Note: You can zoom in/out with ‘Z’+‘I’/‘Z’+‘O’.

Contents

1 Grammar . . . . . .

2 Quantifiers . . . . . .

3 Keywords / Actions . . . . . .

4 Terminals . . . . . .

5 Examples / Implementations . . . . . .

6 Limitations / Assertions . . . . . .

7 Output . . . . . .

8 Final Notes: . . . . . .

8.1 Bonuses . . . . . .

8.2 How to Use >_next(log): . . . . . .

1 Grammar

```
⟨rules⟩             ::= ⟨rules⟩ ⟨rule⟩ ‘;’ ⟨comment⟩?
                      | ⟨ε⟩ ⟨comment⟩?

⟨rule⟩              ::= ‘if’ ⟨identifier⟩ ⟨expr⟩ ‘then’ ⟨action⟩
                      | ‘if’ ⟨rt_expr⟩ ‘then’ ⟨action⟩

⟨expr⟩              ::= ‘#’ ⟨identifier⟩ ⟨equality⟩ ⟨value⟩
                      | ‘@’ ⟨identifier⟩ ⟨equality⟩ ⟨value⟩

⟨rt_expr⟩           ::= ‘!’ ⟨identifier⟩ ⟨equality⟩ ⟨value⟩

⟨action⟩            ::= ‘skip’ ⟨identifier⟩
                      | ‘insert’ ⟨identifier⟩ ‘(’ ⟨attributes⟩ ‘)’
                      | ‘insert’ ⟨identifier⟩
                      | ⟨parallel_list⟩
                      | ⟨sequential_list⟩

⟨parallel_list⟩     ::= ⟨parallel_item⟩
                      | ⟨parallel_next⟩

⟨parallel_item⟩     ::= ⟨identifier⟩ ‘+’ ⟨identifier⟩

⟨parallel_next⟩     ::= ‘+’ ⟨identifier⟩
                      | ⟨parallel_next⟩ ‘+’ ⟨identifier⟩

⟨seq_list⟩          ::= ⟨seq_item⟩
                      | ⟨seq_next⟩

⟨seq_item⟩          ::= ⟨identifier⟩ ‘-’ ⟨identifier⟩

⟨seq_next⟩          ::= ‘-’ ⟨identifier⟩
                      | ⟨seq_next⟩ ‘-’ ⟨identifier⟩

⟨attributes⟩        ::= ⟨attributes⟩ ‘#’ ⟨identifier⟩ ⟨value⟩
                      | ⟨attributes⟩ ‘#’ ⟨identifier⟩ ⟨distribution⟩
                      | ‘#’ ⟨identifier⟩ ⟨value⟩
                      | ‘#’ ⟨identifier⟩ ⟨distribution⟩
```

⟨*distribution*⟩ ::= ‘?N’ ‘[’ ⟨*value*⟩ ⟨*value*⟩ ‘]’
| ‘?U’ ‘[’ ⟨*value*⟩ ⟨*value*⟩ ⟨*value*⟩ ‘]’

⟨*identifier*⟩ ::= [a-zA-Z_][a-zA-Z0-9_]*

⟨*value*⟩ ::= [-]?[0-9]+[‘.’[0-9]+]?

⟨*equality*⟩ ::= ‘<=’ | ‘>=’ | ‘<’ | ‘>’ | ‘==’ | ‘!=’

⟨*comment*⟩ ::= ‘//’ [^\n]*

⟨ε⟩ ::= ‘’

2 Quantifiers

- The ‘#’ (THIS_TOK) is a quantifier which means *this* event’s attribute value. Hence, `A#duration` means the duration of A.
- The ‘@’ (ACCUMULATIVE_TOK) is a quantifier which means the accumulative attribute value *at* that event (inclusive).

  Hence, `C@duration` means the total duration of the instance after event C.
- The ‘!’ (RUNNING_TOTAL_TOK) is a quantifier which means the total running attribute value during the instance. This quantifier does not have a event identifier attached to it.

  Hence, `if !duration > 15 then skip D;` means that if at *any* point during the instance, if the running total for duration is > than 15, then `skip` D. Note: if D has already happened by the time the condition becomes true, then the rule is ignored.

