The DrFurby Classifier submission to the Process Discovery Contest @ BPM 2016

H.M.W. Verbeek and F. Mannhardt

Department of Mathematics and Computer Science
Eindhoven University of Technology, Eindhoven, The Netherlands
h.m.w.verbeek@tue.nl, f.mannhardt@tue.nl

Abstract. To check the current state-of-the-art in process discovery, a contest has been set up. In this contest, 10 undisclosed process models were created by the organizers. For every created process model, the organizers created two logs: a training log containing 1000 traces and a test log containing 20 traces. Both logs were disclosed to the participants. The challenge for the participants was to classify every trace from every test log whether or not it fits the corresponding original process model, that is, whether or not it can be replayed perfectly by that process model. As the original process model was not disclosed, the participants had to first discover an as-good-as-possible process model from the training log, and second use this discovered process model to classify every trace from the corresponding test log. The main assumption here is that a better discovery technique discovers a better (that is, more matching to the original) process model, which then results in a better classification. In the end, the one that classifies the most traces correctly, wins the contest. This paper presents a submission to this contest that, in the end, classifies 193 out of 200 traces correctly.

1 introduction

In the area of process mining, *process discovery* has been an important part from the beginning. To check the state-of-the art in process discovery, a contest has been conceived, called the "Process Discovery Contest @ BPM 2016" [5] (simply called the Contest hereafter).

To participate to the Contest, participants needed to propose a discovery algorithm, a resulting model class, and a means to check whether or not a trace fits a discovered model. As an example, the process discovery algorithm could be the often-cited α-miner [3], the model could be a workflow net, [4], and the alignment-based replay [2] could be used to check the fitness.

This paper presents a submission to the Contest, called the *DrFurby Classifier*, where the discovery algorithm is a collection of existing discovery algorithms [7, 12] and decomposition techniques [9], the model is a collection of two accepting Petri nets [9], and the means to check fitness is the decomposed alignment-based replay [9]. In the end, the DrFurby Classifier classifies 193 out of 200 traces correctly. The 7 traces that were misclassified were all traces that did not fit the

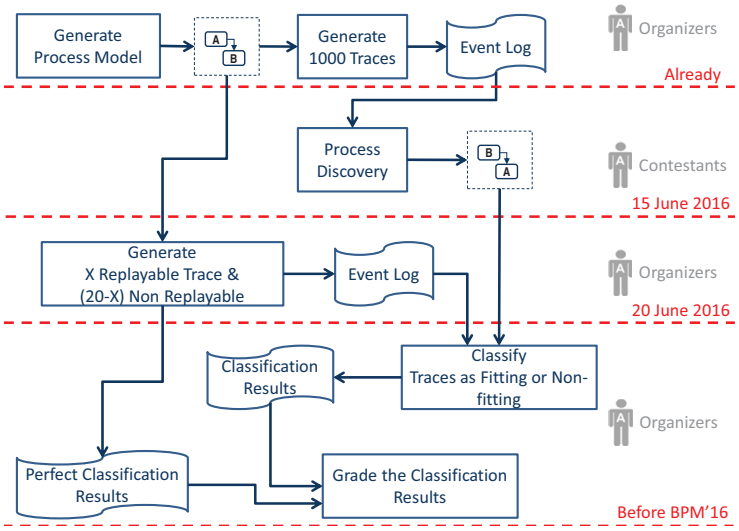

Fig. 1. An overview of the Contest.

original model perfectly, but that did fit the discovered models perfectly. As such, the models discovered by the DrFurby Classifier allowed for more behavior than the original models did. We did not encounter any trace that did fit the original model perfectly, but did not fit the discovered models perfectly.

The remainder of this paper is organized as follows. First, Section 2 provides relevant details on the Contest itself. Second, Section 3 introduces our approach to deal with the challenges as posed by the Contest. As we will see, this approach allows for many discoverers to be used. As we did not want to include all possible discoverers, Section 4 shows how we discarded redundant discoverers. In the end, this resulted in only two remaining discoverers. Then, Section 5 introduces the implementation of our approach in ProM 6.6, which includes a new XES extension [6] to contain the classification results. Using this implementation, Section 6 shows the results we obtained on all logs. Finally, Section 7 concludes the paper.

2 The Contest

Figure 1 shows an overview of the Contest as made by the organizers. In the Contest, 10 undisclosed process models are generated. For every process model, a perfectly fitting (free of noise) *Training* event log containing 1000 traces is generated, which is disclosed. Furthermore, for every process model an undisclosed *June* log is created containing 20 traces, which should be classified as *positive* (if the trace could have been generated by the process model) or *negative* (if the trace could not have been generated by the process model). *The*

winner is/are the contestant(s) whose approach can classify correctly the largest number of traces in all the test event logs. Thus, the goal of the contest is to get the classification for these 10 logs as good as possible, that is, to minimize the number of *false* positives and *false* negatives. A false positive is a negative trace that is classified as positive, whereas a false negative is a positive trace classified as negative.

To help the contestants, for every process model also two test logs are provided, called the *April* log and the *May* log henceforth. Both logs contain 20 traces (like the June log), and for both logs it is known that they contain 10 positives and 10 negatives. Using these April and May logs, the contestants can improve their approach.

Based on the description of the Contest, the following observations can be made, which are vital to the DrFurby Classifier:

No noise The training logs contain no noise. As a result, any trace in the training log should be classified as positive.

Free model The model to be used is not restricted except for the fact that there should be some way to check whether or not a trace fits the model. As a result, a model may consist of multiple submodels, as long as there are ways to check fitness on the submodels.

Classification The submissions are ranked primarily on their classification results, and secondary on the time to need for discovering the model. As a result, there is no reason to go for a model that looks nice. No points will be awarded by a nice model, the model should just classify as good as possible.

3 Approach

The approach taken by the DrFurby Classifier is to minimize the number of false negatives by only using techniques that guarantee perfect fitness, and to minimize the number of false positives by using as many of these techniques as are needed. Note that for the former we rely on the fact that the training logs do not contain noise. By using only discovery techniques that guarantee a perfectly fitting model, we guarantee that every trace from the training will be classified as being positive.

Examples of relevant techniques that guarantee perfect fitness include a number of discovery techniques that guarantee perfect fitness:

Inductive Miner The "Inductive Miner Infrequent" with noise threshold set to 0.0 [7].

