Representing and Evaluating SBVR Specification via ASP

Simone Caruso[1], Carmine Dodaro[2] and Marco Maratea[2]

[1]DIBRIS, University of Genova, Italy
[2]DeMaCS, University of Calabria, Italy

Abstract

Semantics of Business Vocabulary and Rules (SBVR) provide a formal framework for representing business rules in a structured and human-readable manner and they have an important role in aligning business logic with enterprise requirements. However, a recent survey has highlighted several limitations in current approaches to SBVR conflict detection and analysis. In this work, we propose a novel approach based on Answer Set Programming (ASP) that addresses most of such limitations as it offers a simple and declarative way of representing SBVR, a set of robust tools for conflict explanation, and high-performance solvers. Moreover, to assess the scalability of our approach, we conducted an experimental analysis using synthetically generated datasets comprising thousands of conflicts.

Keywords

Answer Set Programming, Controlled Natural Language, Semantics of Business Vocabulary and Rules

1. Introduction

The Semantics of Business Vocabulary and Business Rules (SBVR) [1] is a standard established by the Object Management Group (OMG) for the specification of business rules in a structured, yet human-readable, natural language format. SBVR enables domain experts to formally express business constraints and policies without requiring knowledge of traditional programming languages or formal logics. This makes it an essential component in model-driven development, regulatory compliance, and business process management. By fostering a shared vocabulary and unambiguous rule definitions, SBVR promotes consistency, traceability, and verifiability across enterprise systems [2, 3].

However, rule conflicts, defined as logical contradictions or inconsistencies within a set of sentences, can undermine system reliability, introduce ambiguities, and obstruct automation. Consequently, conflict detection in SBVR is a crucial step in the validation and verification of business rules. As a matter of fact, despite significant interest in this area, a recent systematic survey of existing approaches conducted by [4] reveals that current techniques suffer from the following significant limitations:

- Some strategies do not consider certain types of sentences, but they only include prohibitions and permissions. However, excluding some sentences from the conflict detection can lead to undetected inconsistencies.
- Many approaches have a limited context window and they only compare individual sentences. In this case, it is not possible to determine whether the missing information hidden in the other sentences can lead to inconsistencies.
- The detection is often rule-based or requires training data for supervised machine learning, which leads to two problems: the scarcity of the data due to their high costs and it is unclear how well these methods generalize to diverse regulatory documents, as they may overfit to the specific characteristics of the data they were trained on.
- Many studies define conflict using only deontic logic (e.g., obligation vs. permission), but some situations don't involve full contradictions but discrepancies, such as when one statement is more specific than another.

Joint Proceedings of the Workshops and Doctoral Consortium of the 41st International Conference on Logic Programming, September 9–13, 2025, Rende, Italy

simone.caruso@edu.unige.it (S. Caruso); carmine.dodaro@unical.it (C. Dodaro); marco.maratea@unical.it (M. Maratea)
0000-0002-2724-4342 (S. Caruso); 0000-0002-5617-5286 (C. Dodaro); 0000-0002-9034-25276 (M. Maratea)

- The available datasets include a limited number of conflicts or synthetically generated.

In this paper, we aim to overcome the first four limitations, and partially address the fifth, by introducing a novel approach based on Answer Set Programming (ASP) [5, 6] for SBVR representation and for conflict detection, that we also implemented in an open-source tool called SBVR2ASP. The novel approach offers a number of advantages compared to existing ones. Indeed, ASP is a powerful and expressive formalism that can easily model rules, constraints, and logical dependencies. Thus, our approach is capable of capturing a wide range of SBVR sentence types, supports global reasoning across an unrestricted context window, and operates without supervision or manually defined rules, relying only on the logical specification of the problem. Another advantage of our ASP-based approach lies in the fact that the ASP community has developed a rich ecosystem of tools and methodologies for explanation and debugging [7], employed also in real applications, e.g., Healthcare [8, 9, 10, 11, 12, 13**?** , 14, 15, 16], also in combination with machine learning techniques (see, e.g., [17, 18]).which can be used to analyze and manage conflicts of the SBVR specification. Moreover, modern ASP solvers, as CLINGO [19] and WASP [20], support optimized grounding and solving techniques making them perfectly suitable for large-scale reasoning tasks, and more recent solvers, e.g., [21] work via compilation.

Among the limitations identified by [4], the reliance on synthetic data is particularly difficult to overcome. Unlike other issues, which can potentially be addressed through improved methodologies or technologies, the lack of real-world data is not a problem that can be solved at the technical level. It is inherently tied to the nature of the domain: real regulatory documents, business rules, and compliance policies often contain sensitive, proprietary, or strategically valuable information. As a result, organizations are generally reluctant to share such data, even in anonymized form. Indeed, these documents may include confidential legal clauses, internal governance policies, or competitive business logic that could reveal strategic insights or introduce legal risks if disclosed. Furthermore, in many cases, the rule sets themselves represent a significant intellectual asset with direct economic value, making companies even more protective of their dissemination.

Consequently, unless stakeholders are willing to provide access to such data under specific agreements or within closed evaluation settings, the only viable alternative remains the construction of synthetic datasets. This constraint has been consistently acknowledged by [4], and our work aligns with this trend.

In line with previous studies, we evaluated our approach using publicly available SBVR specifications combined with randomly generated synthetic data. While synthetic, this setup allows for controlled, large-scale experimentation and reproducibility. Our experiments demonstrate that our ASP-based approach is able to efficiently handle large datasets of business rules, even when they contain thousands of conflicts.

