Identifying Anomalies in SBVR-based Business Rules using Directed Graphs and SMT-LIBv2

Sayandeep Mitra, Kritika Anand and Pavan Kumar Chittimalli
Tata Research Development and Design Centre, TCS Innovation Labs, Pune, India

Keywords: Business Rules, Verification, Directed Graph, SMT, Clustering, SBVR, Anomalies.

Abstract: In modern times, business rules have grown exponentially with enterprises becoming more complex in diverse fields. Due to this growth, different forms of anomalies creep into the business rules, causing business enterprise to take wrong decisions, which can impact it's performance and reputation. It is time and resource consuming to examine the rules manually due to the large number of rules intermingled with each other. The process of manual verification is also not free of human induced errors. Thus, automatic verification of business rules is the need of the hour. We present a tool to detect different anomalies in business rules represented in SBVR format. The tool uses a combination of Directed Graphs and SMT solvers to perform the verification task. We show the implementation of our tool along with it's evaluation on industry level benchmarks.

1 INTRODUCTION

Business Rules are operational regulations, decision rules that are followed by a business organization to perform their day to day activities. The business rules are usually embodied in the system artifacts such as governing policies, guidelines, operating procedures, legacy source code, etc.

Humans and information systems together are involved in various business operations, with the corresponding business rules distributed across the enterprise in different forms (policy documents, operational procedures and in the source code of the information systems). Varying market conditions and external regulatory reasons cause constant changes to the business structure, policies and strategies. Business transformation is a process of adjusting business activities to accommodate the above changes. The aging business information systems may also require changes in order to respond to changing business environment, i.e., competition and growth for superior business products and services. Due to the large number of rules of a business enterprise co-existing together in such a state of constant flux, various form of anomalies such as *conflicts, redundancies, duplicates* and *circularity* may creep in. Thus both IT transformation and business transformation force the enterprises to revisit their business rules, pushing forward the need for automatic verification of business rules.

In recent time, various epresentation of business rules have been used (IBM, 2017; JBoss, 2017; OMG, 2013). We have selected SBVR (OMG, 2013) in our tool due to its base in First Order Logic (FOL) and similarity to natural language. The SBVR model has been presented as result of the request for proposal on Business Semantics of Business Rules (BSBR) made by OMG, which is a part of the business model layer in the Model Driven Architecture (MDA). The purpose of SBVR is to describe formally and without ambiguities the semantics of a business model which in turn benefits business analysts and modelers, as well as business vocabulary & rules administrators and software tool developers. SBVR works as a bridge between business enterprises and Information Technology (IT), aiming to provide a way to express business knowledge to the IT group unambiguously using natural language. SBVR meta-model is used to represent business knowledge as:

1. Specifying business vocabularies.
2. Specifying business rules.

Organizations or communities specify the conduct of business using a cohesive set of interconnected concepts known as Business vocabulary. These concepts are entities represented through *name, term,* and *verb* while *fact* is expressed as relation between these concepts. SBVR Structured English (SSE) is a popular textual representation of SBVR, providing the option to write business rules in plain English. Detailed explanation about SBVR can be found at (OMG, 2013).

Mitra, S., Anand, K. and Chittimalli, P.
Identifying Anomalies in SBVR-based Business Rules using Directed Graphs and SMT-LIBv2.
DOI: 10.5220/0006669802150222
In *Proceedings of the 20th International Conference on Enterprise Information Systems (ICEIS 2018)*, pages 215-222
ISBN: 978-989-758-298-1

Previously, works like (He et al., 2003; Chavarria-Baez and Li, 2006; Nazareth and Kennedy, 1991; Ramaswamy et al., 1997) focused on verification of knowledge base or rule base verification. The approaches presented in these works suffered from the problems of large computational complexity, which if applied on huge and complex rule systems existing at present will be extremely inefficient. Presently, some work has been done aiming at verification of business rules and specifications. Some of these works are (dos Santos Guimarães et al., 2014; Karpovič et al., 2016; Reynares et al., 2014; Solomakhin et al., 2013). However most of them fail to present results on complex and real life business rules, along with their efficiency on the proposed approaches.

In our background study, we observe that most of the work has been done aiming to detect all forms of anomalies (errors) using one particular method. In our opinion, this causes a spike in the computational complexity of the approaches and reduces efficiency. We present our tool using a compliance of two different methods, utilizing them to detect targeted class of anomalies, aiming at high efficiency and low complexity. Initially, a directed graph based technique is used to identify various structural anomalies in the business rule set, followed by modeling the rules in `SMT-LIBv2` (Barrett et al., 2010) and then using a solver to detect a different set of anomalies. In our tool, the targeted elimination of errors from the rule set in each step leads to less complexity and high efficiency in the complete verification task.

