Making sense of temporal data: the DECLARE encoding*

(Discussion/Short Paper)

Chiara Di Francescomarino[1], Ivan Donadello[2], Chiara Ghidini[1,*], Fabrizio Maria Maggi[2] and Williams Rizzi[1,2]

[1]FBK IRST, Via Sommarive 18, Trento, Italy
[2]Free University of Bozen-Bolzano, Bolzano, Italy

1. Introduction

Making sense of data, and extracting the knowledge they contain in explicit formalizations, such as the one provided by conceptual or formal models, is an important step in creating human understandable descriptions of data, and in bringing humans in the loop of taking appropriate decisions based on these data. Process discovery aims at exploiting unsupervised learning in order to discover explicit formulations of so-called *event logs* data. These explicit formulations are usually either expressed in the form of Petri nets or DECLARE patterns. The latter provide a declarative model composed of a set of constraints, usually interpreted in Linear Temporal Logic over finite traces (LTL_f), which is deemed especially important when event logs contain the execution of highly variable processes [2]. In addition to techniques for process discovery, recent works in the field of Deviance Mining and explainable process predictions have paved the way to extract also different types of knowledge from data. This knowledge is able to describe the discrepancies among classes of process executions [3], and to describe why a certain prediction is given, not only for a single trace but also for an entire quadrant of the confusion matrix associated to a (binary) classification Machine Learning (ML) model [4].

Most of the state-of-the-art Deviance Mining and explainable process predictions techniques rely on Machine Learning and different encodings have been proposed in literature [5] so as to exploit different characteristics of the event logs by means of ML techniques. Alas, none of them enables the encoding in terms of temporal logic patterns. In this abstract we recall a recent proposal for a DECLARE encoding of execution traces in the context of Deviance Mining [1], where the problem is the one of explaining, in terms of DECLARE patterns, the differences between a set of *positive* ($\mathcal{L}^+$) and a set of *negative* ($\mathcal{L}^-$) traces.

PMAI@IJCAI22: International IJCAI Workshop on Process Management in the AI era, July 23, 2022, Vienna, Austria
*This abstract refers to a paper currently submitted to Decision Support Systems which can also be found in [1]
*Corresponding author.
dfmchiara@fbk.eu (C. Di Francescomarino); ivan.donadello@unibz.it (I. Donadello); ghidini@fbk.eu (C. Ghidini); maggi@inf.unibz.it (F. M. Maggi); wrizzi@fbk.eu (W. Rizzi)
0000-0002-0264-9394 (C. Di Francescomarino); 0000-0003-1563-4965 (C. Ghidini)

2. Declare

DECLARE [2, 6] is a declarative process modeling language based on Linear Temporal Logic over finite traces (LTL_f) [7]. More specifically, a DECLARE model fixes a set of activities, and a set of constraints over such activities, formalized using LTL_f formulae. The overall model is then formalized as the conjunction of the LTL_f formulae of its constraints. Among all possible LTL_f formulae, DECLARE selects some pre-defined patterns such as the ones represented in Table 1. Each pattern is represented as a DECLARE template, i.e., a formula with placeholders to be substituted by concrete activities to obtain a constraint.

Template	$\mathbf{LTL}_f$
existence(A)	$\Diamond A$
responded existence(A,B)	$\Diamond A \rightarrow \Diamond B$
response(A,B)	$\Box(A \rightarrow \Diamond B)$
alternate response(A,B)	$\Box(A \rightarrow \bigcirc(\neg A\, \mathcal{U}\, B))$
chain response(A,B)	$\Box(A \rightarrow \bigcirc B)$
not co-existence(A,B)	$\Diamond A \rightarrow \neg\Diamond(B)$
not succession(A,B)	$\Box(A \rightarrow \neg\Diamond B)$
not chain succession(A,B)	$\Box(A \rightarrow \neg\bigcirc B)$

Table 1: DECLARE constraints in LTL_f.

