# On Enabling Data-Aware Compliance Checking of Business Process Models*

David Knuplesch[1], Linh Thao Ly[1], Stefanie Rinderle-Ma[3], Holger Pfeifer[2], and Peter Dadam[1]

[1] Institute of Databases and Information Systems
Ulm University, Germany
[2] Institute of Artificial Intelligence
Ulm University, Germany
[3] Faculty of Computer Science
University of Vienna, Austria
david.knuplesch,thao.ly,holger.pfeifer,peter.dadam@uni-ulm.de,
stefanie.rinderle-ma@univie.ac.at

**Abstract.** In the light of an increasing demand on business process compliance, the verification of process models against compliance rules has become essential in enterprise computing. To be broadly applicable compliance checking has to support data-aware compliance rules as well as to consider data conditions within a process model. Independently of the actual technique applied to accomplish compliance checking, data-awareness means that in addition to the control flow dimension, the data dimension has to be explored during compliance checking. However, naive exploration of the data dimension can lead to state explosion. We address this issue by introducing an abstraction approach in this paper. We show how state explosion can be avoided by conducting compliance checking for an *abstract process model* and *abstract compliance rules*. Our abstraction approach can serve as preprocessing step to the actual compliance checking and provides the basis for more efficient application of existing compliance checking algorithms.

**Keywords:** Process verification, Compliance rules, Process data, Abstraction

## 1 Introduction

In many application domains, business processes are subject to compliance rules and policies that stem from domain-specific requirements such as standardization or legal regulations [1]. Examples of compliance rules for order-to-delivery processes are collected in Table 1. Ensuring compliance of their business processes is crucial for enterprises today, particularly since auditing and certification of their business processes has become a competitive edge in many domains. Examples

* This work was done within the research project SeaFlows partially funded by the German Research Foundation (DFG).

**Table 1.** Examples of compliance rules for order-to-delivery processes

| | |
|---|---|
| $c_1$ | After confirming an order, goods have to be shipped eventually. |
| $c_2$ | Production (i.e., local and outsourced production) shall not start until the order is confirmed. |
| $c_3$ | Each order shall either be confirmed or declined. |
| $c_4$ | Local production shall be followed by a quality test. |
| $c_5$ | Premium customer status shall only be offered after a prior solvency check. |
| $c_6$ | Orders with a piece number beyond `50,000` shall be approved before they are confirmed. |
| $c_7$ | For orders of a non-premium customer with a piece number beyond `80,000` a solvency check is necessary before assessing the order. |
| $c_8$ | Orders with piece number beyond `80,000` require additional shipping insurance before shipping. |
| $c_9$ | After confirming an order of a non-premium customer with piece number of at least `125,000`, premium status should be offered to the customer |

include certified family-friendly enterprises being more attractive to prospective employees or clinics proving a certain standard of their audited treatments to patients. Since process models are the common way to represent business processes, business process compliance can be ensured by verifying process models against imposed compliance rules at process buildtime. Such a priori compliance checking might help process designers to define compliant process models and avoid instantiations of non-compliant processes. Further, legacy process models can be checked for compliance, when introducing new compliance rules.

Fig. 1 shows a simplified order-to-delivery process $P$ which might be subject to the rules given in Table 1. For brevity we abstain from modeling the complete data flows of $P$. A closer look at compliance rules $c_1$ to $c_4$ reveals that they basically constrain the execution and ordering of activities and events within a process model. For example, $c_1$ being applied to $P$ means that event `confirm order` has to be eventually followed by the activity `ship goods` in all execution paths of $P$. We can apply approaches from literature to verify $P$ against $c_1$ to $c_4$, (e.g., [2–4]). However, compliance rules $c_6$ to $c_9$ obviously do not only refer to activities and events, but also to process data. In particular, in the context of $P$ process data includes piece number $pn$, customer status $c$, and approved $a$. In order to verify $P$ against *data-aware* compliance rules such as $c_6$ to $c_9$, data flows as well as branching conditions of $P$ have to be considered, i.e., any compliance checking approach should be able to deal with data conditions.

It is notable that although compliance rule $c_5$ does not contain any references to process data of $P$, data-awareness of the compliance checking is still needed to enable correct verification. Verifying $c_5$ while ignoring the data conditions in $P$ would lead to violation of $c_5$ over $P$ since activity `offer premium status` is not always preceeded by activity `check solvency`. However, when having a closer look at the data conditions under which these activities are executed (i.e., the

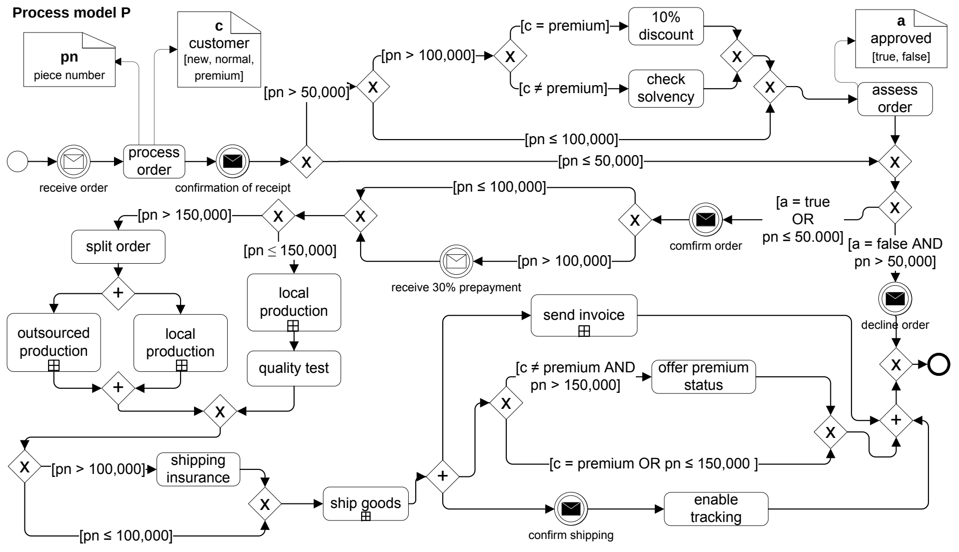


**Fig. 1.** A simplified order-to-delivery process modeled in BPMN

branching conditions) we can see, that $c_5$ is satisfied over $P$. Note that `offer premium status` is executed only for orders of non-premium customers with piece number beyond `150,000`. The correlation of the data conditions assigned to data-based exclusive gateways in $P$ guarantee a prior solvency check.

