Precision-Guided Minimization of Arbitrary Declarative Process Models

Eduardo Goulart Rocha[1,2 (✉) [0009−0000−1184−1188]] and Wil M.P. van der Aalst[2,1 [0000−0002−0955−6940]]

[1] Celonis Labs GmbH, Munich, Germany
[2] Process and Data Science (PADS) Chair, RWTH Aachen University, Germany
e.goulartrocha@celonis.com wvdaalst@pads.rwth-aachen.de

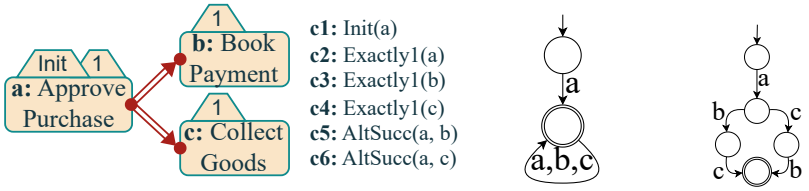
Abstract. Declarative model minimization is a computationally expensive task. State-of-the-art approximation techniques rely on hard-coded heuristic functions based on properties of constraint templates, which requires rework when new templates are added and cannot handle models expressed as arbitrary logical formulas. We present a precision-based heuristic function that requires no pre-configuration and can handle arbitrary constraints, provided they can be mapped to a finite automaton. The approach is evaluated on real-world datasets, where it outperforms state-of-the-art methods while accepting a wider range of inputs.

Keywords: Process Mining · Declarative Process Mining · Constraint Minimization · Redundant Constraints

1 Introduction

Declarative process models are a popular modeling paradigm for processes with high variability [14]. Figure 1a shows a declarative model for a procurement process. It consists of constraints to which the process must adhere. The constraints mandate that the process starts with the approval of the purchase (c_1), that the payment is booked/the goods are collected if and only if the purchase was approved (c_5, c_6), and that all activities happen exactly once (c_2, c_3, c_4).

Declarative models may contain redundancies. In Figure 1a, constraint c_3 can be inferred from $c_1, c_2,$ and c_5 (if a happens exactly once at the beginning and a



(a) A declarative model $M = \{c_1, \cdots, c_6\}$ (b) c_1's DFA (c) M's DFA

Fig. 1: A declarative procurement process (1a) and some related DFAs (1b, 1c).

must be followed by b, then b must happen exactly once). Redundant constraints express hidden dependencies. These are often inserted by an unwary user or by automated process discovery methods. Redundant constraints bloat the model, affecting its understandability, without adding extra information. Thus, it is important to minimize process models by removing as much redundancy as possible.

Redundancy resolution is a difficult task. State-of-the-art techniques approximate its solution by checking for redundant constraints according to a specific order [7]. The ordering criterion affects the approximation quality. State-of-the-art ordering criteria consist of a hard-coded heuristic tailored specifically for the underlying declarative language [7]. However, this approach has a few drawbacks:

1. The method can only be applied to declarative languages based on templates
2. The heuristic must be redesigned every time the language is changed
3. Designing the heuristic function requires domain expertise
4. The heuristic might lead to suboptimal pruning

We present a heuristic function based on model precision for *consistent* declarative models. The precision function measures the "excess behavior" allowed by each constraint with respect to the model. More precise constraints are closer to the model's language and thus also stricter. Stricter constraints are prioritized as they are less likely to be redundant. The proposed method requires no user input and is applicable to any set of constraints expressible as DFAs, including arbitrary $\text{LTL}_f/\text{LDL}_f$ formulas, thus going beyond the DECLARE language. A preliminary evaluation on real-world datasets shows that the approach outperforms the state-of-the-art in terms of runtime and pruning efficiency.

2 Preliminaries

Given a set of activity labels Σ, a trace $\sigma \in \Sigma^*$ is a finite sequence of activities. A *constraint* c is an object whose language $\mathcal{L}(c) \subseteq \Sigma^*$ defines its accepted traces. A *declarative process model* M is a set of constraints M with language $\mathcal{L}(M) = \bigcap_{c \in M} \mathcal{L}(c)$. A model M is *consistent* if $\mathcal{L}(M) \neq \emptyset$, i.e. constraints do not contradict each other. In this work, we consider constraints expressing *regular languages*, which can be defined using Deterministic Finite Automata:

Definition 1. ***(Deterministic Finite Automaton (DFA))*** *Is a tuple* $(Q, \Sigma, \delta, q_0, F)$ *where* Q *is the set of states,* Σ *is a set of symbols,* $\delta : Q \times \Sigma \nrightarrow Q$ *is the transition function,* q_0 *is the initial state and* $F \subseteq Q$ *is the set of final states.*

A DFA $D = (Q, \Sigma, \delta, q_0, F)$ *accepts* a trace $\sigma = l_1 \cdots l_n \in \Sigma^*$ iff. there is a path $(q_0, l_1, q_1) \cdots (q_{n-1}, l_n, q_n) \in (Q \times \Sigma \times Q)^*$ s.t. $q_n \in F$ and $\delta(q_{i-1}, l_i) = q_i \; \forall 1 \leq i \leq n$. The language of D is the set $\mathcal{L}(D) = \{\sigma \in \Sigma^* | \sigma \text{ is accepted by } D\}$.

Template-based declarative languages such as DECLARE [14] are backed by a set of logical formula templates, e.g. LTL_f formulas. Table 1 shows a snippet of the DECLARE language. Templates can be instantiated with concrete parameters, yielding a constraint. For example, setting $x =$ "*Approve Purchase*"

Table 1: A snippet of DECLARE templates [9]. See [4] for the LTL_f semantics.

Template		LTL_f
1. Position		
Init(x)	Trace must start with x	$\Box(start \rightarrow x)$
End(x)	Trace must end with x	$\Box(end \rightarrow x)$
2. Cardinality		
AtMost1(x)	x occurs at most once	$\Box(x \rightarrow \neg \bigcirc \Diamond x)$
Exactly1(x)	x occurs exactly once	$\Diamond x \wedge \Box(x \rightarrow \neg \bigcirc \Diamond x)$
3. Bidirectional Coupling		
AltSucc(x, y)	x and y occur in indirect sequence	$\Box((x \rightarrow \bigcirc(\neg x \text{ U } y)) \wedge (y \rightarrow \blacklozenge x))$
Choice(x, y)	x or y must occur, but not both	$(\Diamond x \wedge \neg\Diamond y) \vee (\neg\Diamond x \wedge \Diamond y)$

for template $Init(x)$ yields constraint c_1 from Figure 1a. We refer to [6] for a discussion on the expressiveness of LTL_f. For now, it suffices to know that LTL_f (thus also DECLARE) is strictly less expressive than DFAs.[3] And that given an LTL_f formula φ, there exists a DFA D accepting exclusively the traces satisfying φ, i.e. $\sigma \models \varphi \Longleftrightarrow \sigma \in \mathcal{L}(D)$. Furthermore, reasoning tasks can also be mapped to DFA operations. Particularly, given a set of formulas $\Phi = \{\varphi_1, \varphi_2, \cdots, \varphi_n\}$ and its respective DFAs $\mathcal{D} = \{D_1, D_2, \cdots, D_n\}$, then $\sigma \models \Phi \Longleftrightarrow \sigma \in \bigcap_{D_i \in \mathcal{D}} \mathcal{L}(D_i)$.

Figure 1 shows the minimal DFAs for constraint c_1 (1b) and model M (1c). Notice how $\mathcal{L}(M) \subset \mathcal{L}(c_1)$, i.e. does not fully describe M. The next section presents an approach for redundancy resolution based on model precision, which aims at preserving constraints that provide the most information.

3 Minimizing Arbitrary Declarative Models

We formalize the problem of declarative model minimization:

Definition 2. ***(Declarative Model Minimization Problem)*** *Given a declarative process model M, the model minimization problem is defined as:*

$$M_{min} = \underset{M' \subseteq M, \mathcal{L}(M') = \mathcal{L}(M)}{\arg\min} |M'| \tag{1}$$

Model minimization searches for a minimal equivalent sub-model. Its exact computation requires checking all subsets $M' \subseteq M$, which is intractable. Therefore, state-of-the-art methods approximate it by greedily checking for redundant constraints based on a specific constraint ordering criterion. In this work, we consider the minimization procedure from [7]. Algorithm 1 presents a simplified and slightly more general version of this algorithm for redundancy resolution.

[3] [6] introduces the Linear Dynamic Logic over Finite Traces (LDL_f), which is as expressive as DFAs. We do not discuss LDL_f formulas, but we notice that methods presented here work with arbitrary DFAs and hence are directly applicable to them.

