Process Mining (ProMi)

Winter 2015/16

Matthias Weidlich

Jan Sürmeli (Substitute)
II. Process Discovery

Mining Algorithms

The Family of Alpha-Algorithms
The Context

process-aware information system

models analyzes

specifies configures implements analyzes

supports/controls

records events, e.g., messages, transactions, etc.

process models

discovery

conformance

extension

event logs
Process Model Discovery

Discovery

• Create process model for observed behaviour
• But: typically not every possible behaviour (i.e., trace) may have been executed and thus recorded

Quality dimensions

• Fitness: model should allow for behaviour seen in log
• Precision – Generalisation trade-off: model does not allow for behaviour completely unrelated to log, but generalizes to some extent what is seen in the log
• Simplicity: discovered model should be as simple as possible
Discovery Example

\[ L_1 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^2, \langle a, e, d \rangle] \]

\( L_1 \) contains the sequence \( <a,b,c,d> \) three times
Discovery Example

\[ L_1 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^2, \langle a, e, d \rangle] \]
Discovery Example

\[ L_1 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^2, \langle a, e, d \rangle] \]
Discovery Example

\[ L_1 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^2, \langle a, e, d \rangle]\]
Discovery Example

\[ L_1 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^2, \langle a, e, d \rangle] \]
Discovery Example

$L_1 = [[a, b, c, d]^3, [a, c, b, d]^2, [a, e, d]]$
Discovery Example

Every trace starts with “a”

\[ L_2 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^4, \langle a, b, c, e, f, b, c, d \rangle^2, \langle a, b, c, e, f, c, b, d \rangle, \langle a, c, b, e, f, b, c, d \rangle^2, \langle a, c, b, e, f, b, c, e, f, c, b, d \rangle] \]
Discovery Example

Every trace ends with “d”

\[ L_2 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^4, \langle a, b, c, e, f, b, c, d \rangle^2, \langle a, b, c, e, f, c, b, d \rangle, \langle a, c, b, e, f, b, c, d \rangle^2, \langle a, c, b, e, f, b, c, e, f, c, b, d \rangle] \]
Discovery Example

“b” and “c” always occur together without a particular order

\[ L_2 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^4, \langle a, b, c, e, f, b, c, d \rangle^2, \langle a, b, c, e, f, c, b, d \rangle, \langle a, c, b, e, f, b, c, d \rangle^2, \langle a, c, b, e, f, b, c, e, f, c, b, d \rangle] \]
Discovery Example

Every “e” is followed by an “f”, every “f” is preceded by an “e”

\[ L_2 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^4, \langle a, b, c, e, f, b, c, d \rangle^2, \langle a, b, c, e, f, c, b, d \rangle, \langle a, c, b, e, f, b, c, d \rangle^2, \langle a, c, b, e, f, b, c, e, f, c, b, d \rangle] \]
Discovery Example

$L_2 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^4, \langle a, b, c, e, f, b, c, d \rangle^2, \langle a, b, c, e, f, c, b, d \rangle, \\
\langle a, c, b, e, f, b, c, d \rangle^2, \langle a, c, b, e, f, b, c, e, f, c, b, d \rangle]$
Discovery Example

Every trace starts with “a”

\[ L_2 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^4, \langle a, b, c, e, f, b, c, d \rangle^2, \langle a, b, c, e, f, c, b, d \rangle, \langle a, c, b, e, f, b, c, d \rangle^2, \langle a, c, b, e, f, b, c, e, f, c, b, d \rangle] \]
Every trace ends with “d”

\[ L_2 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^4, \langle a, b, c, e, f, b, c, d \rangle^2, \langle a, b, c, e, f, c, b, d \rangle, \langle a, c, b, e, f, b, c, d \rangle^2, \langle a, c, b, e, f, b, c, e, f, c, b, d \rangle] \]
Discovery Example