3 Keywords / Actions

For the more intricate details, read the "Examples/Implementations" section further below.

- The keywords `if` and `then` are self-explanatory.
- `skip`
  - This action will skip an event that otherwise *would* have happened.

    Hence, imagine `if A#duration > 15 then skip B;` with BPM 2 (shown below). If an instance’s `A#duration` was greater than 15, then the rule will only be ‘applied’ if the instance was going to choose ‘B’ in the xor gate. Ie, if the instance took the path "_Start->A->C->_End" then the rule will not be applied regardless of A’s duration.
  - Proceeding events’ timestamps will be updated accordingly, they will be ‘shifted’ to the left by the skipped event’s duration. Waiting times will be preserved.
  - If the skipped event was part of a parallel gateway, then the proceeding events’ timestamps will be updated accordingly *if* the skipped event was the last event to finish in the gateway, otherwise nothing is changed - of course, this is different per instance.
- `insert`

- This action will insert the given event *right after* the event that triggered it.

  Hence, `if A#duration > 515 then insert D;` will insert D right after A. If A was in a parallel gateway, then "A->D" is now part of the same gateway.
- Proceeding events' timestamps will be updated accordingly, they will be 'shifted' to the right by the inserted event's duration. Waiting times will be preserved.
- If the inserted event was part of a parallel gateway, then the proceeding events' timestamps will be updated accordingly *if* the inserted event was the last event to finish in the gateway (for that instance).

- `+ // TO_PARALLEL_TOK`
  - This action means for all the events in the given chain, turn them into parallel events.

    Hence, `if A#resourceCost > 10 then B+C+D;` means make the events B,C and D parallel.
  - The chain needs to be such that the events are (directly) in series. For example, looking at BPM 1 below, `... B+C` is allowed, so is `... B+C+D` but `... C+B` and `... B+D` is not.

- `- // TO_SERIES_TOK`
  - This action means for all the events in the given chain, turn them into events in series.

    Hence, `if A#resourceCost > 10 then B-C-D;` means make the events B,C and D series. Note: order is important, ie; if you specify `... then C-D-B` then that will be the order taken, where waiting time will be preserved.

  - Likewise, you can only put events in series if they are in parallel, and *all* events must be transformed to series. Eg, looking at BPM 4, `... D-E-F` is allowed, but `.. D-E` is not (because F is missing).

4 Terminals

- <rules> is your list of rules separated by the ';' token.
- <distribution> is a terminal which allows a user to define an attribute distribution type when inserting an event, the available ones are;
  - `?N (mu sigma)`: A normal distribution, with mean μ and standard deviation σ.
  - `?U (min_val max_val step_size)`: A uniform distribution with the given parameters.
  - Likewise, there is also a "Fixed Distribution" type, but is not actually a <distribution> terminal.
  - For example, if you are inserting the event B, with the attributes a1, a2, a3, a4, a5, you could write the rule;

    ```
    .. insert B (#a1 ?N(20 3) #a2 ?U(10 20 0.2) #a3 10);
    ```

This means B will be inserted with attributes a1 following a normal distribution, a2 with a uniform distribution, a3 will always be 10, and finally a4 and a5 will be set to 0.

Do note; the attribute `#occurred` for the inserted event will always be set to 1 if the event was inserted (otherwise it will be 0). Likewise, `#start_time` and `#end_time` will be created dynamically (with `#start_time` being right after the previous event's `#end_time`). You do not have to do this yourself, these will happen automatically.

- <identifier> is any identifier using the letters from the Roman alphabet, underscores and numbers. It must start with a letter or underscore. These are used for event names and for attribute names.
- <value> is any number, float or integer (positive or negative).
- <comment> is a normal comment, everything after '//' is ignored (until a newline).