The rest of the paper is arranged as follows. In Section 2 we present the classification of anomalies that exist in business rule systems, followed by the detailed explanation of our method in Section 3. Section 4 shows the performance of our tool on industry level benchmarks and snapshots of our tool. We present various related works in Section 5, followed by Future Work and Conclusion in Section 6.

2 CLASSIFICATION OF ANOMALIES

Various works have been made presenting the taxonomy of the different anomalies that exist in rule-based systems (Preece and Shinghal, 1994; Nazareth, 1989). Based on these works, we define the anomalies that exist in Business Rules along with examples. Figure 1 presents a collection of some rules in `SBVR` format.

- **Conflicting Rules:** Business rules conflict with others, when same premise leads to mutually exclusive conclusions. A simple example of conflicting rules occur when the conditions in antecedents are the same, but the consequents are contradictions of each other. e.g., in Figure 1, Rules r_2 and r_6 have the same conditions in their antecedent, however the consequents are the opposite of each other.

r_1 : It is necessary that if car is luxury_car then car has driver.
r_2 : It is necessary that if car has driver then driver owns license.
r_3 : It is necessary that if car has driver and car does not has driver then car is rented_car.
r_4 : It is necessary that if driver owns license then car has driver .
r_5 : It is necessary that if car has driver then driver does not owns license and driver follows traffic_rules.
r_6 : It is necessary that if car has driver then driver does not owns license.

Figure 1: Example of Anomalies in Business Rules.

- **Redundancy:** Different form of redundancies exist in Business Rules, namely *Non-executable Rules*, *Subsuming rules* and *Logical Equivalence*. *Non-executable Rules* are the group of rules which never fire (execute) in a business rule set, given any possible conditions. In broad terms, we can say that non-executable rules are those which have conflicting conditions in their antecedents, thus never allowing the consequents to materialize. These conflicting conditions can exist in a single rule or in form of a chain of rules, the latter being much more harder to detect. In Figure 1, Rule r_3 is an example of non-executable rule.
  Subsuming Rules exist in two different ways, pairwise and in a chain of rules. The former occurs when the antecedents of one rule is a proper subset of another while the consequents are exactly the same and vice versa. The latter occurs due to different inference paths, involving multiple rules, from a given antecedent to a final conclusion. We say that rules R subsumes R', if for some substitution σ, $R' \rightarrow R\sigma$, where $R\sigma$ denotes the instance of R obtained by carrying out substitution σ in R (Preece and Shinghal, 1994). For example, in Figure 1, rules r_5 and r_6 are examples of subsumption, where the consequents of Rule r_6 is a subset of the consequents of Rule r_5, thus causing Rule r_5 to subsume Rule r_6.
  Logical Equivalence is present when two rules effectively convey the same logical meaning, i.e., they logically imply one another. These type of rules are also called *duplicate* rules. We say that rules R and R' are duplicates iff $(R \rightarrow R'\sigma) \wedge (R' \rightarrow R\sigma)$ for some substitution σ.

3 DETAILED METHODOLOGY

We present a detailed explanation of the approach used in our tool to perform automatic verification of business rules. The tool applies two different methods, layered in a systematic way to achieve efficient verification. Initially, the rules are clustered on the `SBVR` *facts* level, aided by the definitions provided in the corresponding `SBVR` vocabulary. Clustering is followed by simultaneous transformation of the business rules provided in natural language to a simplified representation (stripping off any restrictive quantifications that are present in the rules) and `SMT-LIBv2` representation. The transformed rules are used for the verification techniques.

3.1 Clustering

The major concern for business rule verification has been the huge spike in computation time and cost, due to the large size of real life business rule sets. Thus to reduce the number of business rules as input to our tool, we present a method of clustering the business rules beforehand, based on their relationship with each other and the `SBVR` vocabulary definitions. We intend to put 'related' business rules in one cluster, and rules belonging to a particular cluster are taken as input to the verification approaches. Let there be n number of business rules, m clusters are formed and c_i denote the number of rules in cluster i, i.e., $\sum_{i=1}^{m} c_i = n$. Since $c_i < n$, the input size to the tool is reduced. In real life scenarios, rules of a business enterprise are function specific, e.g., a bank will have different rules regarding legal charges and database change for payment failure. Clustering will ensure that we verify the two set of rules separately, rather than together.

