Reactive and Event Based Systems
Lecture 3: DCR Semantics and Process Mining

Tijs Slaats

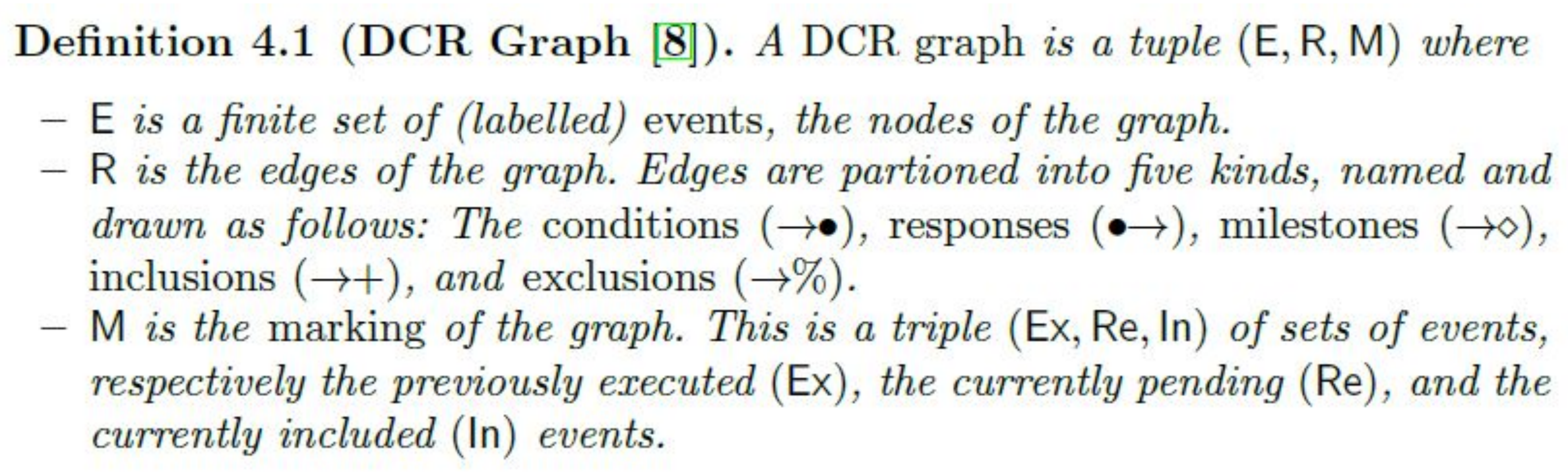
Monday 6th of December 2021

UNIVERSITY OF COPENHAGEN
DEPARTMENT OF COMPUTER SCIENCE

Overview

- **Semantics of DCR Graphs**
- Process Mining
    - Process Conformance
    - Process Discovery
- Assignment 1 Part b

DCR Semantics



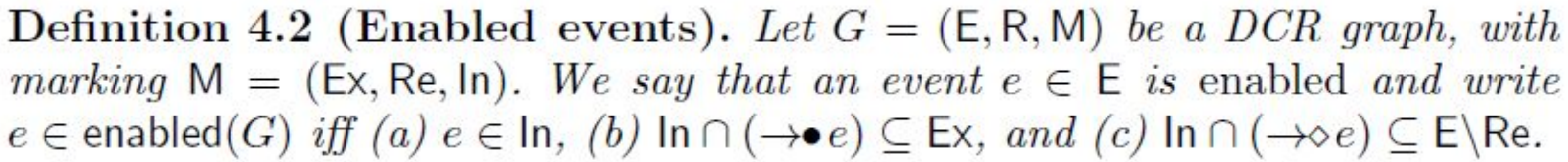
Definition 4.1 (DCR Graph [8]). *A* DCR graph *is a tuple* (E, R, M) *where*

- E *is a finite set of (labelled)* events, *the nodes of the graph.*
- R *is the edges of the graph. Edges are partioned into five kinds, named and drawn as follows: The* conditions ($\rightarrow\bullet$), responses ($\bullet\rightarrow$), milestones ($\rightarrow\diamond$), inclusions ($\rightarrow+$), *and* exclusions ($\rightarrow\%$).
- M *is the* marking *of the graph. This is a triple* (Ex, Re, In) *of sets of events, respectively the previously executed* (Ex)*, the currently pending* (Re)*, and the currently included* (In) *events.*

DCR Semantics

```java
public class DCRGraph {
	// Events
	protected HashSet<String> events = new HashSet<String>();

	// Relations
	private HashMap<String, HashSet<String>> conditionsFor = new HashMap<String, HashSet<String>>();
	private HashMap<String, HashSet<String>> milestonesFor = new HashMap<String, HashSet<String>>();
	private HashMap<String, HashSet<String>> responsesTo = new HashMap<String, HashSet<String>>();
	private HashMap<String, HashSet<String>> excludesTo = new HashMap<String, HashSet<String>>();
	private HashMap<String, HashSet<String>> includesTo = new HashMap<String, HashSet<String>>();

	// Marking
	public DCRMarking marking;
}


public class DCRMarking {
	public HashSet<String> executed = new HashSet<String>();
	public HashSet<String> included = new HashSet<String>();
	public HashSet<String> pending = new HashSet<String>();
}
```

DCR Semantics



Definition 4.2 (Enabled events). *Let* $G = (\mathsf{E}, \mathsf{R}, \mathsf{M})$ *be a DCR graph, with marking* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$. *We say that an event* $e \in \mathsf{E}$ *is* enabled *and write* $e \in \mathsf{enabled}(G)$ *iff (a)* $e \in \mathsf{In}$, *(b)* $\mathsf{In} \cap ({\rightarrow\!\bullet}\, e) \subseteq \mathsf{Ex}$, *and (c)* $\mathsf{In} \cap ({\rightarrow\!\diamond}\, e) \subseteq \mathsf{E} \backslash \mathsf{Re}$.

DCR Semantics

Definition 4.2 (Enabled events). *Let $G = (\mathsf{E}, \mathsf{R}, \mathsf{M})$ be a DCR graph, with marking* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$. *We say that an event $e \in \mathsf{E}$ is* enabled *and write* $e \in \mathsf{enabled}(G)$ *iff (a)* $e \in \mathsf{In}$, *(b)* $\mathsf{In} \cap (\rightarrow\!\bullet e) \subseteq \mathsf{Ex}$, *and (c)* $\mathsf{In} \cap (\rightarrow\!\diamond e) \subseteq \mathsf{E} \backslash \mathsf{Re}$.

e is included

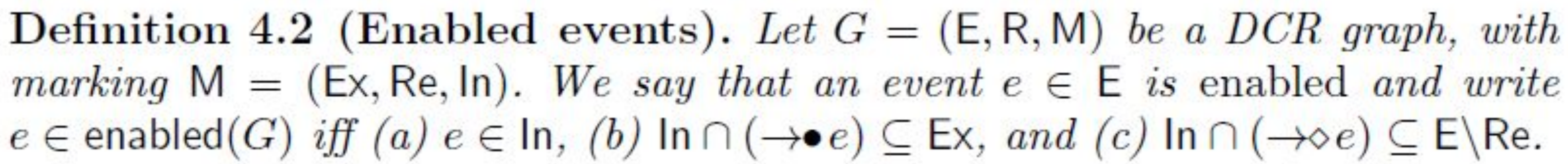
DCR Semantics

Definition 4.2 (Enabled events). *Let* $G = (\mathsf{E}, \mathsf{R}, \mathsf{M})$ *be a DCR graph, with marking* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$. *We say that an event* $e \in \mathsf{E}$ *is* enabled *and write* $e \in \mathsf{enabled}(G)$ *iff (a)* $e \in \mathsf{In}$, *(b)* $\mathsf{In} \cap (\rightarrow\!\bullet e) \subseteq \mathsf{Ex}$, *and (c)* $\mathsf{In} \cap (\rightarrow\!\diamond e) \subseteq \mathsf{E} \backslash \mathsf{Re}$.

e is included

All included conditions for *e* have been executed

DCR Semantics



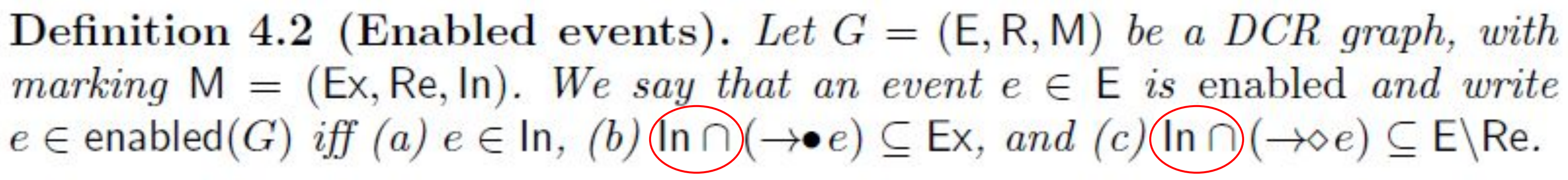
Definition 4.2 (Enabled events). *Let* $G = (\mathsf{E},\mathsf{R},\mathsf{M})$ *be a DCR graph, with marking* $\mathsf{M} = (\mathsf{Ex},\mathsf{Re},\mathsf{In})$. *We say that an event* $e \in \mathsf{E}$ *is* enabled *and write* $e \in \mathsf{enabled}(G)$ *iff (a)* $e \in \mathsf{In}$, *(b)* $\mathsf{In} \cap (\rightarrow\!\bullet e) \subseteq \mathsf{Ex}$, *and (c)* $\mathsf{In} \cap (\rightarrow\!\diamond e) \subseteq \mathsf{E} \setminus \mathsf{Re}$.

e is included

No included milestones for *e* are pending

All included conditions for *e* have been executed

DCR Semantics



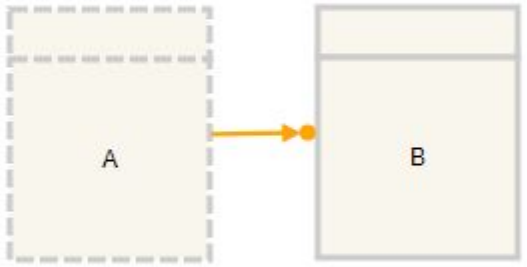
Definition 4.2 (Enabled events). *Let $G = (\mathsf{E}, \mathsf{R}, \mathsf{M})$ be a DCR graph, with marking* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$. *We say that an event $e \in \mathsf{E}$ is* enabled *and write* $e \in \mathsf{enabled}(G)$ *iff (a)* $e \in \mathsf{In}$, *(b)* $\mathsf{In} \cap (\rightarrow\!\bullet e) \subseteq \mathsf{Ex}$, *and (c)* $\mathsf{In} \cap (\rightarrow\!\diamond e) \subseteq \mathsf{E} \backslash \mathsf{Re}$.

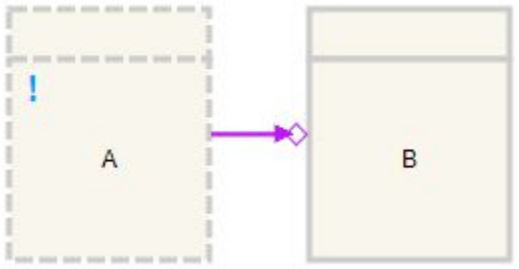
Note: Only included conditions and milestones can block other events!

DCR Semantics

Definition 4.2 (Enabled events). *Let* $G = (\mathsf{E}, \mathsf{R}, \mathsf{M})$ *be a DCR graph, with marking* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$. *We say that an event* $e \in \mathsf{E}$ *is* enabled *and write* $e \in \mathsf{enabled}(G)$ *iff (a)* $e \in \mathsf{In}$, *(b)* $\mathsf{In} \cap (\rightarrow\!\bullet\, e) \subseteq \mathsf{Ex}$, *and (c)* $\mathsf{In} \cap (\rightarrow\!\diamond\, e) \subseteq \mathsf{E} \setminus \mathsf{Re}$.

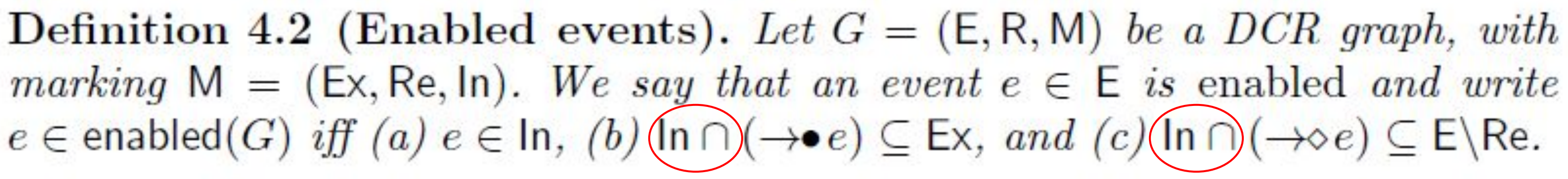
Note: Only included conditions and milestones can block other events!