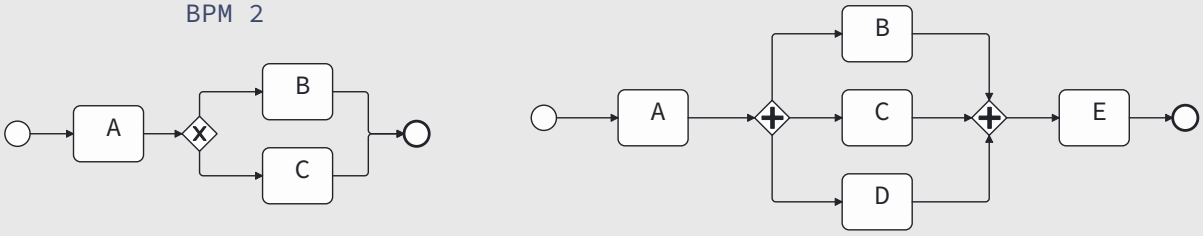
5 Examples / Implementations

Suppose we had the following BPMNs;

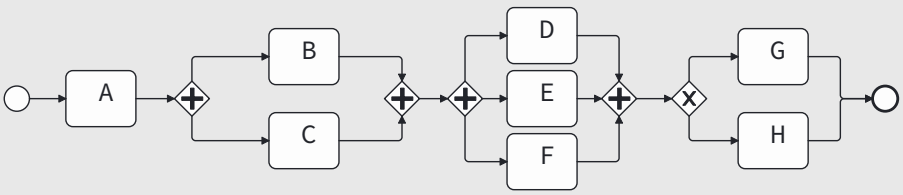
BPM 1

BPM 3

BPM 2



BPM 4



And suppose we had the following rules, where the comments `... // A:1,3; R:2,4` means the >_next(log) parser accepted the rule for BPMs 1 and 3, and rejected the rule for BPMs 2 and 4;

```
1  if A#duration > 515 then skip B;     // A:1,2,3,4
2  if B#duration > 515 then insert K;   // A:1,2,3,4
3  if A#duration > 515 then insert D;   // A:2; R:1,3,4
4  if A#duration > 515 then B+C;        // A:1; R:2,3,4
5  if A#duration > 515 then C+B;        // R:1,2,3,4
6  if A#duration > 515 then B-C;        // A:4; R:1,2,3
7  if A#duration > 515 then C-B;        // A:4; R:1,2,3
8  if A#duration > 515 then D-F-E;      // A:4; R:1,2,3
9  if A#duration > 515 then D-F;        // R:1,2,3,4
10 if !duration > 1000 then B-C;        // A:4; R:1,2,3
```

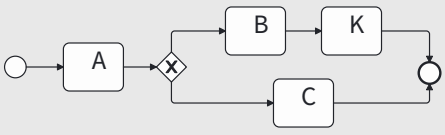
1: `if A#duration > 515 then skip B;  // A:1,2,3,4`

- BPMs 1; the output works as expected. If the rule's condition evaluates to true for that instance, then B will be skipped and all of the proceeding event's timestamps will be shifted accordingly, where waiting times are preserved (you can assume waiting time are always preserved).
- BPMs 2; the rule will only be applied if the instance was going to 'choose' B at the xor gate, otherwise (even if A's duration > 15) then nothing will happen.
- BPMs 3,4; B will be skipped if the rule's condition evaluates to true. And if B was the last event to finish in the parallel gateways, then the proceeding event's timestamps will be shifted to the left (otherwise nothing happens to their timestamps). Of course, this may be different per instance.

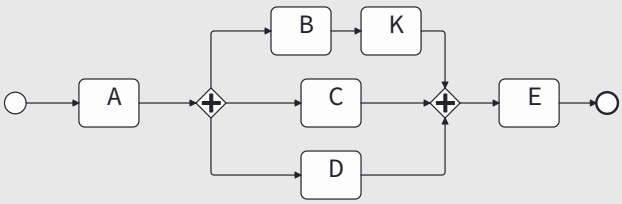
2: `if B#duration > 515 then insert K;  // A:1,2,3,4`

- BPMs 1; the output works as expected. If the rule's condition evaluates to true for that instance, then K will be inserted right after B and all of the proceeding event's timestamps will be shifted accordingly.
- BPMs 2; the rule will only be evaluated if the instance 'chose' B at the xor gate, otherwise the rule isn't even evaluated. Likewise, if the instance chose B, and B's duration was $>$ 15 then K is inserted right after B, with all the proceeding events' timestamps shifted accordingly (see BPM 5 below).
- BPMs 3,4; K will be inserted after B (in series *but* still in the parallel gateway, see BPM 6 below) if the rule's condition evaluates to true. And if K was the last event to finish in the parallel gateway, then the proceeding event's timestamps will be shifted to the right (otherwise nothing happens to their timestamps). Of course, this may be different per instance.