“b” and “c” always occur together without a particular order

\[ L_2 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^4, \langle a, b, c, e, f, b, c, d \rangle^2, \langle a, b, c, e, f, c, b, d \rangle, \langle a, c, b, e, f, b, c, d \rangle^2, \langle a, c, b, e, f, b, c, e, f, c, b, d \rangle] \]
Discovery Example

Every “e” is followed by an “f”, every “f” is preceded by an “e”

$L_2 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^4, \langle a, b, c, e, f, b, c, d \rangle^2, \langle a, b, c, e, f, c, b, d \rangle, \\
\langle a, c, b, e, f, b, c, d \rangle^2, \langle a, c, b, e, f, b, c, e, f, c, b, d \rangle]$
Discovery Algorithms

Simple algorithm: $\alpha$ – algorithm
- Detect concurrent execution of activities
- Relatively easy, certain properties can be proven
- Not robust against *noise*, incomplete or erroneous logs
- Therefore, nice for illustration, but limited use in practice

Extensions: $\alpha+(+)$ – Algorithms
- $\alpha+$ and $\alpha++$ extend $\alpha$ – Algorithm to broaden the spectrum of constructs that may be discovered
- Still not robust against *noise*

Robust algorithms will be discussed later

Workflow Log

General notion of event log

- Format of log entries:
  (timestamp, caseID, activity, additional attributes ...)
- Various abstractions can be applied

One abstraction: the notion of a workflow log

- Set of all distinct sequences of activity executions
- Let $T$ be a set of activities (aka tasks)
- Let $T^*$ the set of all finite sequences over $T$
- $\sigma \in T^*$ is a trace, all tasks in $\sigma$ belong to the same case
- $W \subseteq T^*$ is a workflow log

Assumption: Each task occurs at most once in a process model
Workflow Log

Traces:
Case 1: ABCD
Case 2: ACBD
Case 3: ABCD
Case 4: ACBD
Case 5: EF

Workflow log:
\[ W = \{ABCD, ACBD, EF\} \]
Ordering Relations

Log-based ordering relations for a pair of tasks $a, b \in T$ in a workflow log $W$:

- **Direct successor**
  \[ a >_W b \text{ iff } b \text{ directly follows } a \text{ in at least one trace} \]

- **Causality**
  \[ a \rightarrow_W b \text{ iff } a >_W b \text{ and not } b >_W a \]

- **Concurrency**
  \[ a \parallel_W b \text{ iff } a >_W b \text{ and } b >_W a \]

- **Exclusiveness**
  \[ a \#_W b \text{ iff not } a >_W b \text{ and not } b >_W a \]
Example

Workflow log:
\( W = \{ \text{ABCD, ACBD, EF} \} \)

1) A>B
   A>C
   B>C
   B>D
   C>B
   C>D
   E>F

2) A→B
   A→C
   B→D
   C→D
   E→F

3) B || C
   C || B

4) A # A
   A # D
   A # E
   A # F
   B # B
   B # E
   B # F
   C # C
   C # E
   C # F
   D # D
   D # E
   D # F
   E # E
   F # F
   ...

   | case 1 | task A |
   | case 2 | task A |
   | case 3 | task A |
   | case 3 | task B |
   | case 1 | task B |
   | case 1 | task C |
   | case 2 | task C |
   | case 4 | task A |
   | case 2 | task B |
   | case 2 | task D |
   | case 5 | task E |
   | case 4 | task C |
   | case 1 | task D |
   | case 3 | task C |
   | case 3 | task D |
   | case 4 | task B |
   | case 5 | task F |
   | case 4 | task D |
α-Algorithm Idea

Idea: create workflow net based on the ordering relations, such that the ordering is obeyed by the net

Realisation: derive a Petri net fragment from the each entry of the ordering relations
\(\alpha\)-Algorithm Idea cont.

\[ x \rightarrow y, x \rightarrow z \]
and \( y \parallel z \)

\[ x \rightarrow y, x \rightarrow z \]
and \( y \# z \)

AND-split

XOR-split
α-Algorithm Idea cont.