We denote compliance checking mechanisms that are able to deal with correlations of data-based gateways as well as to verify processes against data-aware compliance rules as *data-aware compliance checking*.

**Challenges.** As our examples show, data-awareness is crucial for applying compliance checking in practice. Independently of *how* compliance checking is actually accomplished (i.e., which techniques, such as model checking [2], are applied), data-awareness poses challenges for compliance checking in general. Data-aware compliance checking has to consider the states that relevant data objects can adopt during process execution. Activity `offer premium status` from Fig. 1, for example, can only be executed under data condition $pn >$ `150,000`. As this example shows, we may have to deal with arbitrary data such as integers that have huge domains. When compliance checking is applied in a straightforward and naive manner, data-awareness can lead to *state explosion* caused by the states that relevant data objects can adopt during process execution. Consider for example data object $pn$ representing the piece number in the order-to-delivery process and compliance rule $c_8$. Let us assume, for example, that the domain of $pn$ ranges from `1` to `500,000`. Then, naive exploration of the data dimension means that process model $P$ is verified against $c_8$ *for each possible state* of $pn$ between `1` and `500,000`. This, in fact, means that the complexity of data-aware compliance checking is `500,000` times the complexity of the non-

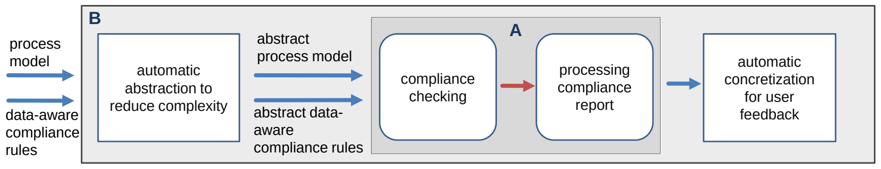

B
process model
data-aware compliance rules
automatic abstraction to reduce complexity
abstract process model
abstract data-aware compliance rules
A
compliance checking
processing compliance report
automatic concretization for user feedback

**Fig. 2.** The overall process of automatic abstraction based on the analysis of the process model and the compliance rules to be verified

data-aware case. Hence, to enable efficient compliance checking, strategies are required that keep the complexity manageable. A further challenge is that data-aware compliance checking also necessitates advanced concepts for user feedback. This is necessary since compliance violations that occur only under certain data conditions are often not as obvious as compliance violations at activity level.

**Contributions.** Although the verification of process models has been addressed by a multitude of recent approaches, data-awareness has not been sufficiently supported yet. Only few approaches consider the data perspective at all. While addressing data-aware compliance rules, [5] only enables data conditions that do not correlate. However, as the order-to-delivery process in Fig. 1 shows, data-based gateways may also contain conditions that are correlated. For example, the data conditions `pn > 100,000` and `pn > 150,000` in $P$ correlate (i.e., `pn > 150,000` implies `pn > 100,000` but not vice versa). In addition, these conditions correlate with data condition `pn > 80,000` ("piece number beyond `80,000`") resulting from compliance rule $c_8$. The limitation to only non-correlating data conditions facilitates the compliance checking problem. However, as our examples indicate, this can be too restrictive for many practical applications.

While existing approaches mainly focus on the compliance checking part (cf. Fig. 2 A), this paper focuses on the pre- and postprocessing steps (cf. Fig. 2 B) that enable data-aware compliance checking by tackling the state explosion problem. The latter can occur when fully exploring the data dimension during compliance checking. In this paper, we introduce abstraction strategies to reduce the complexity of data-aware compliance checking. This is achieved by abstracting from concrete states of data objects to abstract states. Based on the compliance rules to be checked our approach *automatically* derives an *abstract process model* and *abstract compliance rules*. The latter enable more efficient exploration of the data dimension when used as input to the actual compliance checking (cf. Fig. 2 B). Moreover, we discuss how a concretization can be accomplished to provide users with intelligible feedback in case of compliance violations. Our approach is finally validated by a powerful implementation, the SeaFlows compliance checker.

This paper is structured as follows. Related work is discussed in Sect. 2. Fundamentals are introduced in Sect. 3. Sect. 4 discusses how the abstract process model and corresponding abstract compliance rules are derived. The application of our abstraction approach and the proof-of-concept implementation are discussed in Sect. 5. We close with a summary and an outlook in Sect. 6.

## 2 Related Work

Due to its increasing importance compliance verification has been addressed by numerous approaches. Most of them focus on the compliance checking part (cf. Fig. 2 A). They propose a variety of techniques to accomplish the verification of process models against imposed compliance rules, such as model checking [2, 6–9] or the analysis of the process model structure [3, 10]. In our previous work in the SeaFlows project, we addressed the support of activity-level compliance rules throughout the process lifecycle [4, 10]. So far, however, only few approaches have addressed data-awareness. The modeling of data-aware compliance rules is addressed by [4, 5, 7, 11]. Graphical notations that are mapped to logical formulas (e.g., in linear temporal logic) are introduced in [4, 5, 7]. Basically, these approaches support the enrichment of activity-related compliance rules by data conditions. Our work in this paper apply these modeling approaches, since we do not focus on *how* to model data-aware compliance rules but rather on enabling their verification.

[12] introduce an approach for semantically annotating activities with preconditions and effects that may refer to data objects. In addition, [12] discusses an efficient algorithm for compliance verification using propagation. In contrast to this approach, we focus on deriving suitable abstraction predicates from process models and compliance rule. Further [5] allows for verifying process models against compliance rules with data conditions based on linear temporal logic (LTL). However, as previously discussed [5] only addresses data conditions that do not correlate. This is for example not sufficient to support proper verification of the order-to-delivery process from Fig. 1 against compliance rule $c_5$ from Table 1, since we have to deal with the correlation of two data-based exclusive gateways both refering to data object $pn$. In this paper, we propose strategies to deal with such cases. By applying abstraction techniques, our approach accomplishes the basis to apply approaches such as [5].