BPM 5



BPM 6



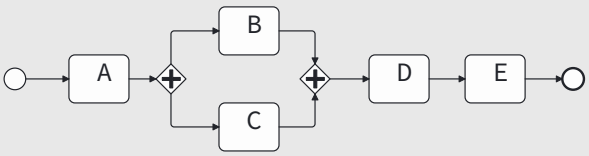
3: `if A#duration > 515 then insert D;  // A:2; R:1,3,4`

- BPMs 2; the output works as expected and works as explained above.
- BPMs 1,3,4; the >_next(log) parser will reject this rule as D already exists in th BPM.

4: `if A#duration > 515 then B+C;  // A:1; R:2,3,4`

- BPMs 1; The output works as expected, if the rule's condition evaluates to true, then the instance's BPM will look like BPM 7 seen below. Of course, the proceeding event's timestamps will be shift to the left accordingly (with waiting time preserved)
- BPMs 2,3,4; the >_next(log) parser will reject this rule as events B and C are not (directly) in series with each other.

BPM 7



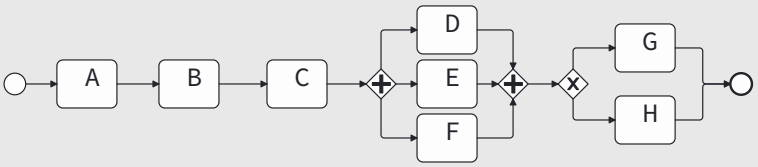

```
5: if A#duration > 515 then C+B;          // R:1,2,3,4
```

- BPMs 1,2,3,4; the >_next(log) parser will reject this rule as events B and C are not (directly) in series with each other.

```
6: if A#duration > 515 then B-C;          // A:4; R:1,2,3
```

- BPMs 4; The output works as expected, if the rule's condition evaluates to true, then the instance's BPM will look like BPM 8 seen below. Of course, the proceeding event's timestamps will be shift to the right accordingly (with waiting time preserved)
- BPMs 1,2,3; the >_next(log) parser will reject this rule as events B and C are not in parallel with each other - note how for BPM 3, the event D is not in the chain.

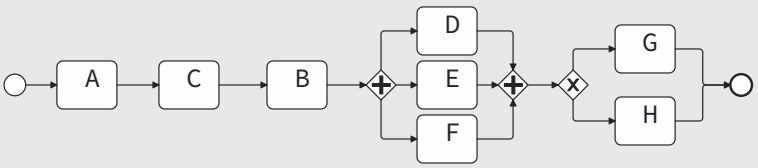
BPM 8




```
7: if A#duration > 515 then C-B;          // A:4; R:1,2,3
```

- BPMs 4; Like before, the output works as expected. However, note the order. If the rule's condition evaluates to true, then the instance's BPM will look like BPM 9 seen below. The original start_time of C is kept and everything is shifted accordingly.
- BPMs 1,2,3; the >_next(log) parser will reject this rule as events B and C are not in parallel with each other - note how for BPM 3, the event D is not in the chain.

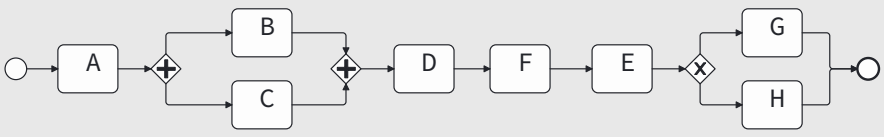
BPM 9




```
8: if A#duration > 515 then D-F-E;        // A:4; R:1,2,3
```

- BPMs 4; Like before, the output works as expected. If the rule's condition evaluates to true, then the instance's BPM will look like BPM 10 seen below.