As mentioned earlier, `SBVR` representation consists of a vocabulary, where the *noun concepts* and *facts* are specified. *Rules* are generated by creating conditions and relations among the *facts*. Thus SBVR provides us with a definition file which simplifies our goal of clustering. We put rules sharing a common *fact* in one cluster. Figure 2 provides us with an example of our clustering method. Cluster 1 has two rules, which are related by the common fact *customer rents car*. The third rule is formed using two *facts* which are not used in any of the rules of Cluster 1, resulting in it being in a separate cluster.

Cluster 1:
It is necessary that if customer rents car then customer has *at least* 2 credit_card.
It is necessary that if customer rents car then customer_age is greater than 18.
Cluster 2:
It is necessary that if customer owns luxury_credit_card then customer gets surprise_gift.

Figure 2: Example of Clustering SBVR rules.

3.2 Transformation

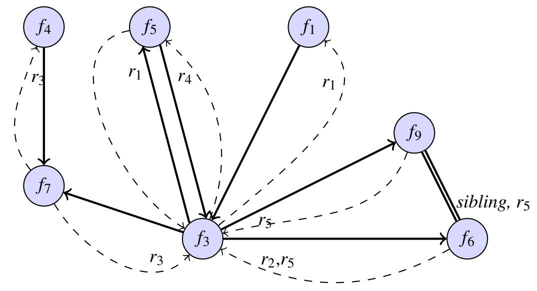
`SBVR` provides the option to apply quantification on rules, enabling the domain experts to represent business rules as close to the real world as possible. Rule r_1 in Figure 3 shows an example of `SBVR` business rules with quantification, where we quantify the number of credit card that a customer has to own in order to rent a car. We undertake two different transformations on the business rules, *preserving quantification* and *discarding quantification*. The latter generates a set of *simplified rules*, unifying the quantification throughout the entire set of rules. We use these newly generated rules as input to our digraph based verification technique. An example of this transformation is shown in Rule r_1 in Figure 3.
Each *simplified rule* is represented as a relation between its underlying base *facts*. The *simplified rules* are made free from disjunctions by transforming rules with disjunction into a set of rules and made to involve implications, without loss of generality. e.g., a simplified business rule of the form $f_1 \vee f_2 \rightarrow f_3$ is represented as $f_1 \rightarrow f_3$ and $f_2 \rightarrow f_3$, where f_1, f_2 and f_3 are `SBVR` *facts*.
To ensure that our tool captures the anomalies involving the quantifications present in modern business rules, we perform another form of transformation. The `SBVR` rules are mapped to `SMT-LIBv2`. The mappings are generated based on the `SBVR` XMI and the generated `SMT-LIBv2` representation incorporates the quantifications that were present in the original rule.

3.3 Directed Graph Based Verification

The use of graphical techniques for verification of rules is attractive, since graphs provide an easy to use framework to represent conceptual dependencies. Complex relationships between rules can be represented as paths in the graph, and the verification problem is reformulated as one of reachability of specific nodes in the graph. From our study, we observe that most of the approaches involve building the entire

r_1 : **SBVR Rule:** It is necessary that if customer rents car then customer owns *at least* 2 credit_card.
Simplified Rule: rents(customer , car) $\rightarrow$ owns(customer , credit_card)

Figure 3: Transformation of SBVR to Simplified Rules.



f_1 :	car is luxury_car	f_6 :	$\neg$ driver owns license
f_3 :	car has driver	f_7 :	car is rented_car
f_4 :	$\neg$ car has driver	f_9 :	driver follows traffic_rules
f_5 :	driver owns license		

Figure 4: Representation of rules (Figure 2) according to our proposed graph structure.

graph initially, followed by performing various computations on the corresponding graph representations (mostly adjacency matrices) to identify errors, which due to the large size of the graphs becomes extremely inefficient. We present a different approach, where we identify errors among the rules while the graph is built in an incremental manner, i.e., rule by rule, without undergoing any complex computations on the complete graph representation at the end.

We represent our set of business rules as a collection of nodes and directed edges, i.e., a directed graph. Each *fact* used in the representation of a rule and it's negation, is represented as a node in the graph. We draw directed edges from the *facts* present in the antecedent to the *facts* forming the consequent of the rule. These edges are called the *successor links*. Similarly, directed edges are drawn from the *facts* belonging to the consequent to the *facts* forming the antecedent. We call these edges *ancestor links*. Both *ancestor* and *successor* relationships are transitive in our graph representation. For example, we draw a *successor link* from x to y for rule $x \rightarrow y$. If y has as successor two nodes p and q, then a *successor link* is drawn from x to both p and q. Thus we ensure that from a node, any of it's ancestor or successor can be visited. We introduce a new form of relationships among the *facts* belonging to the consequent of a rule termed as *sibling* relationship, to show that they are asserted together. e.g., in Rule r_5 in Figure 1, if *car has driver* then *driver does not own license* as well as *driver follows traffic rules*. Both of the *facts* in the consequent are termed *siblings*. Along with *sibling* relationship we further introduce two new relationships *ancestor-sibling* and *successor-sibling*. We say that *facts* which share a sibling relationship with any of ancestors of a fact δ, are *ancestor-sibling* of δ, while *siblings* of any successor of δ are termed as *successor-sibling* of δ.

