Enriched Modeling and Reasoning on Business Processes with Ontologies and Answer Set Programming

Laura Giordano and Daniele Theseider Dupré(✉)

DISIT, Università del Piemonte Orientale, Alessandria, Italy
laura.giordano@uniupo.it, dtd@di.unipmn.it

Abstract. Domain ontologies may provide the proper level of abstraction in modeling semantic constraints and business rules in BPM; in fact, ontologies are intended to define terminologies to be shared within and across organizations and reused in different applications. In this paper we show how Answer Set Programming (ASP), a powerful framework for declarative problem solving, can accommodate for domain ontologies in modeling and reasoning about Business Processes, especially for process verification. Description Logics (DLs) provide the formal counterpart of ontologies, and in our approach knowledge on the process domain is expressed in a low-complexity DL. Terms from the ontology can be used in embedding business rules in the model as well as in expressing constraints that should be verified to achieve compliance by design. Causal rules for reasoning on side-effects of activities in the process domain can be derived, based on knowledge expressed in the DL. We show how ASP can accommodate them, relying on a reasoning about actions and change approach, for process analysis, and, in particular, for verifying formulas in temporal logic.

1 Introduction

In this paper we show how we can accommodate in Answer Set Programming (ASP) several sources of knowledge for reasoning on Business Processes, in particular, for verification purposes, i.e. in order to ensure process compliance. A process model expressed in a standard business process modeling language is enriched with **domain knowledge**, in particular, ontological knowledge describing terms used:

1. in conditions on sequence flow, in particular, conditions in data-based split gateways;
2. in *semantic constraints* on the process, i.e., constraints that express "dependencies such as ordering and temporal relations between activities, incompatibilities, and existence dependencies" [31].

As discussed in [31], semantic constraints are a subset of *business rules*; while this term in widely used in the BPM context, it is a broad one that comprises several types of knowledge about a business domain. Both [36,37] present

M. Weske et al. (Eds.): BPM Forum 2018, LNBIP 329, pp. 71–88, 2018.
https://doi.org/10.1007/978-3-319-98651-7_5

attempts at classifying types of business rules, and the semantic constraints considered in [31] are stated to be *action assertions* in terms of the classification in [36]; in terms of [32,37], semantic constraints can be intended as integrity rules, while derivation and reaction rules are suitable to be embedded in the process description.

As [31] points out, semantic constraints abstract from the way some fact about the case at hand may be actually represented, or computed from stored data, in the process implementation. This is why we believe that ontological knowledge is especially suited for expressing them. In fact, the very idea of the Semantic Web and, in general, of terminological knowledge bases, includes the fact that terminological knowledge about a domain can be shared and reused in several applications. Business Process Management applications can therefore reuse existing terminologies about a whole domain (like the well-known SNOMED-CT medical terminology [29]), and an organization can define its own terminology to be reused in several applications, including management of its own processes. Domain ontologies are also believed to facilitate shared understanding of the process domain across team members [34].

In [31] it is also pointed out that compliance with semantic constraints may be checked at design time (compliance by design [35]), even though not all exceptional situations and process changes (to deal with exceptions) may be considered in the process model, to avoid it becoming too complicated in order to be readable. Therefore, it may be necessary to reason about such constraints at runtime, as part of execution support; i.e., at design time compliance is checked under the assumption that no exception occurs, while at runtime the actual exceptions and process changes occurring in the case at hand are considered; conflicts with semantic constraints should be pointed out as well as possible ways for restoring consistency with them. Semantic constraints are also useful in providing intelligible feedback to users and in supporting traceability, e.g., in order to point out whether and where, in a given process execution, they were violated.

In the work presented in this paper we incorporate contributions from Artificial Intelligence (Logic-based Knowledge Representation and Reasoning) and Formal Methods:

- **Modeling and reasoning based on description logics** [5]**.** Terminological knowledge has been identified, starting from the 1980's, as a form of knowledge which can be expressed in suitable sublanguages of first-order logics, *description logics* (DLs), as well as being useful in formalizing definitions of the terms used in several domains (as pointed out earlier). While full first-order logic is undecidable, DLs offer a trade-off between expressiveness and computational complexity of reasoning, some of them enjoying low complexity while still being able to describe wide terminologies (see, e.g., the already mentioned SNOMED-CT terminology which can be expressed in $\mathcal{EL}$ [3]). As a result, description logics have been chosen as the basis for the Semantic Web, and, in particular, the Web Ontology Language (OWL).
- **Reasoning about action and change** [21], where a domain is described in terms of **fluents**, i.e., propositions whose value can change, possible **actions**

which have preconditions, direct effects in terms of fluents, and static and dynamic **causal laws** which model dependencies between truth values of fluents, or changes in such truth values.
- **Formal verification** based on temporal logics [7], such as LTL.

For the purposes of this paper, all of them (at least, limiting to low-complexity description logics) can be integrated in Answer Set Programming (ASP) [18], a powerful framework for declarative problem solving which combines significant modeling capabilities with efficient solving, relying on inference techniques that include the ones used in SAT solvers.