BPM 10



```
9: if A#duration > 515 then D-F;        // R:1,2,3,4
```

- Likewise, >_next(log) will reject this rule for all of the BPMs for the same reasons described above.

```
10: if !duration > 1000 then B-C;       // A:4; R:1,2,3
```

- BPMs 1, of course >_next(log) will accept this rule for BPM 1. Remember, this rule means "if at any point during this instance, if the running duration value is > 15 then `B-C`". Therefore, if the rule's condition evaluates to true *after* B or C has happened, then this rule is ignored and is not considered to be applied. Likewise, if the condition evaluates to true *before* B or C, then the instance's BPM adapts into BPM 8.
- Likewise, >_next(log) will reject this rule for all of the other BPMs for the same reasons described before.

Note: it's assumed that rules don't 'overlap' each other - ie, if rule x is triggered, it has no impact on whether rule y is triggered. However, in >_next(log), rules are done in sequential order and they *will* be applied with each other. Hence, suppose we have the rules for BPM 1;

```
if A#duration > 515 then insert F (#duration ?N(400 50));
// ^^ Applied like normal
if B@duration > 1000 then skip D;
// ^^ Is much more likely if the first rule was triggered.
```

Again, this was assumed to not happen, but nonetheless I think this is desirable (thus allowing the user to create rules that may trigger other rules).

6 Limitations / Assertions

Of course, the software tool >_next(log) possesses several limitations/assertions that should be taken into consideration before using it;

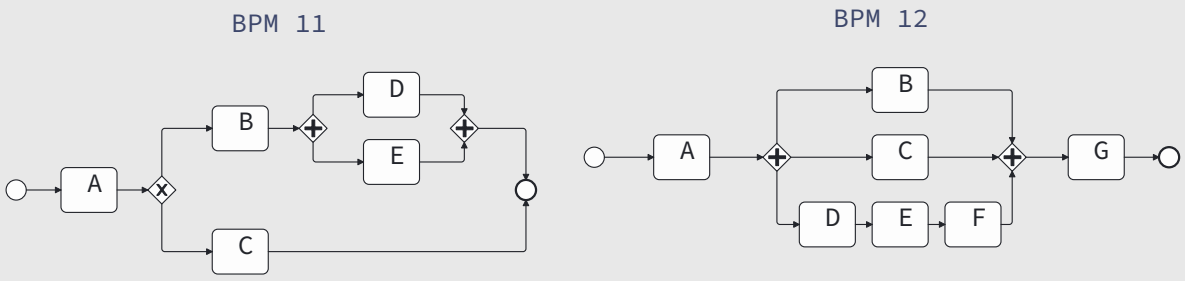
1. **Non-overlapping rules assumption:** As stated above, >_next(log) assumes that the rules defined by the user do not overlap. If there are overlapping rules, unpredictable or ambiguous outcomes may occur, potentially affecting the accuracy and reliability of the generated logs.
2. **Limited compatibility with input files:** While >_next(log) is designed to work with any input .bpmn (Business Process Model and Notation) files, it has been primarily created and tested using files generated with the bpmn.io tool. Similarly, the software should function with any .mxml files (eXtensible Markup Language for Mining), but it has been predominantly tested with files produced by the bimp tool. It is essential to note that when using other BPMN or generative MXML tools, there may be minor issues - however, if problems do arise, they *should* be quite easy to fix in the code.
3. **Assumption about Start and End events:** >_next(log) assumes the standard output format of Start and End nodes/events generated by bpmn.io. Consequently, the software references these events as `_Start` and `_End`. It is important to adhere to this naming convention.

*Therefore, please note: when using bpmn.io, do **not** change the Start/End events default names!*

Likewise, do not name any events "`_Start`" or "`_End`".

4. **Absence of looping in BPMs:** The tool assumes that there are no loops within the Business Process Models (BPMs). If the BPM contains loops, >_next(log) may not generate accurate adapted logs or encounter difficulties in processing the model.
5. **Recognition of specific gateways:** >_next(log) only recognizes parallel and exclusive (XOR) gateways. It does not support other types of gateways, which may limit the tool's applicability to BPMs that utilize different gateway types.
6. **No gateway nesting assumption:** >_next(log) assumes that there is no nesting of gateways within the BPMs. This includes the presence of series events nested within gateways.