2. Preliminaries

SBVR [1], developed by the Object Management Group (OMG), is a standard designed to describe complex systems, such as businesses, in both a formal and natural way. It offers a structured method for defining business vocabulary and rules in a format that is both precise and easy to understand. The SBVR specification consist of two key components: vocabulary and rules. The vocabulary is a set of terms and definitions representing concepts, facts, and relationships within a business domain. More precisely, the vocabulary is made of nouns and verbs. Noun concepts can be classified as either general concepts or individual concepts. A general concept refers to a category that groups things based on shared properties, while an individual concept represents a specific, singular object. Verbs define relationships between two or more noun concepts or describe a characteristic of a noun concept. In the following, we present examples based on the EU-Rent SBVR specification, a car rental company provided by KDM Analytics[1]. EU-Rent operates in multiple countries, renting cars to customers through its branches. The specification includes vocabulary definitions and business rules governing its operations.

[1] https://www.kdmanalytics.com/sbvr/EU-Rent.html

Table 1
SBVR-SE keywords. In the table, n and m represent numbers, whereas p and q denote expressions or propositions that allow to combine the different operators into sentences.

Quantification Operators	Logical Operations	Modal Operations
each	it is not the case that p	it is obligatory that p
some	p and q	it is prohibited that p
at least one	p or q	it is necessary that p
at least n	p or q but not both	it is impossible that p
at most one	if p then q	it is possible that p
at most n	q if p	it is permitted that p
exactly one	p if and only if q	p must q
exactly n	not both p and q	p must not q
at least n and at most m	neither p nor q	p need not q
more than one	p whether or not q	p can q
no		p may q

Example 1 (Vocabulary). *A vocabulary is a sequence of elements of the form:*

```
rental
requested car group
period
rental period
    General Concept: period
full
    Concept type: individual concept
rental includes rental period
rental has requested car group
```

where `rental`, `requested car group`, `period`, `rental period` *and* `full` *are concepts. Moreover,* `rental period` *specializes the* `period` *concept, while* `full` *is an individual concept. Instead,* `includes` *and* `has` *are verbs that define relationships:* `includes` *links* `rental` *to* `rental period`*, and* `has` *connects* `rental` *to* `requested car group`*.*

Business rules, instead, are logical statements that define guidelines, constraints, or conditions that govern how a business operates. They define what can, must, or must not happen in a business process to ensure consistency, compliance, and efficiency. Business rules help enforce policies, regulations, and best practices within an organization.

Example 2 (Business rules). *The following represent two examples of business rules:*

```
It is necessary that each rental has exactly one requested car group.
It is necessary that each rental includes exactly one rental period.
```

It is possible to observe that the concepts introduced in the vocabulary are used in the business rules, together with SBVR keywords, to define constraints. In more details, SBVR allows facts and business rules to be expressed in various ways, including statements, diagrams, or a combination of both, depending on the intended purpose. One common method is through a Controlled Natural Language (CNL), i.e., a simplified subset of natural language (such as English) designed for clarity and consistency. Instead of the full complexity of natural language, a CNL employs a limited set of structures and common words to create a straightforward and structured representation of business knowledge.

In this paper, we focus on SBVR Structured English (SBVR-SE), a CNL described in Annex A of the OMG SBVR specification [1], as SBVR-SE can be seen as an effective compromise for bridging the gap between business experts and information technology professionals.

Table 1 presents the main keywords of the SBVR-SE grammar, while a complete and formal description can be found in [1]. We have three different types of keywords: quantification operators that precede

nouns, logical operations, and modal operations. By combining nouns, verbs, and the operators listed in Table 1, SBVR rules can be formulated.

Example 3 (Modal and quantification operators). *The following business rule uses the modal operator* **`It is obligatory that`** *and the quantification operators* **`each`** *and* **`exactly one`***:*

```
It is obligatory that each rental car is owned by exactly one branch.
```

In this statement, `rental car` *and* `branch` *are nouns, and* `is owned by` *is a verb; these concepts are defined in the vocabulary.*

Moreover, it is possible to use modal operators combined with the keyword **`only if`** to invert the modality.

Example 4 (Modal operators combined with only if). *The following business rules have the same meaning:*

```
A car may be rented only if the car is available.
A car must not be rented if the car is not available.
```

Additionally, SBVR-SE supports other keywords, such as **`the`**, **`a`**, **`an`**, used as quantification or as introduction of a name of an individual thing. The keyword **`that`** has different purposes depending on its position: *(i)* Before a designation for a noun concept, it is a binding to a variable (similar to **`the`**); and *(ii)* After a designation for a noun concept and before a designation for a verb concept, it is used to introduce a restriction on things denoted by the designated entity as shown in the following example.

Example 5 (Usage of `that`**).** *The following two business rules represent an example of the possible usage of the* **`that`** *keyword:*

```
It is necessary that the scheduled pick-up date/time of each advance rental is after the
    booking date/time of the rental booking that establishes the advance rental.
```

Similarly, **`who`** has the same meaning as the second use of **`that`**, but it is specifically used to refer to persons.

Example 6 (Usage of `who`**).** *In the following sentence, the* **`who`** *keyword is used to refer to the renter:*

```
It is permitted that a rental is open only if an estimated rental charge is provisionally
    charged to a credit card of the renter who is responsible for the rental.
```

Additionally, **`of`** establishes a relationship between the two preceding and following nouns. The expression `p` **`of`** `q` is equivalent to `q has p`.