Algorithm 1: Adaptation of the MINERful minimization algorithm [7]. The steps related to conflict resolution were removed and the checks for the subsumption hierarchy were replaced by a generic inclusion check.

```
1  input A declarative model M ⊆ C and an ordering criterion <_h
2  output A declarative model M' ⊆ M s.t. M' ⇒ M
3  Γ ← ∅;
4  foreach (c_i, c_j) ∈ (M × M) ∧ c_i ≠ c_j do
5  |  if c_i, c_j ∉ Γ ∧ L(c_i) ⇒ L(c_j) then Γ ← Γ ∪ {c_j} ;
6  M_0 ← sort([c_i ∈ M | c_i ∉ Γ], <_h)
7  foreach i ∈ [1, 2, ··· , |M_0|] do
8  |  if L({c_1, c_2, ··· , c_{i-1}}) ⊆ L(c_i) then Γ ← Γ ∪ {c_i} ;
9  M_1 ← sort([c_i ∈ M' | c_i ∉ Γ], <_h)
10 foreach i ∈ [|M_1|, ··· , 2, 1] do
11 |  if L(M \ (Γ ∪ {c_i})) ⊆ L(c_i) then Γ ← Γ ∪ {c_i} ;
12 M_2 ← M \ Γ
13 return M_2
```

The procedure accepts a model M and an ordering criteria $<_h$. Initially, a preliminary pass (lines 4-6) removes trivial constraint implication pairs. Next, a first redundancy resolution pass (line 7-9) removes constraints that are implied by constraints preceding it in $<_h$. Finally, a second pass (lines 10-12) ensures that the returned set is redundancy-free by checking each constraint against all others. This is done starting from the largest (according to $<_h$) constraints.

The correctness of Algorithm 1 follows directly from the fact that only redundant constraints are removed. Its optimality depends on $<_h$. A good ordering criterion $<_h$ should sort constraints with the highest pruning potential first. In [7], a series of ordering criteria are presented based on properties of the constraint templates such as constraint type, activation linkage, and relevance metrics (see [7] for more details). This approach requires a redesign of $<_h$ every time the set of templates is extended, which is time-consuming and error-prone. Furthermore, it is only applicable to models built from a set of fixed templates. To address these deficiencies, we propose to define $<_h$ using a precision function. A precision function is a function $p : \Sigma^* \times \Sigma^* \to \mathbb{R}$, where $p(L_1, L_2)$ estimates the fraction $\frac{|L_2 \setminus L_1|}{|L_1|}$. A precision function p induces an ordering criteria $<_{M,p}$ as follows:

Definition 3. ***(Precision-Based Constraint Ordering)*** *Let M be a declarative model and $p : \Sigma^* \times \Sigma^* \to \mathbb{R}$ a precision function. The ordering $<_{M,p}$ induced by p with respect to M is defined as:*

$$c_1 <_{M,p} c_2 \Longleftrightarrow p(\mathcal{L}(M), \mathcal{L}(c_1)) > p(\mathcal{L}(M), \mathcal{L}(c_2)) \tag{2}$$

Intuitively, the more precise a constraint $c_i \in M$ with respect to M, the stricter it is and the more likely it is to prune subsequent constraints. Multiple precision measures have been proposed in the process mining literature (see

Table 2: Comparison of pruning efficiency using the hard-coded function from [7] that orders constraints by increasing type priority and decreasing activation linkage, and the precision-based Escaping Edges Precision (ETC).

$<$	prio.	link.	const.	$\in M_1(M_2)$	$ETC(M, c_i)$	const.	$\in M_1(M_2)$
-	1	3	Init(a)	✓(✓)	0.46	AltSucc(a, b)	✓(✓)
↓	2	3	Exactly1(a)	✓(✓)	0.46	AltSucc(a, c)	✓(✓)
↓	2	2	Exactly1(b)	✓(✓)	0.40	Exactly1(a)	✓(✓)
↓	2	2	Exactly1(c)	✓(✓)	0.40	Exactly1(b)	×(×)
↓	3	3	AltSucc(a, b)	×(×)	0.40	Exactly1(c)	×(×)
+	3	3	AltSucc(a, c)	×(×)	0.35	Init(a)	×(×)

Section 5 for an in-depth discussion). We propose using the Escaping Edges Precision (ETC) [11] as it accepts arbitrary DFAs as inputs, it can be efficiently computed given both DFAs and it performs well when $\mathcal{L}(M) \subseteq \mathcal{L}(c_i)$. A drawback is that the ETC precision only works when the model is consistent.