Orthogonal strategies to reduce complexity of compliance checking are discussed in [6, 7]. [7] sequentializes parallel flows in order to avoid analyzing irrelevant interleavings. [6] limits the complexity of compliance checking by reducing process models to the relevant parts. These abstraction strategies operating at the structural level are orthogonal to our abstraction approach. They can be applied to complement the approach introduced in this paper.

## 3 Fundamentals

Independently of the actual technique applied to accomplish compliance checking, data-awareness means that in addition to the control flow dimension the data dimension must be explored during compliance checking. However, this can lead to state explosion since a potentially huge number of states of data objects has to be explored. In Sect. 4, we show how the state explosion can be avoided by applying suitable abstraction strategies. Before discussing these, we provide some fundamentals.

A process domain representing a particular business domain typically consists of process artifacts (e.g., activities and events). The process domain notion in Def. 1 provides the basis for both process models and compliance rules.

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

- *$\mathbb{A}$ is the set of activity types,*
- *$\mathbb{E}$ is the set of event types,*
- *$\mathbb{O}$ is the set of data objects,*
- *$\mathbb{D}$ is the set of data domains, and*
- *$dom : \mathbb{O} \to \mathbb{D}$ is a function assigning a data domain to each data object.*
- *We further define $\Omega_{\mathcal{D}} := \bigcup \mathbb{D}$ as the set of all values (i.e., data states) of $\mathcal{D}$.*

**Example 1.** Consider process model $P$ from Fig. 1. The related process domain may be $\mathcal{D} = (\mathbb{A}, \mathbb{E}, \mathbb{O}, \mathbb{D}, dom)$, where
$\mathbb{A} := \{$`process order, 10% discount, check solvency, assess order, ...`$\}$
$\mathbb{E} := \{$`receive order, confirmation of receipt, confirm order, ...`$\}$
$\mathbb{O} := \{pn, c, a\}$
$\mathbb{D} := \{\ \mathbb{N} = \{$`0, 1, 2, ...`$\}, \{$`new, normal, premium`$\}, \mathbb{B} = \{$`true, false`$\}\ \}$
$dom(pn) := \mathbb{N}$; $dom(c) := \{$`new, normal, premium`$\}$; $dom(a) := \mathbb{B}$

**Data-Aware Compliance Rules.** It is not our intention to introduce an approach to model data-aware compliance rules. Hence, we rely on existing work such as [4, 5, 13]. Since our approach is not restricted to a particular compliance rule modeling language, we come up with a general notion of data-aware compliance rules in Def. 3 that is applicable to a multitude of existing approaches.

Compliance rules typically contain conditions on activities and events of certain types (e.g., cf. compliance rules $c_1$ to $c_4$). We denote these as *type conditions*. In addition to type conditions, data-aware compliance rules contain conditions on the states of data objects, so-called *data conditions*. Formalization of type conditions and data conditions is given in Def. 2.

**Definition 2 (Type Condition, Data Condition).** *Let $\mathcal{D} = (\mathbb{A}, \mathbb{E}, \mathbb{O}, \mathbb{D}, dom)$ be a process domain and let $t \in \mathbb{A} \cup \mathbb{E}$ be an activity type or an event type. Then*

- *a type condition is an expression of the form: $(type = t)$.*

*Let further $o \in \mathbb{O}$ be a data object, $v \in dom(o)$ a certain value of the related domain, and $\otimes \in \{=, \neq, <, >, \leq, \geq, \ldots\}$ a relation. Then*

- *a data condition is an expression of the form: $(o \otimes v)$.*

*Moreover, we define:*

- *$TC_{\mathcal{D}} \subseteq \{\ (type = t) \mid t \in \mathbb{A} \cup \mathbb{E}\}$ as the set of all type conditions over $\mathcal{D}$.*
- *$DC_{\mathcal{D}} \subseteq \{\ (o \otimes v) \mid o \in \mathbb{O}, v \in dom(o), \otimes := \{=, \neq, <, >, ...\}\}$ as the set of all data conditions over $\mathcal{D}$.*

**Example 2.** Consider compliance rule $c_9$ from Table 1. Here, $c_9$ yields the following type conditions ($TC_{\mathcal{D}}$) and data conditions ($DC_{\mathcal{D}}$) where $pn \in \mathbb{O}$ is the data object representing the piece number and $c \in \mathbb{O}$ is the data object representing the customer status (cf. Fig. 1).

| Phrase | Corresponding condition | |
|---|---|---|
| After **confirming an order** | $(type = \texttt{confirm order})$ | $\in TC_{\mathcal{D}}$ |
| of a **non-premium customer** with | $(c \neq \texttt{premium})$ | $\in DC_{\mathcal{D}}$ |
| **piece number of at least 125,000** | $(pn \geq \texttt{125,000})$ | $\in DC_{\mathcal{D}}$ |
| **premium status** should be **offered** to the customer | $(type = \texttt{offer premium status})$ | $\in TC_{\mathcal{D}}$ |

Finally, a general data-aware compliance rule is defined as follows:

**Definition 3 (Data-Aware Compliance Rule).** *Let $\mathcal{D} = (\mathbb{A}, \mathbb{E}, \mathbb{O}, \mathbb{D}, dom)$ be a process domain. Then, a data-aware compliance rule is a tuple $c = (C, \Delta)$, with:*

- *$C = TC_c \cup DC_c$ is finite set of conditions that is partitioned into the set of type conditions $TC_c \subseteq TC_{\mathcal{D}}$ and the set of data conditions $DC_c \subseteq DC_{\mathcal{D}}$.*
- *$\Delta$ an expression defining temporal (ordering) and logical relations over the conditions in $C$.*