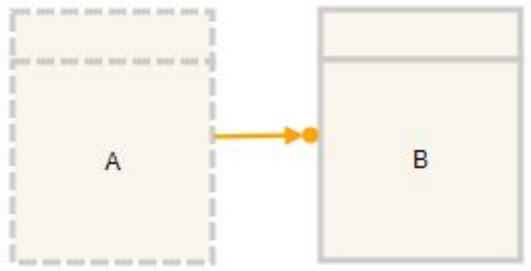
Question: Can we execute B in these models?

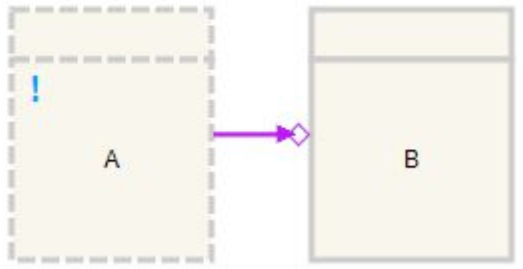
DCR Semantics



Definition 4.2 (Enabled events). *Let* $G = (\mathsf{E}, \mathsf{R}, \mathsf{M})$ *be a DCR graph, with marking* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$. *We say that an event* $e \in \mathsf{E}$ *is* enabled *and write* $e \in \mathsf{enabled}(G)$ *iff (a)* $e \in \mathsf{In}$, *(b)* $\mathsf{In} \cap (\to\!\bullet\, e) \subseteq \mathsf{Ex}$, *and (c)* $\mathsf{In} \cap (\to\!\diamond\, e) \subseteq \mathsf{E} \setminus \mathsf{Re}$.

Note: Only included conditions and milestones can block other events!





Question: Can we execute B in these models?

Yes!

DCR Semantics

```java
public Boolean enabled(final DCRMarking marking, final String event) {

        // Open world assumption: if an event doesn't exist in the graph it must be enabled.
        if (!events.contains(event)) { return true; }

        // check included
        if (!marking.included.contains(event)) { return false; }

        // Select only the included conditions
        final Set<String> inccon = new HashSet<String>(conditionsFor.get(event));
        inccon.retainAll(marking.included);
        // Check if all included conditions have been executed
        if (!marking.executed.containsAll(inccon)) { return false; }

        // Select only the included milestones
        final Set<String> incmil = new HashSet<String>(milestonesFor.get(event));
        incmil.retainAll(marking.included);
        // Check if any included milestone has a pending response
        for (final String p : marking.pending) {
                if (incmil.contains(p)) { return false; }
        }

        return true;
}
```

DCR Semantics

Definition 4.3 (Execution). *Let* $G = (\mathsf{E}, \mathsf{R}, \mathsf{M})$ *be a DCR graph with marking* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$. *Suppose* $e \in \mathsf{enabled}(G)$. *We may* execute *e obtaining the* resulting *DCR graph* $(\mathsf{E}, \mathsf{R}, \mathsf{M}')$ *with* $\mathsf{M}' = (\mathsf{Ex}', \mathsf{Re}', \mathsf{In}')$ *defined as follows.*

1. $\mathsf{Ex}' = \mathsf{Ex} \cup e$
2. $\mathsf{Re}' = (\mathsf{Re} \setminus e) \cup (e {\bullet\!\!\rightarrow})$
3. $\mathsf{In}' = (\mathsf{In} \setminus (e {\rightarrow}\%)) \cup (e {\rightarrow}{+})$

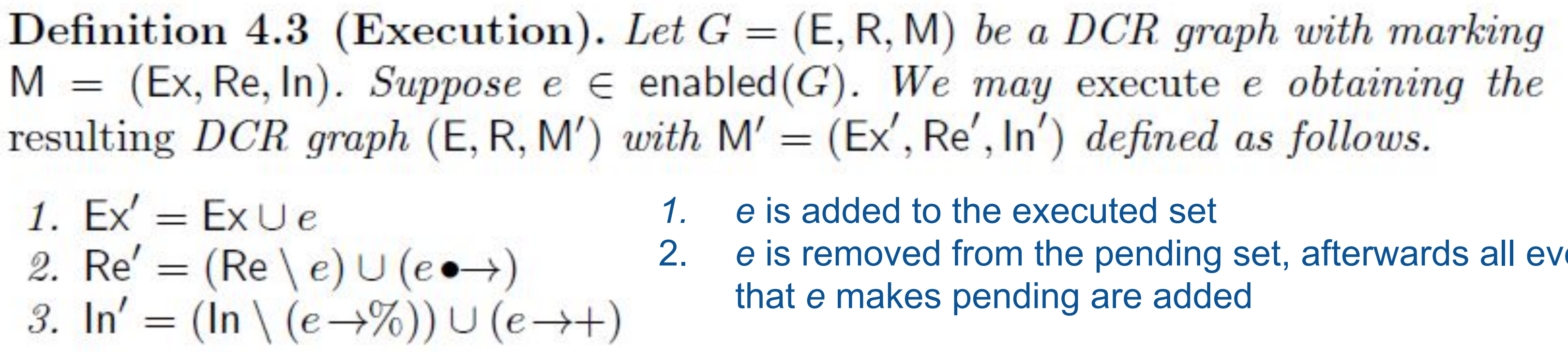
DCR Semantics

Definition 4.3 (Execution). *Let* $G = (\mathsf{E}, \mathsf{R}, \mathsf{M})$ *be a DCR graph with marking* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$. *Suppose* $e \in \mathsf{enabled}(G)$. *We may* execute e *obtaining the* resulting *DCR graph* $(\mathsf{E}, \mathsf{R}, \mathsf{M}')$ *with* $\mathsf{M}' = (\mathsf{Ex}', \mathsf{Re}', \mathsf{In}')$ *defined as follows.*

1. $\mathsf{Ex}' = \mathsf{Ex} \cup e$
2. $\mathsf{Re}' = (\mathsf{Re} \setminus e) \cup (e{\bullet}\!\!\rightarrow)$
3. $\mathsf{In}' = (\mathsf{In} \setminus (e{\rightarrow}\%)) \cup (e{\rightarrow}{+})$

1. *e* is added to the executed set

DCR Semantics

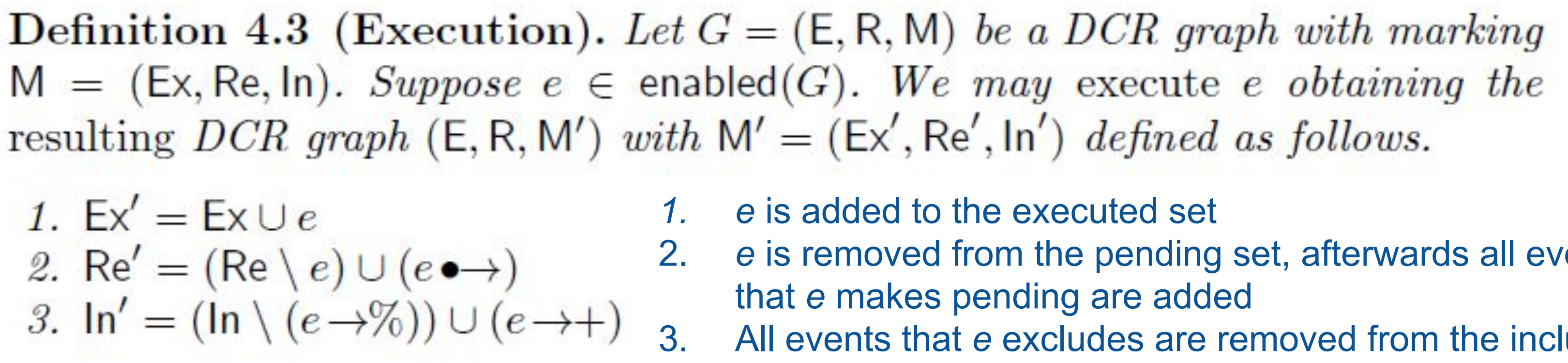


Definition 4.3 (Execution). *Let* $G = (\mathsf{E}, \mathsf{R}, \mathsf{M})$ *be a DCR graph with marking* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$. *Suppose* $e \in \mathsf{enabled}(G)$. *We may* execute e *obtaining the* resulting *DCR graph* $(\mathsf{E}, \mathsf{R}, \mathsf{M}')$ *with* $\mathsf{M}' = (\mathsf{Ex}', \mathsf{Re}', \mathsf{In}')$ *defined as follows.*

1. $\mathsf{Ex}' = \mathsf{Ex} \cup e$
2. $\mathsf{Re}' = (\mathsf{Re} \setminus e) \cup (e \bullet\!\!\rightarrow)$
3. $\mathsf{In}' = (\mathsf{In} \setminus (e \rightarrow\%)) \cup (e \rightarrow+)$

1. *e* is added to the executed set
2. *e* is removed from the pending set, afterwards all events that *e* makes pending are added

DCR Semantics



Definition 4.3 (Execution). *Let* $G = (\mathsf{E}, \mathsf{R}, \mathsf{M})$ *be a DCR graph with marking* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$. *Suppose* $e \in \mathsf{enabled}(G)$. *We may* execute e *obtaining the resulting DCR graph* $(\mathsf{E}, \mathsf{R}, \mathsf{M}')$ *with* $\mathsf{M}' = (\mathsf{Ex}', \mathsf{Re}', \mathsf{In}')$ *defined as follows.*

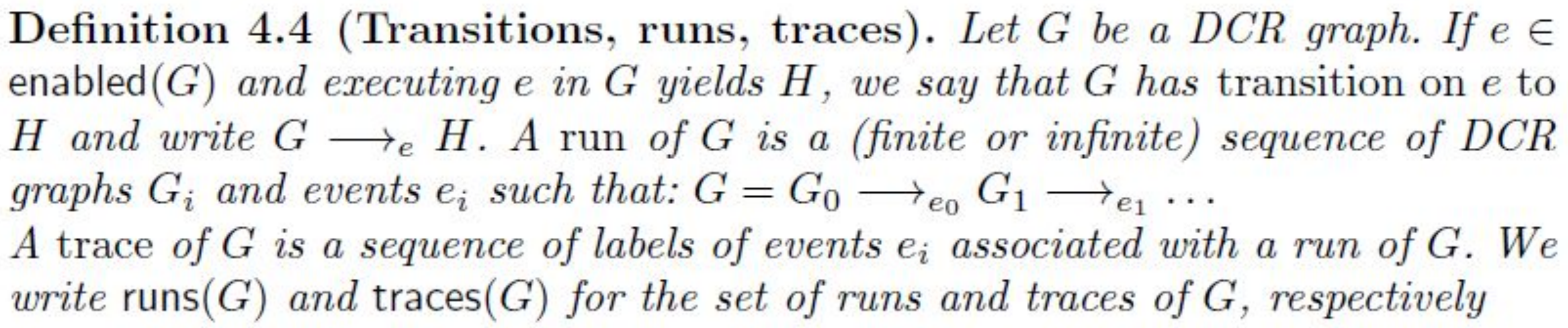
1. $\mathsf{Ex}' = \mathsf{Ex} \cup e$
2. $\mathsf{Re}' = (\mathsf{Re} \setminus e) \cup (e \bullet\!\!\rightarrow)$
3. $\mathsf{In}' = (\mathsf{In} \setminus (e \rightarrow\!\%)) \cup (e \rightarrow\!+)$

1. *e* is added to the executed set
2. *e* is removed from the pending set, afterwards all events that *e* makes pending are added
3. All events that *e* excludes are removed from the included set, afterwards all events that *e* includes are added

DCR Semantics

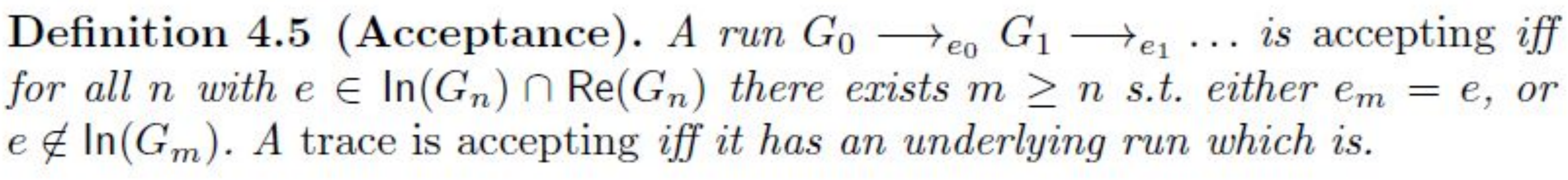
```java
public DCRMarking execute(final DCRMarking marking, final String event) {
        // Check if the event exists
        if (!events.contains(event)) { return marking; }

        // Check if the event is enabled
        if (!this.enabled(marking, event)) { return marking; }

        // Create a new marking
        DCRMarking result = marking.clone();

        // Add the event to the set of executed events.
        result.executed.add(event);

        // Remove the event from the set of pending events.
        result.pending.remove(event);

        // Add all new responses.
        result.pending.addAll(responsesTo.get(event));

        // Remove all excluded events
        result.included.removeAll(excludesTo.get(event));

        // Add all included events
        result.included.addAll(includesTo.get(event));

        return result;
}
```

