ASP and Ontologies for Reasoning on Business Processes

Laura Giordano[1] and Daniele Theseider Dupré[1]

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

Abstract. In this paper we show that Answer Set Programming (ASP) can accommodate for domain ontologies in modeling and reasoning about Business Processes, especially for process verification. In this work, knowledge on the process domain is expressed in a low-complexity description logic (DL), and 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 reasoning about actions and change, for process analysis, and, in particular, for verifying formulas in temporal logic.

1 Introduction

In this paper (an abridged version of [11]) we consider the case of process models expressed in a standard business process modeling language, enriched with **domain knowledge**, in particular, ontological knowledge describing terms used in conditions on sequence flow and in *semantic constraints* on the process, i.e., constraints that express "dependencies such as ordering and temporal relations between activities, incompatibilities, and existence dependencies" [17]. As [17] 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 consistent with the idea of sharing terminological knowledge about a domain and reusing it in several applications (consider, e.g., the well-known SNOMED-CT medical terminology [15]).

In this work we incorporate contributions from:

- logic-based Knowledge Representation and Reasoning: modeling and reasoning based on Description Logics (DLs) and Reasoning about actions and change;
- Formal Verification based on Temporal Logics.

For the purposes of this work, all of them (limiting to a low-complexity description logic) can be integrated in Answer Set Programming (ASP) [7]. In fact, in our previous work we showed the following:

1. ASP (which has been used for reasoning about actions already in [3, 12]) 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 [10].

2. The previous framework can be used for reasoning on business processes, in particular, for verifying process properties in temporal logic [8].
3. Reasoning about actions performed in ASP can rely on domain knowledge in a low-complexity DL [9]. Axioms in the DL describe static knowledge on a domain; 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.

Building on these contributions, we describe 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. This enriches process modeling and analysis, given that it provides expressive modeling at the semantic level and relies on ASP solvers for efficient inference.

2 Sources of knowledge

As sources of knowledge we consider the following ones.

- A **domain knowledge base** formalized as a set of description logic axioms and causal rules, detailed in sections 2.1 and 2.2. Some of the class predicates and properties are *fluents*, i.e., may change their truth values as effect of process activities.
- A model for the **sequence flow** of the process, using conventional gateways. We refer to the BPMN standard, limiting our consideration to models using activities, exclusive and parallel gateways (XOR splits and joins, AND split and joins). *Conditions* on data can be attached to the sequence flow, out of gateways, in particular, exclusive gateways, thus providing *data-based exclusive gateways*. We consider data-based condition expressions that may use terms from the domain knowledge.
- **Data objects** in the process and their **states**. 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, while side effects can be obtained relying on the domain knowledge base.

2.1 Terminological domain knowledge

We consider, as in [9], terminological domain knowledge expressed in the fragment $\mathcal{EL}^{\bot}$ of the description logic $\mathcal{EL}^{++}$ [1]. The choice is motivated by the fact that reasoning about action and change with domain knowledge in $\mathcal{EL}^{\bot}$ can be performed in ASP, with no need for a DL reasoner [9]. In $\mathcal{EL}^{\bot}$, concepts can be constructed from class names, *nominals* $\{a\}$ (i.e., the concept of "being a"), $\top$ and $\bot$, using intersection ($\sqcap$) of concepts and *existential restriction* $\exists r.C$ (the individuals which are in relation r with some member of the concept C).

Examples of concepts are:

- $\exists Teaches.Course$, the domain elements who teach a course;

- $\exists Teaches.\{cs101\}$, the ones who teach the individual course $cs101$;
- $UndergraduateCourse \sqcap ComputerScienceCourse$, the concept of undergraduate courses in computer science, expressed as the intersection of undergraduate courses and computer science courses.

A knowledge base contains concept inclusions $C_1 \sqsubseteq C_2$. Examples are:

- $\exists Teaches.Course \sqsubseteq Lecturer$: the ones who teach some course are lecturers;
- $Course \sqcap \exists HasSubject.ComputerScienceSubject \sqsubseteq ComputerScienceCourse$, 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

Reasoning about such actions and changes can be defined [9] to take into account background knowledge about the domain expressed as concept inclusions in $\mathcal{EL}^{\bot}$, regarded as *state constraints*, i.e., conditions that must hold in all states. We consider an action theory including *action laws* describing the direct effects of actions, such as:

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

(this can be an instance of a parametric action with x in place of $john$). Such laws can be used to define postconditions of activities in a business process. Non-deterministic effects of actions can be defined using default negation in the body of action laws.

