ASP-based Declarative Process Mining

Francesco Chiariello[1], Fabrizio Maria Maggi[2], Fabio Patrizi[1]

[1] DIAG - Sapienza University of Rome, Italy
[2] KRDB - Free University of Bozen-Bolzano, Italy
chiariello@diag.uniroma1.it, maggi@inf.unibz.it, patrizi@diag.uniroma1.it

Objective

- To show how Answer Set Programming can solve problems from Declarative Process Mining.
- Three problems considered: Log Generation, Conformance Checking, and Query Checking.
- Encodings share a common part.

Process Mining

- Intersection of Business Process Management and Data Mining.
- Getting insights into processes analyzing event logs.

Process Mining Terminology

- An event consists of an activity, a timestamp and (possibly) other attributes.
- A case (or trace) is an observed sequence of events.
- An event log is a collection of cases.
- A process model is a specification of properties of cases.

Process Perspectives

- Control-flow perspective focuses on activities.
- Data-aware perspective focuses on attributes.
- Time perspective focuses on timestamps.

Problems

- *Log generation* [Sky+18] is the problem of generating a log compliant with a process model.
- *Conformance checking* [BMS16] is the problem of checking whether traces are compliant with a process model.
- *Query checking* [Räi+14] is the problem of finding properties of a process from the associated event log.

Declarative Process Mining

- Processes are set of constraints.
- Formalism used are:
  - DECLARE [APS09]
  - Linear Temporal Logic on finite traces (LTL_f) [DV13]

DECLARE

Template	Meaning
Absence(*a*)	Activity *a* never happens
Existence(*a*)	Activity *a* happens at least 1 time
Response(*a*, *b*)	If *a* happens, *b* happens afterwards
NotResponse(*a*, *b*)	If *a* happens, *b* doesn't happen afterwards
RespondedExistence(*a*, *b*)	If *a* happens, *b* happens
AlternateResponse(*a*, *b*)	If *a* happens then *b* happens without any *a* inbetween
Precedence(*a*, *b*)	If *b* happens, then *a* happened before it

LTL_f: Syntax

- Given a set $\mathcal{P}$ of propositional symbols, the syntax is defined by the following grammar:

$$\varphi ::= A \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{X}\varphi \mid \varphi_1 \mathbf{U} \varphi_2$$

  with $A \in \mathcal{P}$.
- Common abbreviations used are:
  - *true*, $\rightarrow$, $\vee$
  - $\mathbf{F}\varphi \equiv \mathit{true}\mathbf{U}\varphi$
  - $\mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi$
  - $\varphi_1 \mathbf{W} \varphi_2 \equiv \varphi_1 \mathbf{U} \varphi_2 \vee \mathbf{G}\varphi_1$

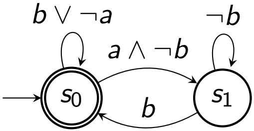
LTL_f: Semantics

- Given a formula φ, a trace $\pi = \pi_1, \pi_2, \ldots, \pi_{len(\pi)} \in (2^{\mathcal{P}})^+$, and a time instant i, with $1 \leq i \leq len(\pi)$, the semantics is defined as follows:
    - $\pi, i \models A$ iff $A \in \pi_i$,
    - $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$,
    - $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$,
    - $\pi, i \models \mathbf{X}\varphi$ if $i < len(\pi)$ and $\pi, i+1 \models \varphi$,
    - $\pi, i \models \varphi_1 \mathbf{U} \varphi_2$ iff $\pi, j \models \varphi_2$ for some j, with $i \leq j \leq len(\pi)$, and $\pi, k \models \varphi_1$ for all $k = i, \ldots, j-1$.
- A formula φ is true in π, and we write $\pi \models \varphi$, if $\pi, 1 \models \varphi$.

Template	LTL_f Formula
$Absence(a)$	$\neg\mathbf{F}a$
$Existence(a)$	$\mathbf{F}a$
$Response(a, b)$	$\mathbf{G}(a \rightarrow \mathbf{F}b)$
$NotResponse(a, b)$	$\mathbf{G}(a \rightarrow \neg\mathbf{F}b)$
$RespondedExistence(a, b)$	$\mathbf{F}a \rightarrow \mathbf{F}b$
$AlternateResponse(a, b)$	$\mathbf{G}(a \rightarrow \mathbf{X}(\neg a\mathbf{U}b))$
$Precedence(a, b)$	$\neg b\mathbf{W}a$

LTL_f and Automata

- For each LTL_f formula φ there exists a NFA A_φ that accepts exactly the traces satisfying φ.
- For example to $\varphi = \mathbf{G}(a \rightarrow \mathbf{F} b)$ is associated



Framework

- An *activity* is an expression of the form $A(a_1, \ldots, a_{n_A})$, where A is the *activity name* and each a_i is an *attribute name*.
- An *event* is an expression of the form $e = A(v_1, \ldots, v_{n_A})$, where v_i is a element of the set $D_A(a_i)$ of possible values of a_i.
- A *process trace* is a finite sequence of events $\pi = e_1 \cdots e_n$.
- An event log is a finite set of traces.

