We use four data-value functions for our evaluation and calculate the number of places and transitions in the Petri net.

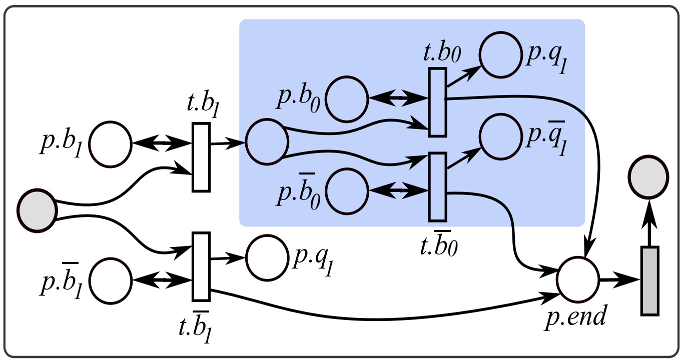

Fig. 6. Example of OBDD into Petri Nets

These four functions are: $f(x) = x + 2$, $f(x, y) = x + y$, $f(x) = x \times 1.2$, and $f(x, y) = max(x, y)$. To study the scalability of our approach, we vary the size of the input domain of each function from $[0 .. 2^{10}]$ to $[0 .. 2^{20}]$. Table V-A shows how many places and transitions are needed to transform each function to a Petri net with varying domain sizes. Multiplication is more expensive than the other functions, i.e., it requires more places and transitions. In general, the number of places and transitions grows logarithmically with the size of the data domain. This is a significant improvement over existing work [3], where these numbers grow linearly with the size of the domain. For example, 2^{20} places and transitions are needed to transform Data-Value Function $y = x + 2$ when $domain(x) = [0 .. 2^{20}]$ using the existing approaches.

B. Simultaneous Multi-Round Auction

We have chosen spectrum auctions as the use case for our evaluation for three reasons: (1) Spectrum auctions have a significant impact on the economy. They earned 50.8 billion Euros in Germany in 2000 [22]. (2) Verification of auctions allows detecting 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 larger than $[0 .. 10]$, due to state-space explosion [3]. (3) Modification of data values, e.g., increasing the price of bids, are essential to verify properties of auction models. In our evaluation, we focus the lowest possible *revenue* for the auctioneer. We also study the run time of the verification process.

a) Evaluation Setting: We model the SMR auction in BPMN. Table III shows the number of BPMN elements of the SMR auction model. The process model consists of four bidders who compete to win six products – this has been exactly the size of the German 4G spectrum auction to sell the 800 MHz band [11]. Similarly to [12], we assign a random budget to each bidder for a certain product in the range $[2; 100]$. 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 a capacity point of 2, Bidders 2 and 4 have capacity points of 3 and 1, respectively. This assignment of capacity points is according to the given assignment at the 4G German spectrum auction.

b) Transformation of SMR Process Model to Petri Nets: We have implemented a framework to verify process models of SMR auctions with our binary approach. The input of our framework is a data property and a process model in BPMN format. The SMR process model in the form of BPMN consists of data-value functions such as *get maximum price*, *place bids*, and *decrease capacity*. Our framework takes this process model and transforms it to a Petri Net. Table IV shows the size of mappings created to transform a single data-value function into Petri Nets. Table V lists the number of places and transitions with our approach and compares this with the naive transformation [3]. The table reveals that our approach transforms the SMR process model to a much smaller Petri Net and state space than the naive approach. We are able to generate a Petri Net including the semantics of modifications of data values with only 576 places and 786 transitions, while these numbers originally are 2912 and 15421, respectively.

c) Verification of SMR Process Models: Given the reduced state space, it is now possible to verify properties of SMR auctions with reasonably large domains. We detect the lowest possible revenue of the auctioneer. 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, analogous to Example 9.

If the property is not satisfied, we verify a new one with an increased price, until there is a state that fulfills the property. The sum of the lowest prices found for each product is the lowest *auctioneer's revenue*. As discussed in Section IV-D, we have to calculate the lowest revenue in 720 different combinations of products. The lowest revenue among all combinations is the lowest final *auctioneer's revenue*. To achieve the lowest final revenue, we have verified 130,292 properties using the model checking algorithm of LoLA [18]. The average time for this verification has been less than 7 seconds, while this simply has not been possible before. Our verification also reveals 13 different assignment of products to bidders which lead to the lowest final revenue. Table VI shows some of these assignments. An auction designer can now take this as a starting point for improvement. For instance, one can

TABLE II
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

TABLE III
THE NUMBER OF BPMN ELEMENTS IN BPMN MODEL OF THE SMR AUCTION

activities	gateways	annotations	data objects	data association
73	303	319	66	128

TABLE IV
THE NUMBER OF PLACES AND TRANSITIONS REQUIRED TO MAP FUNCTIONS

type of function	places	transitions
get maximum price	157	285
place bids	104	188
decrease capacity	10	15

TABLE V
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

detect the lowest possible revenue for an auction with different *capacity points* of bidders and apply our method to alternative designs and compare the results.