In fact, in our previous work we showed the following:

1. ASP (which has been used for reasoning about action already in [8,15,16,26]) can be used for verification (with Bounded Model Checking) of properties, expressed in an extension of Linear Temporal Logic, of an action domain modeled in terms of fluents, action laws providing direct effects of actions, and causal laws [24].
2. The previous framework can be used for reasoning on business processes, in particular, for verifying process properties in temporal logic; the process can be modeled in terms of the widely adopted workflow-like languages (as well as in declarative ones). The approach is described in [22], where, relying on Constraint Answer Set Programming [19], the framework in [24] is extended to deal with conditions on numerical variables, used in the process model in data-based conditions on exclusive splits, and in the formulae to be verified. Process activities correspond to actions, and fluents are used in modeling the enabling of activities (according to the workflow model) as well as further pre/post conditions for activities, expressed in terms of process variables and further background fluents (in particular, postconditions correspond to *annotations* proposed already in [28]).
3. Reasoning about actions performed in ASP can rely on domain knowledge in a low-complexity DL [23]. More precisely, axioms in the DL describe static knowledge on a domain, e.g.: someone that teaches a university course is a lecturer. Causal laws should be associated with such knowledge to control which fluents may change as side effects of other changes, in order for the axiom to still hold, after an action whose direct effects are explicitly stated. Extra knowledge may be necessary to avoid all potential ways for restoring truth of the axiom; e.g., starting from the situation where John teaches university course CS101 (and is then inferred to be a lecturer, according to DL knowledge), if an action (such as John retiring or being fired) has the direct effect he is no longer a lecturer, we would like to infer as a side effect that he does no longer teach CS101 (nor any other course he was teaching), without considering the scenario where CS101 ceases to be a university course which would, in principle, be another way of restoring consistency.

Building on these contributions, in this paper we propose an approach to process modeling and semantic analysis that is able to exploit terminological knowledge in relying process activities to semantic constraints, via the definition of effects and preconditions of activities, and domain knowledge that relates such effects to the terms used in semantic constraints. The proposed approach exploits for business process verification the Bounded Model Checking verification methodology in ASP developed in [24].

We believe that this can indeed enrich process modeling and analysis in the BPM field, given that it provides expressive modeling at the semantic level and relies on the power of ASP solvers for efficient inference.

In the next section we summarize the sources of knowledge of our approach, and how they are expressed. In particular, in Sect. 2.1 we describe how the terminological domain knowledge base is represented, and in Sect. 2.2 we discuss how such a knowledge base can be used in reasoning about action and change. In Sect. 3 we describe how a process model can be described in terms of the framework in the preceding section. Section 4 is devoted to explaining how the model can be encoded in ASP, and how it can be used for process verification. We finally discuss the properties of our contribution especially in comparison with related work in the literature.

2 Sources of Knowledge

As sources of knowledge we consider the following ones.

- A **domain knowledge base** describing terms in the process domain: unary predicates (*classes*) describing entities of the domain, and binary predicates, called *roles* or *properties*, describing relations among domain entities. The knowledge base is formalized as a set of description logic axioms and causal rules, detailed in Sects. 2.1 and 2.2. At least some of the class predicates and properties are *fluents*, i.e., may change their truth values as effect of the process activities.
- A model for the **sequence flow** of the process is given, using conventional gateways. In particular, we refer to BPMN, and in the following we limit our consideration to models using activities, exclusive and parallel gateways (i.e., XOR splits and joins, and AND split and joins).
  Following BPMN, *conditions* on data can be attached to the sequence flow, out of gateways, in particular, exclusive gateways, thus providing *data-based exclusive gateways*. BPMN allows for specifying (in the `expressionLanguage` attribute of a model) a language to be used for expressing such data-based conditions. In this paper we do not detail the specification of one such language, but we intend that data-based condition expressions may use terms from the domain knowledge base.
- **Data objects** in the process and their **states**, which are also part of the BPMN standard, to model, e.g., an "order" whose set of states includes, for example, "pending" and "confirmed". The domain knowledge base may

mention such data objects and relate them to other entities in the process domain.
- **Pre- and postconditions** for activities. Postconditions are used to model the direct effects of activities in terms of the process domain. Postconditions include state changes for data objects, or, more generally, the case where a BPMN data object is output of an activity, i.e., the activity creates or writes the data object (represented as an output data association in BPMN). Similarly, preconditions include the ones that are represented in the BPMN process model by input sets for activities (input data associations).

2.1 The Domain Knowledge Base

We consider, as in [23], domain knowledge bases expressed in a fragment of the description logic $\mathcal{EL}^{++}$ [3]. The fragment, $\mathcal{EL}^{\perp}$, includes the concept $\perp$ (the empty concept, which is false for all individuals) as well as nominals, i.e., concepts corresponding to single individuals.

As for other description logics, the language of $\mathcal{EL}^{\perp}$ is based on a set N_C of concept names (class names), a set N_R of role names (names for properties, i.e., binary relations) and a set N_I of individual names. A concept in $\mathcal{EL}^{\perp}$ is defined as follows:

$$C := A \mid \top \mid \perp \mid C \sqcap C \mid \exists r.C \mid \{a\}$$

where $A \in N_C$, $r \in N_R$ and $a \in N_I$. That is:

- Concept expressions include class names (named concepts) and the concepts "true" and "false".
- A concept can be an intersection ($\sqcap$) of concepts (i.e., named concepts or concept expressions).
- A concept can be built from a role name r and a concept C using an *existential restriction*: the instances of concept $\exists r.C$ are the individuals x which are in relation r with some member y of the concept C.
- A concept can be the *nominal* $\{a\}$, i.e., the concept of "being a".

A knowledge base in $\mathcal{EL}^{\perp}$ is a pair $(\mathcal{T}, \mathcal{A})$, where:

- $\mathcal{T}$ (a *TBox*, i.e., the *terminological* part) is a finite set of concept inclusions $C_1 \sqsubseteq C_2$, where C_1 and C_2 are concepts,
- $\mathcal{A}$ (an *ABox*, the *assertional* part) is a set of assertions of the form $C(a)$ and $r(a, b)$, where C is a concept, $r \in N_R$ and $a, b \in N_I$.

The TBox can be expressed in a normal form [4] where axioms only have the forms: $C_1 \sqsubseteq D$, $C_1 \sqcap C_2 \sqsubseteq D$, $C_1 \sqsubseteq \exists r.C_2$, $\exists r.C_2 \sqsubseteq D$, where C_1, C_2 are from BC_{KB}, i.e., the set of concepts containing $\top$, all the named concepts occurring in KB and all nominals $\{a\}$, for any individual name a occurring in KB; and D is in $BC_{KB} \cup \{\perp\}$.