Example 7 (Usage of `of`**).** *In the following, an example of a relation established by the* **`of`** *keyword:*

```
If the renter of a rental requests a price conversion then it is obligatory that the rental
    charge of the rental is converted to the currency of the price conversion.
```

In this sentence, we have an implication introduced by the operators **`If`** `p` **`then`** `q` *and the modal operator* **`it is obligatory that`***. Additionally, the* **`of`** *operator is again used to express relationships between nouns.*

3. Handling SBVR with ASP

In this section we describe our ASP-based approach and its implementation, SBVR2ASP, whose architecture is shown in Figure 1.

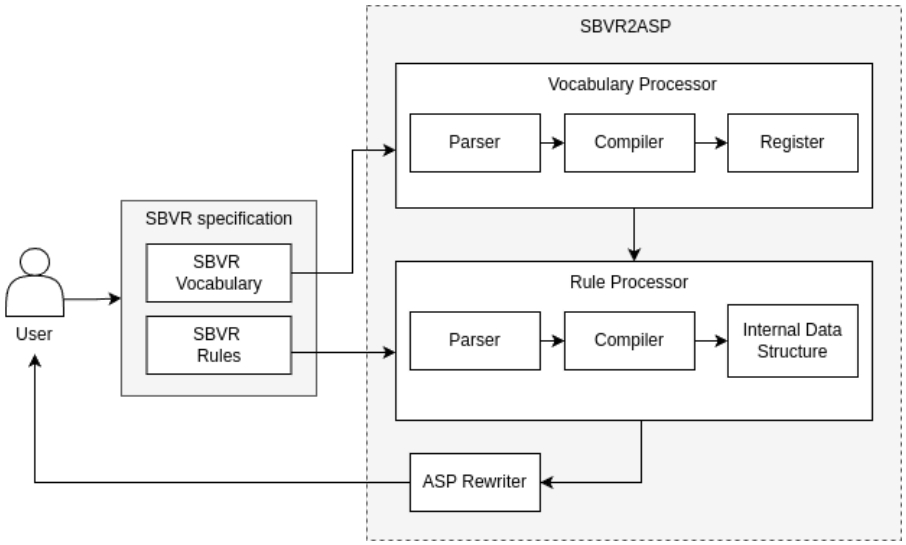

Figure 1: Architecture of SBVR2ASP.

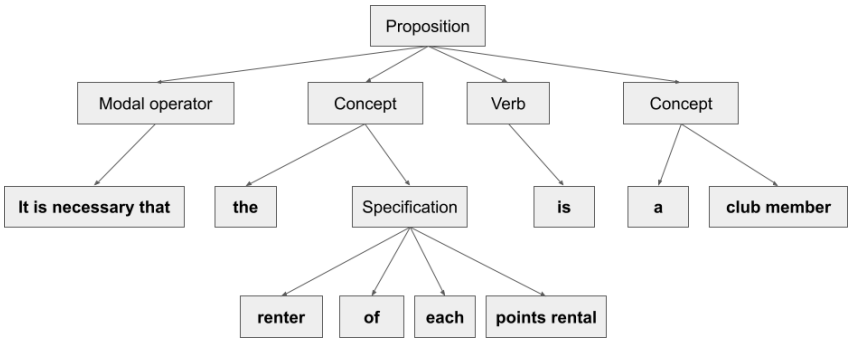

Figure 2: Example of an AST.

SBVR2ASP processes a SBVR specification written in SBVR-SE, which, as mentioned above, consists of two main elements: vocabulary and (business) rules, which are handled by the two main components of the tool, i.e., the Vocabulary Processor and the Rule Processor, respectively.

The Vocabulary Processor includes a Parser that tokenizes the input vocabulary and constructs an abstract syntax tree (AST). The AST is then processed bottom-up by the Compiler, which initializes a data structure, called Register, that tracks all entities of the specification and their relationships. In particular, this data structure saves the information retrieved from the vocabulary. Since SBVR nouns can be multi-word expressions of arbitrary length (e.g., `requested car group` from Example 1 is a single concept), the Parser alone cannot determine where a noun ends and a new token begins. Indeed, recognizing an undetermined sequence of characters can lead to ambiguity. To resolve this issue, the tool first assigns a unique identifier (ID) to every noun and verb in the Register. Then, these identifiers temporarily replace the corresponding terms within the SBVR specification ensuring that each concept is a single word during parsing, while still allowing the original expressions to be retrieved later using their assigned IDs.

Next, the Rule Processor parses the business rules using another Parser, constructing an AST. For example, Figure 2 shows a possible target AST for the following business rule:

`It is necessary that the` `renter` **`of each`** `points rental is` **`a`** `club member.`

However, as already mentioned, for the Parser, understanding that `points rental` is a single node is difficult; the parser could recognize, for example, **`each`** `points rental` or `renter` **`of each`** `points rental` as a single concept. Our solution is that each concept must be a single word; thus, `points rental` is temporarily replaced with its unique ID during parsing and then substituted back to its original form

in the final AST. Once the AST is built, it is processed bottom-up. This step initializes an Internal Data Structure, which facilitates data manipulation by reconstructing and organizing relationships between entities. Finally, the processed data is passed to the ASP Rewriter, which converts the extracted business rules into ASP rules. In the following, we show the main type of sentences for business rules using the EU-Rent specification introduced in Section 2. Due to space constraints, we only include sentences that illustrate distinct SBVR-SE constructs (see https://github.com/simocaruso/sbvr2asp for the full specification). Moreover, we assume the reader to be familiar with ASP.