ILP MIner The "ILP Miner" [11].

Hybrid ILP Miner (Default) The "Hybrid ILP Miner" with default settings [12].

Hybrid ILP Miner (Single) The "Hybrid ILP mIner" that uses only a single variable per causal relation [12].

Nevertheless, it also includes the "Divide and Conquer" decomposition technique, as this *preserves* perfect fitness [1].

Trace	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IM-100	+	−	−	−	+	+	+	+	+	−	−	+	+	+	+	+	−	−	+	+
HIM-0	+	−	+	−	+	+	+	+	+	+	−	+	+	+	+	−	+	−	+	+
Knock-out	+	−	−	−	+	+	+	+	+	−	−	+	+	+	+	−	−	−	+	+

Table 1. The results of IM-100, HIM-0, and their combination on May log #3.

In our approach, we run the various discoverers with various decomposition settings on the training log, and replay the test log on the resulting *accepting Petri net* using the decomposed replayer from the "Divide and Conquer" framework. An accepting Petri net is a Petri net with an initial marking, a final marking[1], and where transitions may be labeled with an activity from the log. Unlabeled transition are considered to be silent transitions, i.e., they do not correspond to any activity from the log. Note that we could also have used the monolithical replay [2], but the decomposed replay preserves perfect fitness [1] and is often much faster [8].

The approach employs a knock-out scheme. If one of the decomposed discoverers classifies a trace as negative, then it will be classified as negative. As such, this decomposed discoverer 'knocks-out' this trace to the negative side. A trace is only classified as positive if all decomposed discoverers classify the trace as positive.

4 Configuration

We have used the April and May logs to determine a minimal set of decomposed discoverers that provides maximal result. For this sake, we tested all discovery techniques, and various decomposition settings (Do not decompose, maximal decomposition, Decompose by 80%, by 60%, by 40%, ...). In the end, two decomposed discoverers remained that *occlude* the negative classifications of all others:

- The Inductive Miner with maximal decomposition (called *IM-100* henceforth).
- The Hybrid ILP Miner without decomposition (called *HIM-0* henceforth).

Table 1 shows why the combination of both techniques is useful: Only HIM-100 classifies the traces 3, 10, and 17 as negative (which is true), while only HIM-0 classifies trace 16 as negative (which is also true). Appendix A shows all models discovered by both IM-100 and HIM-0.

[1] In fact, an accepting Petri net has a *set of final markings*, but for this submission we only need to consider a single final marking (as every discoverer used results in a single final marking).

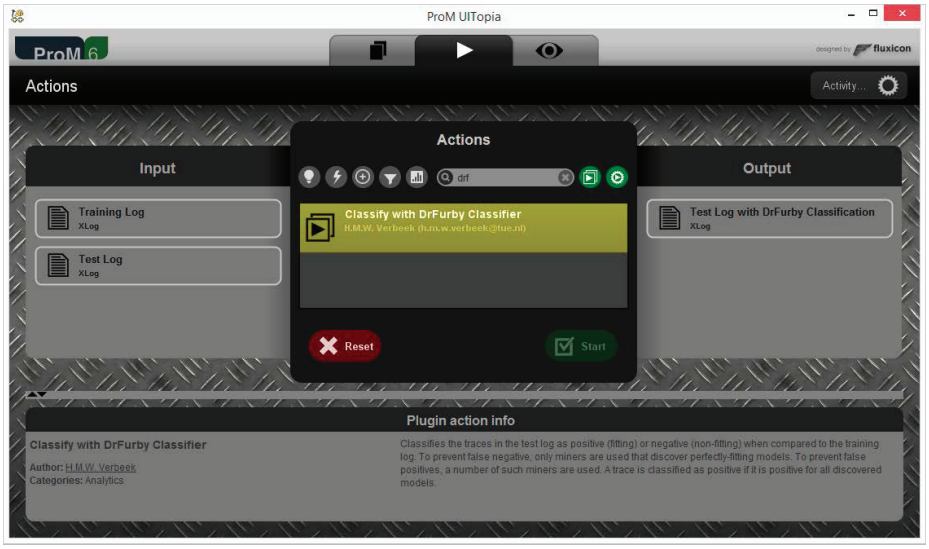

Fig. 2. The DrFurby Classifier plug-in with inputs and outputs.

5 Implementation

5.1 DrFurby Classifier plug-in

The DrFurby Classifier has been implemented as the "DrFurby Classifier" plug-in in ProM 6.6 [10]. This plug-in has been implemented in the "DivideAndConquer" package, as this package is an umbrella package for both discoverers and the decomposition. Fig. 2 shows the plug-in in ProM 6.6. The plug-in takes two event logs as input: a training log and a test log, and it produces a *classified* test log as output.

5.2 DrFurby Extension

To enrich the classified test log with the necessary classification attributes, we have implemented a *DrFurby Extension*, which uses the prefix "drfurby". Table 2 shows the attributes as defined by this extension.

5.3 Example

To showcase the plug-in, we use it on the imported training log #3 and May log #3 (note that we manually renamed both imported logs to "Train-03" and "May-03" to be able to tell them apart). Fig. 3 shows the results.

To start the plug-in, we select the "Start" button. In the end, this results in Fig. 4 and (after having selected the "Log Attributes" tab) Fig. 5.

The accepting Petri nets discovered from the training log can be visualized by selecting them in the "Resource" tab in ProM 6. See Fig. 6 for the nets as discovered for the "Train-03" log.

Attribute	Level	Type	Description
classification	trace	String	Classification of the trace ("positive" or "negative")
him0Costs	trace	Continuous	The costs of replaying this trace on the accepting Petri net as discovered by the HIM-0 discoverer.
im100Costs	trace	Continuous	The costs of replaying this trace on the accepting Petri net as discovered by the IM-100 discoverer.
millis	log	Discrete	The number of milliseconds to it took classify the log.
name	log	Literal	The name of the classified log.
negative	log	Discrete	The number of traces classified negative.
positive	log	Discrete	The number of traces classified positive.
totalCosts	trace	Continuous	The accumulated costs of replaying this trace on all discovered accepting Petri nets.

Table 2. Attributes as defined by the DrFurby Extension.