LTL_f with local conditions (or L-LTL_f)

Given a finite set of activities *Act*, the formulas φ of L-LTL_f over *Act* are inductively defined as follows:

$$\varphi = true \mid A \mid a \odot a' \mid a \odot v \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \varphi\mathbf{U}\varphi,$$

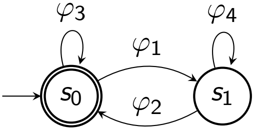
where: a and a' are attribute names from some activity in *Act*, $v \in D_A(a)$, for some $A \in Act$, $\odot$ is an operator from $\{<, \leq, =, \geq, >\}$, and A is an activity name from *Act*.

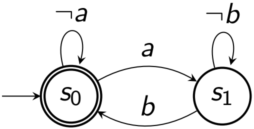
Automata Representation of L-LTL$_f$ formulas

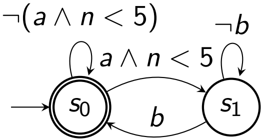
- For every L-LTL$_f$ formula φ there exists a finite-state automaton (FSA) $\mathcal{A}_\varphi$ that accepts exactly the traces that satisfy φ (see [DV13]).
- Such automata are standard FSA with transitions labelled by event formulas (i.e. without temporal operators).

Example

The L-LTL$_f$ formula $\varphi = \mathbf{G}(a \rightarrow \mathbf{F} b)$ can be extended as
$\varphi' = \mathbf{G}((a \wedge n < 5) \rightarrow \mathbf{F} b)$







Approach

- Convert specifications into automata.
- Represent automata in ASP.
- Represent traces in ASP.
- Modeling how automata read trace.
- Add generation and test rules.

ASP for DPM: data

Predicates:

- $act(A)$: A is an activity.
- $has\_attr(A, N)$: activity A has attribute N.
- $val(N, V)$: a possible value of attribute N is V.

Activities $a_1(int, cat)$ and $a_2()$, with $int \in \{1, \ldots, 10\}$ and $cat \in \{c_1, c_2, c_3\}$ becomes:

- $act(a_1)$. $has\_attr(a_1, int)$. $has\_attr(a_1, cat)$.
- $act(a_2)$.
- $value(int, 1..10)$.
- $value(cat, c1)$. $value(cat, c2)$. $value(cat, c3)$.

ASP for DPM: traces

Predicates:

- $trace(A, T)$: activity A happens at time T.
- $has\_value(N, V, T)$: attribute N has value V at time T.

Trace $a_2(), a_1(2, c_3), a_2()$ becomes:

- $trace(a_2, 1)$.
- $trace(a_1, 2)$. $has\_value(int, 2, 2)$. $has\_value(cat, c_3, 2)$.
- $trace(a_2, 3)$.

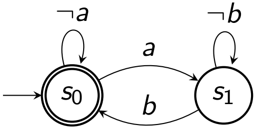
ASP for DPM: automata

- $init(S)$: S is the initial state.
- $acc(S)$: S is an accepting state.
- $trans(S, F, S')$: there exists a transition from state S to state S' labeled with event formula F.

- $holds(F, T)$: event formula F holds at time T.

Example

The ASP encoding of the formula $\varphi = \mathbf{G}(a \rightarrow \mathbf{F}b)$ is given by:



- $init(s_0).$
- $acc(s_0).$
- $trans(s_0, 1, s_1).$
- $holds(1, T) \leftarrow trace(a, T).$
- $trans(s_1, 2, s_0).$
- $holds(2, T) \leftarrow trace(b, T).$
- $trans(s_0, 3, s_0).$
- $holds(3, T) \leftarrow \texttt{not}\ holds(1, T), time(T).$
- $trans(s_1, 4, s_1).$
- $holds(4, T) \leftarrow trace(A, T), A \neq b.$

Example (cont'd)

For the data-aware formula $\varphi' = \mathbf{G}((a \wedge n < 5) \rightarrow \mathbf{F} b)$ it is sufficient to modify the rule for $holds(1, T)$ as follows:

- $holds(1, T) \leftarrow trace(a, T), has\_value(n, V, T), V < 5.$

ASP for DPM: reading trace

Predicate *state* models execution of automaton on trace

- $state(S, T)$: S is current state at time T.

and updated as

- $state(S, 0) \leftarrow init(S)$.
- $state(S', T) \leftarrow state(S, T - 1), trans(S, F, S'), holds(F, T)$.