3.1. Simple and complex sentences

Simple sentences are translated into ASP in a quite intuitive way, where each noun, i.e., the one declared in the vocabulary, is mapped to a predicate with a field serving as an identifier. On the other hand, verbs are represented as atoms of arity two, linking the corresponding nouns that define their relationship. As an example the following SBVR sentences:

```
It is necessary that each rental has exactly one requested car group.
It is necessary that each rental includes exactly one rental period.
It is necessary that each rental has exactly one return branch.
```

are translated into:

```
:- rental(REN), #count{requested_car_group(REQCARGRO):
    has(rental(REN),requested_car_group(REQCARGRO))} != 1.
:- rental(REN), #count{rental_period(RENPER):
    includes(rental(REN),rental_period(RENPER))} != 1.
:- rental(REN), #count{return_branch(RETBRA): has(rental(REN),return_branch(RETBRA))} !=
    1.
```

Note that the quantifier **each** does not require a direct translation in ASP due to grounding, while **exactly one** is represented with an aggregate. However, since the SBVR sentence begins with **It is necessary that** and ASP only supports integrity constraints, the condition of **exactly one** is flipped to be represented as *different from one*.

SBVR2ASP also supports more complex sentences that incorporate additional constructs, as the following one:

```
It is necessary that the scheduled pick-up date/time of each advance rental is after the
    booking date/time of the rental booking that establishes the advance rental.
```

As in the case of simple sentences, verbs and nouns are converted into atoms. Here, the keyword **after** is translated using the operator $<=$ due to the presence of **It is necessary that** (as it is part of an ASP constraint). Additionally, the keyword **of** establishes a relationship between two entities, e.g., between `rental date/time` and `rental booking`, though the relation itself is not explicitly named. To address this, the vocabulary is processed first, as explained in the tool architecture above, and the relevant information is stored in the Register. This allows us to dynamically retrieve the relationship and initialize the appropriate atom. Thus, the relationship:

```
the booking date/time of the rental booking
```

is translated into:

```
booking_date_time(BOODATTIM), rental_booking(RENBOO),
    has(rental_booking(RENBOO),booking_date_time(BOODATTIM))
```

Therefore, the translation of the whole sentence is the following:

```
:- scheduled_pick_up_date_time(SPUDT), booking_date_time(BDT), SPUDT <= BDT,
    advance_rental(ADR), has(advance_rental(ADR),scheduled_pick_up_date_time(SPUDT)),
    rental_booking(RB), has(rental_booking(RB),booking_date_time(BDT)),
    establishes(rental_booking(RB),advance_rental(ADR)).
```

It is important to observe that sentences may also contain conjunctions, which are translated into different ASP rules. As example, consider the following sentence:

It is obligatory that at **the** actual return date/time **of each** in-country rental and **each** international inward rental **the** local area **of the** return branch **of the** rental owns **the** rented car **of the** rental.

This is translated into two different ASP rules:

```
:- in_country_rental(ICR), actual_return_date_time(ARDT), has(in_country_rental(ICR),
    actual_return_date_time(ARDT)), #count{rented_car(RENCAR):
    owns(local_area(LA),rented_car(RENCAR)), local_area(LA), rented_car(RENCAR),
    in_country_rental(ICR), has(in_country_rental(ICR),rented_car(RENCAR))} < 1,
    return_branch(RETBRA), has(in_country_rental(ICR),return_branch(RETBRA)),
    is_included_in(return_branch(RETBRA),local_area(LA)).

:- international_inward_rental(IIT), actual_return_date_time(ARDT),
    has(international_inward_rental(IIT), actual_return_date_time(ARDT)),
    #count{rented_car(RENCAR): owns(local_area(LA),rented_car(RENCAR)), local_area(LA),
    rented_car(RENCAR), international_inward_rental(IIT),
    has(international_inward_rental(IIT),rented_car(RENCAR))} < 1,
    return_branch(RETBRA), has(international_inward_rental(IIT),return_branch(RETBRA)),
    is_included_in(return_branch(RETBRA),local_area(LA)).
```

where the first rule handles `in-country rentals`, while the second one applies the same rule to `international inward rentals`.

3.2. Only if and implication

As mentioned before, **`only if`** is used as a logical connector with a very specific meaning, aligned with formal logic, as it corresponds to a necessary condition. The following example shows the usage of such a construct:

It is permitted that a rental is open **only if an** estimated rental charge is provisionally charged to **a** credit card **of the** renter **that** is responsible for **the** rental.

This is translated as the following ASP rule:

```
:- rental(REN), open(OPE), REN = OPE, #count{credit_card(CC):
    is_provisionally_charged_to(estimated_rental_charge(ERC), credit_card(CC)),
    estimated_rental_charge(ERC), credit_card(CC), renter(RENT), has(renter(RENT),
    credit_card(CC)), is_responsible_for(renter(RENT),rental(REN))} < 1.
```

The resulting translation is that we cannot have a rental open if the second part of the sentence does not hold; that is, an estimated rental charge must be provisionally charged to a renter's credit card. The negation is handled by an aggregate in which it is ensured that at least one credit card provisionally charges the rental. On the other hand, a sentence of the form **`if`** ... **`then`** ... is an implication. An example of a sentence of such kind is the following:

If the renter **of a** rental requests **a** price conversion **then** it is **obligatory that the** rental charge **of the** rental is converted to **the** currency **of the** price conversion.