To give an example of our graph representation, let us consider the rules in Figure 1. The set of rules represented according to our proposed graph structure is shown in Figure 4. *Successor* relationships are shown by the solid edges, *ancestor* relationships by the dashed edges and the *sibling* relationship by double lines. The rule numbers corresponding to the relationships are mentioned between the dashed and solid edges.

Each node in the graph has the following information stored: *list of ancestors, list of siblings, list of successors, list of successor-siblings and list of ancestor-siblings* along with the corresponding rule sequences through which these relations have been formed. A *fact* can have the same relationship with another *fact* in multiple rules, thus leading to multiple rule sequences being stored corresponding to each *fact* in each of the relation list. For a new rule, edges are established and the new nodes (if any) exchange information from their immediate ancestor or immediate successor node. For example, if there are two nodes n_1 and n_2 and we encounter a new rule $r_1 : n_1 \rightarrow n_2$, an *ancestor edge* from n_2 to n_1 is created and the ancestor list from n_1 is added to n_2. A *successor edge* will be created from n_1 to n_2 with the successor list of n_2 being added to that of n_1. This updated successor list will be propagated to all the ancestors of n_1. Thus for each node $x \in \text{ancestor}(n_1)$, now $x \in \text{ancestor}(n_2)$ through the union of rule number r_1 and the chain of rules via which $x \in \text{ancestor}(n_1)$. Similarly, each x will have n_2 in its successor list. Next the updated ancestor information from n_2 is passed onto each of it's successors. Each $x \in \text{ancestor}(n_2)$, is now an ancestor of every $y \in \text{successor}(n_2)$ with rule r_1 added to the corresponding rule sequence, while y is added to successor(x). e.g., n_2 has a successor n' via rule numbers r_λ, r_β. Then the ancestor n_1 of n_2, will store n' in it's successor list with corresponding rule sequence as r_1, r_λ, r_β, while $n' \in \text{successor}(n_1)$ through rule sequence r_1, r_λ, r_β.

We use this graphical representation to identify *non-executable* conditions and *circularity* among business rules. While the graph is being generated in an incremental fashion, we check for anomaly conditions among the different lists stored in each node. As explained earlier, non-executable rules are those where conflicting *facts* are present in the antecedent of a business rule, or a sequence of rules, resulting in the consequent of some rule to be never asserted. After the addition of a new rule, non-executable conditions can exist in the following combinations.

1. Node n previously has node n_α as it's ancestor and $\neg n_\alpha$ is added to the ancestor list.
2. We add $\neg n$ to the ancestor/ successor list of n or vice versa.

3. Node n has n_α in it's ancestor list and $\neg n_\alpha$ is added as a sibling to some ancestor (ancestor-sibling) of n, or vice versa.

Circularity condition exists when a node n has the same node n_γ in it's ancestor and successor list, ancestor-sibling and successor list or ancestor and successor-sibling list.

If a `SBVR` *fact* $n_\delta \in$ `ancestor`(n), due to the presence of multiple rule sequences, i.e., $\left|\texttt{RID}^a_n(n_\delta)\right| > 1$, and n_δ is a direct ancestor (rule sequence of size 1) of n at least once, we say that *chained subsumption* exists.

When any of the above conditions are detected, it flags the presence of a possible anomaly. The set of rules causing the possible anomaly are found out by combining the rule sequences via which the relations were established to the conflicting nodes. e.g., $n' \in \text{ancestor}(n)$ via rule sequences R_ϕ, R_ψ and $\neg n' \in \text{ancestor}(n)$ via rule sequence R_ξ. The possible rule combinations generating the non-executable conditions are $\left\langle R_\phi, R_\xi \right\rangle$ and $\left\langle R_\psi, R_\xi \right\rangle$.