*Further, we define:*

- *$conditionsCR_c : \mathbb{O} \rightarrow 2^{DC_c}, o' \mapsto \{(o \otimes v) | o = o' \wedge (o \otimes v) \in DC_c\}$ as a function returning all data conditions of $c$ that affect a certain data object.*

**Example 3.** To illustrate our examples, we use linear temporal logic (LTL). LTL is applied to model compliance rules by numerous approaches [5, 13]. Note, however, that our approach is not restricted to a particular compliance rule modeling language. Using LTL we can model $c_9$ from Table 1 as follows:

$$c_9 : \mathbf{G} \ ( \ ((type = \texttt{confirm order}) \wedge (pn \geq \texttt{125,000}) \wedge (c \neq \texttt{premium})) \Rightarrow \ \mathbf{F} \ (type = \texttt{offer premium status}) \ )$$

According to Def. 3 this means $c_9 = (C_9, \Delta_9)$, where

$C_9 = \{tc_1 = (type = \texttt{confirm order}), tc_2 = (type = \texttt{offer premium status})$
$dc_1 = (pn \geq \texttt{125,000}), dc_2 = (c \neq \texttt{premium})\}$
$\Delta_9 = \mathbf{G} \ ( \ (tc_1 \wedge dc_1 \wedge dc_2) \ \Rightarrow \ \mathbf{F} \ tc_2)$
$TC_{c_9} = \{(type = \texttt{confirm order}), (type = \texttt{offer premium status})\}$
$DC_{c_9} = \{(pn \geq \texttt{125,000}), (c \neq \texttt{premium})\}$

Based on the above notion of data-aware compliance rules, we later show how automatic abstraction is conducted to enable data-aware compliance checking.

**Processes.** A process model, commonly represented by a process graph, can be composed using activities, events, and data objects from a process domain. Since our approach is not restricted to a particular process definition language, we provide a general definition of process graphs in Def. 4 following common notations, such as BPMN:

**Definition 4 (Process Graph).** *Let $\mathcal{D} = (\mathbb{A}, \mathbb{E}, \mathbb{O}, \mathbb{D}, dom)$ be a process domain. Then, a process graph is a tuple with $P = (N, F, O, I, type, con)$, 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 $G_P$.*
- *$F \subseteq N \times N$ represents the sequence flow relation between nodes.*
- *$O \subseteq \mathbb{O}$ 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 $P$, where holds: $a \in A_P \Rightarrow type(a) \in \mathbb{A}$ and $e \in E_P \Rightarrow type(e) \in \mathbb{E}$*
- *$con : F \to 2^{DC_{\mathcal{D}}}$ is a function assigning a (maybe empty) set of data conditions to each sequence flow.*

*Further, we define:*

- *$DC_P := \{(o \otimes v) | \exists f \in F : (o \otimes v) \in con(f)\} \subseteq DC_{\mathcal{D}}$ as the set of data conditions in $P$.*
- *$conditionsPG_P : O \to 2^{DC_P}, o' \mapsto \{(o \otimes v) | o = o' \wedge (o \otimes v) \in DC_P\}$ as a function returning all data conditions of $P$ on the associated data object.*

**Example 4.** Process model $P$ from Fig. 1 contains the following data conditions over $pn$:

$$conditionsPG_P(pn) = \{(pn \leq \texttt{50,000}), (pn > \texttt{50,000}), (pn \leq \texttt{100,000}), (pn > \texttt{100,000}), (pn \leq \texttt{150,000}), (pn > \texttt{150,000})\}$$

## 4 On Enabling Data-Aware Compliance Checking

As discussed the full exploration of the data dimension can lead to state explosion, when conducting compliance checking. The basic idea to achieve more efficient *data-aware compliance checking* of process models and to limit the state explosion problem is to abstract from states that are irrelevant for the verification of a particular compliance rule. Consider, for example, compliance rule $c_8$ from Table 1. Concerning the satisfaction/violation of $c_8$ it is not relevant whether $pn = \texttt{120,000}$, $pn = \texttt{120,001}$, $pn = \texttt{120,002}$, ..., *or* $pn = \texttt{130,000}$ holds when executing the order-to-delivery process from Fig. 1. Hence, it is not necessary to differentiate between these cases when verifying $P$ against $c_8$. These potential states of $pn$, namely `120,000`, ..., `130,000`, could be treated as one "merged" state. The merged state can be described by the *abstraction predicates* $(pn \geq \texttt{120,000}) \wedge (pn \leq \texttt{130,000})$ and be applied to data-aware compliance checking (e.g., by applying model checking techniques). Other irrelevant states of $pn$ can be merged in a similar manner to derive a more compact set of states that serve as domain of $pn$ in the *abstract process model*. In fact, a differentiation between the concrete states of $pn$ beyond `100,000` is not necessary for verifying $P$ against $c_8$. Hence, all states of $pn$ beyond `100,000` can be merged to one abstract state $pn > \texttt{100,000}$.

In general, by abstracting from states of data objects irrelevant for the verification of a compliance rule, less cases have to be explored in the verification procedure. This helps to reduce complexity of compliance checking. However, abstracting from states must not lead to incorrect verification results. Consider, for example, again the order-to-delivery process $P$ and compliance rule $c_8$ from Table 1. To verify particularly $c_8$ it is not sufficient to only consider whether $pn > \texttt{100,000}$ or $pn \leq \texttt{100,000}$ holds. The challenge of automatic abstraction is to identify adequate *abstraction predicates* that enable us to "merge" states wihtout falsifying verification results

In literature, abstracting from concrete states to abstract predicates is common practice for dealing with state explosion [14–17]. This is particularly relevant in engineering domains, where large systems have to be verified against safety properties. In many applications, abstraction constitutes a task that requires human interaction. In particular, domain experts are required to find the right abstraction. By analyzing the dependencies between a process model $P$ and a compliance rule $c$ our abstraction approach *automatically* derives an abstract process model $P_{abstract}$ and an abstract compliance rule $c_{abstract}$. These can serve as input to the actual compliance checking. We want to find *conservative* abstraction predicates such that holds: $P \models c \Leftrightarrow P_{abstract} \models c_{abstract}$

The data-based abstraction introduced in this paper can be combined with structural abstraction strategies (cf. Sect. 2) to achieve further reduction of the compliance checking complexity.