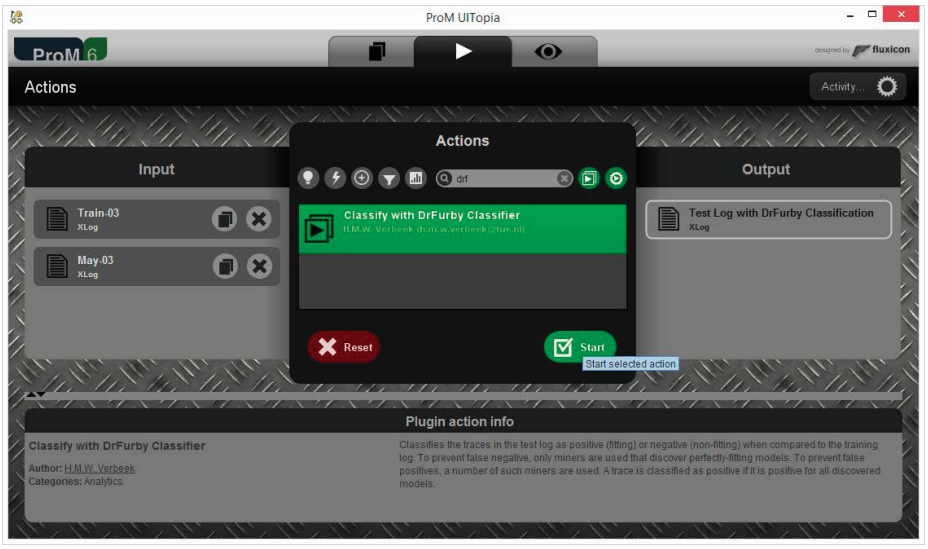

Fig. 3. The DrFurby Classifier plug-in with "Train-03" and "May-03" selected as inputs.

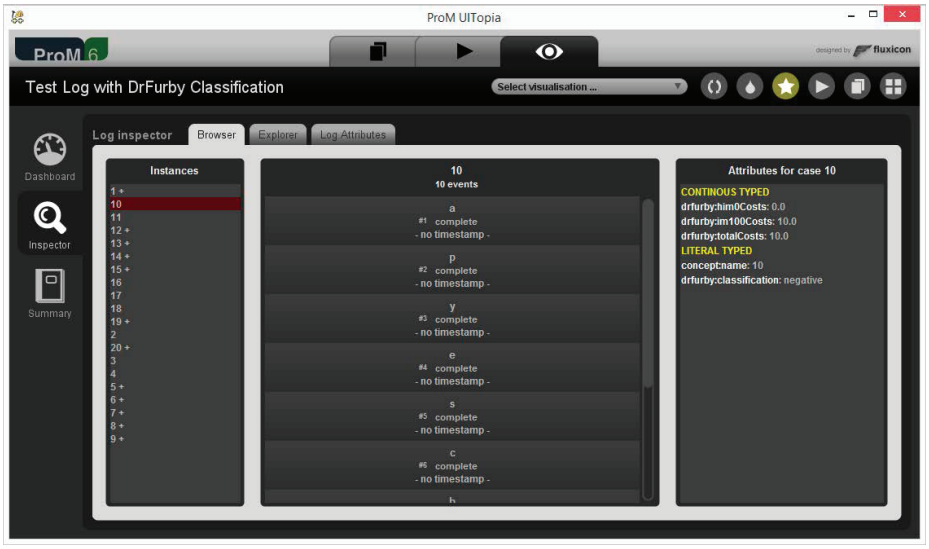

Fig. 4. The DrFurby Classifier output with "Train-03" and "May-03" selected as inputs. Note that in the top view the + signs in the "Instances" list provide an easy overview of which traces are classified as positive. The trace "10" has been selected. Attributes at the right-had side provide information on the classification. For example, the IM-100 discoverer classified it as negative (costs > 0.0), and the IM-0 discoverer as positive (costs $= 0.0$).

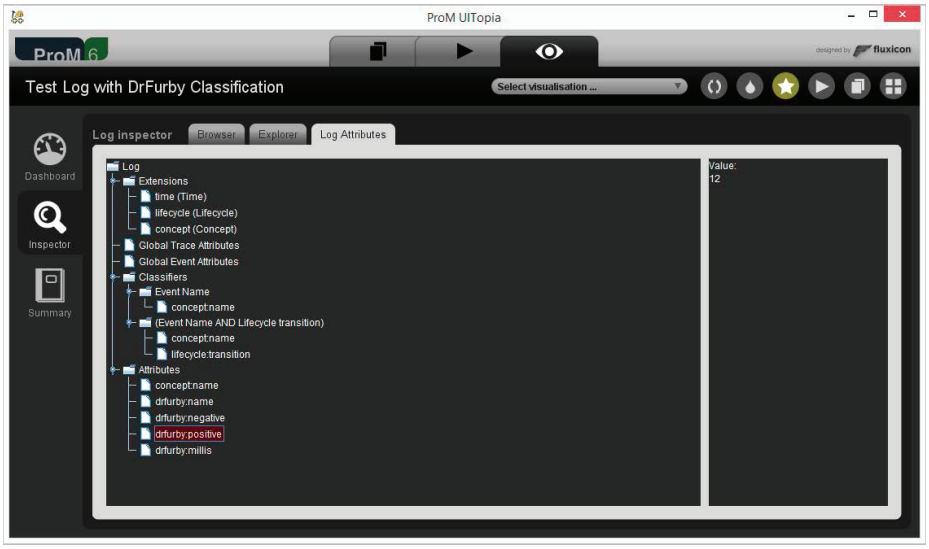

Fig. 5. The log attributes as added by the DrFurby Classifier. The "drfurby:positive" attribute is selected, which shows at the right-hand side that it 12 traces were classified as positive.