Implementation: The digraph proposed in this work involves each node storing copious amount of information. It is evident that traditional methods of graph representation, i.e., adjacency matrix and lists will not suffice. We use a modified version of adjacency lists for our implementation, with the aim of fast access to enable rapid information exchange between multiple nodes for a rule. To ensure direct and fast access to the different lists of each node and quick checks for anomaly conditions between the lists after each update, we represent each `SBVR` *fact* by an *odd number*. The next *even number* denotes the negation of the *fact*. Equation 1 presents a formal representation of the mapping, where $\mathcal{S}$ is a set of `SBVR` *facts* and their negation.

$$\begin{aligned} \mathcal{S} &= \{f_1, \neg f_1, f_2, \neg f_2, \ldots, f_n, \neg f_n\} \\ \mathcal{O} &= \{1, 2, 3, 4, \ldots, 2n-1, 2n\} \\ \text{We define a function } \mathcal{F}&\text{, such that} \\ &\mathcal{F} : \mathcal{S} \mapsto \mathcal{O} \end{aligned} \tag{1}$$

Due to this representation the checks for anomaly condition become very straightforward, e.g., for non-executable condition while inserting an even number to a list l we need to check for the presence of the previous odd number in list l' (based on the error condition check) and vice versa. Also, the use of numbers as node representation makes access to the lists of a particular node in the graph similar to how an array position is accessed given the index, i.e., in a direct manner.

3.4 SMT-LIBv2 based Verification

Modern day business rules along with structural anomalies contains anomalies due to quantifications. Graphical techniques are not sufficient to detect these form of anomalies, leading to a different verification approach.

3.4.1 Mapping SBVR XMI to SMTLibv2

From the `SBVR` vocabulary shown in Figure 1, it can be seen that the rule r : 'It is obligatory that if rented car has driver then driver has license' is based on `SBVR` *fact* f : 'car has driver'. The binary fact $bf \rightarrow n_1 v_1 n_2$, where n_1 (subject) and n_2(object) are *noun concepts* related through *verb concept* v_1 is mapped as,

(declare-fun v_1 (Thing Thing) Bool)

The concepts n_1 and n_2 are declared as constants of sort 'Thing'. `SBVR` also permits a noun concept n_2 to be derived from another noun concept n_1. For instance, noun concept 'rented_car' is derived from another noun concept 'car'. The corresponding `SMT-LIBv2` representation is generated as:

```
(declare-const rented_car Thing)
(declare-fun rented_car_fun ( Thing ) Bool)
(assert (rented_car_fun rented_car ) )
(assert (forall((x Thing))
    (implies(rented_car_fun x)(car_fun x))))
```

In Figure 1, the execution of rules r_2 and r_6 can create an ambiguity in business system resulting in inconsistent state of knowledge base.

An instance of `SBVR` rule mapping to `SMT-LIBv2` for rules r_2 and r_6 is given below:

```
smt (r2 :)
(assert (forall((x Thing)(y Thing)(z Thing))
(implies
  (and (car_fun x)(driver_fun y)
        (license_fun z) ( has x y))
  (owns y z))))
smt (r6 :)
(assert (forall((x Thing)(y Thing)(z Thing))
(implies
  (and (car_fun x)(driver_fun y)
        (license_fun z) ( has x y))
  (not(owns y z)))))
```

The aim of mapping `SBVR` rules to `SMT-LIBv2` is to be able to use SMT solvers for verification. This approach has the following advantages.

1. SMT generalizes Boolean Satisfiabilty (SAT) by adding equality reasoning, arithmetic, fixed size bit vectors, arrays, quantifiers and other logics and useful first order theories (Barrett et al., 2009; De Moura and Bjørner, 2011). SMT solvers are efficiently able to handle such theories, thus providing the support for verification considering quantifications.

2. SMT provides a provision to check satisfiability, validity and un-satisfiabilty of the formulas represented in SMT-LIBv2 format which is captured in our approach to detect conflicting rules.

Mappings have been completed for 50 `SBVR` constructs to the corresponding `SMT-LIBv2` which is an extension of the work performed by (Chittimalli and Anand, 2016) which includes `SBVR` vocabulary, rules, definitions and various other concepts (e.g. synonym, synonymous form, inverse verb concept, atleast-n , atmost-n, exactly-n, universal and existential quantifications).