DCR Semantics



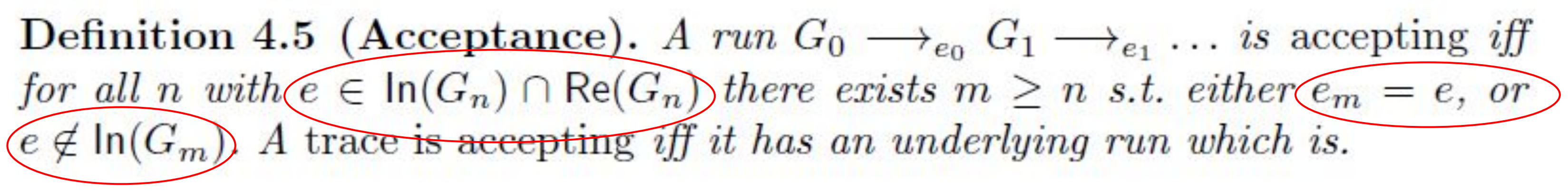
Definition 4.4 (Transitions, runs, traces). *Let G be a DCR graph. If $e \in$* $\mathsf{enabled}(G)$ *and executing e in G yields H, we say that G has* transition on e to *H and write $G \longrightarrow_e H$. A* run *of G is a (finite or infinite) sequence of DCR graphs G_i and events e_i such that: $G = G_0 \longrightarrow_{e_0} G_1 \longrightarrow_{e_1} \ldots$*
A trace *of G is a sequence of labels of events e_i associated with a run of G. We write* $\mathsf{runs}(G)$ *and* $\mathsf{traces}(G)$ *for the set of runs and traces of G, respectively*

DCR Semantics



Definition 4.5 (Acceptance). *A run* $G_0 \longrightarrow_{e_0} G_1 \longrightarrow_{e_1} \ldots$ *is* accepting *iff for all* n *with* $e \in \mathsf{In}(G_n) \cap \mathsf{Re}(G_n)$ *there exists* $m \geq n$ *s.t. either* $e_m = e$, *or* $e \notin \mathsf{In}(G_m)$. *A* trace is accepting *iff it has an underlying run which is.*

DCR Semantics



Definition 4.5 (Acceptance). *A run* $G_0 \longrightarrow_{e_0} G_1 \longrightarrow_{e_1} \ldots$ *is* accepting *iff for all* n *with* $e \in \mathsf{In}(G_n) \cap \mathsf{Re}(G_n)$ *there exists* $m \geq n$ *s.t. either* $e_m = e$, *or* $e \notin \mathsf{In}(G_m)$. *A* trace is accepting *iff it has an underlying run which is.*

If an event *e* at some point becomes pending, then at some point after, it is either executed or excluded.

DCR Semantics

```
public Set<String> getIncludedPending()
{
        // Select those included events that are also pending
        HashSet<String> result = new HashSet<String>(marking.included);
        result.retainAll(marking.pending);
        return result;
}

public boolean isAccepting()
{
        // Check if there are any included pending responses
        return getIncludedPending().isEmpty();
}
```

DCR Semantics

Definition 4.6 (Language). *The* language *of a DCR graph G is the set of its accepting traces. We write* $\mathsf{lang}(G)$ *for the language of G.*

Sample DCR Engine

DEMO

BREAK

Overview

- Semantics of DCR Graphs
- **Process Mining**
  - **Process Conformance**
  - **Process Discovery**
- Assignment 1

Process Mining

- In the past:
  - Models designed based on how we **believe processes work**
- Process Mining
  - Look at real-world *event logs* to find out how **processes actually work**

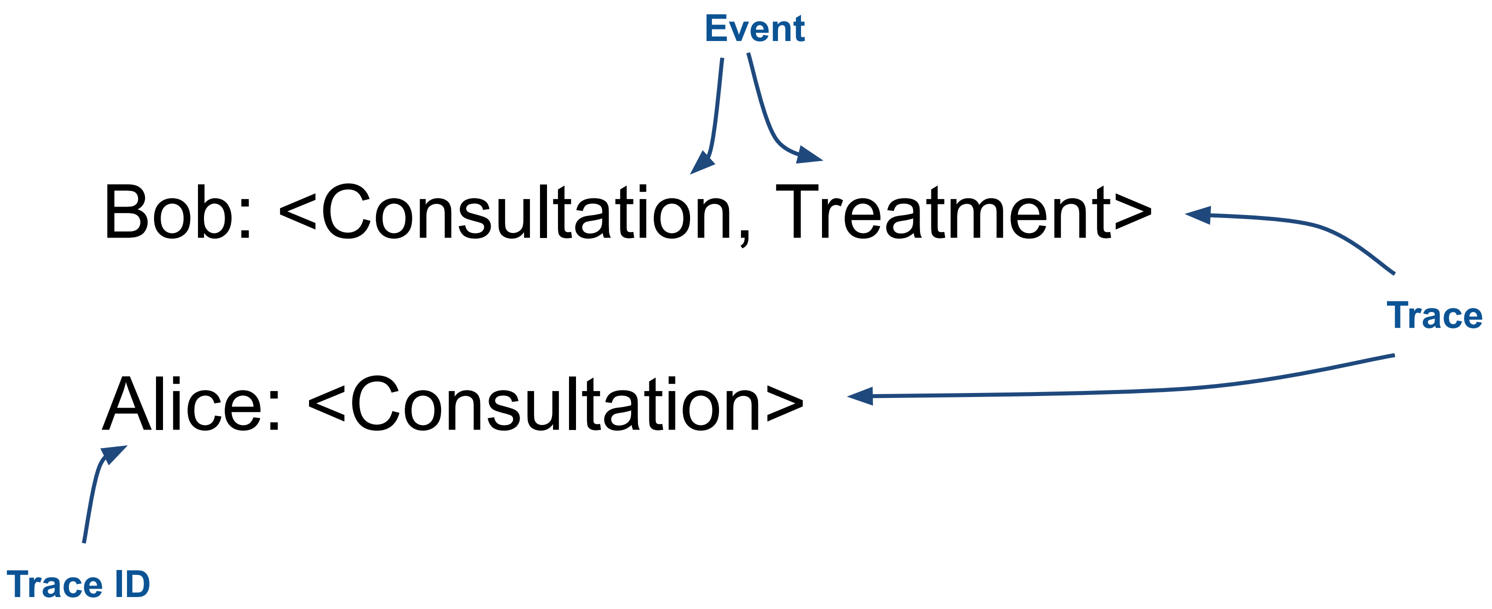
Process Mining

- **<u>Event Log</u>**: A list / table containing *events*, consisting of:
  - *Activities* (e.g. Consultation, Treatment)
  - *Cases* (e.g. a Treatment or Patient ID)
  - An *ordering* of the events (e.g. timestamps)
  - Optionally: any other data relevant to the process

Activity	**Patient**	**Time**	**...**
Consultation	Bob	12.11.2013, 9:19:57	...
Consultation	Alice	12.11.2013, 9:30:01	...
Treatment	Bob	12.11.2013, 15:40:45	...

Process Mining

- **Event Log**: A set of traces

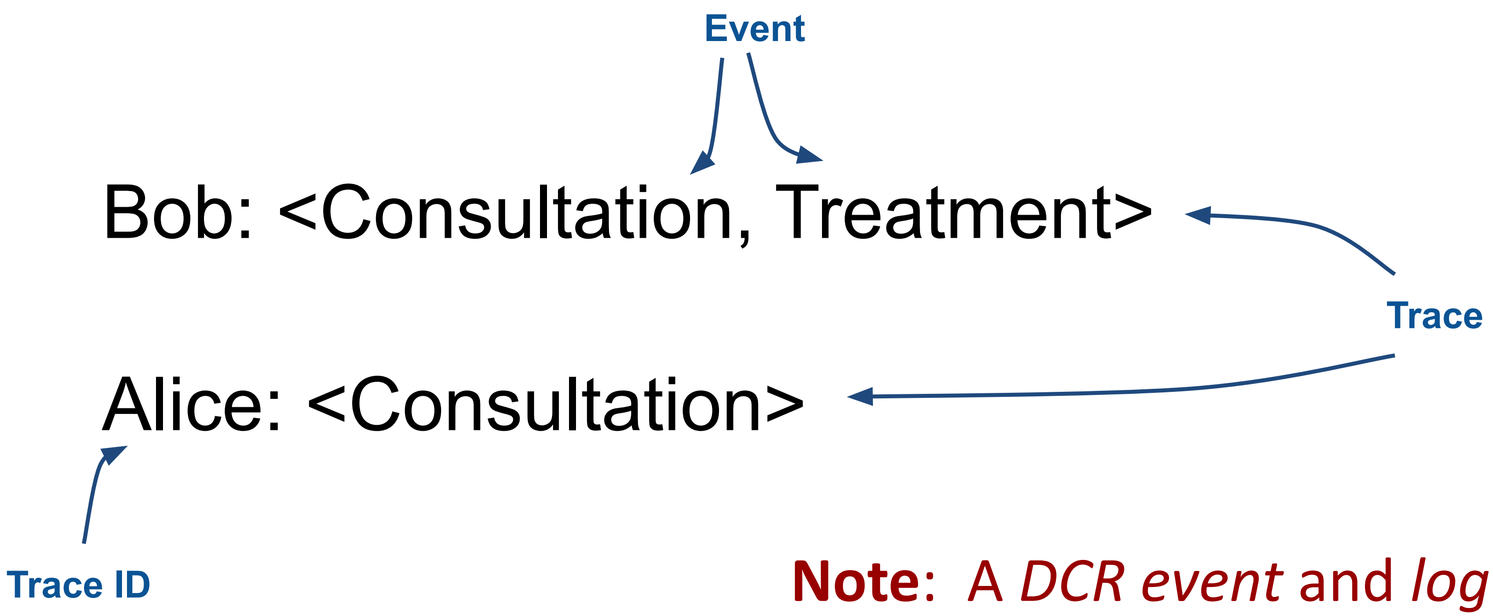


Bob: <Consultation, Treatment>

Alice: <Consultation>

Process Mining

- **<u>Event Log</u>**: A set of traces



Bob: <Consultation, Treatment>

Alice: <Consultation>

Note: A *DCR event* and *log event* are <u>not</u> the same thing!

DCR events may occur more than once, log events are always unique

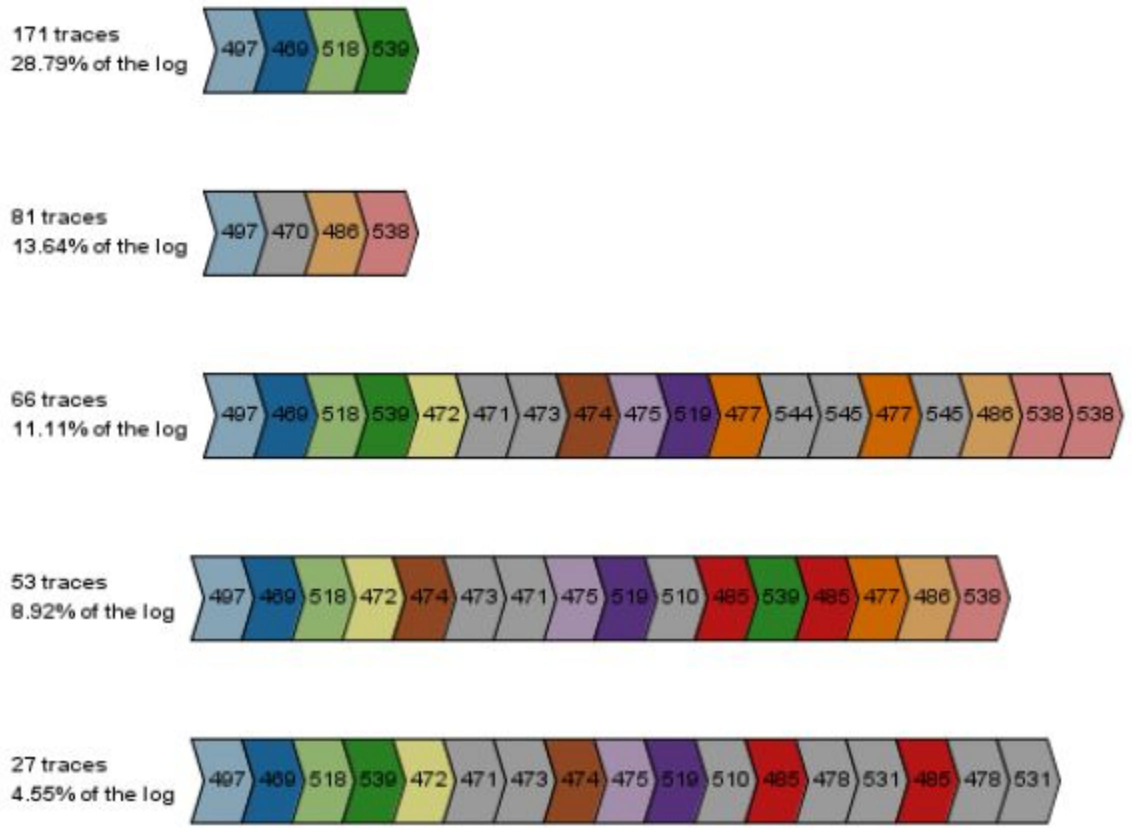
Process Mining

Logs bear valuable information to answer questions like:

- How many patients did we treat in November 2013?
- In what different ways do we treat patients for colon-cancer?
- Which are the most common treatments?
- Are there treatments we never use?