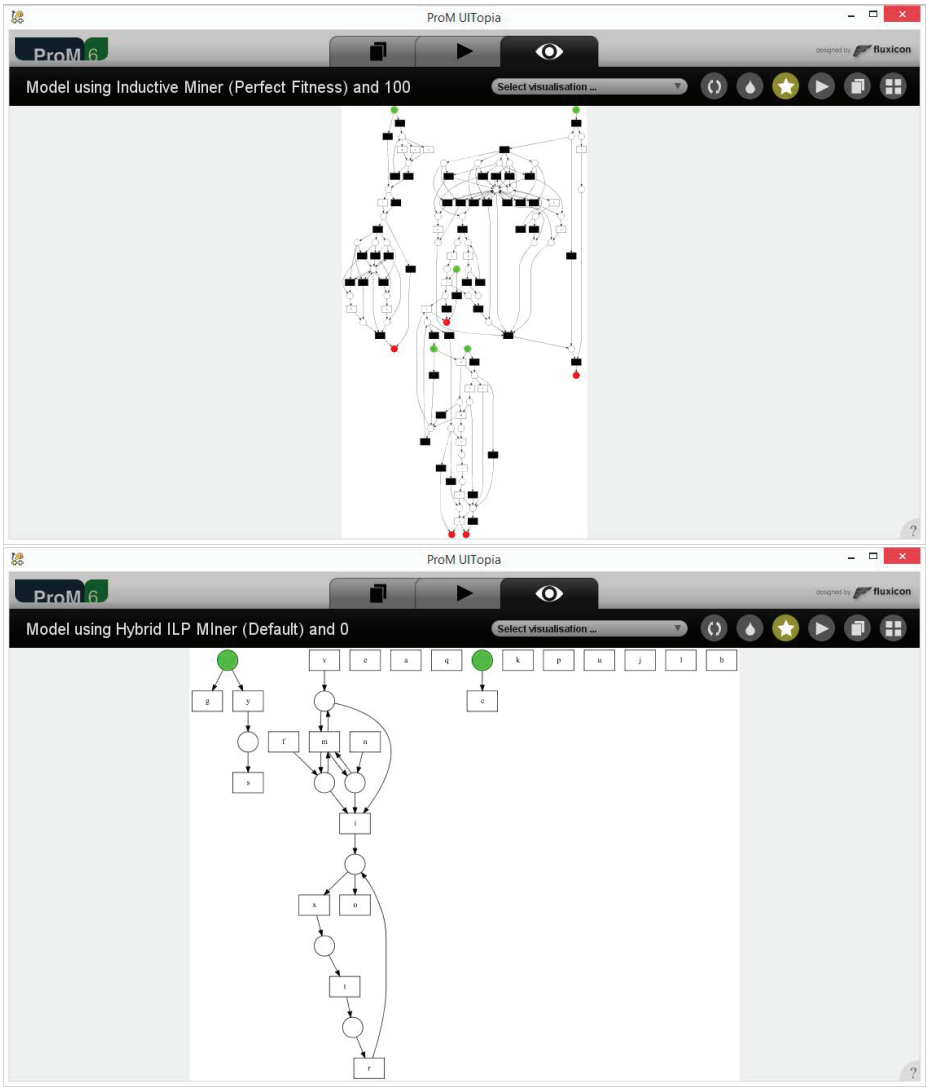

Fig. 6. The accepting Petri nets discovered by the DrFurby Classifier with "Train-03" and "May-03" selected as inputs.

5.4 Installation

Please follow the following steps to install the DrFurby Classifier:

1. Download a version of ProM 6.6 from `http://www.promtools.org/doku.php?id=prom66`.
2. Install the version on your local computer.
3. Run ProM PM 6.6 (that is, the Package Manager of ProM 6.6).
4. Select the "DivideAndConquer" package, and have it installed. There is no need to install the "RunnerUpPackages", as this only installs additional packages that are not needed for the DrFurby Classifier.
5. Set the memory option to 4GB (unless you're using a 32-bit version of ProM).
6. Close ProM PM 6.6.

The "DivideAndConquer" package containing the DrFurby Classifier is now installed and can be used in ProM 6.6.

6 Results

	#Class.		#False		←Trace→																			
Log↓	+	−	+	−	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
1	10	10	0	0	+	+	−	+	−	−	−	+	−	−	+	−	−	+	−	+	+	−	+	+
2	10	10	0	0	−	+	−	+	−	−	−	−	−	+	+	+	+	+	−	+	−	+	−	+
3	10	10	0	0	+	−	−	−	−	+	+	+	−	−	+	−	+	+	+	+	−	−	+	−
4	10	10	0	0	−	+	−	−	−	−	+	−	+	−	+	+	+	−	+	−	−	+	+	+
5	10	10	0	0	+	−	+	+	−	−	−	−	−	−	+	+	+	+	+	−	+	−	−	+
6	10	10	0	0	−	+	+	−	−	+	−	+	+	−	+	+	+	+	−	+	−	−	−	−
7	10	10	0	0	+	−	+	+	−	−	−	−	−	−	+	−	+	−	+	+	+	+	+	−
8	11	9	1	0	−	−	+	+	−	+	+	−	−	−	+	+	+	+	+	+	−	+	−	−
9	10	10	0	0	+	−	+	+	−	−	+	+	+	−	−	+	+	+	−	−	−	−	−	+
10	10	10	0	0	+	+	−	+	−	+	+	+	+	−	+	+	−	−	−	−	−	−	−	+

Table 3. The results of the DrFurby Classifier on the April logs.

Table 3 shows the results for the April logs, whereas Table 4 shows them for the May logs. As we know that every log contains 10 positives and 10 negatives, we immediately know that we have misclassifications for both the April and the May logs. In the April log #8, we have classified 11 traces as positive, which is one too many. As such, we have at least 1 misclassification for the April logs. In a similar way, we have at least 5 misclassifications for the May logs (2 for log #3, 1 for log #5, and 2 for log #8). Later on, the organizers confirmed that for the April log we had indeed 1 misclassification, and 5 for the May logs. From this, it can be easily concluded that all out misclassifications were false positives, and that we had no false negatives. Clearly, our approach works very well in

	#Class.		#False		←Trace→																			
Log↓	+	−	+	−	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
1	10	10	0	0	+	−	+	−	+	−	+	+	−	−	−	−	+	+	−	−	−	+	+	+
2	10	10	0	0	−	+	−	−	+	+	+	−	+	+	−	−	+	+	−	−	−	+	+	−
3	12	8	2	0	+	−	−	−	+	+	+	+	+	−	−	+	+	+	+	−	−	−	+	+
4	10	10	0	0	+	−	+	+	+	−	+	−	+	−	−	−	+	−	+	−	−	+	−	+
5	11	9	1	0	+	+	−	−	−	−	−	+	−	+	−	+	+	+	+	+	+	−	+	−
6	10	10	0	0	+	−	+	+	+	+	−	−	−	−	+	+	+	+	−	+	−	+	−	−
7	10	10	0	0	−	−	+	+	−	+	−	+	−	+	+	−	+	−	−	−	+	−	+	+
8	12	8	2	0	−	−	−	−	+	+	+	+	−	+	+	−	+	+	+	+	−	+	−	+
9	10	10	0	0	−	−	−	+	−	−	−	+	+	+	+	−	+	+	+	+	+	−	−	−
10	10	10	0	0	+	−	+	−	+	−	−	+	+	+	−	+	−	+	+	−	−	−	+	−

Table 4. The results of the DrFurby Classifier on the May logs.

minimizing the number of false negatives, as we have none in both the April and the May logs. Unfortunately, the approach is less successful in minimizing the numbers of false positives.