Log Generation

Problem: given an L-LTL_f formula *varphi* and trace length t, generate a trace of length t satisfying φ

Generate traces as follows

- $\{trace(A, T) : activity(A)\} = 1 \leftarrow time(T).$
- $\{has\_value(K, V, T) : value(K, V)\} = 1 \leftarrow trace(A, T), has\_attribute(A, K).$

Test traces as follows

- $sat \leftarrow state(S, t), accepting(S).$
- $\leftarrow \texttt{not } sat.$

Conformance Checking

It is given a set of traces.

- Add the trace index i to predicate sat.
- Check whether $sat(i)$ holds.

Query Checking

The following predicates are introduced

- $var(V)$: V is a variable.
- $assgnmt(V, A)$: activity A is assigned to variable V.

The body of the rule for *holds* is modified by replacing $trace(act, T)$ with $trace(A, T), assgnmt(v, A)$, with v being the variable in place of activity act.

Then for generating

- $\{assgnmt(V, A) : activity(A)\} = 1 \leftarrow var(V).$

and for testing we check that the formula is satisfied by the trace.

Example

Consider formula $\varphi = \mathbf{G}((?A \wedge number < 5) \rightarrow \mathbf{F}b)$.

Rule for $holds(1, T)$ is:

$$holds(1, T) \leftarrow trace(A, T), assgnmt(varA, A), has\_value(n, V, T), V < 5.$$

Experiments: Log Generation

# constr. → / Trace len ↓	3	5	7	10
10	35975	35786	36464	37688
15	50649	51534	54402	54749
20	69608	70342	73122	73222
25	85127	85598	87065	89210
30	101518	101882	106062	107520
10	595	614	622	654
15	876	894	904	956
20	1132	1155	1178	1250
25	1364	1413	1444	1543
30	1642	1701	1746	1874

Table: Time (in ms) for generating a log of 10000. Above: Results obtained with MP-Declare Log Generator, a state-of-the-art tool. Below: our results.

Experiments: Conformance Checking

Tool → Trace Len ↓	ASP	Declare Analyzer
10	665	598
15	1100	805
20	1456	1092
25	2071	1273
30	2407	1337

Table

Time (in ms) for checking a log of 1000 traces against a model of 10 constraints.

Experiments: Query Checking

Constraints → Trace len ↓	Existence	Responded Existence	Response	Chain Response	Absence	Not Resp. Existence	Not Resp.	Not Chain Response
10	521	736	534	503	566	783	602	385
15	704	1113	801	788	784	1180	879	606
20	1321	1675	1143	1128	1373	1821	1304	865
25	1397	3218	1528	1561	1562	2823	1807	1104
30	1674	2878	1824	1906	1905	2784	2028	1301

Table: Time (in ms) for checking different DECLARE constraints (with both activation and target activity, if any, unknown) against a log of 1000 traces

Results

Our approach

- outperforms the SoA tool MP-Declare Log Generator [Sky+18]
- shows results comparable wrt SoA tool Declare Analyzer [BMS16]
- show the feasibility of data-aware Query checking

Note

- more general specifications.
- code not optimized.

Conclusion

Provided

- ASP encoding of data-aware Log Generation, Conformance Checking, and Query Checking
- Performance evaluation wrt state-of-the-art

Future work

- add time-perspective (i.e. timestamp)
- correlation condition

Bibliography I

[APS09] Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. "Declarative workflows: Balancing between flexibility and support". In: *Comput. Sci. Res. Dev.* 23.2 (2009), pp. 99–113.

[BMS16] Andrea Burattin, Fabrizio Maria Maggi, and Alessandro Sperduti. "Conformance checking based on multi-perspective declarative process models". In: *Expert Syst. Appl.* 65 (2016).

[DV13] Giuseppe De Giacomo and Moshe Y. Vardi. "Linear Temporal Logic and Linear Dynamic Logic on Finite Traces". In: *Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence*. IJCAI/AAAI, 2013.

Bibliography II

[Räi+14] Margus Räim et al. "Log-Based Understanding of Business Processes through Temporal Logic Query Checking". In: *On the Move to Meaningful Internet Systems: OTM 2014 Conferences*. 2014, pp. 75–92.

[Sky+18] Vasyl Skydanienko et al. "A Tool for Generating Event Logs from Multi-Perspective Declare Models". In: *Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018*. 2018.