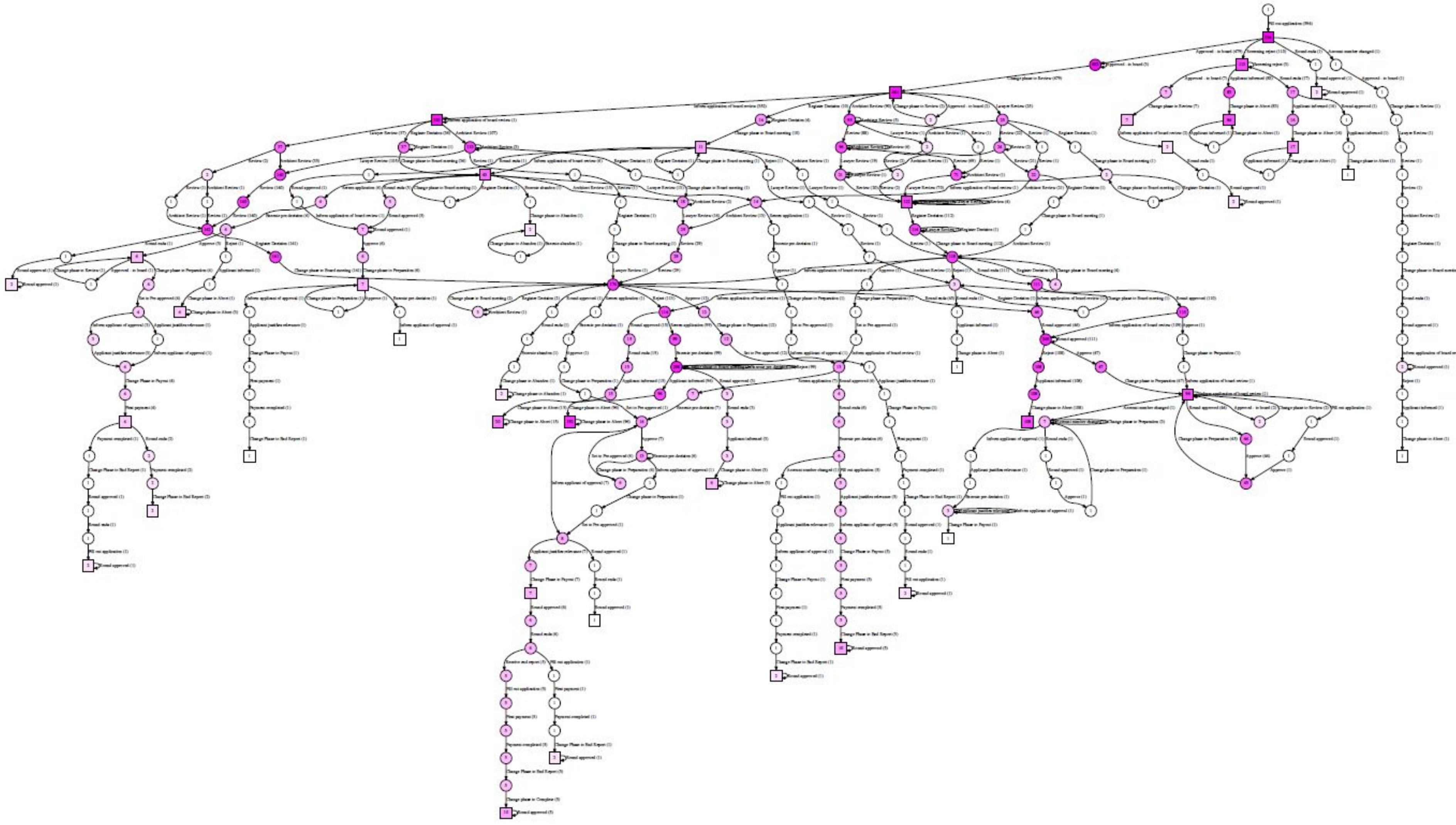
Process Mining

Inspect most common executions:



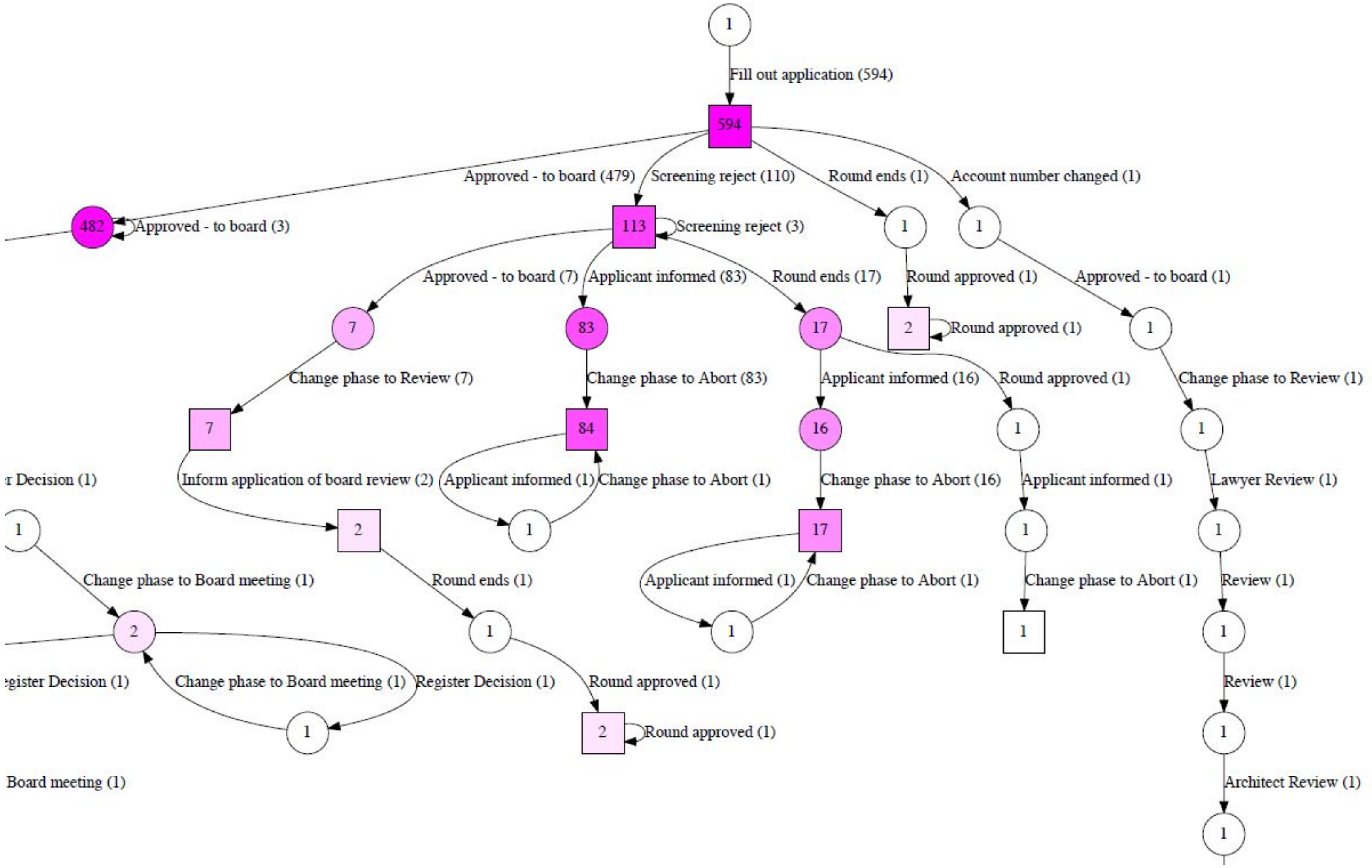
Process Mining

Visualize a model of all executions:



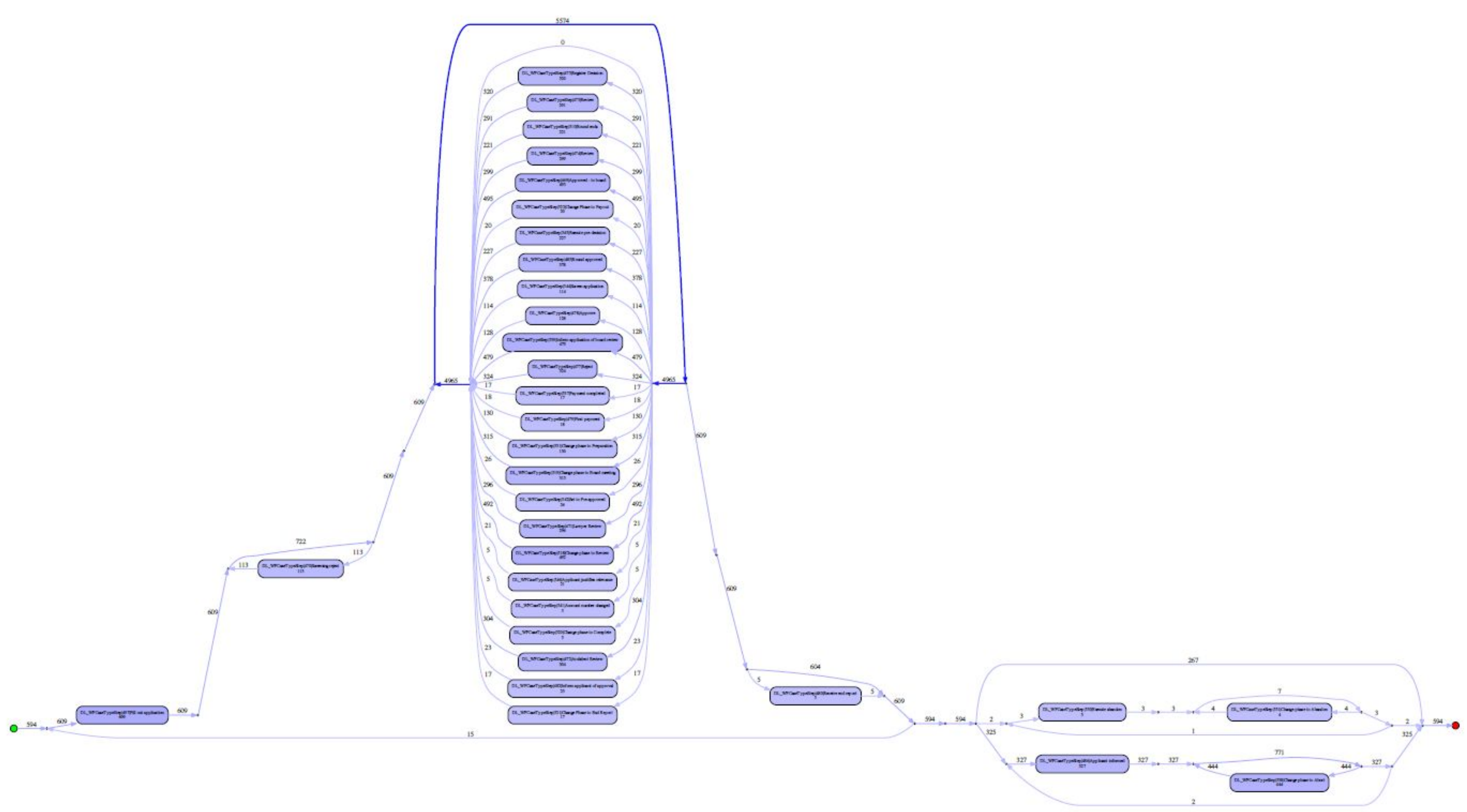
Process Mining

Visualize a model of all executions:



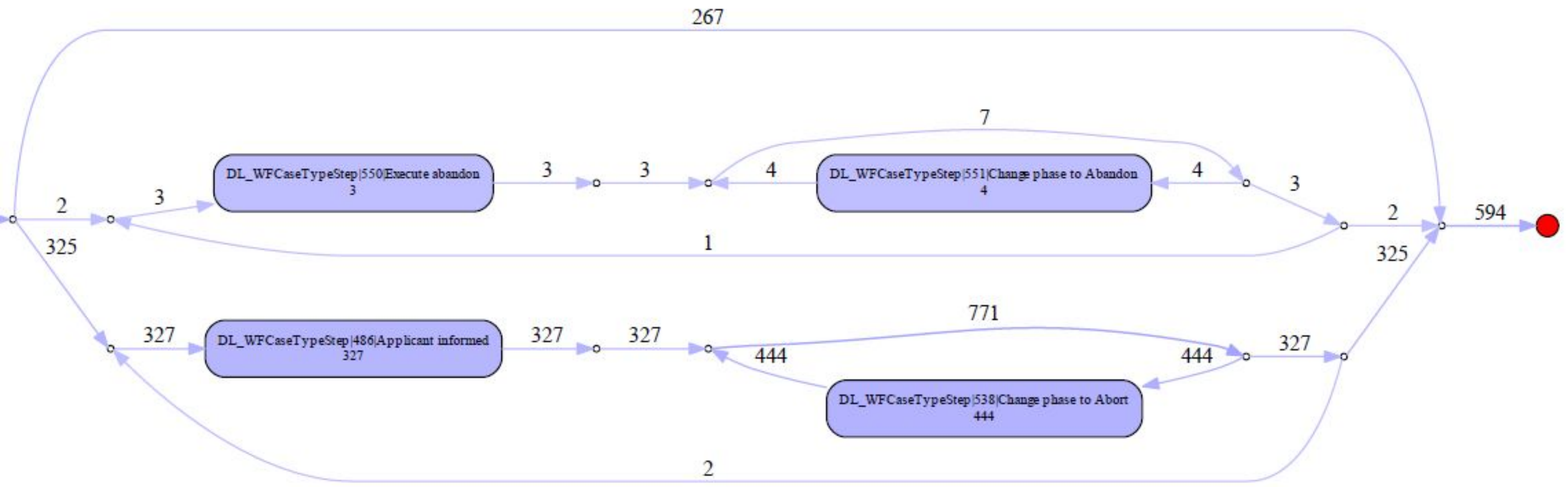
Process Mining

Visualize an imprecise, but simplified model:



Process Mining

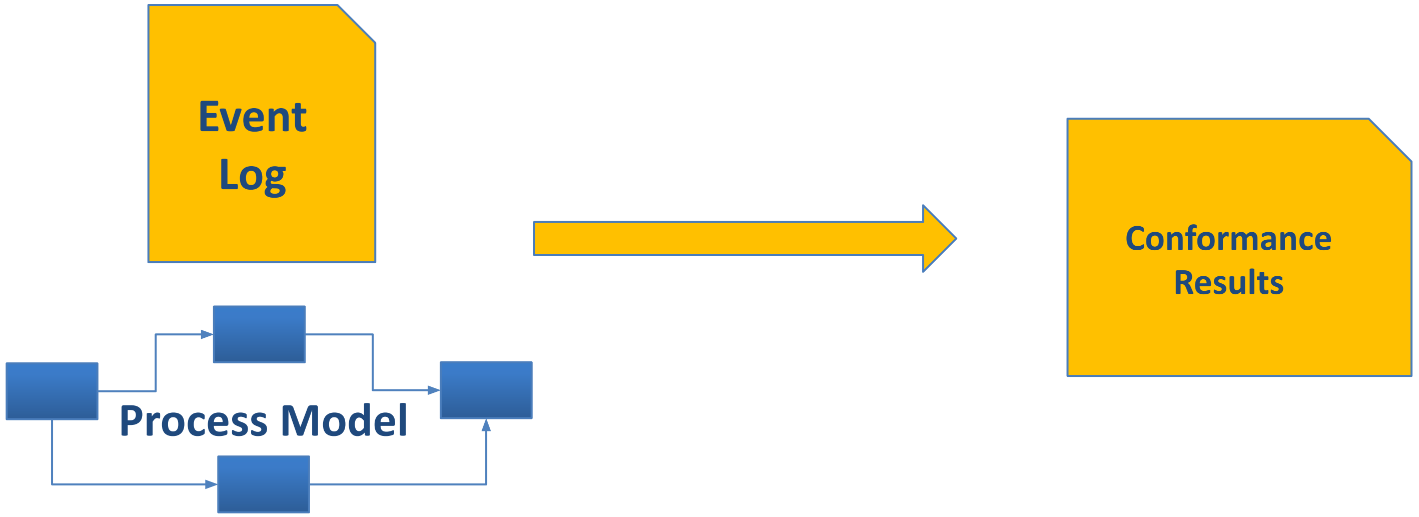
Visualize an imprecise, but simplified model:


267
7
DL_WFCaseTypeStep|550|Execute abandon 3
DL_WFCaseTypeStep|551|Change phase to Abandon 4
3
3
4
4
3
2
3
2
594
325
1
325
771
DL_WFCaseTypeStep|486|Applicant informed 327
327
327
327
444
444
327
DL_WFCaseTypeStep|538|Change phase to Abort 444
2

Process Mining

Three types of process mining techniques:

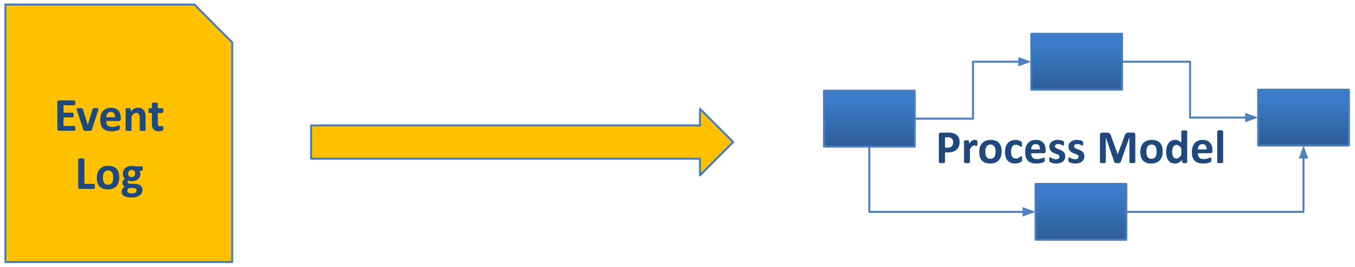
- **Conformance Checking**
  - Check if real processes conform to our models
- Process Discovery
  - Create models based on real-world behaviour
- Process Enhancement
  - Improve our models based on observed real-world behaviour



Process Mining

Three types of process mining techniques:

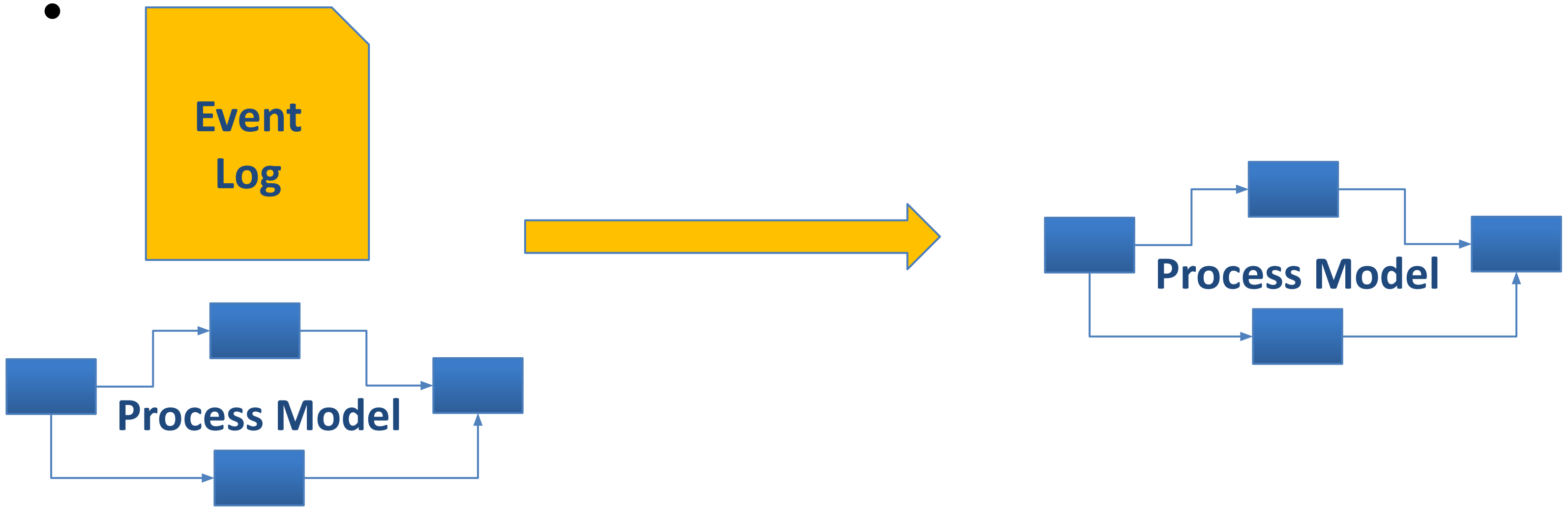
- Conformance Checking
  - Check if real processes conform to our models
- **Process Discovery**
  - Create models based on real-world behaviour
- Process Enhancement
  - Improve our models based on observed real-world behaviour
-



Process Mining

Three types of process mining techniques:

- Conformance Checking
  - Check if real processes conform to our models
- Process Discovery
  - Create models based on real-world behaviour
- **Process Enhancement**
  - Improve our models based on observed real-world behaviour
-



Conformance Checking with DCR Graphs

Event Log:

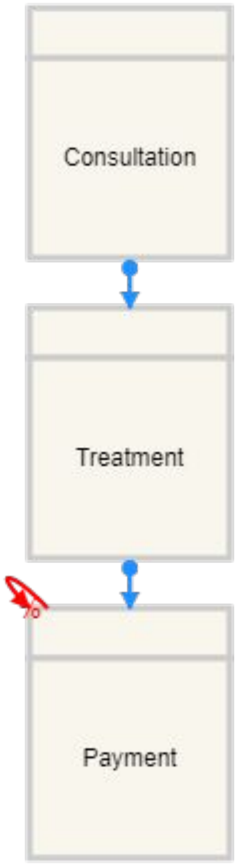
Bob: <Consultation, Treatment, Payment>

Alice: <Consultation, Treatment>

Cooper: <Consultation, Treatment, Payment, Payment>

David: <Consultation, Treatment, Treatment, Payment>

Model:



Result:

Bob: satisfied

Alice: not-satisfied

Cooper: not-satisfied

David: ....

<u>Question</u>: What is the result for David?

Conformance Checking with DCR Graphs

Event Log:

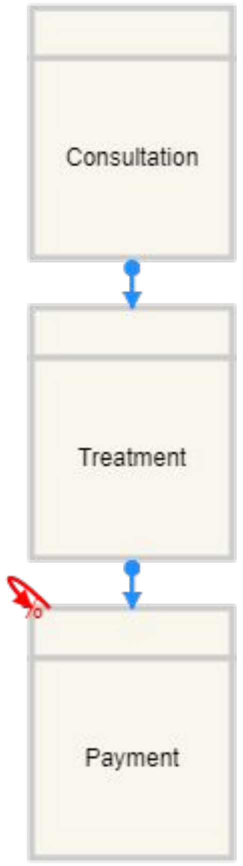
Bob: <Consultation, Treatment, Payment>

Alice: <Consultation, Treatment>

Cooper: <Consultation, Treatment, Payment, Payment>

David: <Consultation, Treatment, Treatment, Payment>

Model:



Result:

Bob: satisfied

Alice: not-satisfied

Cooper: not-satisfied

David: satisfied

Trace-based Conformance: 50%

Conformance Checking with DCR Graphs

Event Log:

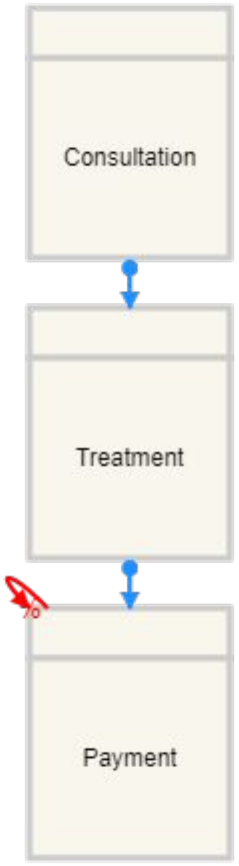
Bob: <Consultation, Treatment, Payment>

Alice: <Consultation, Treatment>

Cooper: <Consultation, Treatment, Payment, Payment>

David: <Consultation, Treatment, Treatment, Payment>

Model:



Result:

Bob: satisfied

Alice: not-satisfied

Cooper: not-satisfied

David: satisfied

Trace-based Conformance: 50%

Note: Good if we want to know what percentage of cases conforms to the model, but not very fine grained...