	#Class.		#False	←Trace→																			
Log↓	+	−		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
1	10	10	≥ 0	+	−	+	+	−	+	−	−	+	+	−	+	+	−	+	−	+	−	−	−
2	10	10	≥ 0	−	−	+	+	−	−	+	+	−	−	+	−	+	+	−	−	+	+	+	−
3	13	7	≥ 3	+	+	+	+	−	−	+	+	+	+	+	−	−	−	−	+	+	−	+	+
4	10	10	≥ 0	−	+	−	+	+	+	−	+	−	−	+	+	−	−	−	+	+	−	+	−
5	10	10	≥ 0	−	+	−	−	−	−	−	−	+	−	+	+	+	−	+	+	+	+	−	+
6	11	9	≥ 1	−	+	−	+	+	+	+	−	−	−	−	+	−	−	+	+	+	+	+	−
7	11	9	≥ 1	+	+	+	−	−	+	−	+	+	+	−	+	−	−	−	−	+	+	+	−
8	12	8	≥ 2	+	−	+	+	+	−	+	+	+	+	+	+	−	+	−	−	−	−	+	−
9	10	10	≥ 0	+	+	+	+	+	−	−	+	−	+	−	+	−	−	−	+	−	−	−	+
10	10	10	≥ 0	+	+	−	−	−	+	−	+	+	+	+	−	+	+	−	−	+	−	−	−

Table 5. The results of the DrFurby Classifier on the June logs.

Table 5 shows the final result: the classification of every of the 20 traces in the 10 June logs. Although this was not mentioned in the original call for participation, every June log contains 10 positives and 10 negatives (like the April and May logs). As such, this table shows that we have at least 7 misclassifications, as we have classified at least 7 traces too many as being positive. Assuming that our previous result (no false negatives) also holds for the June logs, we have *correctly classified 193 out of 200 traces*[2] using the DrFurby Classifier.

[2] Later, it as confirmed by the organizers of the Contest that we indeed had correctly classified 193 out of 200 traces.

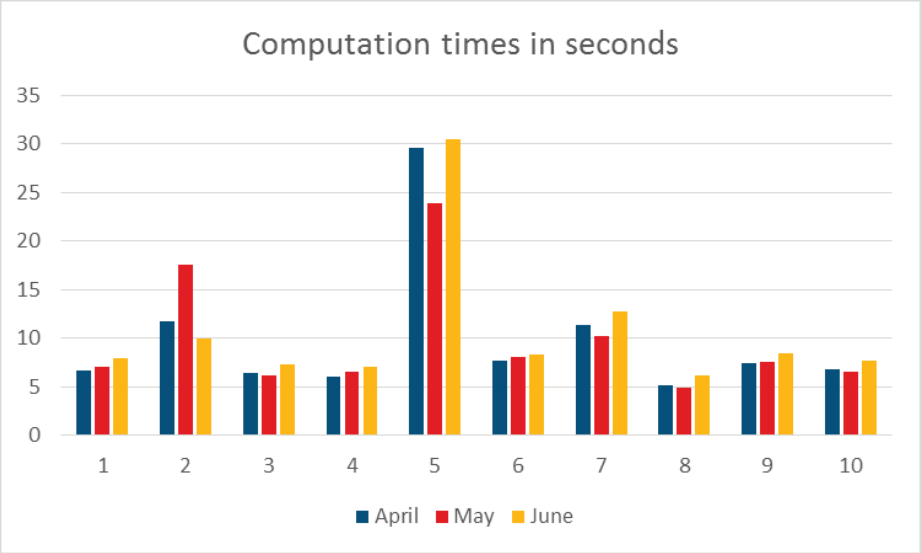

Fig. 7. Computation times for all logs. Times obtained on a Dell E4310 with Intel(R) Core(TM) i7 CPU M620@2.67 GHz, 8 GB of installed RAM, running 64 bits Windows 8.1 Enterprise.

6.1 Computation Times

Fig. 7 shows the computation times required by the DrFurby Classifier on all (April, May, and June) logs. Note that these computation times include the time required to discover the models for the training log. This figure shows that most of the logs required less than 10 seconds to classify, and that all required less than about 30 seconds.

7 Conclusions

The DrFurby Classifier seems to be a good approach to avoid false negatives. At least, there were no false negatives in the April, May, and June logs. This is due to the fact that we only use techniques that guarantee perfectly fitting models. As a result, any trace from the training is classified as positive. Of course, it is still possible that false negatives do occur, but at least so far we have not seen them.

The DrFurby Classifier seems also to be a good approach to avoid false positives. By combining two discoverers that guarantee perfect fitness and a decomposition approach that preserves perfect fitness, a better result was achieved than by any of the discoverers (be it with or without decomposition) on its own. As such, combining discoverers that guarantee perfect fitness makes sense. Apparently, different discoverers may have a different notion of generalization, which we can exploit nicely by combining them. Having said that, we of course

need to mention that it is not yet perfect, that is, that we can still improve if some other such discoverer would appear. Unfortunately, we do not know of such other discoverers, so for the time being we have to accept the fact that some positives will be false positives.

Timewise, the DrFurby Classifier is doing just fine. All logs were classified within half a minute, which included the time required to discover the accepting Petri nets from the training log.

Looking ahead, the DrFurby Classifier could be used for a similar contest next year, provided that the assumptions made by classifier are not violated:

- The training logs contain no noise.
- The model is allowed to contain multiple submodels.
- No points are awarded for how nice a model looks.

As soon as one of these assumptions does not hold, the DrFurby Classifier will be less usable (or even unusable).