The semantics of $\mathcal{EL}^{\perp}$ is defined in the usual way for description logics, based on a domain (a set) Δ, an interpretation of individuals as elements of Δ, of concept names as subsets of Δ, of role names as binary relations on Δ. The interpretation of concept expressions is defined formalizing the description given above for the meaning of such expressions. An concept inclusion $C_1 \sqsubseteq C_2$ is satisfied in an interpretation if the interpretation of C_1 is a subset of the interpretation of C_2 (see [3,23] for the formal definitions).

Examples of concepts are:

- $\exists \mathit{Teaches}.\mathit{Course}$, whose instances are the domain elements who teach a course;
- $\exists \mathit{Teaches}.\{\mathit{cs101}\}$, the ones who teach the individual course *cs101*;
- $\mathit{UndergraduateCourse} \sqcap \mathit{ComputerScienceCourse}$, the concept of undergraduate courses in computer science, which is expressed as the intersection of undergraduate courses and computer science courses.

Examples of concept inclusions are:

- $\exists \mathit{Teaches}.\mathit{Course} \sqsubseteq \mathit{Lecturer}$, which states that the ones who teach some course are lecturers;
- $\mathit{Course} \sqcap \exists \mathit{HasSubject}.\mathit{ComputerScienceSubject} \sqsubseteq \mathit{ComputerScienceCourse}$, which states that a course, which has as subject a computer science subject, is a computer science course. Adding the inverse inclusion would provide a definition of *ComputerScienceCourse*.

2.2 Reasoning About Actions with Terminological Knowledge

Given a domain which is modeled with a knowledge base in a description logic, as above, when reasoning about a process in the domain involving actions and changes, we will assume that the Tbox axioms do not change and must be satisfied during all the process execution, even though, in the long term, it could be appropriate to allow for the knowledge base to evolve with new concepts and new axioms about them, while other axioms may cease to hold, or may be modified, e.g., in order to provide coverage of cases that were not considered before. Presumably, the process model itself should change as well, and automated reasoning may support process redesign, but we do not address the issue in this paper and assume that both the process model and the Tbox do not change.

Of course, we do consider that Abox assertions may change as a result of actions (they are *fluents*). We summarize in the following the way reasoning about such actions and changes can be defined [23] to take into account background knowledge about the domain expressed in an $\mathcal{EL}^{\perp}$ Tbox. The axioms in the Tbox can be regarded as *state constraints*, the term used in the literature in reasoning about actions and change to describe conditions that must hold in all states.

The language we consider for reasoning about action and change involves predicate symbols and constants. Such symbols include the concept names, role names and individual names occurring in the $\mathcal{EL}^{\perp}$ domain knowledge base.

The *fluents* $\mathcal{F}$ are ground atomic propositions $p(a_1,\ldots,a_k)$ where p is a predicate symbol and $a_1,\ldots,a_n$ are constants.

A *fluent literal* l is a fluent f or its explicit negation $-f$. Two literals f and $-f$ are the *complement* of each other. We denote by Lit the set of fluent literals.

If a concept $\exists r.C$ occurs in the KB, the predicate names in the action theory include a name $\exists r.C$, so that, for a individual name a, the fluent literals $(\exists r.C)(a)$ and $-(\exists r.C)(a)$ belong to Lit.

A *state* S is a set of literals in Lit. A state S is *consistent* if it is not the case that both a literal and its complement belong to S. A state S is *complete* if for any fluent literal l, S contains l or its complement.

For describing an action theory, laws are introduced in a notation in the line of the literature of reasoning about actions and change [8,16,26]. *Action laws* describe the direct effects of actions. They have the form:

$$\alpha \ \textbf{causes} \ \phi \ \textbf{if} \ \psi_1 \ \textbf{after} \ \psi_2$$

meaning that the execution of action α in a state in which ψ_2 holds causes ϕ to hold in the new state as a direct effect, if ψ_1 holds in the new state as well. The action name α corresponds to an activity in a process model, ϕ is a literal in Lit and $\psi_i = L_1 \wedge \ldots \wedge L_m, \mathit{not}\ L_{m+1} \wedge \ldots \wedge \mathit{not}\ L_n$ is a conjunction of literals $L_i \in Lit$ or their default negations. The informal meaning of *default negation* in $\mathit{not}\ L_j$ is that "L_j is not believed", its formal semantics is the stable model semantics [20].

The action name can have parameters also occurring in ϕ, ψ_1 and ψ_2, and a parametric action law is a shorthand for all its instances with individual names; the same applied to other types of laws described below. An example (instance) of action law is:

$$\mathit{retire}(\mathit{john}) \ \textbf{causes} \ -\mathit{Lecturer}(\mathit{john})$$

Non-deterministic effects of actions can be defined using default negation in the body of action laws. For instance, after flipping a coin, the result may be head or not:

$$\mathit{flip} \ \textbf{causes} \ \mathit{head} \ \textbf{if} \ \mathit{not}\ -\mathit{head}$$
$$\mathit{flip} \ \textbf{causes} \ -\mathit{head} \ \textbf{if} \ \mathit{not}\ \mathit{head}$$

Causal laws describe indirect effects of actions. They have the form:

$$\textbf{caused} \ \phi \ \textbf{if} \ \psi_1 \ \textbf{after} \ \psi_2$$

meaning that ψ_1 causes ϕ to hold whenever ψ_2 holds in the previous state; ϕ is a literal in Lit and ψ_1 and ψ_2 are as in action laws. If the condition ψ_2 is $\top$, the causal law is said to be *static*, since it only involves conditions on a single state, and the **after** part is omitted.