## Automatic Abstraction for Data Conditions

To achieve automatic abstraction, we have to accomplish three steps:

1) Identify data objects potentially relevant to $c$ and the data conditions on them in $c$ and in the data-based gateways of $P$
2) Identify abstraction predicates for relevant data objects
3) Application of abstraction predicates to obtain $P_{abstract}$ and $c_{abstract}$

Altogether, the states of each data object $o$ can be represented by a set of abstraction predicates beeing relevant for proper verification of the associated compliance rule over the process model. This is accomplished by analyzing the data conditions in the process model and in the compliance rule. For the identified set of abstraction predicates we can identify combinations of predicates whose conjunction is satisfiable (i.e., evaluated with `true`). Each such combination represents a potential *abstract state* of the corresponding data object $o$:

**Definition 5 (Abstraction for Data Conditions).** *Let $\mathcal{D} = (\mathbb{A}, \mathbb{E}, \mathbb{O}, \mathbb{D}, dom)$ be a process domain, $P = (N, F, O, I, type, con)$ be a process model, and $c$ be a compliance rule over $\mathcal{D}$. Let further $o \in O$ be a data object in $P$. Then,*

- *$predicates^c_P : O \to DC_{\mathcal{D}}, o \mapsto predicates^c_P(o)$ with $predicates^c_P(o) := conditionsCR_c(o) \cup conditionsPG_P(o)$ is a function returning the set of all data conditions in $c$ and $P$ that affect $o$ and, thus, constitute relevant abstraction predicates.*

– *solve* : $2^{DC_{\mathcal{D}}} \times \Omega_{\mathcal{D}} \rightarrow 2^{DC_{\mathcal{D}}}, (C, v') \mapsto solve((C, v'))$ *with* $solve(C, v') := \{(o \otimes v) \mid (v' \otimes v) = \texttt{true} \wedge (o \otimes v) \in C\}$ *is a function returning the particular subset of predicates that are satisfied by* $v$.
– $allocations^{c}_{P} : O \rightarrow 2^{2^{DC_{\mathcal{D}}}}, o \mapsto allocations^{c}_{P}(o)$ *with* $allocations^{c}_{P}(o) := \{S \mid \exists v \in dom(o) : S = solve(predicates^{c}_{P}(o), v)\}$ *is a function returning a set of sets of predicates such that for each value* $v \in dom(o)$ *there is a set in* $allocations^{c}_{P}(o)$ *containing all predicates over* $o$ *that are satisfied by* $v$.

Note that for deriving the abstraction predicates $predicates^{c}_{P}(o)$ not only the data conditions of $P$, but also the data conditions of $c$ are considered. Based on the predicates $predicates^{c}_{P}(o)$ for a data object $o$, we can narrow the data domain of $o$ which has to be explored during data-aware compliance checking. In particular, instead of exploring the complete domain of $o$, which may cause a state explosion, only the corresponding set of abstract states (i.e., the elements of $allocations^{c}_{P}(o)$) has to be explored in the compliance checking procedure. Due to the Def. 5 $|allocations^{c}_{P}(o)| \leq |dom(o)|$ always holds. For a large data domain $dom(o)$ typically, $allocations^{c}_{P}(o)$ contains significantly less elements. Hence, to be able to narrow the domain of $o$ to the set of abstract states of $o$ being relevant for the verification of the actual compliance rule is a crucial step to avoid the state explosion problem.

**Dealing with Large Domains.** Although it is easy to derive $allocations^{c}_{P}(o)$ for small finite domains by calculating $solve(C, v)$ for each $v \in dom(o)$, this procedure is not feasible for large data domains, such as $D = \mathbb{N}$. However, if $D$ is a totally ordered domain (e.g. $\mathbb{N}$, $\mathbb{Z}$, or $\mathbb{R}$) and all conditions $(o \otimes v) \in predicates^{c}_{P}(o)$ are using ordering relations $\otimes \in \{<, \leq, =, \geq, >, \neq\}$, as it is the case with *pn* and corresponding data conditions, we can efficiently calculate $allocations^{c}_{P}(o)$ as follows:

First, we determine $(v_i)_{1 \leq i \leq n} =< v_1, \ldots, v_n >$ the ascendingly sorted finite sequence of such values $v$ with $\exists(o \otimes v) \in predicates^{c}_{P}(o)$ without any multiple occurrences. Now it is sufficient to limit the calculation of $solve(predicates^{c}_{P}(o), v)$ to the following cases of $v$

1. the values $v_1, \ldots, v_n$ that are the limits of the relevant abstraction predicates,
2. for any two successive values $v_i$ and $v_{i+1}$, a value $w_i$ with $v_1 < w_i < v_{i+1}$,
3. a value $s < v_1$ smaller than any $v_i$ and $b > v_n$ bigger than any $v_i$

Obviously, it is sufficent to use one $w_i$ with $v_i < w_i < v_{i+1}$, since all values of this interval exactly fulfill and violate the same conditions of $predicates^{c}_{P}(o)$. For the same reason the use of one $s$ and one $b$ is sufficent. Note that sometimes there may be no $s \in D$ with $s < v_1$ or no $w_i \in D$ with $v_i < w_i < v_{i+1}$ (i.e. $D = \mathbb{N}$ and $v_1 = 0, v_2 = 1$). Then, the corresponding cases have to be ignored.

The calculation of $allocations^{c}_{P}(o)$ may also be delegated to a SMT-Solver (e.g., Yices [18]) that is even able to deal with large domains and conditions using linear arithmetics.