References

1. Aalst, W.M.P.v.d.: Decomposing Petri nets for process mining: A generic approach. Distrib. Parallel Dat. 31(4), 471–507 (2013)
2. Aalst, W.M.P.v.d., Adriansyah, A., Dongen, B.F.v.: Replaying history on process models for conformance checking and performance analysis. Data Min. Knowl. Disc. 2(2), 182–192 (2012)
3. Aalst, W.M.P.v.d., Weijters, A.J.M.M., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. on Knowl. and Data Eng. 16(9), 1128–1142 (Sep 2004), http://dx.doi.org/10.1109/TKDE.2004.47
4. Aalst, W.M.P.v.d.A.: The application of Petri nets to workflow management. J. Circuit. Syst. Comp. 8(1), 21–66 (1998)
5. Carmona, J., de Leoni, M., Depair, B., Jouck, T.: Process Discovery Contest @ BPM 2016, http://www.win.tue.nl/ieeetfpm/doku.php?id=shared:process_discovery_contest, accessed on September 2, 2016
6. Günther, C.W., Verbeek, H.M.W.: XES standard definition. Tech. Rep. BPM-14-09 (2014), http://bpmcenter.org/wp-content/uploads/reports/2014/BPM-14-09.pdf
7. Leemans, S.J.J., Fahland, D., Aalst, W.M.P.v.d.: Discovering block-structured process models from event logs - a constructive approach. In: Colom, J.M., Desel, J. (eds.) Application and Theory of Petri Nets and Concurrency. Lect. Notes Comput. Sc., vol. 7927, pp. 311–329 (2013)
8. Verbeek, H.M.W.: Decomposed replay using hiding and reduction. In: Cabac, L., Kristensen, L., Rölke, H. (eds.) PNSE 2016 Workshop Proceedings. Torun, Poland (June 2016), http://www.win.tue.nl/~hverbeek/wp-content/papercite-data/pdf/verbeek16a.pdf, accepted for publication
9. Verbeek, H.M.W., Aalst, W.M.P.v.d., Munoz-Gama, J.: Divide and conquer. BPM Center Report BPM-16-06, BPMCenter.org (2016), http://bpmcenter.org/wp-content/uploads/reports/2016/BPM-16-06.pdf
10. Verbeek, H.M.W., Buijs, J.C.A.M., Dongen, B.F.v., Aalst, W.M.P.v.d.: ProM 6: The process mining toolkit. In: Proc. of BPM Demonstration Track 2010. vol. 615, pp. 34–39. CEUR-WS.org (2010)

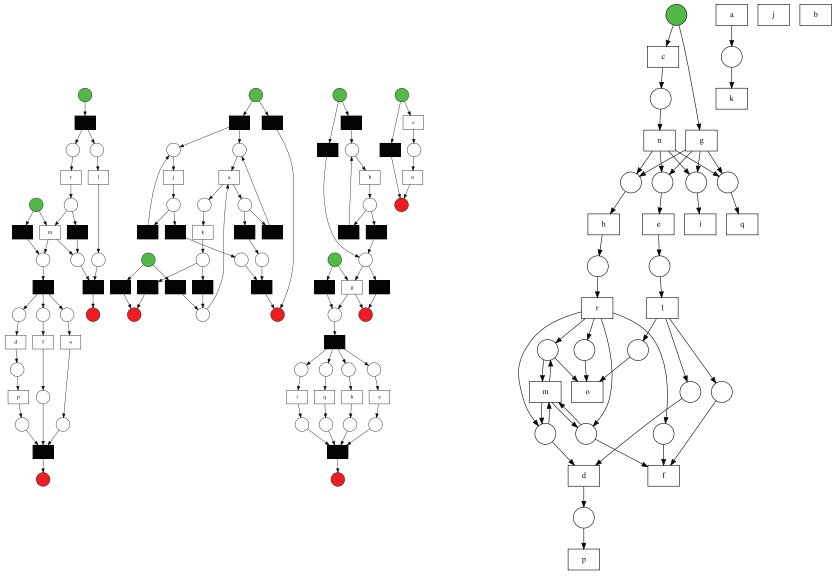

Fig. 8. Petri net discovered from training log 1 using **IM-100** (left) and **HIM-0** (right).

11. Werf, J.M.E.M.v.d., Dongen, B.F.v., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. Fundam. Inf. 94(3-4), 387–412 (Aug 2009), http://dl.acm.org/citation.cfm?id=1662594.1662600
12. Zelst, S.J.v., Dongen, B.F.v., Aalst, W.M.P.v.d.: ILP-based process discovery using hybrid regions. In: Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data, ATAED 2015, Satellite event of the conferences: 36th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2015 and 15th International Conference on Application of Concurrency to System Design ACSD 2015, Brussels, Belgium, June 22-23, 2015. vol. 1371, pp. 47–61. CEUR-WS.org (2015)

A Discovered models

This appendix contains figures for all discovered models. Every discovered model is shown as two accepting Petri nets: one discovered by IM-100 and one discovered by HIM-0. For a trace to be accepted by the model, it needs to be replayed perfectly (costs = 0.0) by both accepting Petri nets. The replay of a trace on the accepting Petri net starts from the initial marking, and ends if (and only if) the final marking is reached. The initial marking includes all places colored green, the final marking all places colored red. Note that some nets have an empty initial marking (no green places) and/or an empty final marking (no red places). As usual, the transitions colored black are silent transitions, while the white transitions are the visible transitions that carry an activity as label.

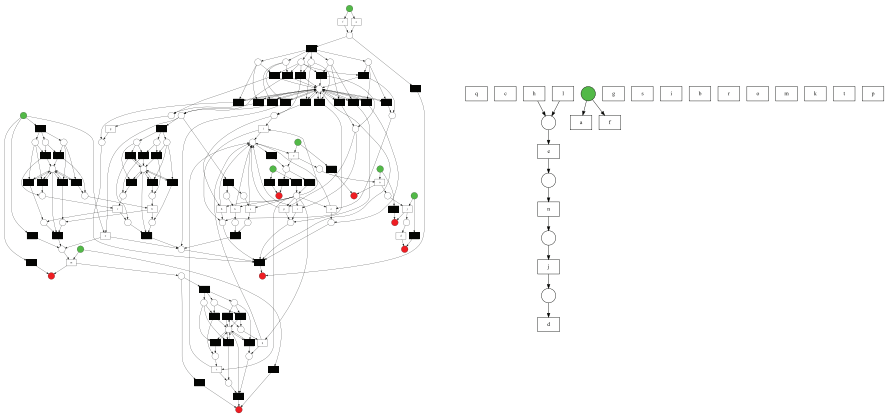

Fig. 9. Petri net discovered from training log 2 using **IM-100** (left) and **HIM-0** (right).