An example causal law, that, as we shall see, could be associated with the Tbox axiom $\exists \mathit{Teaches}.\mathit{Course} \sqsubseteq \mathit{Lecturer}$, is:

$$\textbf{caused} \ \mathit{Lecturer}(x) \ \textbf{if} \ \mathit{Teaches}(x, y) \ \wedge \ \mathit{Course}(y)$$

Precondition laws describe the executability conditions of actions. They have the form: α **executable if** ψ, meaning that the execution of action α is possible in a state where the precondition ψ holds; α is an action name and ψ is a conjunction of literals or default negations of literals. An example is: $retire(x)$ **executable if** $aged(x)$.

The *constraints* define conditions that must be satisfied by all states. They have the form: $\perp$ **if** ψ, meaning that any state in which ψ holds is inconsistent.

Initial state laws are needed to introduce conditions that have to hold in the initial state. They have the form: **Init** ϕ **if** ψ. When $\phi = \perp$, we get the a constraint on the initial state **Init** $\perp$ **if** ψ.

Most fluents are intended to be *frame* fluents, i.e., their truth value persists across action occurrences. For all such fluents p, the following causal laws, said *persistency laws*, are introduced:

$$\textbf{caused}\ p\ \textbf{if}\ \mathit{not}\ -p\ \textbf{after}\ p$$
$$\textbf{caused}\ -p\ \textbf{if}\ \mathit{not}\ p\ \textbf{after}\ -p$$

meaning that, if p holds in a state, then p will hold in the next state, unless its negation $-p$ is caused to hold (and similarly for $-p$). Persistency of a fluent is blocked by the execution of an action which causes the value of the fluent to change, or by a nondeterministic action which may cause it to change.

Persistency laws are not provided for literals such as $(\exists r.C)(a)$; their value in a state is rather derived, using causal laws, from the one of literal with "simple" predicate names, i.e., $(\exists r.B)(x)$ is caused if $r(x, y) \wedge B(y)$.

Initial state laws that correspond to what is known about the initial state may incompletely specify it. As we want to reason about all the possible complete states, the laws:

$$\textbf{Init}\ p\ \textbf{if}\ \mathit{not}\ -p$$
$$\textbf{Init}\ -p\ \textbf{if}\ \mathit{not}\ p$$

for completing the initial state are introduced for all "simple" literals p.

In [23] a **semantics** is defined for action execution. Given a state (a set of literals) S which is consistent and complete (i.e., it contains either l or $-l$ for all fluent literals), such a semantics defines which are the possible resulting states if an action α is executed in S, and is based on the answer set semantics [18].

We assume to start action execution in a state which satisfies the Tbox $\mathcal{T}$; however, in general there is no guarantee that, if an action α is applied to a state satisfying $\mathcal{T}$, the state that can result, according to the semantics, will still satisfy $\mathcal{T}$.

However, suitable causal laws can be associated with (normalized) axioms in $\mathcal{T}$ in order to guarantee this[1] [23]; here we describe part of them. For inclusions $A \sqsubseteq B$, two causal laws are needed:

$$\textbf{caused } B(x) \textbf{ if } A(x) \text{ and } \textbf{caused } -A(x) \textbf{ if } -B(x)$$

For an axiom $\exists r.B \sqsubseteq A$, the laws:

$$\begin{array}{l} \textbf{caused } A(x) \textbf{ if } (\exists r.B)(x) \\ \textbf{caused } -(\exists r.B)(x) \textbf{ if } -A(x) \end{array}$$

and at least one of:

$$\begin{array}{l} \textbf{caused } -r(x,y) \ \textbf{ if } \ -A(x) \ \wedge \ B(y) \\ \textbf{caused } -B(y) \ \textbf{ if } \ -A(x) \ \wedge \ r(x,y) \end{array}$$

should be introduced.

For example, an axiom $\exists \mathit{approved\_by}.\mathit{examiner} \sqsubseteq \mathit{approved}$ relative to insurance claim processing, has the associated causal law:

$$\textbf{caused } \mathit{approved}(x) \textbf{ if } (\exists \mathit{approved\_by}.\mathit{examiner})(x)$$

where $(\exists \mathit{approved\_by}.\mathit{examiner})(x)$ is in turn caused, if $\mathit{approved\_by}(x,y)$ and $\mathit{examiner}(y)$). If we admit that the claim, after being approved by an examiner, can be made $-\mathit{approved}$ by a manager, the causal law:

$$\textbf{caused } -\mathit{approved\_by}(x,y) \textbf{ if } -\mathit{approved}(x) \ \wedge \ \mathit{examiner}(y)$$

is introduced, while the other possible causal law is not, because we do not expect $\mathit{examiner}(y)$ to become false as a side effect of $\mathit{approved\_by}(x,y)$ becoming false.

There is an option also for the case of an axiom $A \sqcap B \sqsubseteq D$; besides the law **caused** $D(x)$ **if** $A(x) \wedge B(x)$, at least one of:

$$\begin{array}{l} \textbf{caused } -A(x) \textbf{ if } -D(x) \ \wedge B(x) \\ \textbf{caused } -B(x) \textbf{ if } -D(x) \ \wedge A(x) \end{array}$$

should be introduced.

The presence of such options requires further pieces of domain knowledge, besides the axioms in $\mathcal{T}$. The choice of causal rules to be discarded should in general be made for each single axiom, while in some cases it can be derived from more general knowledge. In the literature about ontologies, and, in particular, Temporal Description Logics, a distinction is introduced between *rigid* and *temporal* concepts and roles [1], i.e., the ones that are supposed not to change their truth values across time, and the ones that may change. If such a distinction is present, it can be used to discard optional causal rules: if a concept or role

[1] As a consequence of the introduction of the causal laws for the axioms in $\mathcal{T}$, there is no need to exploit a DL reasoner, as each state is guaranteed to satisfy $\mathcal{T}$.