**Example 5.** Consider process model $P$ from Fig. 1 and compliance rule $c_9$ from Table 1 over process domain $\mathcal{D}$. As described above, to calculate $allocations_P^{c_9}(pn)$ it is sufficient to consider `50,000, 100,000, 125,000, 150,000` as well as `75,000, 112,500, 137,500` and `49,999, 150,001`. So we receive the following abstraction predicates for the data object $pn$.

$$\begin{aligned}
predicates_P^{c_9}(pn) &= conditionsCR_{c_9}(pn) \cup conditionsPG_P(pn)\\
&= \{(pn \geq \texttt{125,000})\} \cup \{(pn \leq \texttt{50,000}), (pn > \texttt{50,000}), (pn \leq \texttt{100,000}),\\
&\quad (pn > \texttt{100,000}), (pn \leq \texttt{150,000}), (pn > \texttt{150,000})\}\\
&= \{(pn \geq \texttt{125,000}), (pn \leq \texttt{50,000}), (pn > \texttt{50,000}), (pn \leq \texttt{100,000}),\\
&\quad (pn > \texttt{100,000}), (pn \leq \texttt{150,000}), (pn > \texttt{150,000})\}
\end{aligned}$$

$allocations_P^{c_9}(pn) = \{\alpha, \beta, \gamma, \delta, \epsilon\}$, where
$\alpha := \{(pn \leq \texttt{50,000}), (pn \leq \texttt{100,000}), (pn \leq \texttt{150,000})\}$
$\beta := \{(pn > \texttt{50,000}), (pn \leq \texttt{100,000}), (pn \leq \texttt{150,000})\}$
$\gamma := \{(pn > \texttt{50,000}), (pn > \texttt{100,000}), (pn \leq \texttt{150,000})\}$
$\delta := \{(pn \geq \texttt{125,000}), (pn > \texttt{50,000}), (pn > \texttt{100,000}), (pn \leq \texttt{150,000})\}$
$\epsilon := \{(pn \geq \texttt{125,000}), (pn > \texttt{50,000}), (pn > \texttt{100,000}), (pn > \texttt{150,000})\}$

The sets of predicates in $allocations_P^{c_9}(pn)$ constitute properties describing the "merged" sets of original states of $pn$. $\{\alpha, \beta, \gamma, \delta, \epsilon\}$ may be used as *abstract data domain* of $pn$ for verification against $c_9$. During compliance checking of $P$ against $c_9$, only these abstract states have to be explored. Compared to the original domain of $pn$, this constitutes a significant reduction of the complexity for exploring the data dimension.

In Def. 6 we transfer the above results into process domain, process graph and compliance rule and, therefore, formalize the *abstract process domain*, the *abstract process graph*, and the *abstract compliance rule* (cf. Fig. 2).

**Definition 6 (Abstract Process Domain, Abstract Process Graph, Abstract Compliance Rule).** *Let $\mathcal{D} = (\mathbb{A}, \mathbb{E}, \mathbb{O}, \mathbb{D}, dom)$ be a process domain with a process graph $P = (N, F, O, I, type, con)$ and a compliance rule $c = (C, \Delta)$. The abstract process domain $\mathcal{D}_{abstract}$ of $\mathcal{D}$ with respect to $P$ and $c$ is defined as $\mathcal{D}_{abstract} = (\mathbb{A}, \mathbb{E}, \mathbb{O}, \mathbb{D}_{abstract}, dom_{abstract})$, where*

- $\mathbb{D}_{abstract} := \{allocations_P^c(o) | o \in \mathbb{O}\}$
- $dom_{abstract}(o) := allocations_P^c(o)$

*Then the abstract process graph $P_{abstract}$ of $P$ with respect to $c$ is defined as $P_{abstract} = (N, F, O, I, type, con_{abstract})$, where for $f \in F$ holds*

- $con_{abstract}(f) := \{\ (''(o \otimes v)'' \in o) \mid ''(o \otimes v)'' \in con(f)\ \}$

*The abstract compliance rule $c_{abstract}$ of $c$ with respect to $P$, is defined as $c_{abstract} = (C_{abstract}, \Delta)$ where:*

- $C_{abstract} := TC_c \cup \{\ (''(o \otimes v)'' \in o)\ \mid ''(o \otimes v)'' \in DC_c)\ \}$

**Example 6.** Consider process model $P$ from Fig. 1 and compliance rule $c_9$ over process domain $\mathcal{D}$ (cf. Example 1). Due to space limitation, we apply the abstraction only for data object *pn*. Based on Example 5 we obtain the abstract process domain $\mathcal{D}_{abstract} = (\mathbb{A}, \mathbb{E}, \mathbb{O}, \mathbb{D}_{abstract}, dom_{abstract})$, with

- $\mathbb{D}_{abstract} := \{\ allocations_P^{c_9}(pn), \ldots\}$
  $= \{\ \{\alpha, \beta, \gamma, \delta, \epsilon\}\ ,\ \ldots\ \}$
- $dom(pn)_{abstract} := allocations_P^{c_9}(pn) = \{\alpha, \beta, \gamma, \delta, \epsilon\}$

The corresponding abstract process graph $P_{abstract} = (N, F, O, I, type, con_{abstract})$ is depicted in Fig. 3). Further, we obtain the corresponding compliance rule $c_{abstract} = (C_{abstract}, \Delta)$, where:

$$C_{abstract} := TC_c \cup \{\ ((o \otimes v) \in o)\,|(o \otimes v) \in DC_c)\}$$
$$= \{tc_1 = (type = \texttt{confirm order}), tc_2 = (type = \texttt{offer premium status})$$
$$dc_{1abstract} = ((pn \geq \texttt{125,000}) \in pn),\ \ldots\ \}$$

## 5 Analyis and Implementation

We applied our approach to enable data-aware compliance checking of process models without running into intractable state explosion problems. To accomplish the actual compliance checking step (cf. Fig. 2 A), we applied model checking techniques (e.g., by using SAL [19]). Model checking is applied to compliance verification by numerous approaches in literature [2, 6, 7]. It comprises techniques for automatic verification of a model specification against predefined properties.