\[ x \rightarrow z, \quad y \rightarrow z \]
and \[ x || y \]

\[ X \rightarrow z, \quad y \rightarrow z \]
and \[ x \# y \]
α-Algorithm

Let $W$ be a workflow log over $T$. $\alpha(W)$ is defined as follows:

1. $T_W = \{ t \in T \mid \exists_{\sigma \in W} t \in \sigma \}$,
2. $T_I = \{ t \in T \mid \exists_{\sigma \in W} t = \text{first}(\sigma) \}$,
3. $T_O = \{ t \in T \mid \exists_{\sigma \in W} t = \text{last}(\sigma) \}$,
4. $X_W = \{ (A,B) \mid A \subseteq T_W \land B \subseteq T_W \land \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \land \forall_{a_1,a_2 \in A} a_1 \#_W a_2 \land \forall_{b_1,b_2 \in B} b_1 \#_W b_2 \}$,
5. $Y_W = \{ (A,B) \in X \mid \forall_{(A',B')} \in X A \subseteq A' \land B \subseteq B' \Rightarrow (A,B) = (A',B') \}$,
6. $P_W = \{ p_{(A,B)} \mid (A,B) \in Y_W \} \cup \{ i_W, o_W \}$,
7. $F_W = \{ (a,p_{(A,B)}) \mid (A,B) \in Y_W \land a \in A \} \cup \{ (p_{(A,B)},b) \mid (A,B) \in Y_W \land b \in B \} \cup \{ (i_W,t) \mid t \in T_I \} \cup \{ (t,o_W) \mid t \in T_O \}$,
8. $\alpha(W) = (P_W, T_W, F_W)$. 
**α-Algorithm**

Let $W$ be a workflow log over $T$. $\alpha(W)$ is defined as follows:

1. $T_W = \{ t \in T \mid \exists_{\sigma \in W} t \in \sigma \}$,
2. $T_I = \{ t \in T \mid \exists_{\sigma \in W} t = \text{first}(\sigma) \}$,
3. $T_O = \{ t \in T \mid \exists_{\sigma \in W} t = \text{last}(\sigma) \}$,
4. $X_W = \{ (A,B) \mid A \subseteq T_W \land B \subseteq T_W \land \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \land \forall_{a_1,a_2 \in A} a_1 \not\#_W a_2 \land \forall_{b_1,b_2 \in B} b_1 \not\#_W b_2 \}$,
5. $Y_W = \{ (A,B) \in X \mid \forall_{(A',B') \in X} A \subseteq A' \land B \subseteq B' \Rightarrow (A,B) = (A',B') \}$,
6. $P_W = \{ p_{(A,B)} \mid (A,B) \in Y_W \} \cup \{ i_W, o_W \}$,
7. $F_W = \{ (a,p_{(A,B)}) \mid (A,B) \in Y_W \land a \in A \}
\cup \{ (p_{(A,B)},b) \mid (A,B) \in Y_W \land b \in B \}
\cup \{ (i_W,t) \mid t \in T_I \} \cup \{ (t,o_W) \mid t \in T_O \}$,
8. $\alpha(W) = (P_W, T_W, F_W)$.

**Result is a WF-net:**

- $P_W$ : Set of places
- $T_W$ : Set of transitions
- $F_W$ : Flow relation
α-Algorithm

Let $W$ be a workflow log over $T$. $\alpha(W)$ is defined as follows:

1. $T_W = \{ t \in T \mid \exists \sigma \in W \ t \in \sigma \}$,
2. $T_I = \{ t \in T \mid \exists \sigma \in W \ t = \text{first}(\sigma) \}$,
3. $T_O = \{ t \in T \mid \exists \sigma \in W \ t = \text{last}(\sigma) \}$,
4. $X_W = \{ (A,B) \mid A \subseteq T_W \land B \subseteq T_W \land \forall a_1,a_2 \in A \ a_1 \#_{W} a_2 \land \forall b_1,b_2 \in B \ b_1 \#_{W} b_2 \}$,
5. $Y_W = \{ (A,B) \in X \mid \forall (A',B') \in X \ A \subseteq A' \}$,
6. $P_W = \{ p_{(A,B)} \mid (A,B) \in Y_W \} \cup \{ i_{W}, o_{W} \}$,
7. $F_W = \{ (a,p_{(A,B)}) \mid (A,B) \in Y_W \land a \in A \} \cup \{ (p_{(A,B)},b) \mid (A,B) \in Y_W \land b \in B \} \cup \{ (i_{W},t) \mid t \in T_I \} \cup \{ (t,o_{W}) \mid t \in T_O \}$,
8. $\alpha(W) = (P_W,T_W,F_W)$.

Derive set of transitions $T_W$ from all traces.

$\text{first}(\sigma)$ [$\text{last}(\sigma)$] is the first [last] transition in trace $\sigma$.

$T_I [T_O]$ is the set of all initial [final] transitions.

$W = \{ABCD, ACBD, EF\}$

$T_W = \{A, B, C, D, E, F\}$

$T_I = \{A, E\}$

$T_O = \{D, F\}$
**α-Algorithm**

\[ T_I = \{ A, E \} \]

\[ T_O = \{ D, F \} \]

\[
\{ (i_W, t) \mid t \in T_I \} = \{(i_W, A), (i_W, E)\}
\]

\[
\{ (t, o_W) \mid t \in T_O \} = \{(D, o_W), (F, o_W)\}
\]

5. \[ Y_W = \{ (A,B) \in X \mid \forall (A',B') \in X \ A \subseteq A' \land B \subseteq B' \Rightarrow (A,B) = (A',B') \} \],

6. \[ P_W = \{ p_{(A,B)} \mid (A,B) \in Y_W \} \cup \{ i_W, o_W \}, \]

7. \[ F_W = \{ (a,p_{(A,B)}) \mid (A,B) \in Y_W \land a \in A \} \cup \{ (p_{(A,B)}, b) \mid (A,B) \in Y_W \land b \in B \} \cup \{ (i_W, t) \mid t \in T_I \} \cup \{ (t, o_W) \mid t \in T_O \}, \]

8. \[ \alpha(W) = (P_W, T_W, F_W). \]

\(i_W\) is initial place, \(o_W\) is final place.

Step 7 connects them to transitions in \(T_I\) and \(T_O\).
**α-Algorithm**

Let $W$ be a workflow log over $T$. $\alpha(W)$ is defined as follows:

1. $T_w = \{ t \in T \mid \exists \sigma \in W \ t \in \sigma \}$,
2. $T_I = \{ t \in T \mid \exists \sigma \in W \ t = \text{first}(\sigma) \}$,
3. $T_O = \{ t \in T \mid \exists \sigma \in W \ t = \text{last}(\sigma) \}$,
4. $X_w = \{ (A,B) \mid A \subseteq T_w \wedge B \subseteq T_w \wedge \forall a_1,a_2 \in A \ a_1 \#_W a_2 \wedge \forall b_1,b_2 \in B \ b_1 \#_W b_2 \}$,
5. $Y_w = \{ (A,B) \in X \mid \forall (A',B') \in X \ A \subseteq A' \wedge B \subseteq B' \Rightarrow (A,B) = (A',B') \}$,
6. $P_w = \{ p_{(A,B)} \mid (A,B) \in Y_w \} \cup \{ i_w,o_w \}$,
7. $F_w = \{ (a,p_{(A,B)}) \mid (A,B) \in Y_w \wedge a \in A \} \cup \{ (p_{(A,B)},b) \mid (A,B) \in Y_w \wedge b \in B \} \cup \{ (i_{W},t) \mid t \in T_I \} \cup \{ (t,o_{W}) \mid t \in T_O \}$,
8. $\alpha(W) = (P_w,T_w,F_w)$.