Trace Alignment

Event Log:

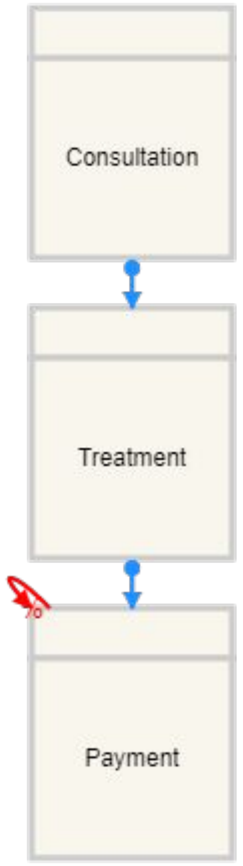
Bob: <Consultation, Treatment, Payment>

Alice: <Consultation, Treatment>

Cooper: <Consultation, Treatment, Payment, Payment>

David: <Consultation, Treatment, Treatment, Payment>

Model:



Result:

Bob: <Consultation, Treatment, Payment>

Alice: <Consultation, Treatment, Payment>

Cooper: <Consultation, Treatment, Payment, ~~Payment~~>

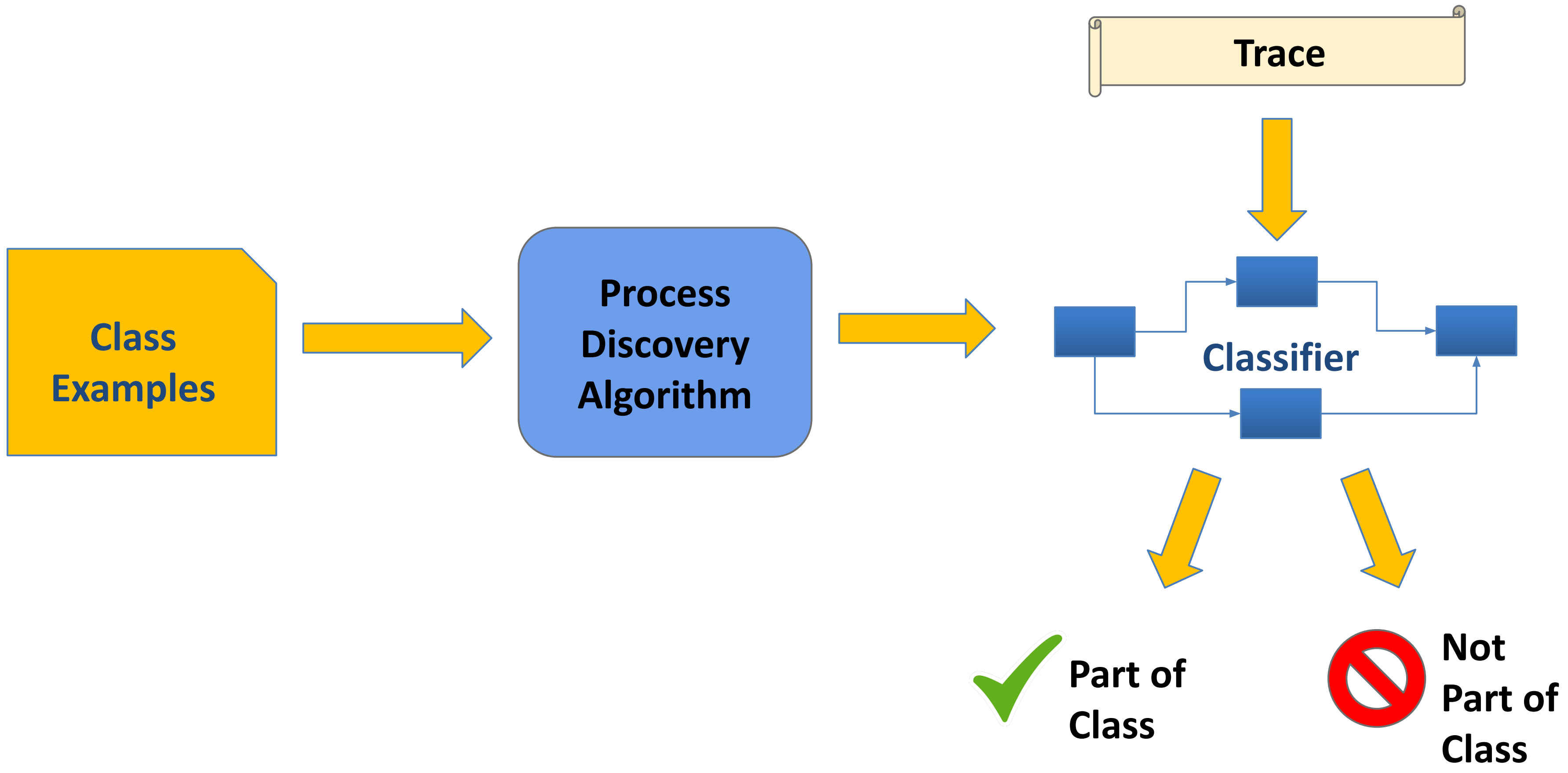
David: <Consultation, Treatment, Treatment, Payment>

Alignment cost: 1 insertion, 1 deletion.

Alignment-based conformance: (12/14) = 85.7%

A Machine Learning Perspective on Process Discovery

- Process discovery has (almost) exclusively been treated as a **one-class classification** task:



Process Discovery for DCR Graphs

- First DCR miner: The *UlrikHovsgaard Miner*: [1]
  - Creates a model containing all possible relations
  - Removes relations that may conflict with traces in the log
  - Produces heavily constrained spaghetti models, making them difficult to understand and not helpful.
  - Discovered models sometimes mimic flow-diagrams.

[1] Debois, S., Hildebrandt, T.T., Laursen, P.H. and Ulrik, K.R., 2017, April. Declarative process mining for DCR graphs. In Proceedings of the Symposium on Applied Computing (pp. 759-764). ACM.

Process Discovery for DCR Graphs

- First DCR miner: The *UlrikHovsgaard Miner*: [1]
  - Creates a model containing all possible relations
  - Removes relations that may conflict with traces in the log
  - Produces heavily constrained spaghetti models, making them difficult to understand and not helpful.
  - Discovered models sometimes mimic flow-diagrams.

- **Observation**:
  - When looking for a *flexible* model, it's counter-productive to start from a fully constrained model, which allows no behaviour

[1] Debois, S., Hildebrandt, T.T., Laursen, P.H. and Ulrik, K.R., 2017, April. Declarative process mining for DCR graphs. In Proceedings of the Symposium on Applied Computing (pp. 759-764). ACM.

DisCoveR

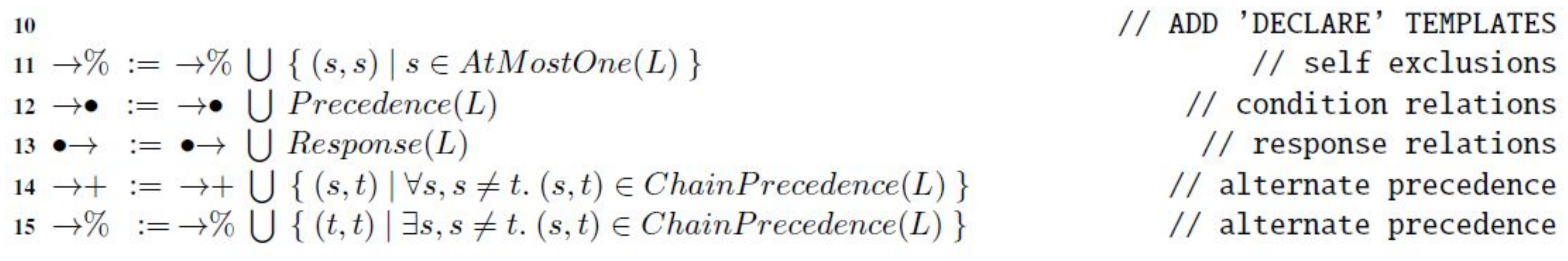
- Goals:
  - Find a good balance between simplicity and precision
  - Guarantee fitness
  - Short runtime (minutes instead of hours)

- Basic approach:
  - Start from a fully unconstrained model
  - Add relations to the model based on an analysis of the log

DisCoveR

Basic Patterns | Exclusions | Dynamic Behaviour | Additional Conditions

An algorithm that finds a number of basic declarative patterns, inspired by the Minerful algorithm.

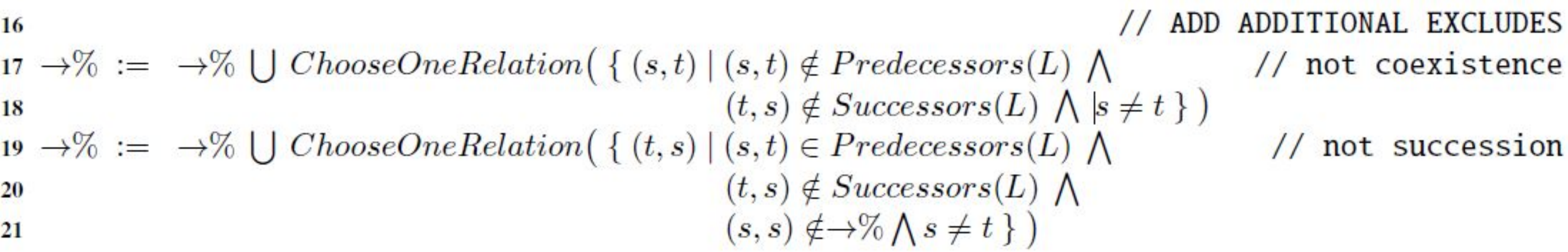


```
                                                                              // ADD 'DECLARE' TEMPLATES
$\rightarrow\%$ := $\rightarrow\%$ $\bigcup$ { $(s,s) \mid s \in AtMostOne(L)$ }                              // self exclusions
$\rightarrow\bullet$ := $\rightarrow\bullet$ $\bigcup$ $Precedence(L)$                                       // condition relations
$\bullet\rightarrow$ := $\bullet\rightarrow$ $\bigcup$ $Response(L)$                                        // response relations
$\rightarrow+$ := $\rightarrow+$ $\bigcup$ { $(s,t) \mid \forall s, s \neq t.\ (s,t) \in ChainPrecedence(L)$ }   // alternate precedence
$\rightarrow\%$ := $\rightarrow\%$ $\bigcup$ { $(t,t) \mid \exists s, s \neq t.\ (s,t) \in ChainPrecedence(L)$ }  // alternate precedence
```

DisCoveR

Basic Patterns → Exclusions → Dynamic Behaviour → Additional Conditions

An algorithm for finding exclusion relations not covered by the previous algorithm.



```
                                                                         // ADD ADDITIONAL EXCLUDES
$\rightarrow\% := \rightarrow\% \bigcup ChooseOneRelation(\{ (s,t) \mid (s,t) \notin Predecessors(L) \bigwedge$      // not coexistence
                                             $(t,s) \notin Successors(L) \bigwedge s \neq t \})$
$\rightarrow\% := \rightarrow\% \bigcup ChooseOneRelation(\{ (t,s) \mid (s,t) \in Predecessors(L) \bigwedge$        // not succession
                                             $(t,s) \notin Successors(L) \bigwedge$
                                             $(s,s) \notin \rightarrow\% \bigwedge s \neq t \})$
```

DisCoveR

Basic Patterns | Exclusions | Dynamic Behaviour | Additional Conditions

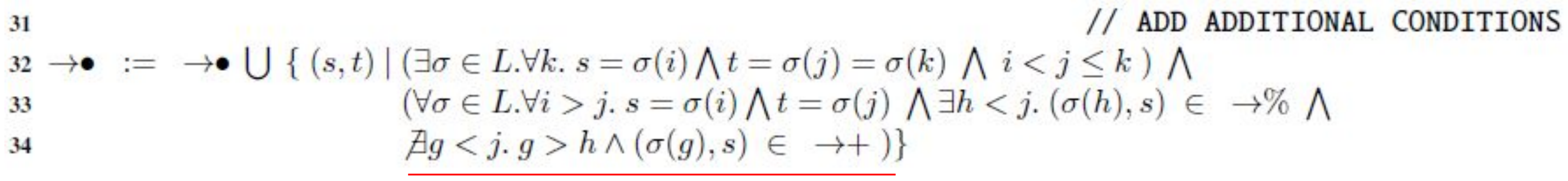
An algorithm that adds additional exclusions and inclusions to further constrain when activities are enabled

```
                                                  // ADDITIONAL INCLUDES/EXCLUDES
$\to\% := \to\% \bigcup NotChainSuccession(L)$                        // not chain succession
$\to+ := \to+ \bigcup \{ (u,t) \mid \exists s.\ (s,t) \in NotChainSuccession(L) \bigwedge (s,u,t) \in Between(L) \}$
```

DisCoveR

Basic Patterns → Exclusions → Dynamic Behaviour → Additional Conditions

An algorithm that finds a small amount of additional conditions, which for some logs provide significant improvements on precision.
Based on the complex interplay between exclusions and conditions.



```
                                                    // ADD ADDITIONAL CONDITIONS
$\rightarrow\!\bullet \;:=\; \rightarrow\!\bullet \bigcup \{\, (s,t) \mid (\exists \sigma \in L. \forall k.\ s = \sigma(i) \bigwedge t = \sigma(j) = \sigma(k) \ \bigwedge\ i < j \leq k\,) \ \bigwedge$
                    $(\forall \sigma \in L. \forall i > j.\ s = \sigma(i) \bigwedge t = \sigma(j)\ \bigwedge \exists h < j.\ (\sigma(h), s) \in\ \rightarrow\%\ \bigwedge$
                    $\underline{\nexists g < j.\ g > h \wedge (\sigma(g), s)\ \in\ \rightarrow\!+\ )\}}$
```

DisCoveR

Basic Patterns

An algorithm that finds a number of basic declarative patterns, inspired by the Minerful algorithm.