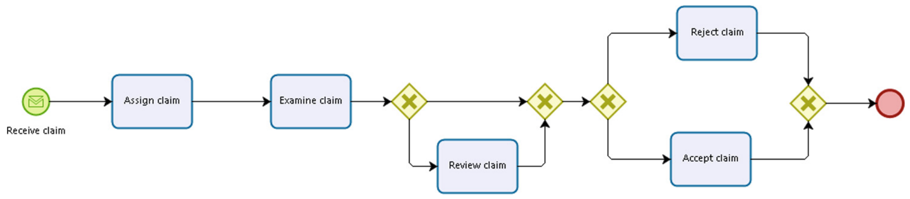

Fig. 1. Example process model

is identified as rigid, optional causal rules where concept or role occurs in the head (the "caused" part) will not be included. Transformation rules, a type of business rules identified in [32,37], can also represent a source of knowledge for selecting optional causal rules.

As detailed in [23], the representation of an action domain with terminological knowledge can be encoded in ASP, and an ASP solver can be used to determine, e.g., whether a given literal holds in some of the resulting states, or in all the resulting states, after executing a given sequence of actions. In Sect. 4 we present such an encoding and show that it can be combined with the ASP representation of the sequence flow of a process model, and an ASP representation of Bounded Model Checking, in order to verify semantic constraints.

Before that, in the next section we show how an action domain description can be derived, in a semi-automated translation, from a process model described using the basic elements of BPMN.

3 Process Models as Action Domains

Consider a simple process model for insurance claim processing whose control flow is described in Fig. 1 (additional knowledge is not shown since only part of it can be represented in BPMN). In this model, a claim is assigned to a claims examiner, who provides a (preliminar) acceptance or rejection, and then possibly reviewed by a claims manager. We do not detail the accept/reject final part in terms of sending letters or performing payment.

All activities refer to a data object *Claim*, which is output of the start event *Receive claim* and is both input and output of all the other activities.

The activity *Assign claim* also has as output the examiner who had the claim assigned and the manager who should possibly review the claim. Examiner and manager are input to the activities executed by them (alternatively, swim lanes could be used to represent actors in the process).

In general, in the representation of activities as actions in the action domain, a choice should be made on which parameters are introduced for the action. As a default, if an activity *a* has a data object as input, it will have it as a parameter. Therefore, all activities in the process have as parameters a claim identifier, and the person executing the activity.

When executing the process, it is of course necessary to assign specific values to data objects (the claim identifier, the examiner and manager, as well as other variables that may appear in the model). When reasoning about the process, considering all possible values that a data object can assume could be unnecessary (other than, of course, source of intractability or undecidability). In the example model, the actual value of the claim identifier and the actual names of the examiner and manager are not supposed to influence process execution (as we shall see, they do not occur in the data-based conditions), and are therefore irrelevant.

Then, when an object is output of an activity, we represent its value with an individual name with the only constraint that it should be different from other names. We will then consider the parametric activities to be instantiated with such parameters. This can be considered as a default in an automatic translation, but, in general, such a translation can only be semi-automated, given that the choice of a suitable abstraction on data is essential for the model to be useful. As discussed before, ontologies can help to this purpose: in the example considered here, suppose there are two classes of examiners, expert ones and in training, and that process execution depends on the class of the examiner, but not on who actually the examiner is. Then, in the model, two alternative, nondeterministic postconditions in terms of examiner type can be considered, rather than all possible individual names.

In the example we will use as values the names *claim, examiner, manager* of the data objects themselves. To avoid redundancy, we remove "claim" from the name of the activity. The action instances that are considered in the action domain corresponding to the process model are then:

$assign(claim)$
$examine(examiner, claim)$
$review(manager, claim)$
$reject(examiner, claim)$
$accept(examiner, claim)$

The control flow of the process model can be represented with action laws and precondition laws resulting from an automated translation similar to the one described in [22] (Appendix A) for a subset of YAWL, analogous to the subset of BPMN used in this paper. In the following we use the terms "activity" (in the process model) and "action" (in the action domain) interchangeably. Fluents are introduced to represent the *enabling* of activities, which is a precondition for the action. In case of two activities in sequence, execution of the first one disables itself, and enables the next one. With a parallel split, all outgoing flows are enabled, and with parallel join, enabling is necessary from all incoming flows. With nondeterministic exclusive split, the pattern for nondeterministic actions is used.

Data-based conditions for exclusive splits are the most interesting case for the approach proposed in this paper, since, as pointed out in the introduction, they are the place in the model where terms from the domain knowledge (in particular, concept names and role names applied to data objects) can be conveniently used. For the model in Fig. 1, we suppose that the condition for reviewing a claim it that it is approved by the examiner and the customer is suspect of being a fraudster (in a variation of the example, another sufficient condition could be that the examiner is in training). The condition can be expressed as $\mathit{PossiblyFraudolentClaim}(\mathit{claim})$ where the concept is defined in the domain knowledge base as $\mathit{Claim} \sqcap \exists \mathit{HasCustomer}.\mathit{SuspectFraudster}$. How a customer is actually suspected to be a fraudster (also due to previous claim history) could indeed be one of the cases mentioned in the introduction, where in the model we want to abstract from the way this is explicitly stored or computed[2]. Notice that, given that the initial state is made complete (see Sect. 2.2), the possible complete initial states will contain either $\mathit{PossiblyFraudolentClaim}(\mathit{claim})$ or its complement.

Further action laws state that:

- $\mathit{examine}(\mathit{examiner}, \mathit{claim})$ has an effect $\mathit{examined}(\mathit{claim})$ and a nondeterministic effect $\mathit{approved\_by}(\mathit{examiner}, \mathit{claim})$ or $-\mathit{approved\_by}(\mathit{examiner}, \mathit{claim})$
- $\mathit{review}(\mathit{manager}, \mathit{claim})$ has a nondeterministic effect $\mathit{approved}(\mathit{claim})$ or $-\mathit{approved}(\mathit{claim})$.