The translation is similar to the previous case, and also here an aggregate ensures that there exists a currency to which the rental charge is converted. Therefore, the resulting ASP rule is the following:

```
:- renter(RENT), price_conversion(PRICON),
    requests(renter(RENT),price_conversion(PRICON)), rental(REN),
    has(rental(REN),renter(RENT)), #count{currency(CUR):
    is_converted_to(rental_charge(RENCHA),currency(CUR)), rental_charge(RENCHA),
    currency(CUR), price_conversion(PRICON),
    has(price_conversion(PRICON),currency(CUR))} < 1, rental(REN),
    has(rental(REN),rental_charge(RENCHA)).
```

3.3. Arithmetic operations and time constraints

SBVR allows the usage of arithmetic operations, as shown in the following example:

It is necessary that the booking date/time **of a** points rental is at least 5 days before **the** scheduled start date/time **of the** rental.

where at least 5 days before is an arithmetic operation. This sentence is translated nicely into ASP by simply adding the number 5 to the booking date/time to enforce the required condition. The resulting rule is the following:

```
:- booking_date_time(BDT), scheduled_start_date_time(SSDT), points_rental(PR), BDT+5 >=
    SSDT, has(points_rental(PR),booking_date_time(BDT)),
    has(points_rental(PR),scheduled_start_date_time(SSDT)).
```

There are also sentences involving generic time constraints, as the following:

It is **obligatory that the** start date **of each** reserved rental is in **the** future.

In this case, we consider future as being after a constant representing the current date/time, denoted as now. Therefore, the corresponding ASP rule is the following:

```
:- start_date(STADAT), STADAT <= now, reserved_rental(RESREN),
    has(reserved_rental(RESREN),start_date(STADAT)).
```

3.4. Values and labels

SBVR sentences may include specific values that are defined in the vocabulary. The following sentence contains the term full, which is declared in the vocabulary as a possible value for the fuel level:

At **the** actual start date/time **of each** rental it is **obligatory that the** fuel level **of the** rented car **of the** rental is full.

In ASP, this is handled by using a constant named full. The resulting translation ensures that the fuel level cannot be anything other than full at the start of the rental, as follows:

```
:- rental(REN), actual_start_date_time(ASDT),
    has(rental(REN),actual_start_date_time(ASDT)), fuel_level(FUELEV), FUELEV != full,
    rental(REN), rented_car(RENCAR), has(rental(REN), rented_car(RENCAR)),
    has(rented_car(RENCAR), fuel_level(FUELEV)).
```

Finally, SBVR allows for labels that distinguish different instances of the same concept, as shown in the following example:

If rental1 is **not** rental2 and **the** renter **of** rental1 is **the** renter **of** rental2 **then** it is **obligatory that the** rental period **of** rental1 does **not** overlap **the** rental period **of** rental2.

The following is translated into the following ASP rule:

```
:- rental(REN_1), rental(REN_2), REN_1 != REN_2, renter(RENT_1), renter(RENT_2), RENT_1
    = RENT_2, has(rental(REN_1),renter(RENT_1)), has(rental(REN_2),renter(RENT_2)),
    rental_period(RENPER_1), rental_period(RENPER_2),
    overlap(rental_period(RENPER_1),rental_period(RENPER_2)),
    includes(rental(REN_1),rental_period(RENPER_1)),
    includes(rental(REN_2),rental_period(RENPER_2)).
```

It is important to observe that the two rental instances are assigned different variables, and they are explicitly treated as distinct (REN_1 != REN_2), ensuring that the constraint applies only when different rentals share the same renter.

4. Experiments

The performance of SBVR2ASP has been empirically evaluated on three publicly available datasets containing a SBVR specification, including vocabularies and rules, namely EU-Rent [1], Loan, and Photo Equipment[2] [22, 23]. The first one has been used as running example in this paper and describes a fictional car rental company operating across multiple countries, renting vehicles to customers through

[2]Loan and Photo Equipment can be downloaded from https://s2o.isd.ktu.lt/about.php.

Table 2
Average times (in seconds) for evaluating instances with no conflicts.

EU-Rent		Loan		Photo Equipment	
#Rentals	Time	#Loans	Time	#Photos	Time
500	0.29	100	0.01	10 – 100	0.01
5500	7.83	1 000	0.10	500 – 1 000	0.10
10 500	24.85	5 000	0.50	2 500 – 10 000	0.86
15 500	51.31	10 000	1.05	12 500 – 20 000	2.54
20 500	88.26	15 000	1.64	22 500 – 30 000	4.06
25 500	143.63	20 000	2.24		
30 500	169.14	25 000	3.03		
		30 000	3.50		

Table 3
Average times (in seconds) for detecting conflicts.

Number of conflicts	EU-Rent	Loan	Photo Equipment
0 – 1 500	0.28	0.05	0.09
1 501 – 3 000	7.43	0.55	0.78
3 001 – 4 500	27.10	1.18	1.51
4 501 – 6 000	55.13	1.77	2.26
6 001 – 7 500	87.35	2.27	3.27
7 501 – 9 000	152.87	3.07	3.86
9 001 – 10 000	156.47	3.53	4.52

its various branches. The second one includes a limited set of rules regarding interactions among debtors, banks, and loans. Finally, the Photo Equipment specification contains simple rules related to components of cameras.

As previously discussed, [4] identified several limitations in existing datasets. In particular, although they are publicly available, there are no real-world instances. Furthermore, the available synthetic datasets include only a limited number of conflicts (typically no more than a few hundred). Therefore, in designing our experiment, we generated synthetic data while deliberately increasing both the dataset size and the number of conflicts. In more details, for the EU-Rent dataset, we created instances with up to 30 500 rentals, varying the number of branches, countries, and customers. For the Loan dataset, we generated instances with up to 30 000 people and up to 20 different banks. Finally, for the Photo Equipment dataset, we modeled up to 30 000 cameras and related components. Concerning conflicts, we identified a set of potential inconsistencies within each dataset. Then, we assigned a probability (approx. 4%, as we observed it was a good empirical parameter to produce a sufficient number of conflicts) to each type of conflict, determining whether it would be activated and included in a given instance. This resulted in a number of conflicts up to 10 000 in the largest instances.