All other places are denoted by $p_{(A,B)}$, with $A$ and $B$ being preceding / succeeding transitions.

Place is created, iff $a \rightarrow_w b$. 
α-Algorithm

Let $W$ be a workflow log over $T$.

1. $T_w = \{ t \in T \mid \exists \sigma \in W \ t \in \sigma \}$
2. $T_I = \{ t \in T \mid \exists \sigma \in W \ t = \text{first}(\sigma) \}$
3. $T_O = \{ t \in T \mid \exists \sigma \in W \ t = \text{last}(\sigma) \}$
4. $X_W = \{ (A,B) \mid A \subseteq T_w \land B \subseteq T_w \land \forall a \in A \forall b \in B \ a \to_W b \land \forall a_1,a_2 \in A \ a_2 \#_W a_1 \land \forall b_1,b_2 \in B \ b_2 \#_W b_1 \}$
5. $Y_W = \{ (A,B) \in X \mid \forall (A',B') \in X \ A \subseteq A' \land B \subseteq B' \Rightarrow (A,B) = (A',B') \}$
6. $P_W = \{ p \in X \mid (A,B) \in X \}$

Some places are joint to represent XOR-splits/joins instead of AND-splits/joins. To distinguish those, relations $X_W$ and $Y_W$ are defined. $(A,B) \in X_W$ if causality between all $A$ and all $B$ and elements in $A$ ($B$) are never direct successors.

| A→B | B||C |
| A→C | C||B |
| B→D | |

$T_W = \{ A, B, C, D, E, F \}$

$X_W = \{ (\{A\},\{B\}), (\{A\},\{C\}), (\{B\},\{D\}), (\{C\},\{D\}), (\{E\},\{F\}) \}$

Since $B||C$, tuple $(\{A\}, \{B,C\})$ and $(\{B,C\},\{D\})$ are not in $X_W$
α-Algorithm

Let $W$ be a workflow log over $T$. $\alpha(W)$ is defined as follows:

1. $T_W = \{ t \in T \mid \exists \sigma \in W \ t \in \sigma \}$,
2. $T_I = \{ t \in T \mid \exists \sigma \in W \ t = \text{first}(\sigma) \}$,
3. $T_O = \{ t \in T \mid \exists \sigma \in W \ t = \text{last}(\sigma) \}$,
4. $X_W = \{ (A,B) \mid A \subseteq T_W \land B \subseteq T_W \land \forall a \in A \forall b \in B \ a \rightarrow_W b \land \forall a_1,a_2 \in A \ a_1 \#_W a_2 \land \forall b_1,b_2 \in B \ b_1 \#_W b_2 \}$,
5. $Y_W = \{ (A,B) \in X \mid \forall (A',B') \in X A \subseteq A' \land B \subseteq B' \Rightarrow (A,B) = (A',B') \}$,
6. $P_W = \{ p_{(A,B)} \mid (A,B) \in Y_W \} \cup \{ i_W, o_W \}$,
7. $F_W = \{ (a,p_{(A,B)}) \mid (A,B) \in Y_W \land a \in A \} \cup \{ (p_{(A,B)},b) \mid (A,B) \in Y_W \land b \in B \} \cup \{ (i_W,t) \mid t \in T_I \} \cup \{ (t,o_W) \mid t \in T_O \}$,

$Y_W = \{ ((A),{B})), ((A),{C})), ((B),{D})), ((C),{D})), ((E),{F})) \}$

It holds $Y_W = X_W$ since $\forall (A,B) \in X_W : |A| = 1 \land |B| = 1$
\( \alpha \)-Algorithm

\[ Y_W = \{ (\{A\},\{B\}), (\{A\},\{C\}), (\{B\},\{D\}), (\{C\},\{D\}), (\{E\},\{F\}) \}\]

\[ P_W = \{ p_{(\{A\},\{B\})}, p_{(\{A\},\{C\})}, p_{(\{B\},\{D\})}, p_{(\{C\},\{D\})}, p_{(\{E\},\{F\})}, i_W, o_W \}\]