The causal laws in Sect. 2.2, associated with the domain knowledge axiom $\exists \mathit{approved\_by}.\mathit{examiner} \sqsubseteq \mathit{approved}$, imply that if the claim is approved by the examiner and does not undergo review, it will remain approved; while if it is made not approved by the manager's review, it will no longer be considered as approved by the examiner.

The second exclusive split is (obviously) conditioned on $\mathit{approved}(\mathit{claim})$.

4 ASP Representation

An action domain, including the one derived from a process model as described in the previous section, can be represented in ASP as follows [23].

States are represented as integers, starting with the initial state 0. The predicate $\mathit{occurs}(\mathit{Action}, \mathit{State})$ represents the fact that Action occurs in State; occurrence of exactly one action in each state must be imposed:

$$-\mathit{occurs}(A, S) \leftarrow \mathit{occurs}(A1, S), \mathit{action}(A), \mathit{action}(A1), A \neq A1, \mathit{state}(S).$$
$$\mathit{occurs}(A, S) \leftarrow \mathit{not}\ -\mathit{occurs}(A, S), \mathit{action}(A), \mathit{state}(S).$$

(in state-of-the art ASP solvers [18], this can also be expressed with *choice rules*, whose syntax we do not introduce here).

In order to represent the fact that a literal holds in a state, we use different predicates:

- $\mathit{holds\_inst}(\mathit{Concept}, \mathit{Name}, \mathit{State})$ represents the fact that an assertions of the form $C(a)$ holds in a state.
- $\mathit{holds\_triple}(\mathit{Role}, \mathit{NameA}, \mathit{NameB}, \mathit{State})$ is used for role assertions $r(a, b)$

[2] It could be modeled separately in a decision model, an issue we do not address in this paper.

– $holds(Fluent, State)$ is used for other fluents (for an action domain derived from a process model, they are used to model control flow).

An *action law*:

α **causes** L_0 **if** ψ_1 **after** ψ_2

where $\psi_1 = L_1 \wedge \ldots \wedge L_m, not\ L_{m+1} \wedge \ldots \wedge not\ L_n$ and $\psi_2 = L'_1 \wedge \ldots \wedge L'_m, not\ L'_{m+1} \wedge \ldots \wedge not\ L'_n$ is translated to:

$$h_0 \leftarrow state(S), S' = S+1, occurs(a,S), h_1 \ldots h_m, not\ h_{m+1} \ldots not\ h_n,$$
$$h'_1 \ldots h'_m, not\ h'_{m+1} \ldots not\ h'_n$$

where $h_0 = (-)holds\_inst(C_0, a_0, S')$ if $L_0 = (-)C_0(a_0)$, $h_0 = (-)holds\_triple$ (r_0, a_0, b_0, S'), if $L_0 = (-)r_0(a_0, b_0)$, and $h_0 = (-)holds(p(a_1, .., a_n), S')$ if $L_0 = (-)\ p(a_1, .., a_n)$ and similarly for the h_i's and h'_j's, using S' for the h_i's and S for the h'_j's; where C_0 in $holds\_inst$ stands for the ground term representing C_0, and similarly for $p(a_1, .., a_n)$.

Other laws can be translated to ASP in a similar way.

In [24] we showed (for a variant of the action language used in this paper, which can be similarly encoded in ASP) that, given an action domain, temporal properties of the domain, expressed in Dynamic Linear Time Temporal Logic [27], an extension of Linear Time Temporal Logic [7], can be verified in ASP in a Bounded Model Checking (BMC [9]) approach. The ASP encoding in [24] is suitable for verifying systems with infinite computations which can be finitely represented (with a loop back to a previously reached state). The approach relies on the definition of a predicate $sat(T\_alpha, S)$, where T_alpha is a term representing a temporal logic formula α, and S is a state, which corresponds to the fact that α holds in S. The predicate can be defined inductively on the structure of α.

In [22] we showed how the approach can be adapted to the verification of properties of finite executions of a business process model. Model checking should either find a counterexample for the formula to be verified (an interpretation falsifying the formula) or ensure that no counterexample exists. BMC is in general a partial decision procedure for model checking: it considers executions of bounded length, iteratively increasing the bound; if no model exists, in general the procedure would not stop. There are, however, cases where a *completeness threshold* can be identified (a value such that, if a counterexample exists, it can be found using such a value as bound). An obvious case is the one where the workflow of a business process model is loop-free. Appendix B of [22] reports results that demonstrate the scalability of the approach, also considering non-loop-free workflows. Processes with up to 200 activities (a size which is in line with the one of real-world processes in [17]) and run length of more than 100 activities are considered. Properties in LTL are verified, while DLTL can be useful for the declarative specification of process models.

The same approach[3] can be used for verifying LTL properties of action domains in this paper, where LTL formulae can be built from fluents, including

[3] The work in [22] allows for conditions on numerical data – e.g., the piece number in an order is larger than 50000 – to be used in the model and in the formulae to

assertions in the language of domain knowledge. The analysis is performed on the finite domain represented by the set of constants in the ASP encoding. This is without loss of generality as regards the domain knowledge, given that it is expressed in $\mathcal{EL}^{\perp}$; but it relies on the assumption that the domains for data objects are assumed to be finite.

As an example, the formula:

$$\Box(\mathit{examined}(\mathit{claim}) \wedge \neg\mathit{approved}(\mathit{claim}) \rightarrow \neg\Diamond\mathit{approved}(\mathit{claim}))$$

corresponds to the property that an examined claim which is not approved cannot become approved. In the model described in Sect. 3, it indeed holds, because the claim is reviewed only if it was approved by the examiner (and the customer is suspected to be fraudolent), while if was not approved by the examiner, it does not undergo review and its approval is not modified. The formula can be verified to hold using the approach described above.