Each specification was translated into an ASP encoding using SBVR2ASP. We then used CLINGO (version 5.4.1) to check the satisfiability of each generated instance. All experiments were executed on a machine with an AMD Ryzen 7 5825U CPU @ 2.0GHz and 16 GB of RAM.

Table 2 reports the average solving times (in seconds) for instances without conflicts, across the three benchmark domains. CLINGO is able to process conflict-free instances efficiently, with solving times remaining low even as instance sizes grow significantly. In the EU-Rent setting, instances with up to 30 500 rentals are solved in less than 3 minutes. Moreover, in the Loan and Photo Equipment domain, even the largest tested instance (30 000 loans/photos) requires less than 5 seconds to be evaluated. Table 3 shows the average solving times for instances with conflicts, grouped by increasing conflict counts. As expected, solving times grow with the number of conflicts, but remain within reasonable limits across all domains. In the EU-Rent domain, the increase is more pronounced, with solving times reaching approximately 2.5 minutes for instances with up to 10 000 conflicts. Conversely, the Loan

and Photo Equipment datasets exhibit lower absolute times: even at the highest conflict levels tested, CLINGO completes the analysis in under 5 seconds for Photo Equipment and under 4 seconds for Loan. It is important to note that the solving times reported for both conflict-free and conflicting instances are largely dominated by the grounding phase performed by CLINGO. In our encodings, the solving process itself is essentially trivial: the programs are fully deterministic and do not involve any non-deterministic choices. As a result, once grounding is complete, the actual computation of the answer set is immediate. The observed execution times, therefore, reflect the cost of instantiating the rules over the (potentially large) input datasets.

5. Related Work

Several studies concern the translation of SBVR into logical formalisms with the aim of detecting conflicts and inconsistencies. For example, [24] proposed a method to translate SBVR, represented using Object-Role Modeling diagrams, into first-order deontic-alethic logic. Similarly, [25] translated SBVR-SE specification into the declarative modeling language Alloy, focusing on the validation of service choreographies. However, their approach is limited to deontic rules, reflecting its domain-specific orientation. Other approaches, instead, employ SMT (Satisfiability Modulo Theories) for conflict detection. [26] and [27] developed a tool that converts SBVR-SE into SMT [28] and uses SMT solvers (see, e.g., [29, 30]) for verification. However, their method does not consider all rules simultaneously in the verification phase. Instead, it first clusters business rules based on their relationships and SBVR vocabulary definitions to reduce the computational load. While this clustering improves efficiency, it may lead to undetected conflicts due to the exclusion of certain sentences during verification. In contrast, our approach, based on ASP, does not suffer from this limitation, as only grounded rules are processed by the solver. Alternative approaches employ machine learning techniques to identify inconsistencies. For example, [31] developed an approach to detect conflicts using sentence embeddings, obtaining an accuracy of 95%. In [32], the authors introduced a two-phase approach using traditional machine learning and convolutional neural network to compare business rules in documents and detect conflicts. They obtained an accuracy of 84%. Similarly, [33] developed NeuralConflict, a convolutional neural network-based model, and tested it on English and Chinese datasets, achieving F1-scores of 0.979 and 0.958, respectively. A significant limitation of these approaches lies in the scarcity of training data, and researchers must often construct custom datasets, which are typically limited in size. For a comprehensive review of existing approaches and their limitations, readers are referred to [4].

Concerning the translation of CNLs into ASP, [34] proposed a CNL specifically designed for solving logic puzzles. As in our approach, their CNL is automatically converted into ASP rules. [35] showed the process of translating the English sentence “A prime is a natural number greater than 1 that is not a product of two smaller natural numbers." into ASP code. Similarly, [36] defined the language PENG^{ASP}, a CNL that is automatically converted into ASP. Notably, PENG^{ASP} is designed to allow a conversion from CNL to ASP and then back in the other direction. More recently, [37] proposed a tool, called CNL2ASP, for converting a CNL into ASP. The resulting CNL also includes *domain definitions* with a role similar to the SBVR vocabulary, but is not fully compatible with the SBVR specification as it is more oriented towards solving hard combinatorial problems. Also recently, Deontic Answer Set Programming [38, 39] has been introduced, which integrates deontic logic concepts like obligation, permission, and prohibition into ASP. There is also ongoing research into using large language models (LLMs) to translate natural language into ASP. However, LLMs currently struggle to produce syntactically and semantically correct logic programs consistently. Due to their probabilistic nature, LLMs cannot guarantee the precision and reliability required in knowledge representation tasks, especially in critical contexts such as business rule consistency checking. KR languages like ASP are also highly sensitive to syntax and semantics, meaning that even minor errors (common in LLMs) can lead to unusable programs, or, even worse, incorrect programs. [40] suggested that LLMs such as GPT-3 can function as few-shot semantic parsers, transforming natural language into logical forms for ASP. However, as pointed out by the authors, some results are still unpredictable, and the LLM does not behave as intended. Other works focus on

specific tasks, as [41], which translates NL sentences into ASP facts, or [42], which supports some simple patterns. Thus, despite their promise, LLMs are not yet capable of reliably producing arbitrary ASP programs. CNLs instead remain a more dependable solution in domains requiring high accuracy, such as business rule inconsistency detection. However, a possible solution to this problem can be the approach proposed by [43], where natural language sentences are translated into ASP using the CNL2ASP system. The techniques introduced in their work could be adapted to rewrite natural language sentences in the SBVR CNL, and then SBVR2ASP could be used to convert those sentences into ASP. This would bring SBVR even closer to natural language, enhancing its accessibility and usability. Finally, it is important to note that a comparative evaluation across these tools is challenging. Many of them are not publicly available, and the datasets used in their evaluations are often inaccessible, limiting reproducibility and benchmarking.