3.4.2 Conflicting Rules Detection

The presence of conflicting conditions in a set of rules shall cause the latter to be inconsistent. Therefore it is important to check for conflicting conditions within individual rules and between different rules. Conflicting rules can exist in one of the following two scenarios:

1. When a rule is enumerated using `AND` operator, i.e., r_1: 'It is obligatory that f_1 and f_2 and.....f_3' (f_i*'s* are fact types) and conjunction of some fact types contradicts with another fact type in the rule. Formally, the conflicting condition can be represented as $fol(f_i) \wedge fol(f_j) ....\wedge fol(f_k) \Rightarrow \neg fol(f_m)$, where $fol(f_i)$ is the FOL formula corresponding to f_i.
2. A rule base R consists of rules = $\{r_1, r_2,.... r_n\}$ and the conjunction of some rules in R is contradictory with some other rule/ rules in R, i.e.,
$fol(r_i) \wedge fol(r_j) ....\wedge fol(r_k) \Rightarrow \neg fol(r_m)$ where $fol(r_i)$ is the FOL formula corresponding to r_i.

Our tool aims to find the minimal set of rules that are inconsistent with respect to each other, i.e., the minimal unsatisfiable cores, aided by the Satisfiable Modulo Theories (SMT) solvers. We use the definition of `validity` and `satisfiability` that exists in First Order Logic in our verification approach.

DPLL($\mathcal{T}$)-BASED SMT SOLVERS checks for the satisfiability of the SMT formulas (generated from `SBVR` rules). The underlying logic is that if formulas given as input to the solvers are consistent, i.e., the business rules have no conflict present among them then `sat` will be returned by the solvers else `unsat`. The working of the DPLL($\mathcal{T}$)-BASED SMT SOLVERS can be visualized as that of a transition system. The initial state of the transition system is given as $\{\emptyset, F_0, \emptyset\}$, where F_0 is a given set of clauses to be checked for satisfiability (i.e., the input formula). When F_0 is unsatisfiable in $\mathcal{T}$ the expected final states is *fail* or $\emptyset$. Here $\mathcal{T}$ is the background theory considered while mapping from `SBVR` to `SMT-LIBv2`, i.e., effectively the transformations (Katz et al., 2016).

A conflicting set of rules can be found by putting `assert` statements in `SMT-LIBv2`, which compels the solvers to check the satisfiability until the point of invocation. We add `labels` to the assertions to identify the exact rules which caused the unsatisfiability (conflict) to occur. The command '`get-unsat-core`' use these `labels` to generate UNSAT core output in a representation which is easily understandable by logic experts.

3.4.3 Redundancy : Subsuming and Duplicate Rules Detetction

We use the generated `SMT-LIbv2` formulas to detect the pair of subsuming rules involving quantifications in business rule set. A formula F subsumes another formula F' (F$\succ F'$) if for each interpretation I, $I \models F'$ *implies* $I \models F$ (Lukichev, 2010). We use this definition to devise our verification approach of detecting subsumption. Let smt(r_i) be the `SMT-LIBv2` representation of rule $r_i \in R$. As per the definition of subsumption provided in Section 2, in order to check whether a rule $r' \in R$ is subsumed by another rule $r \in R$, we propose to `assert` the negation of the formula obtained by smt(r) →smt(r').

$$\neg(smt(r) \rightarrow smt(r'))$$
$$\equiv \neg(\neg smt(r) \vee smt(r')).........[MaterialImplication]$$
$$\equiv (smt(r) \wedge \neg smt(r')).........[DeMorgan'sLaw]$$

Our assumption is if r subsumes r', then the solver should find a solution to the formula smt(r) →smt(r'). Since we check for the satisfiability of the *negation* of the formula, if the solver returns `UNSAT`, i.e., no solution exists where smt(r) is satisfied but smt(r')is not, it is deduced that r subsumes r'. Figure 5 shows rules which are subsumed due to quantifications.

r_3: It is permitted that if car rental *is insured by* at least 2 credit cards then car rental *is luxury car rental*.
r_4: It is permitted that if car rental *is insured by* exactly 3 credit cards then car rental *is luxury car rental*.

Figure 5: SBVR Rules created from EURent vocabulary that are subsuming.

While checking for the satisfiability of *smt*(r_3) $\wedge$ ¬*smt*(r_4), the SMT solvers fail to find a value $x \in$ Thing that satisfies *smt*(r_3) but does not satisfy *smt*(r_4), resulting in `unsat`. This happens because if a constant satisfies the antecedent of rule r_4 then by default it is going to satisfy the antecedent of r_3 as the formula *EqualsTo(credit_card.quantity (x), 3)* is subsumed by *GreaterThan(credit_card.quantity (x),*

Table 1: Performance of our Tool on Business Rules.