Exclusions

An algorithm for finding exclusion relations not covered by the previous algorithm.

Additional Conditions

An algorithm that finds a small amount of additional conditions, which for some logs provide significant improvements on precision.

Based on the complex interplay between exclusions and conditions.

Dynamic Behaviour

An algorithm that adds additional exclusions and inclusions to further constrain when activities are enabled

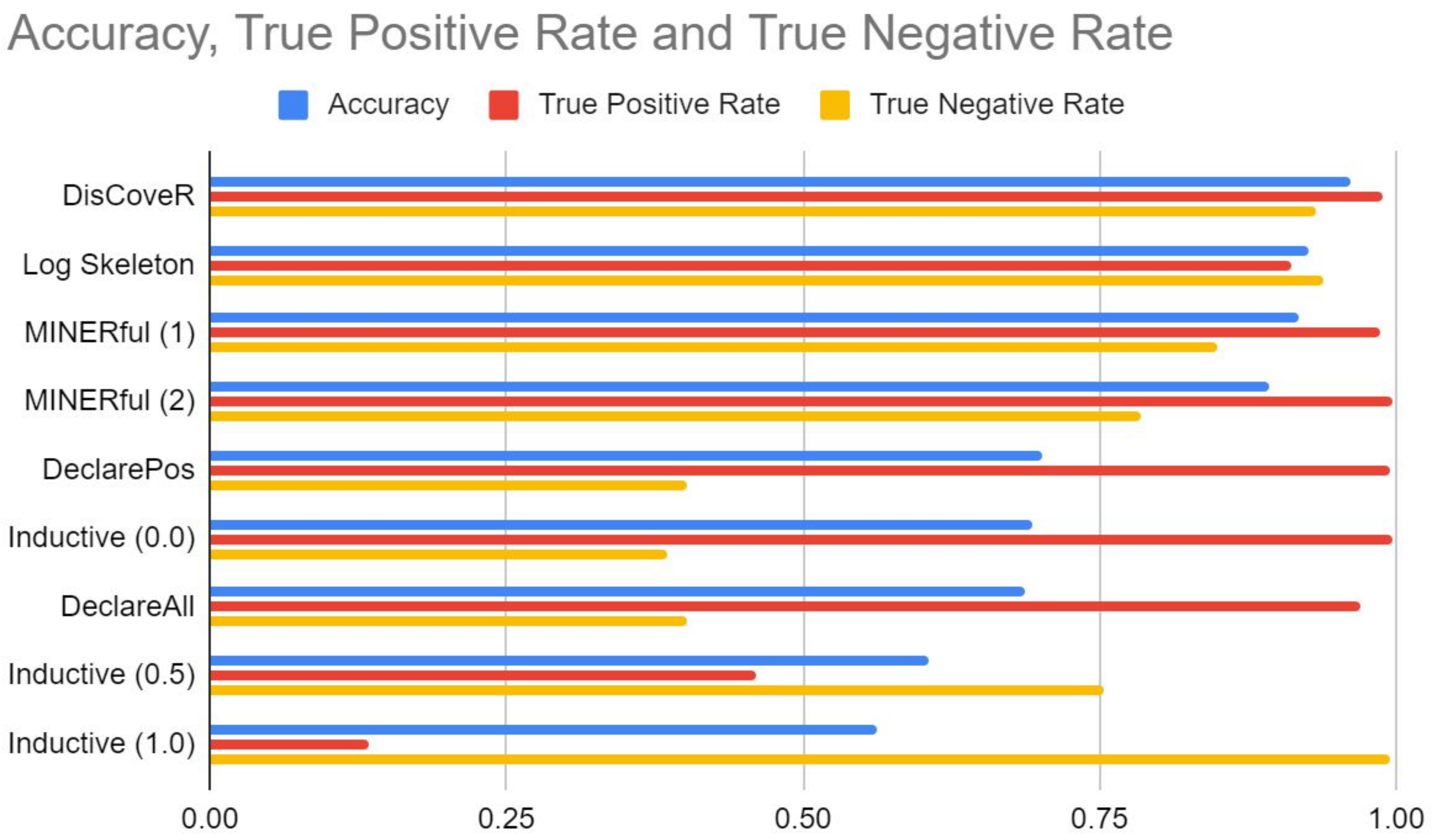
Evaluation

Process Discovery Challenge 2019: mine 10 logs generated from synthetic models, evaluate based on a labelled test set (with both positive and negative outcomes)

DisCoveR (full):

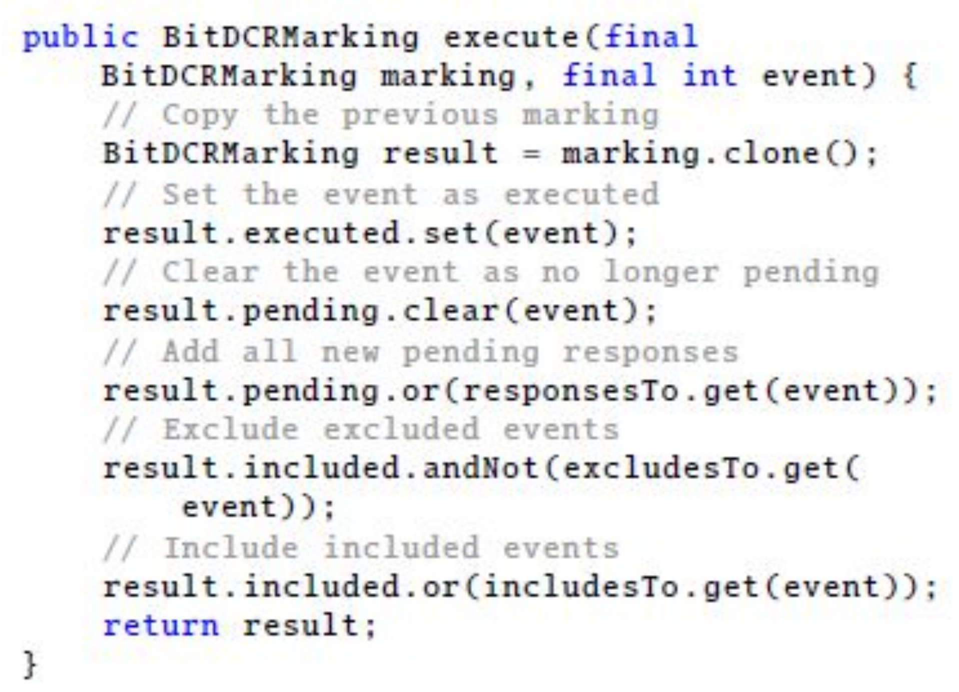
- Correctly classified 865 out of 900 traces (96% accuracy)
- 5 false negatives, 30 false positives
- Used only the control-flow perspective (there was also a data perspective in the ground truth model and logs)

DisCoveR Accuracy

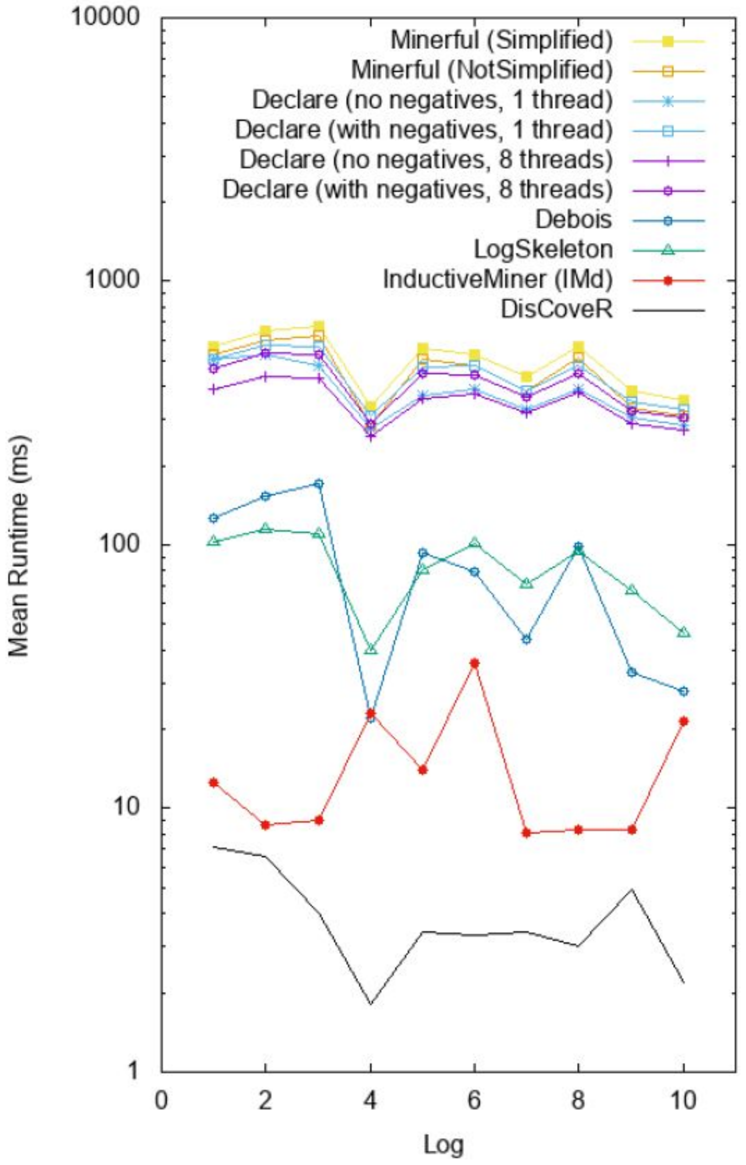


Runtime performance

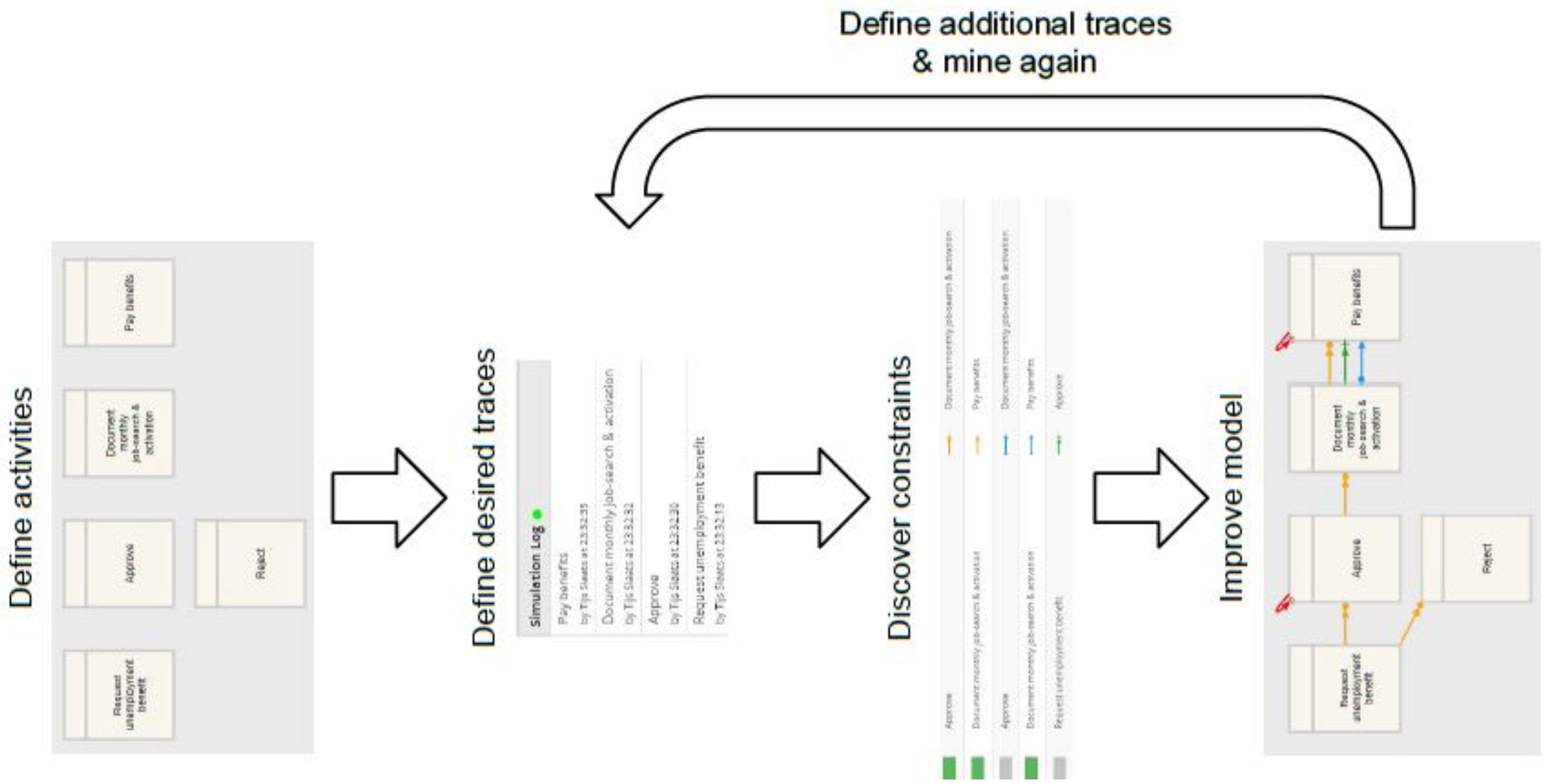
Faster than any other tested miner due to implementation of large parts of the algorithm and DCR semantics as bit vector operations.



```java
public BitDCRMarking execute(final
    BitDCRMarking marking, final int event) {
    // Copy the previous marking
    BitDCRMarking result = marking.clone();
    // Set the event as executed
    result.executed.set(event);
    // Clear the event as no longer pending
    result.pending.clear(event);
    // Add all new pending responses
    result.pending.or(responsesTo.get(event));
    // Exclude excluded events
    result.included.andNot(excludesTo.get(
        event));
    // Include included events
    result.included.or(includesTo.get(event));
    return result;
}
```