Table 2 compares both heuristics when minimizing the model M from Figure 1. Notice that the ETC heuristic leads to better pruning. Both heuristics differ in how they order the *Init* and *AltSucc* constraints.

4 Experimental Evaluation

We evaluate the proposed approach on the example task of discovering a declarative model from a procedural specification. Given a procedural model and a set of constraint templates, we instantiate all templates with all possible parameter combinations and check if the procedural model satisfies the constraints. The result is an "equivalent" declarative version of the original model. We use manually designed process models for the Road Fines (RF), and BPI Challenge 2020 (BPI-20) processes, derived from their respective process specifications, and a model discovered from the BPI Challenge 2017 log (BPI-17).[4]

We consider the DECLARE[5] (DEC) templates [9] with the hard-coded heuristic function specified in [7] (HC), as well as the templates by Ramezani et al. [16] (RMZ). The latter includes constraint templates with varying numbers of arguments (k) and subsumes the set of DECLARE templates. For that, we design a heuristic function inspired by the one described in [7]. We consider the minimization with the ETC precision function ([11]) and with a random shuffle of the constraints as a baseline (RND).

Quantitative Evaluation We assess the pruning and runtime efficiency of each heuristic. We report the number of constraints after the preliminary pass ($|M_0|$), and after the first/second minimization passes (1st/2nd #min), along with the runtime for each run. The approach is implemented in pure Python and each experiment is assigned a memory limit of 4 GBs. Table 3 summarizes the results.

[4] The datasets can be found at github.com/EduardoGoulart1/BPMDS24

[5] Extracted from github.com/cdc08x/MINERful/commit/8311258

Table 3: Comparison of random (RND), hard-coded (HC), and precision-based (ETC) heuristic functions in terms of pruning performance (#min) and computation time in seconds. We report the numbers for two runs of the algorithm, performing only a first pass and performing both passes over the constraints.

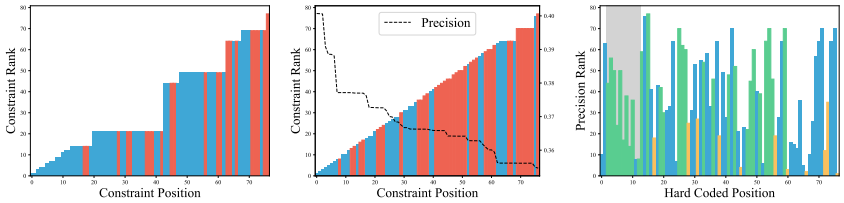
DS	LIB	K	$\|M_0\|$	RND [1st|2nd]				HC [1st|2nd]				ETC [1st|2nd]			
				#min		time(s)		#min		time(s)		#min		time(s)	
RF	DEC	2	77	56	26	11.3	20.5	50	**23**	4.6	27.9	**36**	**23**	17.3	43.8
	RMZ	2	151	109	72	7.9	38.0	105	**68**	7.6	42.2	**82**	69	5.6	13.7
		3	197	94	62	4.8	48.8	98	**59**	4.0	22.4	**86**	60	3.3	7.4
		4	2623	!	!	!	!	**173**	**58**	17.4	283.6	!	!	!	!
BPI-20	DEC	2	37	30	**17**	0.15	1.24	36	23	0.61	12.6	**19**	18	1.34	2.16
	RMZ	2	85	59	35	0.24	2.41	53	**34**	0.78	2.30	**41**	35	0.49	0.58
		3	92	60	33	0.48	2.81	49	**31**	0.30	1.30	**36**	**31**	0.38	0.52
		4	122	76	32	2.25	3.17	74	32	0.48	2.44	**35**	**31**	0.66	1.25
BPI-17	DEC	2	79	65	!	7.8	!	67	**32**	2.6	47.5	**44**	33	8.2	41.1
	RMZ	2	138	88	**62**	5.8	201	95	**62**	7.5	118.6	**81**	**62**	10.2	26.8
		3	145	91	**57**	11.1	83	81	59	3.6	43.0	**78**	58	8.3	19.1
		4	416	121	66	83.1	735	123	**59**	4.8	90.1	**84**	60	36.6	50.9

For the single-pass minimization procedure, the ETC heuristic achieves the best pruning efficiency, being the best-performing method in all but one situation where it ran out of memory. Perhaps most surprisingly, the hard-coded heuristic frequently performs similar or worse than the random heuristic. After the second pass, the difference between the hard-coded and the ETC heuristics diminishes.