   Incorrectly structured BPMs with gateway nesting, as illustrated in BPMs 11 and 12, may result in unexpected behaviour or errors.



   While the software may still function in some cases, its performance and accuracy cannot be guaranteed when encountering nested gateways as its not been tested.

7. **Hardcoded attributes:** >_next(log) includes two hardcoded attributes, namely `#occurred` and `#duration`. Consequently, if these attributes are already defined in the .mxml logs, conflicts may arise. It is important to avoid using these attributes in the .mxml files to prevent any issues during log generation.
8. **Dependency on specific attributes for duration calculation:** The `#duration` attribute in >_next(log) is determined by calculating the difference between the `#start_time` and `#end_time` of each instance defined in the uploaded .mxml logs. Therefore, it is essential to ensure the presence of these two attributes in every .mxml log for accurate duration calculations.
9. **Event definitions:** As seen in the grammar defined earlier, the naming of events within the BPM or in the uploaded .mxml files must adhere to specific naming conventions:

   - Event names should consist of only letters, underscores, and numbers.
   - Event names must start with a letter or an underscore.
   - Hence, event names cannot include spaces, brackets, or any special characters.

By considering the above limitations, users of >_next(log) can effectively assess its applicability, and avoid potential issues when generating logs for adaptive business processes.

7 Output

The output of >_next(log) is a .csv file that includes the following components;

1. **Attribute values:** The file has column headers that list all the events present in the business process model and their associated attributes. These headers facilitate easy identification and understanding of the events and their corresponding attributes.
2. **Instance path:** The .csv file describes the path taken by each instance within the business process. Ie, it outlines the sequence of events followed by each (possibly adapted) instance.
3. **Rule definitions:** For each user-defined rule, a corresponding column is included in the .csv file. The column name represents the rule definition. Each cell in the rule column indicates whether the rule was applied or not for a particular instance. If the rule was applied, then the cell value is 1; otherwise, it is 0.

   It's important to note that even if a rule's condition evaluates to true, it does not guarantee that the rule was applied. For example, in BPM 2, if a specific instance's path was '_Start->A->C->_End' and the rule stated `if A#duration > 515 then skip B;` the rule column would still have a value of 0, regardless of A's duration exceeding 515.
4. **"_Label" column:** The "_Label" column in the .csv file represents a binary representation of all possible combinations of applied rules for each instance. This column provides a concise summary of the rules that were actually applied to the instances, allowing for easy identification of patterns and correlations between different rules.

   Likewise, this allows for seamless integration into ML.log.
5. **Save original:** Finally, >_next(log) also provides the option for users to save the original (unadapted) logs, primarily for debugging and traceability purposes.