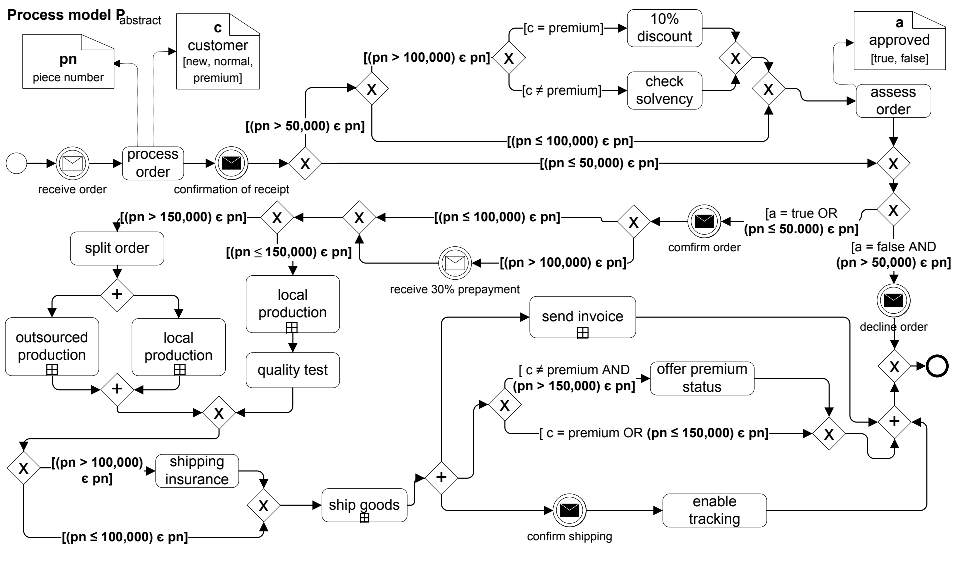


**Fig. 3.** The abstract order-to-delivery process after applying abstraction to *pn*

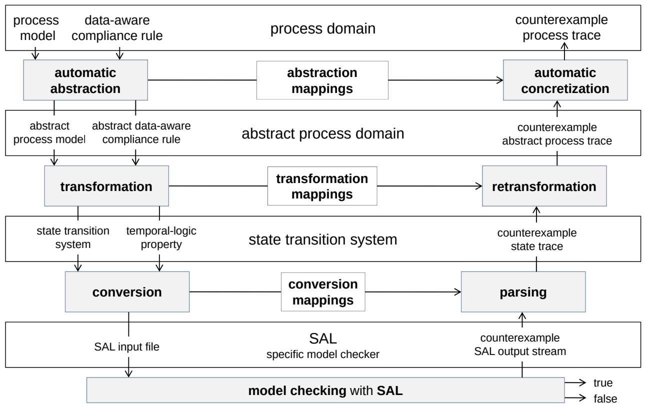


**Fig. 4.** Data-aware compliance checking and generation of counterexample

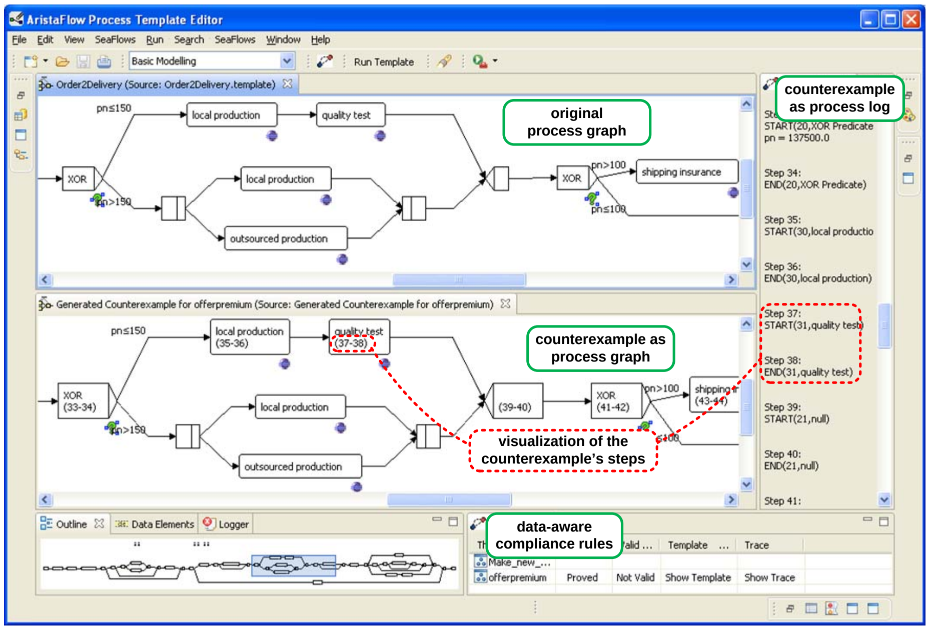
In order to apply model checking, we have to provide a state transition system and a logic property model to the model checker (cf. Fig 4). Therefore, we transform the abstract process model into a state representation and the abstract compliance rule into a logic property. To provide both to the model checker, a conversion with respect to the model checker's specific syntax and restrictions is required. The model checker then performs automatic exploration of the state space and checks for conformance to the compliance rule. In case of a violation, the model checker provides an incompliant execution trace as counterexample.

As previously discussed, a major challenge of data-aware compliance checking is to provide meaningful feedback in case of compliance violations (e.g., data conditions under which a violation occurs). To tackle this we have to memorize the steps taken during the transformation procedure and we need to conduct a retransformation. Fig. 4 shows the steps accomplished by our implementation.

Our proof-of-concept implementation, the SeaFlows compliance checker, is implemented as Java-plug-in for the Aristaflow Process Template Editor which is part of the Aristaflow BPM Suite [20]. 17.000 lines of code and the class hierarchy comprising about 70 interfaces and 210 classes indicate the complexity of the implementation. The SeaFlows compliance checker enables modeling of LTL-based data aware compliance rules using a tree-based editor. Automatic abstraction as discussed in Sect. 4 is supported for domains of numbers. The SeaFlows compliance checker conducts the automatic abstraction, transforms a AristaFlow process model into a state representation, and pass it to the model checker SAL [19]. Counterexamples obtained from SAL can be shown as process logs or visualized as process graph as shown in Fig. 5.