Business Rule Set	Number of Rules	Anomalies Detected		
		Conflicting Rules	Redundant Rules	Circularity
EU-Rent Car Rental	84	14	5	1
Industrial Insurance Application	125	12	6	0
Photo Equipment	35	2	3	0
Loan Contracts	11	2	2	0

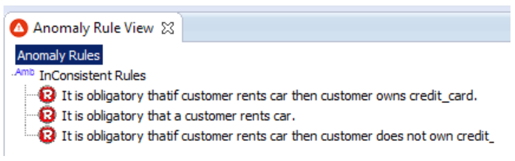
2). Satisfiability check on *smt*(r_4) $\wedge$ $\neg$*smt*(r_3) gives SAT while that on *smt*(r_3) $\wedge$ $\neg$*smt*(r_4) gives an UNSAT, depicting that r_3 subsumes r_4, but the reverse is not true.

As explained earlier in Section 2, *logical equivalence* or *duplicates* are special cases of subsumption. From the corollary of the approach employed to detect subsumption in rules, we propose that two rules are *duplicates* or *logical equivalent* of one another if $\neg\,(smt(r) \rightarrow smt(r'))$ and $\neg\,(smt(r') \rightarrow smt(r))$ yields UNSAT.

4 IMPLEMENTATION AND EXPERIMENTAL RESULTS

We have developed an intelligent editor to specify business rules in SBVR. The editor removes spelling mistakes and SBVR syntactical errors like duplicate fact detection, absence of *term* from SBVR vocabulary but used in *rules* or *facts*. The business rules specified in the editor undergo the pre-processing step, where they are clustered and then transformed to simplified rules and SMT-LIBv2. We check the satisfiability of the generated SMT-LIBv2 formulas using z3 solver (De Moura and Bjørner, 2008), while the graph based verification algorithm has been implemented in Java. For both the approaches, we highlight the set of rules which are the cause of the possible anomaly. While performing experiments of our tool, we felt that there was a dearth of universally accepted benchmarks for this verification problem. The work done by (Mitra and Chittimalli, 2017) also stated about the same drawback in this research field.

Table 1 shows the distribution of anomalies identified by our tool. EU-Rent Car Rental, Photo Equipment and Loan Contracts consists of seeded anomalies, where the anomalies were injected in by a business expert ignorant of our approaches. The tool performed exceptionally well identifying all the anomalies injected. We executed the Industrial Insurance Application Case Study twice. Initially the original version was given as input, i.e., no anomalies were injected from our side. Our tool identified 6 redundancies in the rule set. The business expert could identify only these 6 anomalies on manual inspection of the case study. This experiment enabled us to show that our tool performs efficiently well on real life business rules. Later we executed the Case Study with conflicting conditions injected by our business expert. Our tool could identify 12 conflicting rules while 15 were injected by the expert. 3 rules were not identified by our tool because they involved quantifications via numerical computations (addition, multiplication, etc.), for which we have not yet generated SMT-LIBv2 mappings. Figure 6 shows a snapshot of our tool in work.



It is obligatory thatif customer rents car then customer owns credit_card.
It is obligatory that a customer rents car.
It is obligatory thatif customer rents car then customer does not own credit_

Figure 6: Snapshot of our Tool.

5 RELATED WORK

Various solutions had been proposed for automatic rule base knowledge verification, where the rules were mostly represented in sentential logic form. The approaches included verification with the aid of petri nets (He et al., 2003; Chavarria-Baez and Li, 2006), directed graphs (Nazareth and Kennedy, 1991), directed hypergraph (Ramaswamy et al., 1997), inference graphs (Nguyen et al., 1987) and hypergraphs (Valiente, 1993). These approaches presents extreme computational complexity on present day business rules, along with not detecting anomalies involving data and quantifications. (Yeh and Chu, 2008) proposed a DNA-based computing algorithm to detect structural errors in rule based systems, which had low computational complexity but failed to deal with anomalies involving quantifications. Recently the problem of automatic verification of large and complex business rules has gained popularity. (Mitra and Chittimalli, 2017) presents a survey showing the present state of research for this problem. Conversion of SBVR based business rules to ontologies and then using reasoners to identify anomalies is an approach that has been presented by (Ceravolo et al., 2007; Karpovič et al., 2016; Reynares et al., 2014; Solomakhin et al., 2013). Most of these works fail to show the efficiency and completeness of their approach, and results on real life complex business rules. A different approach was presented by (dos Santos Guimarães et al., 2014), where the business rules were modeled into Alloy notation and the Alloy tool was used to detect anomalies. This approach suffered from high execution time and completely failed to detect anomalies existing in a sequence of rules. (Chittimalli and Anand, 2016) detected only inconsistencies modeling the rules into `SMT-LIBv2`.