The runtime of the single pass for the ETC-based heuristic is dominated by the computation of $\bigcap_{c \in M} \mathcal{L}(c)$ (this is also the step where it runs out of memory). In comparison, the hard-coded heuristic is cheap to compute, and consequentially faster for the single pass. However, when considering the second pass, the situation flips. The time to compute $\bigcap_{c \in M} \mathcal{L}(c)$ is compensated by a subsequent speedup in pruning. Making the ETC heuristic faster than the hard-coded in all but one situation (excluding the out-of-memory). Overall, the experiments show that the ETC-based heuristic provides better pruning than the hard-coded heuristic. With a single pass, ETC prunes almost as many constraints as the hard-coded with two passes, while being much faster. Therefore, the ETC heuristic surpasses the state-of-the-art in terms of pruning efficiency and performance.

Comparing the Orders We now compare the orderings induced by the hard-coded and the ETC-based heuristics. We consider the Road Fines (RF) model and the DECLARE templates (DEC). Figure 2 compares the results of the single-pass minimization procedure produced by the hard-coded and ETC-based heuristics.

Figures 2a and 2b plot the position of each constraint and their ranks according to the hard-coded/ETC heuristics. Constraints colored red are marked as redundant. The hard-coded function contains large plateaus of multiple constraints with the same rank, which explains why it performs similarly to the random ordering. In comparison, ETC can better distinguish the rank of each



(a) Hard-coded (b) ETC (c) Hard-coded vs ETC

Fig. 2: HC vs ETC heuristics on the DEC-2 templates for the Road Fines dataset.

constraint. However, notice that the ETC precision varies in a relatively small range from 0.35 to 0.40. Sometimes, the difference in precision between two constraints is $< 10^{-5}$. This shows the problem with the *low-resolution* of the ETC function and motivates the development of higher-resolution metrics.

Finally, Figure 2c compares the position assigned by the hard-coded ordering criterion with the rank assigned by the ETC function. Constraints colored green are marked as redundant by the ETC heuristic, but not by the hard-coded. Constraints colored orange are marked as redundant by the hard-coded ordering but not the ETC-based. Both orderings seem to have little agreement between them. More interestingly, in Figure 2c there is a cluster of green constraints at the beginning encompassing almost all DECLARE cardinality constraints (gray background). The ETC heuristic assigns them high ranks and marks them as minimal. This suggests that the hard-coded function could be improved by lowering the type priority of cardinality constraints. We tried that in our experiments. To our surprise, the minimization using this modified function ran out of memory. A possible explanation is that not all cardinality constraints are trivial. Assigning a low priority to all cardinality constraints misses important constraints that help to reduce the size of intermediate models, causing out-of-memories. In contrast, the ETC-based heuristic can separate constraints of the same type if it deems that they are not equally relevant in the given context.

Overall, the experiments from this section show that the ETC-based ordering criterion produces fewer ties in the constraint ordering and that even a heuristic function designed by domain experts can be sub-optimal. A possible argument in favor of hard-coded heuristics is that they can incorporate other aspects such as understandability, ensuring that more understandable constraints are prioritized. However, even in these situations, a precision-based heuristic could be used as a secondary ordering criterion, to break ties between constraints.

5 Related Work

DECLARE [14] is the de-facto standard language for declarative process mining. Its greatest advantages are usability and simplicity. Using constraint templates facilitates tasks such as discovery [9, 12] and reasoning [7]. More recently, as the limitations of DECLARE in terms of expressiveness and the link between

DECLARE, LTLf/LDLf, and DFAs became clear [6], more general reasoning methods aiming at the full LTLf/LDLf specifications have been proposed [8,10].

Most precision metrics in process mining require that at least one of its inputs is finite [1,2,5,13]. The ETC precision was first proposed in [13] and later refined in [2] to handle unfitting behavior by using trace alignment. In [11], the approach is extended to handle infinite languages by comparing their respective DFAs. Furthermore, [11] suggests projecting both languages' DFAs to subsets of activities to handle language mismatches. In our setting, the projection step is not necessary since the model's language always fits the constraint's language.

