Dynamic Pattern-based Case Filters using Regular Expressions

Thomas Vogelgesang, Janina Nakladal, Jerome Geyer-Klingeberg, and Peyman Badakhshan

Celonis SE, Munich, Germany
Corresponding author: `j.geyerklingeberg@celonis.com`
`https://www.celonis.com/`

Abstract. Process mining allows for a fact-based analysis of business processes by discovering descriptive process models. However, providing an overall view of the process is not enough. To gain valuable insights and identify potentials for process improvement, it is crucial to filter the underlying event log to a subset of cases of interest. Usually this is done by attribute-based filters, e.g. filtering for a specific vendor, production line or time period. However, sometimes the interesting cases are defined by a more complex pattern, e.g. by a sequence of certain events. In this demo, we present a new feature of Celonis that allows the analysts to filter for such complex patterns by defining a regular expression. Due to its integration into the Celonis Process Query Language, it can be applied to arbitrary analysis components. This allows for ad-hoc filtering by experienced analysts as well as pre-defined filters for business users.

Keywords: Process Mining · Regular Expression · Process Discovery.

1 Introduction

Process mining provides business analysts and process owners a fact-based view on their processes, e.g. by discovering a descriptive process model [2]. However, just providing an overall view of all cases is not sufficient. To gain valuable insights into the processes, the users must be able to focus on particular cases of interest. For example, this can be orders with a delayed shipment or customer journeys that resulted in a low customer satisfaction. To understand what goes wrong in the process and identify possible root causes for this, it is crucial to drill-down the event log to the relevant cases.

Usually, these drill-downs are performed along dimensions stored with the data (e.g. region, vendor, etc.) or based on KPIs like throughput time. However, all of them are based on simple attribute values but do not consider the process behavior, such as the flow of activities. For example, a user might be interested in cases where a delivery block has been set but has never been removed afterwards.

In this demo, we present a novel feature of Celonis Process Mining which allows to filter cases based on user-defined process flow patterns. These patterns are defined as regular expressions that are processed by a specific filter operator

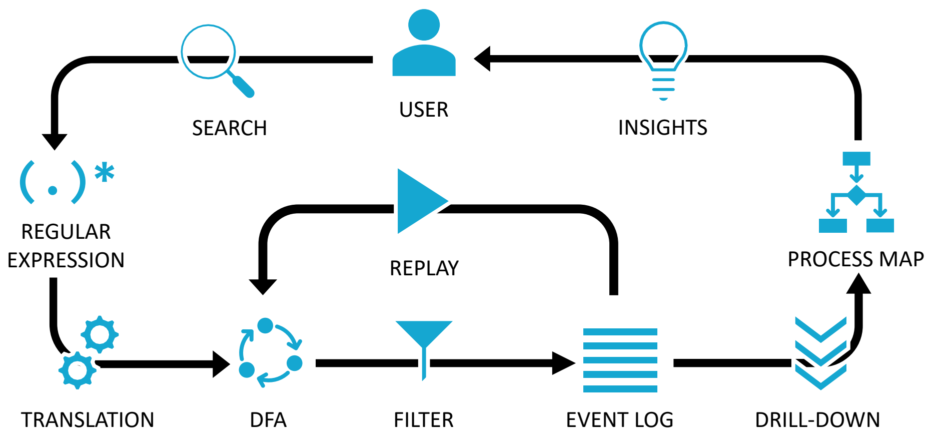

Fig. 1. Concept overview of pattern-based filter.

integrated into the Celonis Process Query Language (PQL). This allows for applying the filter to a specific analysis component (e.g. a table or chart) or even the entire analysis. As shown in this demo, advanced users can also interactively filter the cases by entering a regular expression describing the process flow pattern of interest. This enables them to dynamically explore the processes in depth in order to find potential flaws or undesired behaviors in the process.

In the remainder of this paper, we provide an overview of the pattern-based case filters and its capabilities in Section 2. Section 3 discusses related work and Section 4 describes the presented demo and its setting.

2 Pattern-based Case Filters

The searched behavior is defined as a regular expression – a well-known concept for pattern-matching in programming. In contrast to programming, we do not apply them to strings but to activity sequences. The syntax is inspired by the widely used Perl syntax but has some adoptions to its application to event logs.