We can observe that the (grounding of the) ASP program has polynomial size in the size of the input (ontology and business process). More precisely, let n be the size of the ontology, m the size of the business process, d the size of data domains and annotations, f the size of the formula to be checked by bounded model checking, and k (a constant) the length of the sequence searched for in a BMC verification of a formula (i.e. the number of states). The size of the BP encoding is $O(m \times d \times k)$, while the size of the action theory and BMC encoding is $O((n^2 + f) \times d \times k)$. $O(n^2)$ is an upper bound on the size of causal laws, taking into consideration the number of possible contrapositives of $\mathcal{EL}^{\perp}$ inclusions. From this observation and the fact that the final ASP encoding is a normal (non disjunctive) logic program, it follows that checking satisfiability of a temporal formula over a BP specification is in NP.

5 Conclusion and Related Work

In the paper we described how domain knowledge in the form of ontologies can be accommodated in modeling and reasoning about business processes in Answer Set Programming. We build on contributions in our previous work [22–24], but their combination for the verification of BPMN process models enriched with domain knowledge is novel. As a reasoning task, we emphasized verification of compliance by design, but other reasoning tasks can be accommodated as well. Consider, for example, compliance at runtime which should take into account a specific partial execution (whose events are given, up to a current time), but also exceptional situations occurring in the case at hand (exception not necessarily considered in the general process model). The model description in ASP

be verified. In order to deal with them, without considering all individual values in the – finite but large – numerical domain, it relies on Constraint ASP [19]. In this paper we do not consider this feature, which can however be integrated with the ones addressed here, and would provide another form of abstraction, complementary to the use of ontologies.

is modular, elaboration tolerant, and can easily accommodate, e.g., for additional actions with their enabling conditions (not necessarily related to the basic workflow structure).

Our contribution is related to several ones in the literature.

Ly et al. in [31] provide thorough motivations for the use of semantic constraints in BPM; first-order predicate logic is used as a language for expressing such constraints (while also mentioning description logics as a suitable option) but the paper does not describe the use of automated reasoning based on such a description of semantic constraints in logic. Actually, as pointed out in [31], ontological modeling and reasoning can also be useful to relate specific activities to abstract classes of activities, such as, in the medical domain, "invasive procedures", e.g., to ensure compliance of processes with the constraint that the patient has to be informed prior to invasive procedures. The approach in the paper can be extended with such a feature.

An early approach using logic-based reasoning about actions and change for modeling and verification of business processes is presented in [30], based on the ConGolog language. The work is in the line of declarative modeling of processes, while our work is aimed at enhancing BPMN-like models with semantic knowledge and reasoning.

Awad et al. [2] developed a framework for the verification of compliance of a BPMN process to requirements expressed with visual patterns, mapped to temporal logic. The requirements may involve data objects and their states. Other than being based on different inference machinery, our work allows for ontological knowledge to be accommodated in modeling and reasoning.

In [10] Calvanese et al. present an approach where decision models in the DMN standard [33] are integrated with domain knowledge. Representation and reasoning in such integrated models can be expressed in a version of the $\mathcal{ALC}$ description logic with datatypes; this provides, among other things, complexity results for reasoning tasks on decision models, such as analyzing their completeness. The integration of decision models into knowledge representation formalisms, and, in particular, the FO(.) language, is also studied in [13]. The integration of decision models in the approach presented in this paper is a subject for future work.

Colombo Tosatto et al. [12] study the complexity of the problem of business process regulatory compliance, considering achievement and maintenance obligations, showing that verifying partial compliance is an NP-complete problem, and verifying full compliance is a co-NP-complete problem. While in this paper we do not deal with obligations, it has to be noticed that different kinds of obligations could be modeled in our temporal action language by suitably introducing deontic fluents, as done in [25], where a deontic temporal extension of ASP is developed. We observed in Sect. 4 that the complexity of checking the satisfiability of a temporal formula over the BP specification is in NP; this is in agreement with the complexity result for partial compliance in [12].

In [11] Calvanese et al. study plan synthesis for a variant of Knowledge and Action Bases (KAB), a dynamic framework introduced in [6] where states are

DL knowledge bases and an initial ABox evolves over time due to actions which have conditional effects. In particular, [11] focuses on state bounded KABs, for which plan existence is proved to be decidable and shows that, for lightweight DLs, plan synthesis can be compiled into ADL planning.

De Masellis et al. in [14] describe a framework for business process verification combining a control flow model based on Petri Nets with a data model à la Data Centric Dynamic systems. In particular, they adopt the data interaction formalism in [6,11] and prove the decidability of reachability (which in general is undecidable) under three notions of state boundedness. The framework is then encoded in a $\mathcal{C}$-based action language. Finiteness of the domain is guaranteed by the fact that the model is state-bounded. In our approach we can consider the domain to be finite (for each fixed bound in the BMC), by assuming that the data type of objects in the business process is finite. We uniformly model in ASP the business process, the action language (including the constraints extracted from ontological domain knowledge, which is not considered in [14]) and the bounded model checking verification for general formulas, which subsumes reachability analysis.