\[ F_W = \{ (A,p_{(\{A\},\{B\})}), (A,p_{(\{A\},\{C\})}), (B,p_{(\{B\},\{D\})}), (C,p_{(\{C\},\{D\})}), (E,p_{(\{E\},\{F\})}), (p_{(\{A\},\{B\})},B), (p_{(\{A\},\{C\})},C), (p_{(\{B\},\{D\})},D), (p_{(\{C\},\{D\})},D), (p_{(\{E\},\{F\})},F), (i_W,A), (i_W,E), (D,o_W), (F,o_W) \}\]

3. \( I_O = \{ t \in T \mid \exists \sigma \in W \ t = last(\sigma) \}\),

4. \( X_W = \{ (A,B) \mid A \subseteq T_W \land B \subseteq T_W \land \forall a \in A \forall b \in B \ a \rightarrow_W \ b \land \forall a_1,a_2 \in A \ a_1 \#_W a_2 \land \forall b_1,b_2 \in B \ b_1 \#_W b_2 \}\)

5. \( Y_W = \{ (A,B) \in X \mid \forall (A',B') \in X \ A \subseteq A' \land \forall a \in A \forall b \in B \ a \rightarrow_W \ b \}\)

6. \( P_W = \{ p_{(A,B)} \mid (A,B) \in Y_W \} \cup \{ i_W, o_W \}\),

7. \( F_W = \{ (a,p_{(A,B)}) \mid (A,B) \in Y_W \land a \in A \} \cup \{ (p_{(A,B)},b) \mid (A,B) \in Y_W \land b \in B \} \cup \{ (i_W,t) \mid t \in T_I \} \cup \{ (t,o_W) \mid t \in T_O \}\)

8. \( \alpha(W) = (P_W,T_W,F_W) \).

Use \( Y_W \) to create places and the flow relation.
α-Algorithm Example

case 1 : task A
case 2 : task A
case 3 : task A
case 3 : task B
case 1 : task B
case 1 : task C
case 2 : task C
case 4 : task A
case 2 : task B
case 2 : task D
case 5 : task E
case 4 : task C
case 1 : task D
case 3 : task C
case 3 : task D
case 4 : task B
case 5 : task F

a(W):

![Diagram of an α-Algorithm with tasks A, B, C, D, E, and F connected by arrows indicating dependencies and states.](image-url)
Log Completeness

Which information is required to be in the log?

- Consider the example given for the α – algorithm and assume trace EF for instance 5 is missing in the log
- Then, the correct model cannot be created
- Assume that every traces has to occur in the log?
  - Problem: number of required instances increases dramatically
  - Consider the parallel execution of 10 tasks
  - This yields 10! = 3.628.800 possible traces!
Log Completeness

Required completeness for the $\alpha$-algorithm

- Completeness w.r.t. direct successorship ($\succ_w$)
- Whenever two tasks may occur as direct successors, this must be observed in at least one trace
- Example: concurrent execution of five tasks, A,B,C,D,E
- If ABCDE and BACED are observed, BACDE is not required for log completeness!

This completeness criterion reduced the number of required traces for highly concurrent processes dramatically
Rediscovery Problem

For which class of WF-nets can we guarantee that WF\(_1\) and WF\(_2\) are equivalent if logging is complete according to introduced notion?

Structural and behavioural assumptions on WF\(_1\)
Implicit Places

The presence or absence of implicit places does not change the behaviour of a net system

• Hence, process models with implicit places cannot be re-discovered
• Not really an issue, since there are no consequences for the behaviour
Structured Workflow-Nets

Structured WF-nets (SWF-nets) are structural subclass:

Definition 4.3. (SWF-net) A WF-net $N = (P, T, F)$ is an SWF-net (Structured workflow net) iff:

1. For all $p \in P$ and $t \in T$ with $(p, t) \in F$: $|p \bullet| > 1$ implies $|\bullet t| = 1$.
2. For all $p \in P$ and $t \in T$ with $(p, t) \in F$: $|\bullet t| > 1$ implies $|\bullet p| = 1$.
3. There are no implicit places.