Figure 1 gives an overview of our approach. The user-defined regular expression is translated into a deterministic finite automaton (DFA) by applying the Berry-Sethi-algorithm [3] and powerset construction [5]. Then the event log is replayed on the DFA to solve the acceptance problem. All cases not matching the pattern are filtered out and the process map is drilled-down accordingly.

To use the pattern-based case filters in an analysis, the Celonis PQL provides the `match_process_regex(`*`column, pattern`*`)` operator. The first parameter is the activity column in the data model which stores the executed activity for each event in the event log. The second parameter is the regular expression describing the searched pattern. The result of the operator is a new integer column attached to the case table. For each case matching the pattern, the value is 1 while for non-matching cases the value is 0. This allows for the seamless integration of the operator into any PQL filter formula defined in the analysis. Binding the regular expression to a variable allows to dynamically adjust the pattern by selecting pre-defined patterns or by editing it in an ad-hoc fashion in a text-field. The following language constructs can be used to define regular expressions which can also be nested to define complex patterns.

Activities: Activities are the primitives of the regular expressions and are identified by their single-quoted name (e.g. `'Create Invoice'`). If the activity should match multiple activities with similar name, it is also possible to use wildcard matching for activities (e.g. `LIKE 'Create%'`). Instead of the activity name, the `ANY` keyword can be used to match any arbitrary activity.

Sequences: A sequence defines a directly follows relationship between two regular expressions. Usually, this is expressed by concatenating two symbols of the regular expression. However, as we have activity names as primitives, this would result in a very confusing syntax. Therefore, we use `>>` to express a sequence of two activities (e.g., `'Create Invoice' >> 'Clear Invoice'`).

Choices: To choose between multiple valid regular expressions, we separate the options with `|` (e.g. `'Change Quantity' | 'Change Price'`). Alternatively, we can also give a comma-separated list of activity names surrounded by squared brackets to define a choice (e.g. `['Change Quantity', 'Change Price']`). While the former is applicable to any regular expression, the latter only accepts activities as the primitives. Though the latter allows to invert the set of activities. For example, `[ ! 'Change Quantity', 'Change Price']` matches all activities except *Change Quantity* and *Change Price.*

Quantifiers: With quantifiers (*, `+`, `?`) one can declare that a regular expression should not match only once, but with a certain cardinality. A regular expression must be surrounded by brackets when applying a quantifier to it. While `('Create Invoice')*` matches any arbitrary number of occurrence, `('Create Invoice')+` requires the activity *Create Invoice* to occur at least once. With `('Create Invoice')?` one can mark it as optional, i.e. *Create Invoice* occurs once or not.

Match at start/end: To declare that a regular expression must match at the start (i.e. from the first event of the trace), we can prepend a circumflex (^) to it. Analogously, we can append `$` to the regular expression if it must match at the end. If the regular expression is not marked to match at the start / end, we implicitly prepend / append `(ANY)*` to the regular expression for convenience in order to match the pattern anywhere within a trace. Please note that this may interfere with other quantifiers. For example, the regular expression `('Create Invoice')?` will also match traces having two or more consecutive *Create Invoice* activities, as the additional activities are consumed by the implicitly added `(ANY)*`.

Due to the usage of activities as primitives, regular expressions may become quite long. To reduce the user's effort to construct complex expressions and to improve their readability, regular expressions can be assigned to an alias which then can be referenced in other regular expressions. Note that the regular expressions must be comma-separated and the last regular expression must not have an alias, e.g. `'Change Quantity' | 'Change Price' AS A, 'Create Invoice' AS B, A >> (ANY)* >> A >> B`

Instead of the activity column, it is also possible to use any other string column of the activity table. This enables the user to define search patterns not only over the activities but also over other event attributes like the resource. Due to its integration into PQL, also columns of other type can be converted into strings so effectively each column of the activity table can be used.

The pattern-based case filters are available in the Celonis Intelligent Business Cloud (IBC) and in Celonis Process Mining 4.5. It is accessible to thousands of users and actively used by various Celonis customers.

The Celonis Academic Alliance[1] offers free licenses for academic purposes.

3 Related Work

Filtering event logs by a specific pattern of behavior can also be achieved by other approaches. For example, LTL Checker [1] allows the user to define such patterns in linear temporal logic and filter the event log to cases matching the LTL program. However, regular expressions are usually less verbose than LTL programs. Furthermore, regular expressions are a well-known and widely used concept (e.g. in programming) which makes it easier to adopt by users.