6 CONCLUSION AND FUTURE WORK

In this paper, we present a tool to automatically detect anomalies present in business rules using a assortment of different techniques. We successfully detect anomalies with quantifications along with the ones not involving quantification. We better previous graph based rule verification techniques bypassing the adjacency matrix computations of high complexity, while we present mappings to SMT-LIBv2 enabling use of solvers. As per our knowledge, our tool is the first to use a combined approach to tackle the problem of detecting anomalies in business rules. We show experimental results on standard benchmarks along with some industrial data sets. In the future, the aim is to extend the graph based verification to be able to detect anomalies involving quantifications along with optimizing the performance of the logic solvers. We also intend to test our approaches on more complex real life business systems.

REFERENCES

Barrett, C. et al. (2010). The smt-lib standard: Version 2.0.

Barrett, C. W., Sebastiani, R., Seshia, S. A., and Tinelli, C. (2009). Satisfiability modulo theories. *Handbook of satisfiability*.

Ceravolo, P. et al. (2007). Modeling semantics of business rules. In *2007 Inaugural IEEE-IES Digital EcoSystems and Technologies Conference*, pages 171–176. IEEE.

Chavarria-Baez, L. and Li, X. (2006). Structural error verification in active rule-based systems using petri nets. In *Artificial Intelligence, 2006. MICAI'06*. IEEE.

Chittimalli, P. K. and Anand, K. (2016). Domain-independent method of detecting inconsistencies in sbvr-based business rules. In *Proceedings of the International Workshop on Formal Methods for Analysis of Business Systems*.

De Moura, L. and Bjørner, N. (2008). Z3: An efficient smt solver. In *14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer-Verlag.

De Moura, L. and Bjørner, N. (2011). Satisfiability modulo theories: introduction and applications. *Communications of the ACM*.

dos Santos Guimarães, D. et al. (2014). A method for verifying the consistency of business rules using alloy. In *Proceedings of the Twenty-Sixth International Conference on Software Engineering & Knowledge Engineering*.

Group, B. R. (2016). EU-Rent Car Rental case study. http://www.kdmanalytics.com/sbvr/EU-Rent.html.

He, X. et al. (2003). A new approach to verify rule-based systems using petri nets. *Information and Software Technology*.

IBM (2017). Business action language. https://goo.gl/Ybqj1m.

JBoss (2017). Drools. https://goo.gl/KivdD2.

Karpovič, J. et al. (2016). Experimental investigation of transformations from sbvr business vocabularies and business rules to owl 2 ontologies. *Information Technology And Control*.

Katz, G., Barrett, C., Tinelli, C., Reynolds, A., and Hadarean, L. (2016). Lazy proofs for dpll (t)-based smt solvers. In *Formal Methods in Computer-Aided Design (FMCAD), 2016*. IEEE.

Lukichev, S. (2010). Improving the quality of rule-based applications using the declarative verification approach. *International Journal of Knowledge Engineering and Data Mining*.

MIT (2015). Alloy: A language and tool for relational models. http://alloy.mit.edu/alloy/index.html.

Mitra, S. and Chittimalli, P. K. (2017). A systematic review of methods for consistency checking in sbvr-based business rules.

Nazareth, D. (1989). Issues in the verification of knowledge in rule-based systems. *Int. J. Man-Mach. Stud.*

Nazareth, D. L. and Kennedy, M. H. (1991). Verification of rule-based knowledge using directed graphs. *Knowledge Acquisition*.

Nguyen, T. A. et al. (1987). Verifying consistency of production systems. In *Proceedings of the Third Conference on Artificial Intelligence Applications*, pages 4–8.

OMG (2013). Semantics of business vocabulary and rules 1.2. http://www.omg.org/spec/SBVR/1.2/.

Preece, A. D. and Shinghal, R. (1994). Foundation and application of knowledge base verification. *International Journal of Intelligent Systems*.

Ramaswamy, M. et al. (1997). Using directed hypergraphs to verify rule-based expert systems. *IEEE Trans. on Knowl. and Data Eng.*

Reynares, E. et al. (2014). Sbvr to owl 2 mappings: an automatable and structural-rooted approach. *CLEI Electronic Journal*.

Sirin, E. et al. (2007). Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*.

Solomakhin, D. et al. (2013). Logic-based reasoning support for sbvr. *Fundamenta Informaticae*.

Valiente, G. (1993). Verification of knowledge base redundancy and subsumption using graph transformations. *International Journal of Expert Systems*, 6:341–355.

Yeh, C.-W. and Chu, C.-P. (2008). Molecular verification of rule-based systems based on dna computation. *IEEE Transactions on Knowledge and Data Engineering*.