## 6 Summary and Outlook

Enabling process-aware information systems to support the compliance of process models with imposed data-aware compliance rules can be regarded as one


AristaFlow Process Template Editor
File Edit View SeaFlows Run Search SeaFlows Window Help
Basic Modelling
Run Template
Order2Delivery (Source: Order2Delivery.template)
original process graph
pn≤150
local production
quality test
XOR
pn>150
local production
outsourced production
XOR
pn>100
shipping insurance
pn≤100
Generated Counterexample for offerpremium (Source: Generated Counterexample for offerpremium)
counterexample as process graph
XOR (33-34)
local production (35-36)
quality test (37-38)
(39-40)
XOR (41-42)
shipping ir (43-44)
visualization of the counterexample's steps
counterexample as process log
data-aware compliance rules
Outline
Data Elements
Logger

```
St
START(20,XOR Predicate
pn = 137500.0

Step 34:
END(20,XOR Predicate)

Step 35:
START(30,local productio

Step 36:
END(30,local production)

Step 37:
START(31,quality test)

Step 38:
END(31,quality test)

Step 39:
START(21,null)

Step 40:
END(21,null)

Step 41:
```

| Th | | Valid ... | Template ... | Trace |
|---|---|---|---|---|
| Make_new_... | | | | |
| offerpremium | Proved | Not Valid | Show Template | Show Trace |

**Fig. 5.** Aristaflow Process Template Editor with the SeaFlows compliance checker plug-in for data-aware compliance rule checking

step towards installing business process compliance management in practice. In this paper, we introduced an abstraction approach that enables data-aware compliance checking in a more efficient manner by limiting the state explosion problem that can occur when fully exploring the data dimension during verification. The approach serves as preprocessing step to actual compliance checking and provides the basis for efficient application of existing compliance checking algorithms. Being indepedent of a particular process meta-model and of a particular compliance rule modeling language, our approach is applicable to a variety of existing approaches. To accomplish data-aware compliance checking in a comprehensive manner, we also address the challenge of providing users with intelligible feedback in case of compliance violations. To our best knowledge, we are the first to apply automatic data abstraction in the context of compliance checking of business process models. In future, we will further research on automatic abstraction for other types of domains, also considering relationships among them. Further we will go on in refining the verification output to provide more intelligible feedback to users.

## References

1. Sadiq, S., Governatori, G., Naimiri, K.: Modeling control objectives for business process compliance. In: Proc. of the 5th Int'l Conf. on Business Process Management. Volume 4714 of LNCS., Springer (2007) 149–164

2. Ghose, A., Koliadis, G.: Auditing business process compliance. Service-Oriented Computing–ICSOC 2007 **4749** (2007) 169–180
3. Sadiq, S., Orlowska, M., Sadiq, W.: Specification and validation of process constraints for flexible workflows. Information Systems **30**(5) (2005) 349–378
4. Ly, L.T., Rinderle-Ma, S., Dadam, P.: Design and verification of instantiable compliance rule graphs in process-aware information systems. In: Proc. of the 22nd Int'l Conf. on Advanced Information Systems Engineering. Volume 6051 of LNCS., Springer (2010)
5. Awad, A., Weidlich, M., Weske, M.: Specification, verification and explanation of violation for data aware compliance rules. In: Proc. of the 7th Int'l Joint Conf. on Service-Oriented Computing. Volume 5900 of LNCS., Springer (2009) 500–515
6. Awad, A., Decker, G., Weske, M.: Efficient compliance checking using BPMN-Q and temporal logic. In: Proc. of the 6th Int'l Conf. on Business Process Management. Volume 5240 of LNCS., Springer (2008) 326–341
7. Liu, Y., Müller, S., Xu, K.: A static compliance-checking framework for business process models. IBM Systems Journal **46**(2) (2007) 335–261
8. Yu, J., et al.: Pattern based property specification and verification for service composition. In: Proc. of the 7th Int'l Conf. on Web Information Systems Engineering. Volume 4255 of LNCS. (2006) 156–168
9. Förster, A., Engels, G., Schattkowsky, T.: Activity diagram patterns for modeling quality constraints in business processes. In: Proc of the 8th Int'l Conf. on Model Driven Engineering Languages and Systems. Volume 3713 of LNCS., Springer (2005) 2–16
10. Ly, L.T., Rinderle, S., Dadam, P.: Semantic correctness in adaptive process management systems. In: Proc. of the 4th Int'l Conf. on Business Process Management. Volume 4102 of LNCS., Springer (2006) 193–208
11. Weber, I., Hoffmann, J., Mendling, J.: Semantic business process validation. In: Proc. of the 3rd Int'l Workshop on Semantic Business Process Management. (2008) 22–36
12. Governatori, G., et al.: Detecting regulatory compliance for business process models through semantic annotations. In: Business Process Management Workshops. Volume 17 of LNBIP., Springer (2008) 5–17
13. van der Aalst, W., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: Proc. of the 3rd Int'l Workshop on Web Services and Formal Methods. Volume 4184 of LNCS., Springer (2006) 1–23
14. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. of the 4th ACM Symp. on Principles of Programming Languages, ACM Press (1977) 238–252
15. Graf, S., Saidi, H.: Construction of abstract state graphs with PVS. In: Proc. of the 9th Int'l Conf. on Computer Aided Verification, Springer (1997) 72–83
16. Das, S.: Predicate Abstraction. PhD thesis, Stanford University (2003)
17. Model, F.S., Clarke, E., Lu, Y.: Counterexample-guided abstraction refinement. In: Computer Aided Verification. Volume 1855 of LNCS., Springer (2000) 154–169
18. Dutertre, B., De Moura, L.: The YICES SMT solver. Tool paper at http://yices. csl. sri. com/tool-paper. pdf (2006)
19. Bensalem, S., et al.: An overview of SAL. In: Proc. of the 5th NASA Langley Formal Methods Workshop, NASA Langley Research Center (2000) 187–196
20. Dadam, P., Reichert, M.: The ADEPT project: a decade of research and development for robust and flexible process support. Computer Science-Research and Development **23**(2) (2009) 81–97