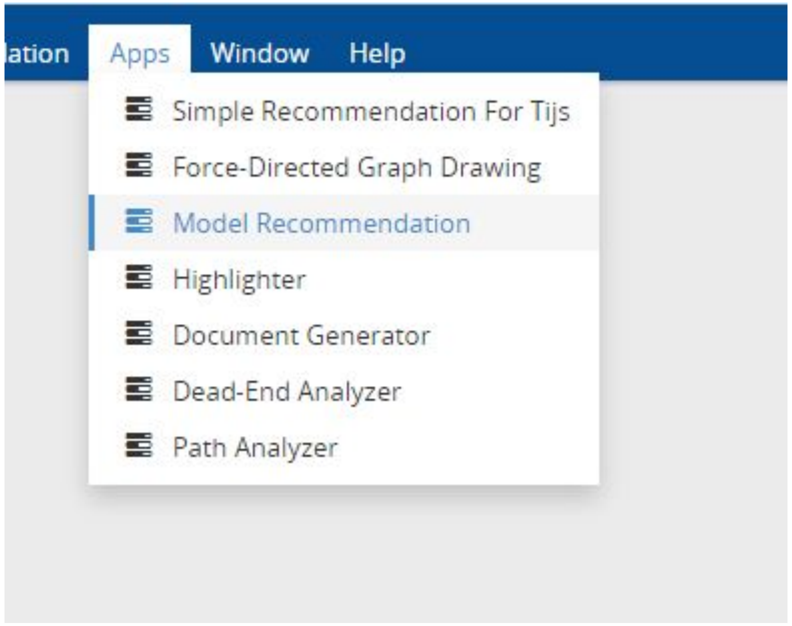


Using DCR Discovery for Model Recommendation


Define additional traces & mine again
Define activities
Pay benefits
Document monthly job-search & activation
Approve
Reject
Request unemployment benefit
Define desired traces
Simulation Log
Pay benefits
by Tijs Slaats at 23:32:35
Document monthly job-search & activation
by Tijs Slaats at 23:32:32
Approve
by Tijs Slaats at 23:32:30
Request unemployment benefit
by Tijs Slaats at 23:32:13
Discover constraints
Improve model

Using DCR Discovery for Model Recommendation

Demo



Overview

- Semantics of DCR Graphs
- Process Mining
  - Process Conformance
  - Process Discovery
- **Assignment 1**

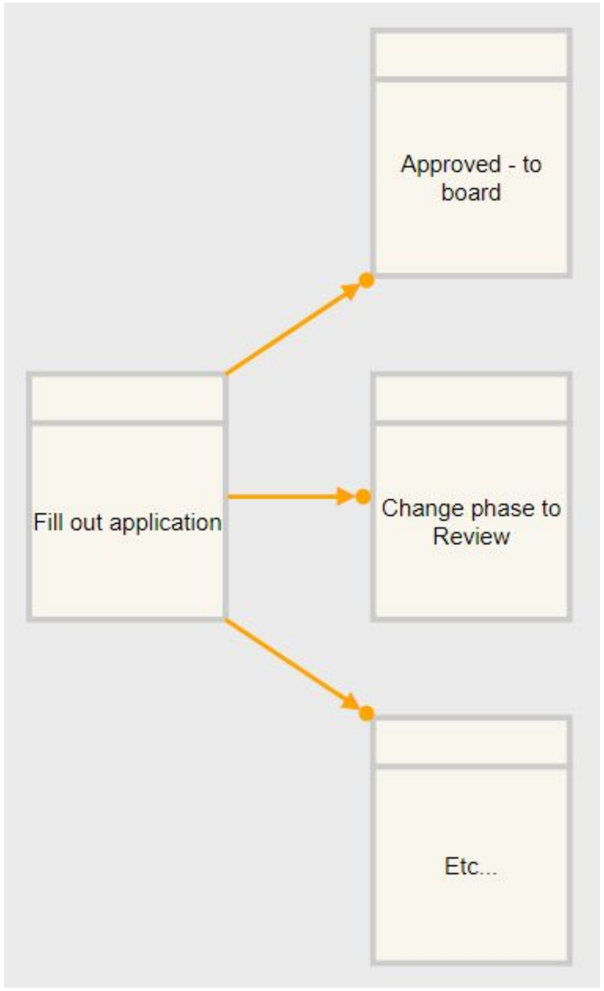
Assignment 1

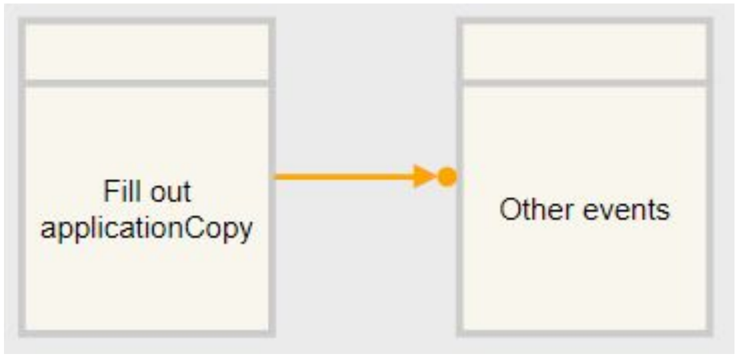
Part A: Hints

- You do not need to model the entire process, only the 4 individual patterns.
- They are meant to be very straightforward / simple.

Assignment 1

- First pattern: *Fill out application should always be the first event of the case*, several ways to model:





Note: Option 2 requires that you have a special case for "Other events" in your conformance checker (part 2).

Assignment 1

Part 2: Implementing a conformance checker for DCR Graphs

Implement a simple conformance checker for DCR Graphs and use it to check the patterns defined in Part 1 against the Dreyers log .
The conformance checker should be able of taking any DCR Graph and log as input, i.e. it will not suffice to hard code the patterns of part 1. For the logs we advise that you use the CSV format of the Dreyers log. For DCR Graphs we leave the input format to you.
As output the program should show for each pattern how many traces in the log satisfy it, and how many do not.

Assignment 1

Part 2: Implementing a conformance checker for DCR Graphs

In principle you may use any language you like, but for your own convenience you’re strongly encouraged to use the sample Javascript engine provided. Note that the code you hand-in should compile and run. If for some reason we cannot compile and run your code on our own machines (e.g. because you used a different programming language), you may be asked to demonstrate that it works on your own machine.

Optional: Have your program output for each trace that does not satisfy a pattern the reason that this is the case.

Assignment 1

Hints for Part 2:

1) Your output will look something like this:

	Satisfied	Not satisfied
Pattern 1	10	20
Pattern 2	15	15
Pattern 3	19	11
Pattern 4	5	25

Assignment 1: Hand-in

1. A ZIP file containing, for Part 1:
   a. One PNG image file for each pattern, named “pattern_x.png”
   b. *Or* one text file for each pattern following the language format used in the sample JavaScript program.
2. A second ZIP file containing, for Part 2:
   a. Your source code.
   b. *And* a script or batch file that runs your program on the 4 patterns of Part 1 and the Dreyers log.
3. A 2-3 page PDF report containing:
   a. A front page stating your group name (“assignments x”) and the names of the group members that contributed to the submission. (Not counted towards the 2-3 pages.)
   b. A short description of how you came to model each pattern (a couple of lines per pattern will suffice).
   c. A short description of how you implemented the conformance checker (please include an overview of your main design decisions).
   d. A screenshot of your program running, including the output of results.
   e. A table showing for each pattern how many traces satisfied it, and how many did not.

The two-page report must use 11-point font of the Times family and be formatted single-column with 2 cm margin.

The Dreyers Log

```
ID;Event;Title;Role;Date;EventName;EventType
14a-027_0;DL_WFCaseTypeStep|497;Fill out application;AnsÃ¸ger;2013-12-25 12:36:38.987;Fill out application;Task
14a-027_0;DL_WFCaseTypeStep|469;Approved - to board;ls;2013-12-30 01:41:21.143;Approved - to board;Task
14a-027_0;DL_WFCaseTypeStep|518;Change phase to Review;suser;2013-12-30 01:41:22.717;Change phase to Review;Task
14a-027_0;DL_WFCaseTypeStep|472;Architect Review;mp;2013-12-30 03:34:46.273;Architect Review;Task
14a-027_0;DL_WFCaseTypeStep|472;Architect Review;mp;2013-12-30 03:35:19.810;Architect Review;Task
14a-027_0;DL_WFCaseTypeStep|474;Review;bry;2014-02-11 17:07:09.263;Review;Task
....
```

Question: Which are the most relevant columns?

The Dreyers Log

Cases / Traces

```
ID;Event;Title;Role;Date;EventName;EventType
14a-027_0;DL_WFCaseTypeStep|497;Fill out application;AnsÃ¸ger;2013-12-25 12:36:38.987;Fill out application;Task
14a-027_0;DL_WFCaseTypeStep|469;Approved - to board;ls;2013-12-30 01:41:21.143;Approved - to board;Task
14a-027_0;DL_WFCaseTypeStep|518;Change phase to Review;suser;2013-12-30 01:41:22.717;Change phase to Review;Task
14a-027_0;DL_WFCaseTypeStep|472;Architect Review;mp;2013-12-30 03:34:46.273;Architect Review;Task
14a-027_0;DL_WFCaseTypeStep|472;Architect Review;mp;2013-12-30 03:35:19.810;Architect Review;Task
14a-027_0;DL_WFCaseTypeStep|474;Review;bry;2014-02-11 17:07:09.263;Review;Task
....
```

The Dreyers Log

Events

```
ID;Event;Title;Role;Date;EventName;EventType
14a-027_0;DL_WFCaseTypeStep|497;Fill out application;AnsÃ¸ger;2013-12-25 12:36:38.987;Fill out application;Task
14a-027_0;DL_WFCaseTypeStep|469;Approved - to board;ls;2013-12-30 01:41:21.143;Approved - to board;Task
14a-027_0;DL_WFCaseTypeStep|518;Change phase to Review;suser;2013-12-30 01:41:22.717;Change phase to Review;Task
14a-027_0;DL_WFCaseTypeStep|472;Architect Review;mp;2013-12-30 03:34:46.273;Architect Review;Task
14a-027_0;DL_WFCaseTypeStep|472;Architect Review;mp;2013-12-30 03:35:19.810;Architect Review;Task
14a-027_0;DL_WFCaseTypeStep|474;Review;bry;2014-02-11 17:07:09.263;Review;Task
....
```

The Dreyers Log

Timestamps

```
ID;Event;Title;Role;Date;EventName;EventType
14a-027_0;DL_WFCaseTypeStep|497;Fill out application;AnsÃ¸ger;2013-12-25 12:36:38.987;Fill out application;Task
14a-027_0;DL_WFCaseTypeStep|469;Approved - to board;ls;2013-12-30 01:41:21.143;Approved - to board;Task
14a-027_0;DL_WFCaseTypeStep|518;Change phase to Review;suser;2013-12-30 01:41:22.717;Change phase to Review;Task
14a-027_0;DL_WFCaseTypeStep|472;Architect Review;mp;2013-12-30 03:34:46.273;Architect Review;Task
14a-027_0;DL_WFCaseTypeStep|472;Architect Review;mp;2013-12-30 03:35:19.810;Architect Review;Task
14a-027_0;DL_WFCaseTypeStep|474;Review;bry;2014-02-11 17:07:09.263;Review;Task
....
```

Questions?