In [15], a family of precision metrics is proposed based on the quotient $\frac{\|\mathcal{L}(M)\cap\mathcal{L}(c)\|}{\|\mathcal{L}(c)\|}$, where $\|.\|$ is a (monotone) language measure, e.g. the language's topological entropy. This metric did not lead to satisfactory results in our experiments since $\mathcal{L}(M) \cap \mathcal{L}(c_i) = \mathcal{L}(M)\ \forall c_i \in M$, and multiple constraints have the same topological entropy $\|c_i\|$, leading to multiple ties in their ordering criteria. In [3] a precision metric based on the comparison of the sets of k-trimmed subtraces (the Markovian abstraction) of M and c_i is proposed. This metric also performed poorly in our experiments since many constraints have very similar abstractions. Similarly, techniques based on behavioral relations [17] also perform poorly in our setup since many constraints have very similar behavioral relations.

6 Conclusion

This paper proposed a method for the minimization of arbitrary declarative process models using model precision. The approach is in itself competitive with state-of-the-art techniques even when applied to declarative models for which a hard-coded heuristic function exists. Moreover, it is the only alternative for models containing arbitrary LTLf/LDLf formulas. This paves the way for the usage of declarative models that go beyond template-based declarative languages.

In future work, we plan to develop "higher resolution" precision functions that produce fewer ties between constraints, to investigate more efficient precision functions that do not require the intersection of all constraint DFAs, and to extend it to inconsistent declarative models. Finally, we would like to evaluate the proposed approach on sets of arbitrary LTLf/LDLf formulas, which is so far not possible due to a lack of suitable LTLf/LDLf miners.

Acknowledgements We thank the Alexander von Humboldt (AvH) Stiftung for supporting our research.

References

1. Adriansyah, A., van Dongen, B., van der Aalst, W.M.P.: Conformance checking using cost-based fitness analysis. In: 2011 IEEE 15th International Enterprise Distributed Object Computing Conference. pp. 55–64 (2011)

2. Adriansyah, A., Munoz-Gama, J., Carmona, J., Dongen, B., Aalst, W.M.P.: Measuring precision of modeled behavior. Information Systems and e-Business Management **13** (01 2014)
3. Augusto, A., Armas-Cervantes, A., Conforti, R., Dumas, M., Rosa, M.L.: Measuring fitness and precision of automatically discovered process models: A principled and scalable approach. IEEE Transactions on Knowledge and Data Engineering **34**(4), 1870–1888 (2022)
4. Cecconi, A., Di Ciccio, C., De Giacomo, G., Mendling, J.: Interestingness of traces in declarative process mining: The janus ltlpf approach. In: Business Process Management. pp. 121–138. Springer International Publishing (2018)
5. Chatain, T., Carmona, J.: Anti-alignments in conformance checking – the dark side of process models. In: Application and Theory of Petri Nets and Concurrency. pp. 240–258. Springer International Publishing (2016)
6. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence. p. 854–860. IJCAI '13, AAAI Press (2013)
7. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: Resolving inconsistencies and redundancies in declarative process models. Information Systems **64**, 425–446 (2017)
8. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: On the relevance of a business constraint to an event log. Information Systems **78**, 144–161 (2018)
9. Di Ciccio, C., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM Trans. Manage. Inf. Syst. **5**(4) (jan 2015)
10. Giacomo, G.D., Masellis, R.D., Grasso, M., Maggi, F.M., Montali, M.: Monitoring business metaconstraints based on ltl and ldl for finite traces. In: International Conference on Business Process Management (2014)
11. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Scalable process discovery and conformance checking. Software and Systems Modeling **17**, 599 – 631 (2016)
12. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient discovery of understandable declarative process models from event logs. In: Advanced Information Systems Engineering. pp. 270–285. Springer Berlin Heidelberg (2012)
13. Munoz-Gama, J., Carmona, J.: A fresh look at precision in process conformance. vol. 6336, pp. 211–226 (09 2010)
14. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: Declare: Full support for loosely-structured processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007). pp. 287–287 (2007)
15. Polyvyanyy, A., Solti, A., Weidlich, M., Di Ciccio, C., Mendling, J.: Monotone precision and recall measures for comparing executions and specifications of dynamic systems. ACM Trans. Softw. Eng. Methodol. **29**(3) (jun 2020)
16. Ramezani, E., Fahland, D., van Dongen, B., van der Aalst, W.M.P.: Diagnostic information in temporal compliance checking. tech. rep., BPM Cent. Rep (2) (2012)
17. Weidlich, M., Polyvyanyy, A., Desai, N., Mendling, J., Weske, M.: Process compliance analysis based on behavioural profiles. Inf. Syst. **36**, 1009–1025 (11 2011)