References

1. Artale, A., Kontchakov, R., Ryzhikov, V., Zakharyaschev, M.: DL-Lite with temporalised concepts, rigid axioms and roles. In: 7th International Symposium on Frontiers of Combining Systems, FroCoS 2009, Trento, Italy, 16–18 September 2009, Proceedings, pp. 133–148 (2009)
2. Awad, A., Weidlich, M., Weske, M.: Visually specifying compliance rules and explaining their violations for business processes. J. Vis. Lang. Comput. **22**(1), 30–55 (2011)
3. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Kaelbling, L., Saffiotti, A. (eds.) Proceedings of IJCAI 2005, pp. 364–369, Edinburgh, Scotland, UK, August 2005
4. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: LTCS-Report LTCS-05-01. Institute for Theoretical Computer Science, TU Dresden (2005)
5. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2007)
6. Hariri, B.B., Calvanese, D., Montali, M., De Giacomo, G., De Masellis, R., Felli, P.: Description logic knowledge and action bases. J. Artif. Intell. Res. **46**, 651–686 (2013)
7. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press, Cambridge (2008)
8. Baral, C., Gelfond, M.: Reasoning agents in dynamic domains. In: Minker, J. (ed.) Logic-Based Artificial Intelligence, pp. 257–279 (2000)
9. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Adv. Comput. **58**, 118–149 (2003)
10. Calvanese, D., Dumas, M., Maggi, F.M., Montali, M.: Semantic DMN: formalizing decision models with domain knowledge. In: Rules and Reasoning - International Joint Conference, RuleML+RR 2017, London, UK, 12–15 July 2017, Proceedings, pp. 70–86 (2017)

11. Calvanese, D., Montali, M., Patrizi, F., Stawowy, M.: Plan synthesis for knowledge and action bases. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9–15 July 2016, pp. 1022–1029 (2016)
12. Colombo Tosatto, S., Governatori, G., Kelsen, P.: Business process regulatory compliance is hard. IEEE Trans. Serv. Comput. **8**(6), 958–970 (2015)
13. Dasseville, I., Janssens, L., Janssens, G., Vanthienen, J., Denecker, M.: Combining DMN and the knowledge base paradigm for flexible decision enactment. In: Supplementary Proceedings of the RuleML 2016 Challenge, Doctoral Consortium and Industry Track Hosted by the 10th International Web Rule Symposium, RuleML 2016 (2016)
14. De Masellis, R., Francescomarino, C.D., Ghidini, C., Montali, M., Tessaris, S.: Add data into business process verification: bridging the gap between theory and practice. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, 4–9 February 2017, San Francisco, California, USA, pp. 1091–1099 (2017)
15. Dovier, A., Formisano, A., Pontelli, E.: Perspectives on logic-based approaches for reasoning about actions and change. In: Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday, pp. 259–279 (2011)
16. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: A logic programming approach to knowledge-state planning: semantics and complexity. ACM Trans. Comput. Logic **5**(2), 206–263 (2004)
17. Fahland, D., Favre, C., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Analysis on demand: instantaneous soundness checking of industrial business process models. Data Knowl. Eng. **70**(5), 448–466 (2011)
18. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Morgan & Claypool Publishers, San Rafael (2012)
19. Gebser, M., Ostrowski, M., Schaub, T.: Constraint answer set solving. In: Hill, P.M., Warren, D.S. (eds.) ICLP 2009. LNCS, vol. 5649, pp. 235–249. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02846-5_22
20. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Logic Programming, Proceedings of the 5th International Conference and Symposium, pp. 1070–1080 (1988)
21. Gelfond, M., Lifschitz, V.: Action languages. Artif. Intell. **2**, 193–210 (1998)
22. Giordano, L., Martelli, A., Spiotta, M., Theseider Dupré, D.: Business process verification with constraint temporal answer set programming. Theory Pract. Logic Prog. **13**, 641–655 (2013)
23. Giordano, L., Martelli, A., Spiotta, M., Theseider Dupré, D.: ASP for reasoning about actions with an EL-bot knowledge base. In: Proceedings of the 31st Italian Conference on Computational Logic, Milano, Italy, 20–22 June 2016, pp. 214–229 (2016)
24. Giordano, L., Martelli, A., Theseider Dupré, D.: Reasoning about actions with temporal answer sets. Theory Pract. Logic Prog. **13**, 201–225 (2013)
25. Giordano, L., Martelli, A., Theseider Dupré, D.: Temporal deontic action logic for the verification of compliance to norms in ASP. In: Proceedings of ICAIL 2013 (2013)
26. Giunchiglia, E., Lifschitz, V.: An action language based on causal explanation: preliminary report. Proc. AAAI/IAAI **1998**, 623–630 (1998)
27. Henriksen, J., Thiagarajan, P.: Dynamic linear time temporal logic. Ann. Pure Appl. logic **96**(1–3), 187–207 (1999)

28. Hoffmann, J., Weber, I., Governatori, G.: On compliance checking for clausal constraints in annotated process models. Inf. Syst. Front. **14**, 155–177 (2009)
29. International Health Terminology Standards Development Organization: SNOMED CT. http://www.ihtsdo.org/snomed-ct/
30. Koubarakis, M., Plexousakis, D.: A formal framework for business process modelling and design. Inf. Syst. **27**(5), 299–319 (2002)
31. Ly, L.T., Rinderle-Ma, S., Göser, K., Dadam, P.: On enabling integrated process compliance with semantic constraints in process management systems - requirements, challenges, solutions. Inf. Syst. Front. **14**(2), 195–219 (2012)
32. zur Muehlen, M., Indulska, M.: Modeling languages for business processes and business rules: a representational analysis. Inf. Syst. **35**(4), 379–390 (2010)
33. Object Management Group: Object Management Group: Decision Model and Notation (DMN) 1.0. http://www.omg.org/spec/DMN/1.0/
34. Roa, H., Indulska, M., Sadiq, S.W.: Effectiveness of domain ontologies to facilitate shared understanding and cross-understanding. In: Proceedings of the International Conference on Information Systems - Exploring the Information Frontier, ICIS 2015, Fort Worth, Texas, USA, 13–16 December 2015
35. Sadiq, S., Governatori, G., Namiri, K.: Modeling control objectives for business process compliance. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 149–164. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75183-0_12
36. The Business Rules Group: Defininig business rules - What are they really? http://www.businessrulesgroup.org/first_paper/BRG-whatisBR_3ed.pdf
37. Wagner, G.: Rule modeling and markup. In: Reasoning Web, First International Summer School 2005, Msida, Malta, 25–29 July 2005, Tutorial Lectures, pp. 251–274 (2005)