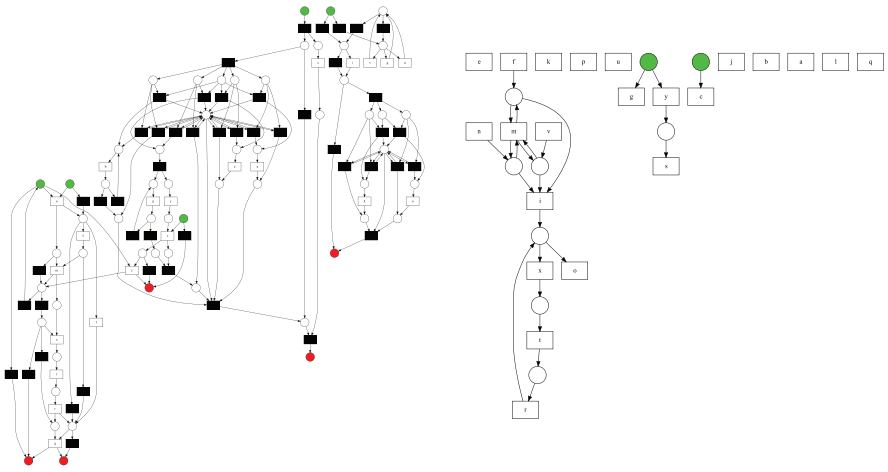

Fig. 10. Petri net discovered from training log 3 using **IM-100** (left) and **HIM-0** (right).

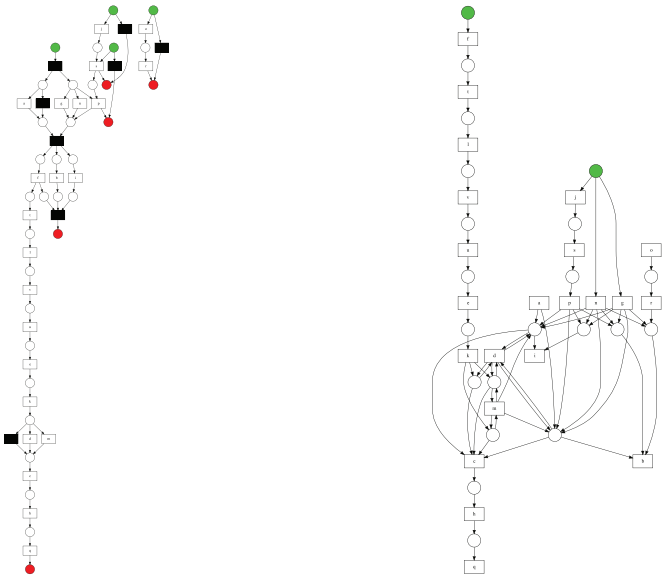

Fig. 11. Petri net discovered from training log 4 using **IM-100** (left) and **HIM-0** (right).

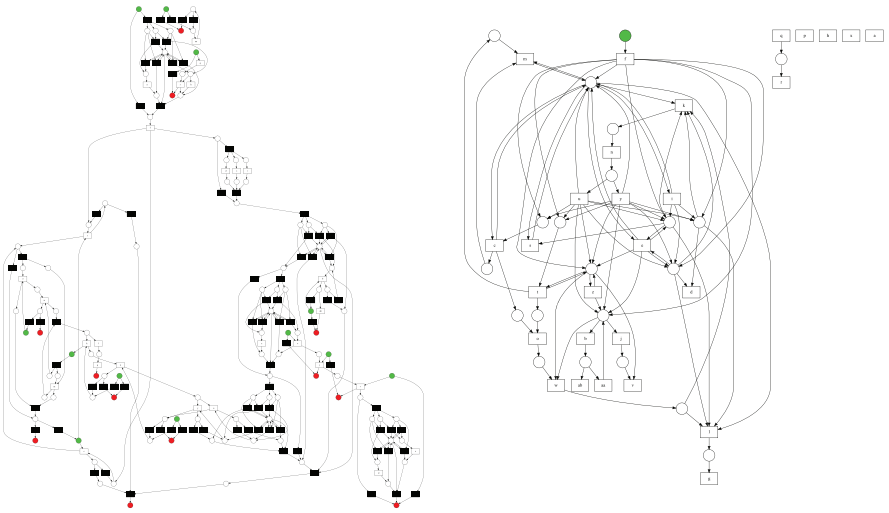

Fig. 12. Petri net discovered from training log 5 using **IM-100** (left) and **HIM-0** (right).

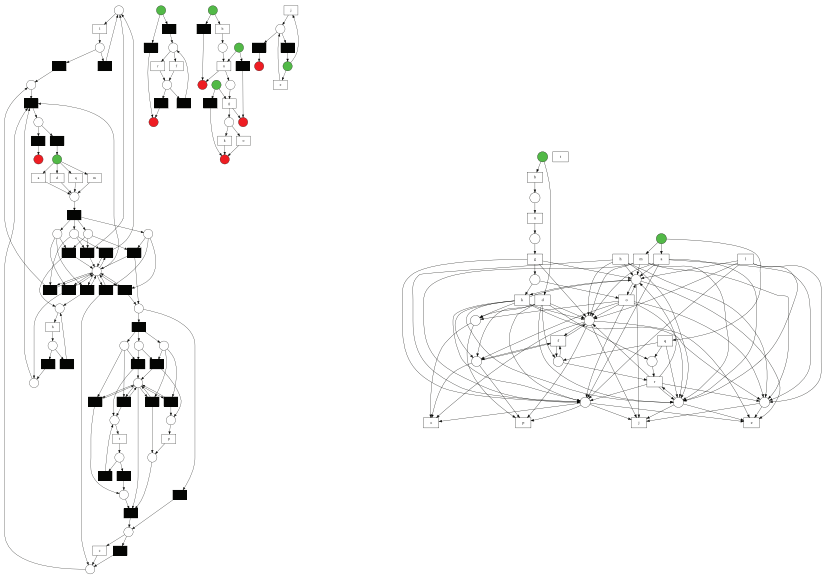

Fig. 13. Petri net discovered from training log 6 using **IM-100** (left) and **HIM-0** (right).