For binary constraints (i.e., constraints specifying a relationship between two activities), one of the two activities is called *activation*, and the other *target*; while testing a trace for conformance over one of these constraints, the presence of the activation in the trace triggers the clause verification, requiring the (non-)execution of an event containing the target in the same trace. The notion of activation is related to the notion of *vacuity detection* in model checking [8]. For example, in the constraint *response[*`doSurgery`*,*`doRehab`*]*, if `doSurgery` never occurs in a trace, then the constraint is "vacuously" satisfied, that is, satisfied without showing any form of interaction with the trace.

3. Encoding traces using $\mathbf{LTL}_f$ temporal patterns

How to encode. Traces in an event log need to be transformed into numerical feature vectors, which can then be used to train a classifier. To this aim, encodings can be used where each element of the feature vector corresponds to the LTL_f temporal pattern derived from a DECLARE constraint (taken from a list of selected constraints). Depending on the need to to distinguish vacuously satisfied constraints from non-vacuously satisfied, we introduce the following parametric encoding values:

- -1, if the corresponding DECLARE constraint is violated in the trace;
- 1, if the corresponding DECLARE constraint is (non vacuously) satisfied in the trace.
- α if the corresponding DECLARE constraint is vacuously satisfied in the trace, with $\alpha = 0$ if we aim at distinguishing vacuously satisfied constraints and $\alpha = 1$ otherwise.

The event log is transformed into a matrix of numerical values where each row corresponds to a trace and each column corresponds to a feature.

Let us assume we aim at distinguishing between satisfied and vacuously satisfied. Given the trace $\langle \mathrm{a, b, c, a, b, c, d, a, b} \rangle$:

- constraint *response*(a, c) is violated, since the third activation (the third occurrence of a) leads to a violation (is not eventually followed by c) and is encoded as 0;
- constraint *response*(a, b) is satisfied and is encoded as 1;

- constraint *response*(e, b) is (vacuously) satisfied and is encoded as 0.

We also adopt the same approach for representing **data-aware** [9] declarative features. E.g., given a trace $\langle$a*{color = white}*, c, b*{color = black}*, c, d, a*{color = white}*, c$\rangle$, where e.g., a *{color = white}* indicates that a has an attribute *color* having value *white* in its payload, we have:

- constraint *response*(a,c,color = white) is satisfied, and is hence encoded as 1;
- constraint *response*(a,d,color = white) is violated, since the second occurrence of a is not eventually followed by d, and is encoded as 0;
- constraint *response*(b,c,color = white) is vacuously satisfied and encoded as 0 (b is not an activation, because the data condition *color = white* does not hold on its payload).

How to build the feature vector. While apparently extremely natural, the usage of the DECLARE encoding requires some preprocessing. In particular, even assuming to restrict to the typical set of DECLARE templates, the number of patterns to be considered present in any trace is, generally, too large to use all of them to build the feature vectors. This triggers the need of a method to select an appropriate set of patterns for building the feature vectors.

To select the DECLARE constraints to be used for building the feature vectors, the first step is to discover a list of constraints from the event log, similarly to what is usually done in process discovery. Since the problem we are addressing is the discriminative explanation of *positive* ($\mathcal{L}^+$) and *negative* ($\mathcal{L}^-$) traces, DECLARE constraints need to be extracted separately from $\mathcal{L}^+$ and $\mathcal{L}^-$. In particular, we first discover frequent activity sets separately in $\mathcal{L}^+$ and in $\mathcal{L}^-$. Then, we build a list of candidates as done for process discovery. Each candidate is checked separately over $\mathcal{L}^+$ and $\mathcal{L}^-$ (depending on whether it is derived from an activity set discovered from $\mathcal{L}^+$ or $\mathcal{L}^-$) to verify if it is satisfied in a percentage of traces that is above a given *support* threshold.