6. Conclusion

In this paper, we proposed a novel approach based on ASP for processing SBVR specification. Our method directly addresses several limitations highlighted in a recent survey by [4], particularly those related to conflict detection in SBVR. To evaluate its effectiveness, we carried out an experimental analysis on synthetically generated datasets containing thousands of conflicts. The results demonstrate that our approach is well-suited for inconsistency checking in SBVR specification. As future work, it might be interesting to support other SBVR languages, as *RuleSpeak*, which is described in the annex H of the SBVR specification. Another promising direction for future work is the validation of SBVR2ASP on real-world scenarios. To overcome the current reliance on synthetic data, a possible way is to pursue collaborations with companies under NDA. Finally, to improve explainability and conflict detection in SBVR, we plan to extend the tool for computing minimal correction sets (MCS) or minimally unsatisfiable subsets (MUS) of business rules, similarly to was recently done for, e.g., configuration [44]. Finally, we remark that our tool and all the material needed to reproduce the experiments are available at https://github.com/simocaruso/sbvr2asp.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT in order to: Grammar and spelling check. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] The Object Management Group, Semantics Of Business Vocabulary And Business Rules. Version: 1.5, 2019.

[2] M. Ceci, F. A. Khalil, L. O'Brien, Making sense of regulations with SBVR, in: Supplementary Proceedings of the RuleML 2016 Challenge, Doctoral Consortium and Industry Track, volume 1620 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2016. URL: https://ceur-ws.org/Vol-1620/paper7.pdf.

[3] E. Abi-Lahoud, T. Butler, D. Chapin, J. Hall, Interpreting regulations with SBVR, in: Proc. of International Rule Challenge, Special Track on Human Language Technology and RuleML DC, volume 1004 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2013. URL: https://ceur-ws.org/Vol-1004/paper6.pdf.

[4] G. Schumann, J. M. Gómez, Detection of conflicts, contradictions and inconsistencies in regulatory documents: A literature review, in: Proc. of IDSTA, IEEE, 2024, pp. 81–88.

[5] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Commun. ACM 54 (2011) 92–103.

[6] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: Proc. of Logic Programming, MIT Press, 1988, pp. 1070–1080.

[7] J. Fandinno, C. Schulz, Answering the "why" in answer set programming - A survey of explanation approaches, Theory Pract. Log. Program. 19 (2019) 114–203.
[8] R. Bertolucci, C. Dodaro, G. Galatà, M. Maratea, I. Porro, F. Ricca, Explaining ASP-based operating room schedules, in: R. D. Benedictis, M. Maratea, A. Micheli, E. Scala, I. Serina, M. Vallati, A. Umbrico (Eds.), Proceedings of IPS'21 and RCRA'21, volume 3065 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021.
[9] C. Dodaro, G. Galatà, M. Maratea, I. Porro, Operating room scheduling via answer set programming, in: AI*IA, volume 11298 of *LNCS*, Springer, 2018, pp. 445–459.
[10] C. Dodaro, G. Galatà, M. Maratea, I. Porro, An ASP-based framework for operating room scheduling, Intelligenza Artificiale 13 (2019) 63–77.
[11] C. Dodaro, G. Galatà, M. K. Khan, M. Maratea, I. Porro, An ASP-based solution for operating room scheduling with beds management, in: P. Fodor, M. Montali, D. Calvanese, D. Roman (Eds.), Proceedings of the Third International Joint Conference on Rules and Reasoning (RuleML+RR 2019), volume 11784 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 67–81.
[12] M. Alviano, R. Bertolucci, M. Cardellini, C. Dodaro, G. Galatà, M. K. Khan, M. Maratea, M. Mochi, V. Morozan, I. Porro, M. Schouten, Answer set programming in healthcare: Extended overview, in: IPS and RCRA 2020, volume 2745 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020.
[13] M. Cardellini, P. D. Nardi, C. Dodaro, G. Galatà, A. Giardini, M. Maratea, I. Porro, A two-phase ASP encoding for solving rehabilitation scheduling, in: S. Moschoyiannis, R. Peñaloza, J. Vanthienen, A. Soylu, D. Roman (Eds.), Proc. of RuleML+RR 2021, volume 12851 of *LNCS*, Springer, 2021, pp. 111–125.
[14] P. Cappanera, M. Gavanelli, M. Nonato, M. Roma, Decomposition approaches for scheduling chronic outpatients' clinical pathways in answer set programming, J. Log. Comput. 33 (2023) 1851–1871.
[15] S. Caruso, G. Galatà, M. Maratea, M. Mochi, I. Porro, Scheduling pre-operative assessment clinic with answer set programming, Journal of Logic and Computation 34 (2023) 465–493.
[16] C. Dodaro, G. Galatà, M. Gebser, M. Maratea, C. Marte, M. Mochi, M. Scanu, Operating room scheduling via answer set programming: Improved encoding and test on real data, Journal of Logic and Computation 34 (2024) 1556–1579.
[17] P. Bruno, F. Calimeri, C. Marte, M. Manna, Combining deep learning and asp-based models for the semantic segmentation of medical images, in: Proc. of RuleML+RR, volume 12851 of *LNCS*, Springer, 2021, pp. 95–110.
[18] P. Bruno, F. Calimeri, C. Marte, Dedudeep: An extensible framework for combining deep learning and asp-based models, in: Proc. of LPNMR, volume 13416 of *LNCS*, Springer, 2022, pp. 505–510.
[19] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, P. Wanko, Theory solving made easy with clingo 5, in: Proc. of ICLP Technical Communications, volume 52 of *OASIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, pp. 2:1–2:15.
[20] M. Alviano, G. Amendola, C. Dodaro, N. Leone, M. Maratea, F. Ricca, Evaluation of disjunctive programs in WASP, in: Proc. of LPNMR, volume 11481 of *LNCS*, Springer, 2019, pp. 241–255.
[21] C. Dodaro, G. Mazzotta, F. Ricca, Blending grounding and compilation for efficient ASP solving, in: KR, 2024.
[22] J. Karpovic, L. Ablonskis, L. Nemuraite, B. Paradauskas, Experimental investigation of transformations from SBVR business vocabularies and business rules to owl 2 ontologies, Inf. Technol. Control. 45 (2016) 195–207.
[23] J. Karpovic, G. Krisciuniene, L. Ablonskis, L. Nemuraite, The comprehensive mapping of semantics of business vocabulary and business rules (SBVR) to OWL 2 ontologies, Inf. Technol. Control. 43 (2014) 289–302.
[24] D. Solomakhin, E. Franconi, A. Mosca, Logic-based reasoning support for SBVR, Fundam. Informaticae 124 (2013) 543–560.
[25] N. A. Manaf, A. Antoniades, S. Moschoyiannis, SBVR2Alloy: An SBVR to Alloy Compiler, in: Proc. of SOCA, IEEE Computer Society, 2017, pp. 73–80.
[26] S. Mitra, K. Anand, P. K. Chittimalli, Identifying anomalies in sbvr-based business rules using