8 Final Notes:

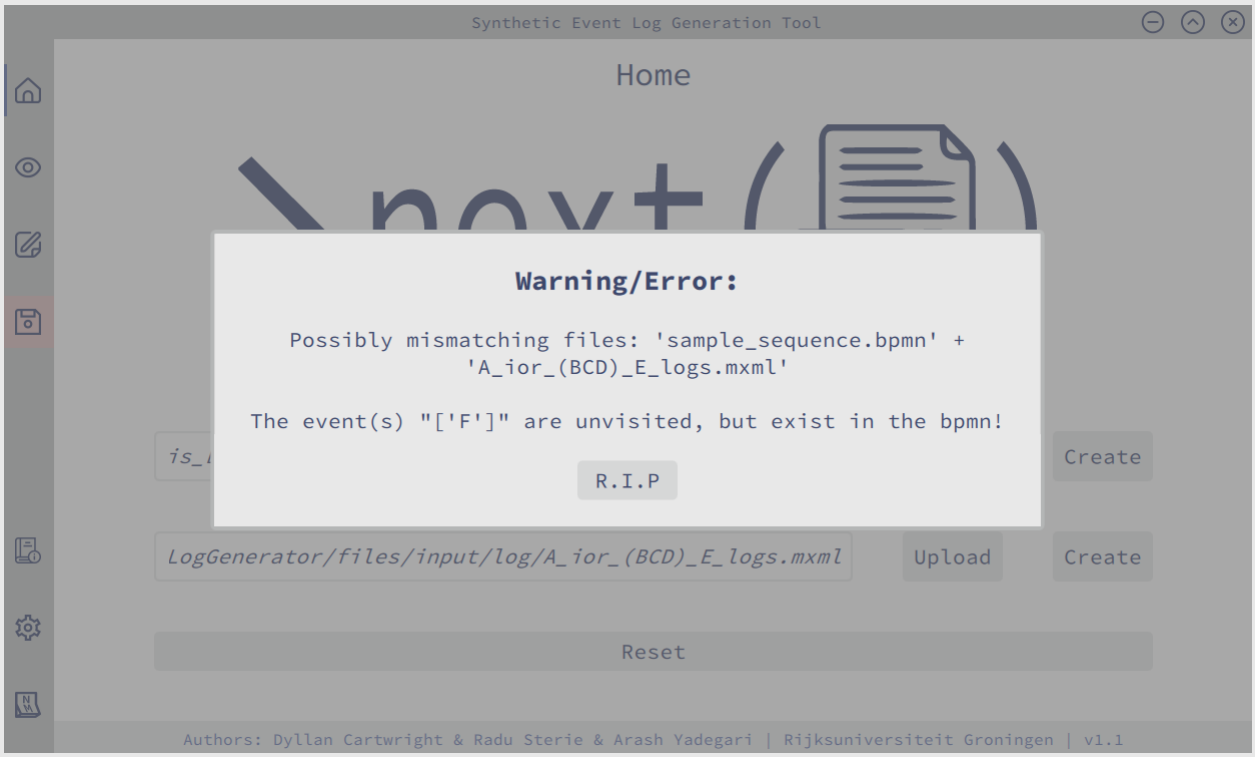
8.1 Bonuses

One of the main goals of this project was that it must be user-friendly, intuitive, and should provide a seamless automated process for generating logs for adaptive processes. Hence, the application was designed to streamline this process and minimize the need for manual intervention or input. >_next(log) not only meets but also exceeds these expectations for several reasons:

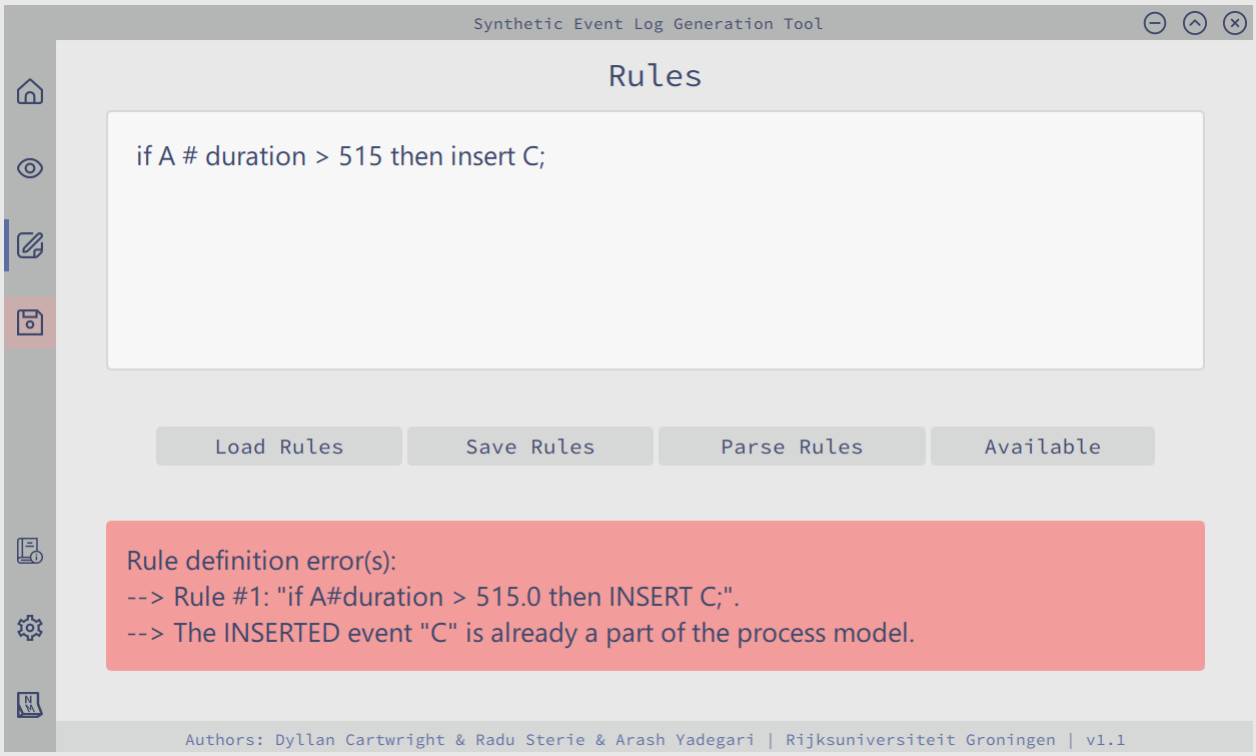
1. **Easy file uploading:** Uploading .mxml and .bpmn files is a straightforward process that only requires a few clicks.