The number of patterns generated from the discovery is, generally, still too large to use all of them to build the feature vectors. Therefore, it becomes important to remove the ones that do not give much value for training the explanatory model. To this aim, the temporal patterns discovered in the previous step are selected by first ranking them according to the Fisher score [10]. The Fisher score for the *j-th* feature (i.e., pattern) is computed as:

$$F_j = \frac{\sum_{i=1}^{\mathcal{C}} n_i(\mu_i - \mu)^2}{\sum_{i=1}^{\mathcal{C}} n_i \sigma_i^2}, \tag{1}$$

where n_i denotes the number of traces in class *i* (in our case the number of *positive* traces in $\mathcal{L}^+$ and the number of *negative* traces in $\mathcal{L}^-$), μ_i and σ_i^2 denote mean and variance of the values of the *j-th* feature for traces in class i, and μ and σ are mean and variance of the values of the *j-th* feature for all traces in the event log. $\mathcal{C} = \{0, 1\}$ denotes the set of classes for our binary classification task. Following the ranking, patterns are selected until every trace satisfies (non-vacuously) at least a fixed number of patterns (*coverage* threshold). A pattern is only chosen if it is non-vacuously satisfied in at least one of the traces not totally covered yet.

4. Using the DECLARE encoding for Deviance Mining.

A hospital carries out the procedures for the treatment of fractures, whose executions are logged in its information system. Some lengthy traces reflect hospital inefficiencies, which might resolve into patients' complaints: the hospital director needs to understand the discrimination between two distinct trace classes - the fast ($\mathcal{L}^+$) and the lengthy ones ($\mathcal{L}^-$) - by characterizing the slow executions with respect to the fast ones. A resulting list of temporal logic patterns will give suggestions on how to intervene so to match the behavior of the desired (fast) traces.

In the context of a scenario like this, the work in [1] investigates how a declarative encoding, paired with feature selection, can accurately discriminate between ($\mathcal{L}^+$) and ($\mathcal{L}^-$) executions of a process using synthetic and real-life event logs from multiple domains. Moreover, the paper analyzes the possible outcomes returned to the users. Two different methods, based on the white-box classifiers Ripper$k$ (Repeated Incremental Pruning to Produce Error Reduction) [11] and decision trees [12] are used to identify the declarative patterns, and compared both in terms of their classification performance and in terms of amount and length of the decision rules returned (to investigate user readability and explanation conciseness).

References

[1] G. Bergami, C. Di Francescomarino, C. Ghidini, F. M. Maggi, J. Puura, Exploring business process deviance with sequential and declarative patterns, arXiv:2111.12454v1, 2022.

[2] M. Pesic, H. Schonenberg, W. M. P. van der Aalst, Declare: Full support for loosely-structured processes, in: Proc. of EDOC, IEEE Computer Society, 2007.

[3] F. Taymouri, M. La Rosa, M. Dumas, F. M. Maggi, Business process variant analysis: Survey and classification, Knowl. Based Syst. 211 (2021) 106557.

[4] W. Rizzi, C. Di Francescomarino, F. M. Maggi, Explainability in predictive process monitoring: When understanding helps improving, in: BPM Forum 2020, Proceedings, volume 392 of *LNBIP*, Springer, 2020, pp. 141–158.

[5] C. Di Francescomarino, C. Ghidini, Predictive process monitoring, in: Process Mining Handbook, volume 448 of *LNBIP*, Springer, 2022, pp. 320–346.

[6] V. Skydanienko, C. Di Francescomarino, C. Ghidini, F. M. Maggi, A tool for generating event logs from multi-perspective declare models, in: Proc. of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018, volume 2196 of *CEUR Workshop Proc.*, CEUR-WS.org, 2018, pp. 111–115.

[7] G. De Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: Proc. of IJCAI, AAAI Press, 2013.

[8] I. Beer, C. Eisner, Efficient detection of vacuity in temporal model checking, Formal Methods in System Design 18 (2001).

[9] M. Montali, F. Chesani, P. Mello, F. M. Maggi, Towards data-aware constraints in declare, in: Proc. of SAC 2013, ACM, 2013, pp. 1391–1396.

[10] X. He, D. Cai, P. Niyogi, Laplacian score for feature selection, in: NIPS 2005, 2005, pp. 507–514.

[11] W. W. Cohen, Fast effective rule induction, in: In Proc. of the 12th Int. Conf. on Machine Learning, Morgan Kaufmann, 1995, pp. 115–123.

[12] J. R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.