Similar results can be achieved by drawing the behavioral pattern as process model, then checking its conformance [4] with the event log and finally filtering the event log to all conforming cases. However, conformance checking algorithms (especially alignment-based approaches) are computational expensive. While creating DFAs from regular expressions is highly complex too, the evaluation of traces on DFAs has linear run-time. To our experience, the fast evaluation usually outweighs the initial effort, especially in real-world scenarios with huge event logs. Besides, drawing process models does not integrate well with query languages.

4 Demo Description

In this demo, we show a ready-to-use analysis running in the Celonis IBC. The components of the analysis are configured to apply a filter containing the `process_match_regex` operator. The regular expressions used in the component filters are bound to a global variable to share the same regular expression among different components. To change this variable one can enter a new regular expression into one of the text fields. Alternatively, it is also possible to select a pre-defined regular expression from the drop-down menu on the left-hand side of the analysis. This shows that the new filter is capable to be used either by power users that interactively filter the cases by defining a regular expression ad-hoc or by less experienced users who select some prepared filters from a menu.

[1] `https://www.celonis.com/academic-signup`

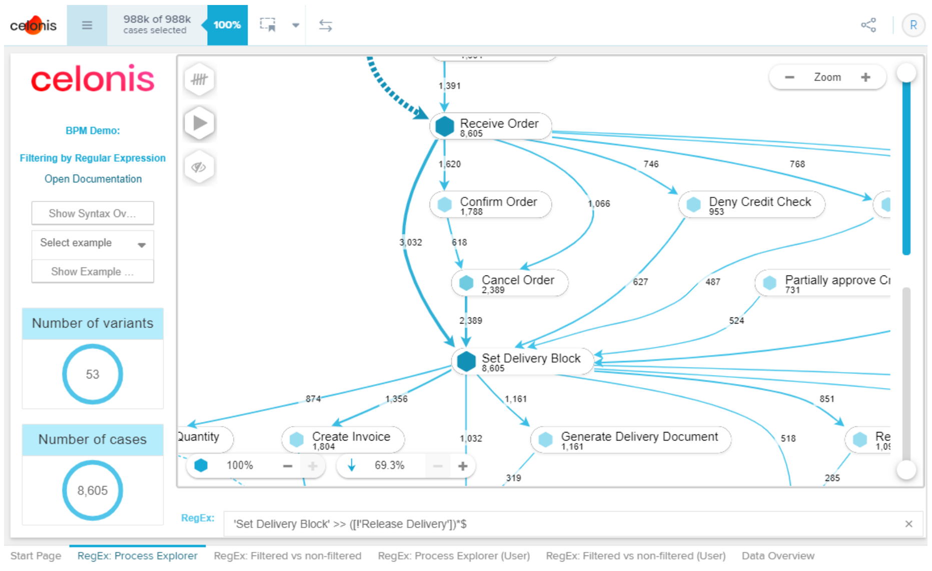

Fig. 2. Filtered process map showing cases with unresolved delivery blocks.

For this demo, we use a demo data set of a standard order-to-cash (O2C) process. The data consists of more than 6,000,000 events of almost 1,000,000 cases with almost 500 different traces.

Figure 2 shows a screenshot of the demo. The process map is filtered to all cases with an unresolved delivery block. This is done by the regular expression `'Set Delivery Block' >> ([! 'Release Delivery'])*$` entered in the text field below. As indicated by the numbers on the left-hand side, the filtered data comprises 8,605 cases forming 53 unique variants (traces). A screencast showing a short walk-through of the demo is available online[2].

References

1. van der Aalst, W., de Beer, H.T., van Dongen, B.: Process mining and verification of properties: An approach based on temporal logic. In: R. Meersman et. al (ed.) On the Move to Meaningful Internet Systems 2005. LNCS, vol. 3760, pp. 130–147. Springer (2005)
2. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)
3. Berry, G., Sethi, R.: From regular expressions to deterministic automata. Theor. Comput. Sci. **48**(3), 117–126 (1986)
4. Carmona, J., van Dongen, B.F., Solti, A., Weidlich, M.: Conformance Checking - Relating Processes and Models. Springer (2018)
5. Rabin, M.O., Scott, D.S.: Finite automata and their decision problems. IBM Journal of Research and Development **3**(2), 114–125 (1959)

[2] `https://bit.ly/2UmC3G5`