   Additionally, the tool provides convenient buttons that quickly link users to the bimp and bpmn.io tools, simplifying the file selection and uploading process.
2. **Built-in BPM Viewer:** >_next(log) includes a built-in BPM viewer, allowing users to effortlessly check the uploaded BPM when generating rules. This feature enhances usability and provides users with a visual representation of the process model.

3. **Extensive error checking:** >_next(log) incorporates error checking mechanisms;
    - For instance, when uploading files, the tool performs compatibility checks to ensure the files match. If any inconsistencies are detected, helpful warnings are displayed to guide users (as seen below).



    - Similarly, the parser in >_next(log) is adept and robust at verifying rule accuracy, providing informative messages when users make errors. This proactive error checking enhances the user experience and helps users avoid mistakes.



4. **Expanded functionality:** >_next(log) has expanded its functionality significantly beyond the initial project description;
    - Originally, the tool only required the `skip` and `insert` actions. However, it now includes additional actions such as `+` (to parallel action) and `-` (to series action).

- Moreover, time shifting capability has been added (with preserved waiting times).
- Initially, "Cycle Time" and event durations were the only expected attributes available to create rules with, but now users can utilize any available attribute. Likewise, multiple quantifiers, including `#`, `@`, and `!`, have been incorporated to further enhance flexibility in rule creation.

  Note: to create a rule using an instance's "Cycle Time", you could use the following; `if _End@duration ... then ...`.

5. **Exemplary UI**: The user interface (UI) of >_next(log) probably surpasses expectations. Its intuitive design and user-friendly layout contribute to a positive user experience. The UI was crafted to provide users with easy navigation, clear instructions, and an overall efficient workflow.

8.2 How to Use >_next(log):

- **Home Page**: On this page, users can easily upload their .bpmn and .mxml files. This page serves as the starting point for the log generation process, allowing users to select the necessary files for adaptation.
- **View Page**: This page provides users with a visual representation of their uploaded BPM. It enables users to inspect and review the structure and flow of the process model, aiding in the understanding of the process when creating rules.
- **Rules Editor Page**: On this page, users can create their rules for log adaptation. The intuitive interface allows users to define rules based on specific conditions and actions. After creating the rules, users can click 'Parse Rules' to verify their validity. By clicking 'Available', users can view all the defined events and attributes to facilitate rule creation.
- **Save Page**: This page is where users can generate and save the adapted logs. Users can specify the desired output file name and save the adapted logs as a .csv file. Additionally, users have the option to save the "original" unadapted logs as a separate .csv file, providing traceability and allowing for debugging purposes if needed.
- **Settings Page**: This page offers users the ability to customize certain preferences according to their requirements. Users can define the default folder location for file uploads. If the default folder location setting is disabled, >_next(log) will remember the last folder used for uploading files and will open there.
- **The Switch Icon**: The Switch Icon allows users to toggle between >_next(log) and ML.log, another tool within the software. This feature provides users with flexibility and the ability to easily switch between the two applications.

Authors:
Dyllan Cartwright; Radu Sterie; Arash Yadegari
(University of Groningen)