Causal laws describe dependencies and can be used to derive indirect effects of actions. An example causal law, that, as we shall see, could be associated with the concept inclusion $\exists Teaches.Course \sqsubseteq Lecturer$, is:

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

Precondition laws describe the executability conditions of actions. An example is: $retire(x)$ **executable if** $aged(x)$.

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:

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

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.

In [9] 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, based on the answer set semantics [7], defines which are the possible resulting states if an action α is executed in S.

Suitable causal laws can be associated with concept inclusions in order to guarantee that if an action is applied to a state satisfying such inclusions, the resulting state will

still satisfy them; as a consequence, there is no need to exploit a DL reasoner, as each state is guaranteed to satisfy concept inclusions [9]. Here we describe part of such causal laws. For inclusions $A \sqsubseteq B$, two causal laws are needed:

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

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

$$\textbf{caused } A(x) \textbf{ if } (\exists r.B)(x);$$
$$\textbf{caused } -(\exists r.B)(x) \textbf{ if } -A(x);$$

and at least one of:

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

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

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

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

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

is introduced, while the other possible causal law is not, because we do not expect $examiner(y)$ to become false as a side effect of $approved\_by(x,y)$ 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 the following should be introduced:

$$\textbf{caused } -A(x) \textbf{ if } -D(x) \wedge B(x)$$
$$\textbf{caused } -B(x) \textbf{ if } -D(x) \wedge A(x)$$

3 Process models as action domains

Consider a simple process model for insurance claim processing whose control flow is described in Figure 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).

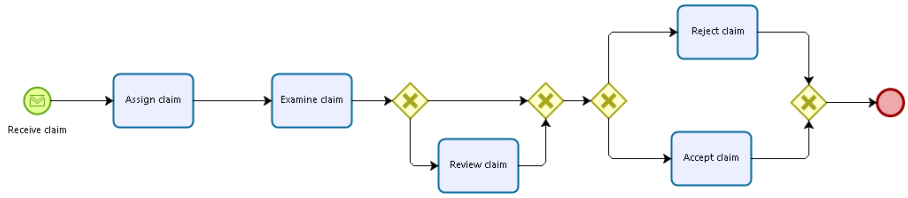

Fig. 1. Example process model

In general, in the representation of activities as actions in the action domain, a choice should be made on the parameters introduced for the action, and the possible values for data objects in the process. As a default, if an activity a has a data object as input or output, 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.

As regards values for data objects (here, a claim identifier, the examiner and manager), when reasoning about the process, considering all possible values that a data object can assume could be unnecessary. In the example model, the actual value of the claim identifier and the actual names of the examiner and manager do not 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. 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, based on the *enabling* of activites, similar to the one described in [8] (appendix A) for a subset of YAWL, analogous to the subset of BPMN used in this paper.

Data-based conditions for exclusive splits are the most interesting case for this paper, since they are the place in the model where terms from the domain knowledge can be conveniently used. For the model in figure 1, we suppose that the condition for reviewing a claim is 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 $PossiblyFraudolentClaim(claim)$ where the concept is defined in the domain knowledge base as $Claim \sqcap \exists HasCustomer.SuspectFraudster$. How a customer is actually

suspected to be a fraudster (also due to previous claim history) could be a case where in the model we abstract from the way this is explicitly stored or computed: this is one of the reasons for introducing a semantic layer in the model.

Further action laws state that:

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

The causal laws in section 2.2, associated with the domain knowledge axiom $\exists approved\_by.examiner \sqsubseteq approved$, 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 $approved(claim)$.

4 Process representation and reasoning in ASP

An action domain, including the one derived from a process model as described in the previous section, can be represented in ASP [9].

States are represented as integers, starting with the initial state 0; $occurs(Action, State)$ represents the fact that $Action$ occurs in $State$; occurrence of exactly one action in each state is imposed. A predicate $holds\_inst(Concept, Name, State)$ is used to represent that an assertion of the form $C(a)$ holds in a state, $holds\_triple(Role, NameA, NameB, State)$ is used for role assertions $r(a, b)$, and $holds(Fluent, State)$ is used for other fluents (used to model control flow in a process model).

Action and causal laws can be translated to ASP rules. As an example, the action law $examine(examiner, claim)$ **causes** $examined(claim)$ is translated to:

$$holds\_inst(examined, claim, S1) \leftarrow$$
$$S1 = S + 1, occurs(examine(examiner, claim), S)$$

while the causal law **caused** $-approved\_by(x, y)$ **if** $-approved(x) \wedge examiner(y)$ is translated to

$$-holds\_triple(approved\_by, X, Y, S) \leftarrow$$
$$state(S), -holds\_inst(approved, X, S), holds\_inst(examiner, Y, S)$$

In [10] 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 [13], an extension of Linear Time Temporal Logic [2], can be verified in ASP in a Bounded Model Checking (BMC [4]) approach.