VI. RELATED WORK

There is a growing body of research addressing the data perspective of workflow verification, see [23] for an overview. In the following, we only refer to work that deals with modification of data values.

[8] enables verification of process models supporting modification of data values. They model the modification using decision tables and then transform the decision-aware processes to colored Petri Nets and use temporal logic model checking to verify properties. This approach is not effective when the decision tables are big, i.e., same as the size of input variables. [24] and [25] also provide an approach to verify decision-aware process models. However, they only verify structural properties, i.e., verification of behavioural properties is not a topic there. A recent method [26] provides a parameterized verification of data-aware BPMN called DAB. There, the process affects data in three possible ways: *Insert&Set*, *Delete&Set* and *Conditional update*. By encoding DABs in an “array-based artifact system framework”, as called by the inventors, SMT-based model checking of array-based systems [27] can be used to verify safety properties of DABs. However, this approach can only specify updates of single data values possibly under conditions, by so-called conditional update specifications. An example of a conditional update is to update the price of a product to 6 if its current price is 5. In [28], the authors present an approach to apply data-centric verification to activity-centric processes. The focus there is on reachability analysis, and there is no checking of arbitrary logic formulas as in our approach. A so-called artifact system [29] 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 auctions.

Various reduction techniques on the level of (1) state space or (2) process models have been proposed to deal with state-space explosion. Approaches like [30], [31] fall into the first category. However, when activities modify values of a data object with a large domain, the state space cannot even be generated. Such reductions might work here – in combination with our approach, but not in isolation. Regarding the second category, there are approaches to detect the relevant data objects used in a process model and remove the irrelevant ones [32], [33]. But their definition of relevance is not valid when control-flow elements manipulate data objects during process execution. Besides this, they do not address state-space explosion when a data object with a large domain is relevant. The authors in [34] transform a data-aware process model into a *Petri Net with Data*. Then they decompose the Petri Net into smaller nets that can be verified more efficiently. However, the state-space explosion can still occur if the subnets contain

TABLE VI
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
final price	60	80	70	70	90	60
winner	bidder 3	bidder 1	bidder 2	bidder 2	bidder 1	bidder 2
final price	70	60	80	70	90	60
winner	bidder 2	bidder 3	bidder 1	bidder 2	bidder 1	bidder 2
final price	70	80	80	70	70	60
winner	bidder 2	bidder 1	bidder 1	bidder 2	bidder 4	bidder 2
final price	70	60	80	70	90	60
winner	bidder 2	bidder 3	bidder 1	bidder 2	bidder 1	bidder 2
final price	70	80	80	70	70	60
winner	bidder 2	bidder 1	bidder 1	bidder 2	bidder 4	bidder 2
final price	60	80	70	70	90	60
winner	bidder 3	bidder 1	bidder 2	bidder 2	bidder 1	bidder 2

activities that modify the values of data objects with large domains. [16] features abstractions from the data domain in the form of intervals. They handle data conditions of XOR gateways in the process model. But they do not consider modifications of data objects used in these conditions.

VII. Conclusions

Verification of process models with modifications of data with large domains is crucial in many settings. Existing techniques often suffer from state-space explosion. In this paper, we have introduced a novel approach to overcome this challenge. We represent data-value functions as Ordered Binary Decision Diagrams (OBDDs). This allows to apply state-of-the-art techniques to reduce OBDDs and, thus, the size of Petri Nets and of the state space of the process models. Our approach allows verifying properties of real-world applications. We have showcased this with the German 4G spectrum auction. We have derived the lowest possible revenue of the auction with reasonable domain sizes. This has not been possible before and is an important milestone from the perspective of auction designers.

Appendix

Figure 7 shows a simplified SMR auction in BPMN notation according to [35]. There are three subprocesses in this model. During the first subprocess, the bidder is assessed for their ability to afford a certain license that they haven't yet won (availability of bidders). If one or more qualified bidders can place a bid in Subprocess *bids of every bidder*, the auction continues. Activity *decrease capacity* decreases the bidder's capacity after winning the product. Activity *issue bid* generates a bid based on the price of the product and the budget of the qualified bidder. Activity *remove bid* removes the bid if the bidder does not have any remaining capacity points. Based on the existing bids, the subprocess *winner determination* outputs the new prices and the winners. A repeat of each of these three subprocesses occurs until no more bids are placed.

References

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

[2] M. Bichler and J. K. Goeree, *Handbook of spectrum auction design*. Cambridge University Press, 2017.

[3] 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*, pp. 181–190, IEEE, 2020.

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

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

[6] 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.

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

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

[9] 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.

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

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

[12] 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.

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

[14] P. Milgrom and P. R. Milgrom, *Putting auction theory to work*. Cambridge University Press, 2004.

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

[16] 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*, pp. 332–346, Springer, 2010.

[17] 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.