directed graphs and smt-libv2, in: Proc. of ICEIS, SciTePress, 2018, pp. 215–222.
[27] P. K. Chittimalli, K. Anand, Domain-independent method of detecting inconsistencies in sbvr-based business rules, in: Proc. of ForMABS@ASE, ACM, 2016, pp. 9–16.
[28] L. M. de Moura, N. S. Bjørner, Satisfiability modulo theories: introduction and applications, Commun. ACM 54 (2011) 69–77.
[29] A. Armando, C. Castellini, E. Giunchiglia, M. Idini, M. Maratea, TSAT++: an open platform for satisfiability modulo theories, Electronic Notes in Theoretical Computer Science 125 (2005) 25–36.
[30] L. M. de Moura, N. S. Bjørner, Z3: an efficient SMT solver, in: Proc. of TACAS, volume 4963 of *LNCS*, Springer, 2008, pp. 337–340.
[31] J. P. Aires, J. Monteiro, R. Granada, F. Meneguzzi, Norm conflict identification using vector space offsets, in: Proc. of IJCNN, IEEE, 2018, pp. 1–8.
[32] J. P. Aires, F. Meneguzzi, Norm conflict identification using a convolutional neural network, in: Proc. of COIN and COINE, volume 12298 of *LNCS*, Springer, 2020, pp. 3–19.
[33] S. Huang, J. Sun, R. Li, Neuralconflict: Using neural networks to identify norm conflicts in normative documents, Expert Syst. J. Knowl. Eng. 41 (2024).
[34] C. Baral, J. Dzifcak, Solving puzzles described in english by automated translation to answer set programming and learning how to do that translation, in: Proc. of KR, AAAI Press, 2012. URL: http://www.aaai.org/ocs/index.php/KR/KR12/paper/view/4562.
[35] V. Lifschitz, Translating definitions into the language of logic programming: A case study, in: Proc. of the ICLP Workshops, volume 3193 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022. URL: https://ceur-ws.org/Vol-3193/short1ASPOCP.pdf.
[36] R. Schwitter, Specifying and verbalising answer set programs in controlled natural language, TPLP 18 (2018) 691–705. URL: https://doi.org/10.1017/S1471068418000327. doi:10.1017/S1471068418000327.
[37] S. Caruso, C. Dodaro, M. Maratea, M. Mochi, F. Riccio, CNL2ASP: converting controlled natural language sentences into ASP, Theory Pract. Log. Program. 24 (2024) 196–226.
[38] P. Cabalar, A. Ciabattoni, L. van der Torre, A preliminary study on deontic answer set programming, in: Proceedings of ACLAI'22, 2022.
[39] G. Governatori, An ASP implementation of defeasible deontic logic, Künstliche Intell. 38 (2024) 79–88.
[40] Z. Yang, A. Ishay, J. Lee, Coupling large language models with logic programming for robust and general reasoning from text, in: ACL, Association for Computational Linguistics, 2023, pp. 5186–5219.
[41] M. Alviano, L. Grillo, Answer set programming and large language models interaction with YAML: preliminary report, in: Proc. of CILC, volume 3733 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3733/short2.pdf.
[42] E. Coppolillo, F. Calimeri, G. Manco, S. Perri, F. Ricca, LLASP: fine-tuning large language models for answer set programming, in: Proc. of KR, 2024.
[43] M. A. Borroto Santana, I. Kareem, F. Ricca, Towards automatic composition of ASP programs from natural language specifications, in: Proc. of IJCAI, ijcai.org, 2024, pp. 6198–6206. URL: https://www.ijcai.org/proceedings/2024/685.
[44] K. Herud, J. Baumeister, O. Sabuncu, T. Schaub, Conflict handling in product configuration using answer set programming, in: J. Arias, R. Calegari, L. Dickens, W. Faber, J. Fandinno, G. Gupta, M. Hecher, D. Inclezan, E. LeBlanc, M. Morak, E. Salazar, J. Zangari (Eds.), Proc. of the International Conference on Logic Programming 2022 Workshops co-located with the 38th International Conference on Logic Programming (ICLP 2022), volume 3193 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022.