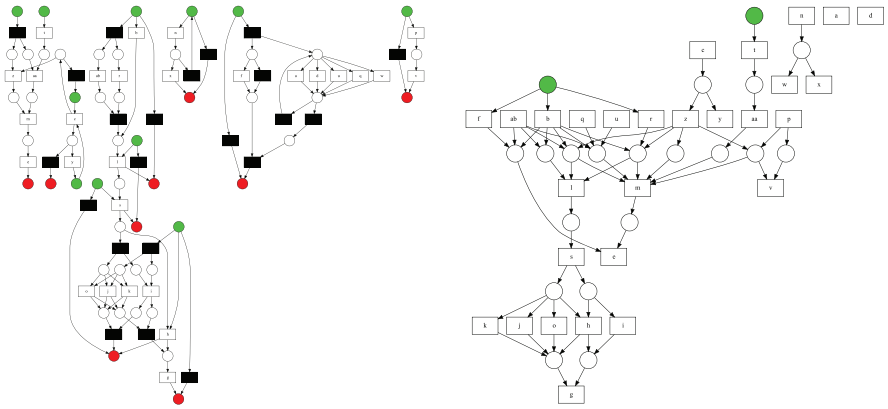

Fig. 14. Petri net discovered from training log 7 using **IM-100** (left) and **HIM-0** (right).

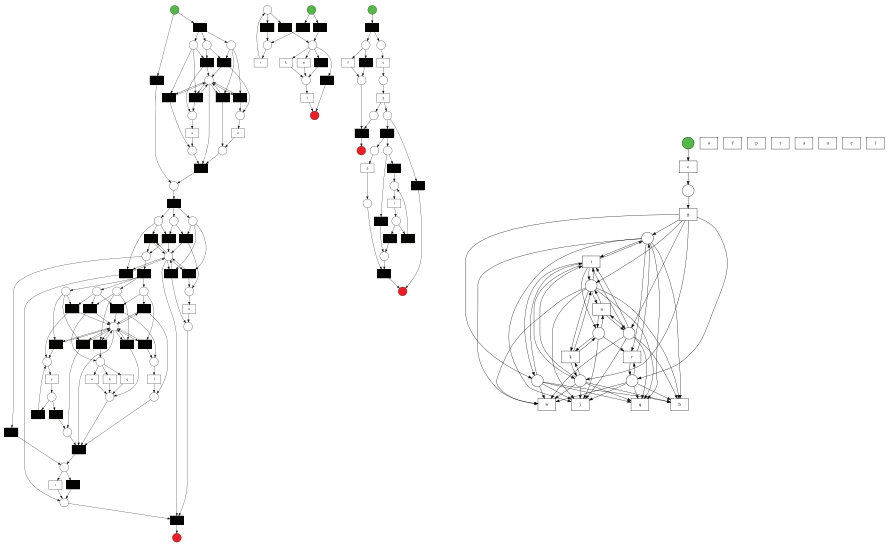

Fig. 15. Petri net discovered from training log 8 using **IM-100** (left) and **HIM-0** (right).

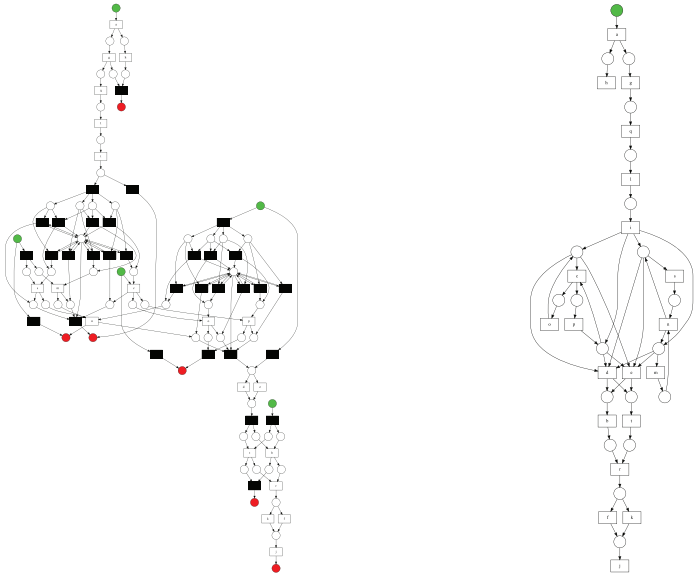

Fig. 16. Petri net discovered from training log 9 using **IM-100** (left) and **HIM-0** (right).

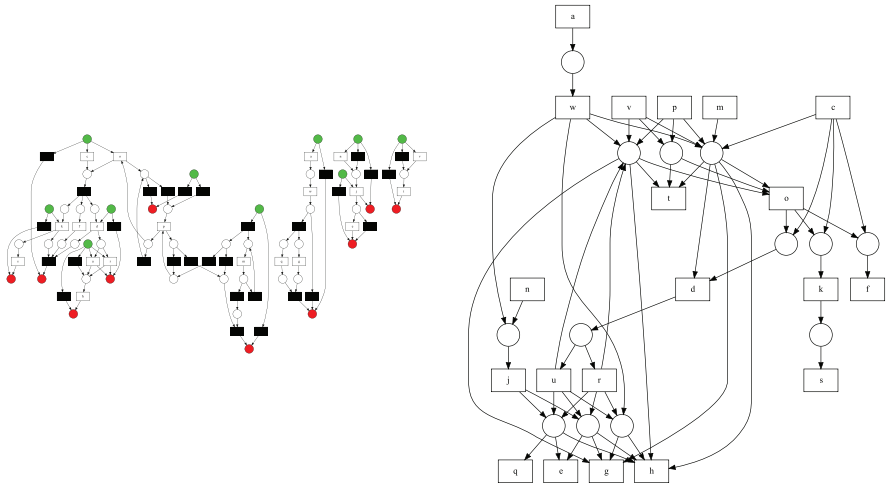

Fig. 17. Petri net discovered from training log 10 using **IM-100** (left) and **HIM-0** (right).