In [8] we showed how the approach can be adapted to the verification of properties of finite executions of a business process model, and, in Appendix B therein, that processes with up to 200 activities and run length of more than 100 activities can be dealt with.

The same approach can be used for verifying LTL properties of action domains in this paper, where LTL formulae can be built from fluents, including 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 $\mathcal{EL}^{\bot}$; but it relies on the assumption that the domains for data objects are assumed to be finite. As an example, the formula:

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

corresponds to the property that an examined claim which is not approved cannot become approved. In the model described in section 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.

5 Conclusion and related work

In the paper, building on contributions in our previous work [10, 8, 9], we described how domain knowledge in the form of ontologies can be accommodated in modeling and reasoning about business processes in Answer Set Programming.

Our contribution is related to several ones in the literature.

Ly et al. in [17] provide thorough motivations for the use of semantic constraints – represented in first-order logic – in BPM, but the paper does not describe the use of automated reasoning based on such constraints.

An early approach using logic-based reasoning about actions and change for modeling and verification of business processes is presented in [16], 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.

The idea of adding *semantic annotations* to a process model was proposed already in [14]. In that paper, domain knowledge is in the form of clauses, rather than relying on DLs, which are now commonly used for semantic layers; side effects of actions are obtained based on the PMA approach [18], while we rely on causal rules, which were introduced for that purpose in reasoning about action and change.

De Masellis et al. in [5] 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 prove the decidability of reachability (which in general is undecidable) under three notions of state boundedness. The framework is 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 [5]) and the bounded model checking verification for general formulas, which subsumes reachability analysis.

In a related paper [6] the authors rely on the formulation of a business process in terms of an action language in order to take advantage of automated planners in order to solve reachability problems, in particular, in order for repairing incomplete traces of execution.

References

1. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Kaelbling, L., Saffiotti, A. (eds.) Proc. IJCAI 2005. pp. 364–369. Edinburgh, Scotland, UK (August 2005)
2. Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008)
3. Baral, C., Gelfond, M.: Reasoning agents in dynamic domains. In: Logic-Based Artificial Intelligence, pp. 257–279 (2000)
4. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers 58, 118–149 (2003)
5. De Masellis, R., Di Francescomarino, C., 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. pp. 1091–1099 (2017)
6. De Masellis, R., Francescomarino, C.D., Ghidini, C., Tessaris, S.: Enhancing workflow-nets with data for trace completion. In: Business Process Management Workshops - BPM 2017 International Workshops. pp. 89–106 (2017)
7. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Morgan & Claypool Publishers (2012)
8. Giordano, L., Martelli, A., Spiotta, M., Theseider Dupré, D.: Business process verification with constraint temporal answer set programming. Theory and Practice of Logic Programming 13, 641–655 (2013)
9. 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. pp. 214–229 (2016)
10. Giordano, L., Martelli, A., Theseider Dupré, D.: Reasoning about actions with temporal answer sets. Theory and Practice of Logic Programming 13, 201–225 (2013)
11. Giordano, L., Theseider Dupré, D.: Enriched modeling and reasoning on business processes with ontologies and answer set programming. In: Business Process Management Forum - BPM Forum 2018, Sydney. pp. 71–88 (2018)
12. Giunchiglia, E., Lifschitz, V.: An action language based on causal explanation: Preliminary report. In: Proc. AAAI/IAAI 1998. pp. 623–630 (1998)
13. Henriksen, J., Thiagarajan, P.: Dynamic linear time temporal logic. Annals of Pure and Applied logic 96(1-3), 187–207 (1999)
14. Hoffmann, J., Weber, I., Governatori, G.: On compliance checking for clausal constraints in annotated process models. Information Systems Frontiers (2009)
15. International Health Terminology Standards Development Organization: SNOMED CT. http://www.ihtsdo.org/snomed-ct/
16. Koubarakis, M., Plexousakis, D.: A formal framework for business process modelling and design. Inf. Syst. 27(5), 299–319 (2002)
17. 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. Information Systems Frontiers 14(2), 195–219 (2012)
18. Winslett, M.: Reasoning about action using a possible models approach. In: Proc. AAAI, St. Paul, MN, August 21-26, 1988. pp. 89–93 (1988)