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

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

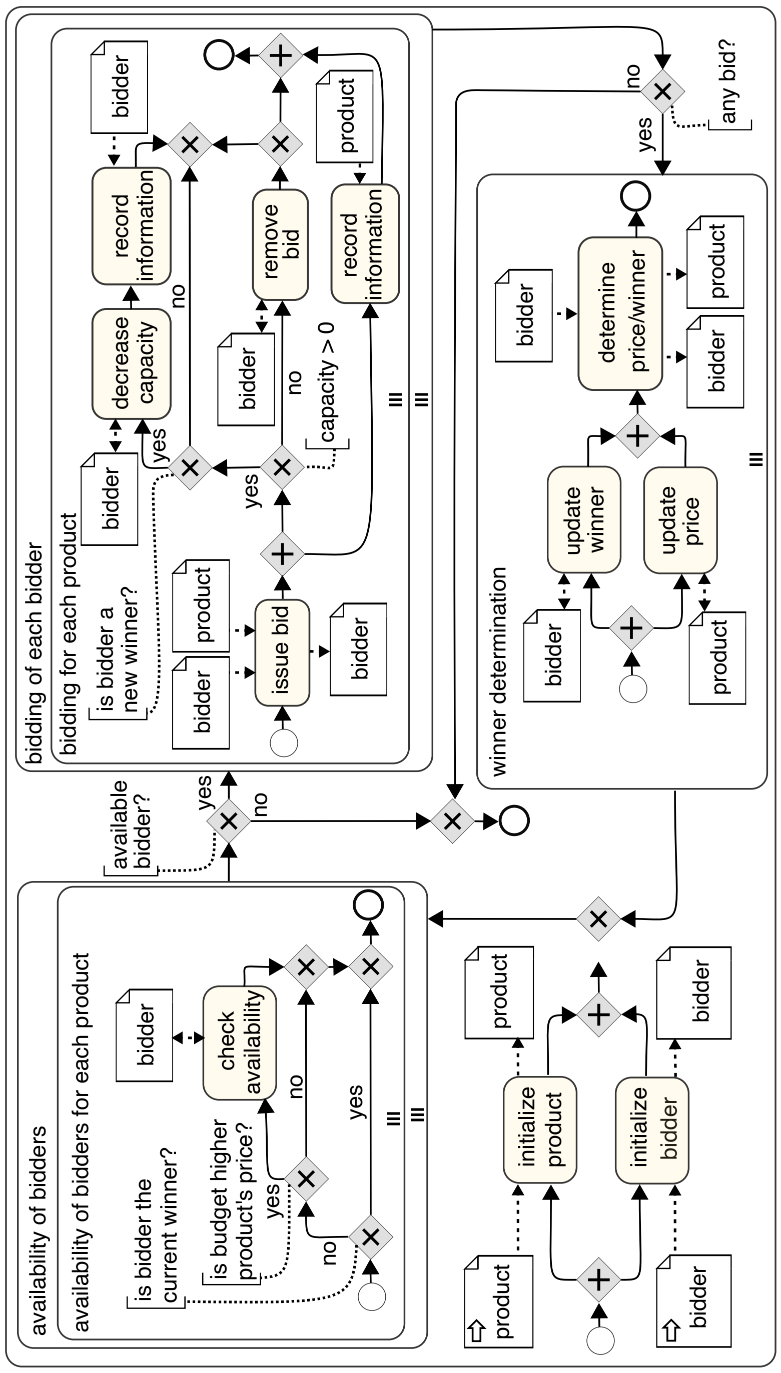

Fig. 7. Simplified Process Model of Simultaneous Multi-Round Auction in BPMN notation

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

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

[22] 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.

[23] R. Dell’Aversana, “Verification of data aware business process models: a methodological survey of research results and challenges,” in *Distributed Computing and Artificial Intelligence, 12th International Conference*, pp. 393–397, Springer, 2015.

[24] 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*, pp. 219–235, Springer, 2018.

[25] 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)*, pp. 82–91, IEEE, 2019.

[26] 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*, pp. 157–175, Springer, 2019.

[27] 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.

[28] 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.

[29] 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*, pp. 252–267, 2009.

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

[31] R. Gerth, R. Kuiper, D. Peled, and W. Penczek, “A partial order approach to branching time logic model checking,” in *Proceedings Third Israel Symposium on the Theory of Computing and Systems*, pp. 130–139, IEEE, 1995.

[32] 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.

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

[34] M. de Leoni, J. Munoz-Gama, J. Carmona, and W. M. van der Aalst, “Decomposing alignment-based conformance checking of data-aware process models,” in *OTM Confederated International Conferences” On the Move to Meaningful Internet Systems”*, pp. 3–20, Springer, 2014.

[35] E. Ordoni, J. Bach, and A.-K. Fleck, “Analyzing and predicting verification of data-aware process models – a case study with spectrum auctions,” *IEEE Access*, pp. 1–1, 2022.