Note:
Sufficiently expressive to model most process-related control-flow structures, sequences, concurrency, exclusive choices, etc.
Soundness

Behavioural correctness criterion for WF-nets

- Processes terminate in proper final state
- Final state is indeed characterised unambiguously
- All activities can contribute to process execution

Recall: WF-net has initial place $i$ and final place $o$

- Overload notation and refer to $i$ and $o$ also as the markings that put one token into $i$ and $o$, respectively, and no token in any other place
**Soundness Definition**

**Definition 6.3** A workflow system \((PN, i)\) with a workflow net \(PN = (P, T, F)\) is **sound**, if and only if

- For every state \(M\) reachable from state \(i\) there exists a firing sequence leading from \(M\) to \(o\), i.e.,
  \[
  \forall M(i \rightarrow^* M) \implies (M \rightarrow^* o)
  \]
- State \(o\) is the only state reachable from state \(i\) with at least one token in place \(o\), i.e.,
  \[
  \forall M(i \rightarrow^* M \land M \geq o) \implies M = o
  \]
- There are no dead transitions in the workflow net in state \(i\), i.e.,
  \[
  (\forall t \in T) \; \exists M, M' : i \rightarrow^* M \xrightarrow{t} M'
  \]
What can be rediscovered?

Consider Loops

Original net:
What can be rediscovered?

Short loops are a problem:

Original net:

Discovered net

α-algorithm
Short Loops

Short loops (length 1 or 2) pose an issue for rediscoverability by the $\alpha$-algorithm:
What can be rediscovered?

There is a proof for: The $\alpha$-algorithm can rediscover a sound SWF-net without short loops, if the event log is complete according to introduced notion (based on direct successorship)

Next:

- Extend $\alpha$-algorithm, so that sound SWF-nets with short loops can be rediscovered
- This yields the $\alpha^+$-algorithm
**α+ - Algorithm**

Loops of length 2

- Issue: concurrent execution of activities and their execution in a loop cannot be distinguished
- Approach: adapt log completeness criterion
- Completeness so far: all direct successorships $xy$ of transitions are observed at least once
- Loop completeness: all successorship triples of transitions $xyx$ are observed at least once

Example for WF-net system

- $...cdc...$
- $...dcd...$
α+ - Algorithm

Redefine ordering relations

• Direct successor $a >_w b$ and exclusiveness $a \#_w b$ as before

• Causality
  
  $a \rightarrow_w b$ iff $a >_w b$ and (not $b >_w a$ or there exists a trace containing $aba$)

• Concurrency
  
  $a \parallel_w b$ iff $a >_w b$ and $b >_w a$ and there exists no trace containing $aba$

Hence: loops of length 2 and concurrency can be distinguished
α+ - Algorithm

Issue: Loops of length 2 and 1 cannot be distinguished

Both nets produce

- ...cdc...
- ...cdcd...
α+ - Algorithm

Solution to cope with loops of length 1

• Pre-processing: identify transitions in loop of length 1 by sequence \(...a a\)...
• Remove all these transitions from all observed sequences
• Mining: Proceed as before with redefined ordering relations and the assumption of loop completeness of the log
• Post-processing: insert loops of length 1 into constructed net
• Find place \(p\) to insert loop for transition \(a\)
  • Find transitions, which are direct predecessors of \(a\), but not direct successors of \(a\)
  • Find transitions, which are direct successors of \(a\), but not direct predecessors of \(a\)
  • Input/output places of these transitions indicate where the loop has to be inserted
\( \alpha+ - \text{Algorithm} \)

Log:
- \( ab, acdcb, acccb, adcddb, adddb \)

After pre-processing:
- \( ab, ab, ab, ab, ab \)
- Short loops of length 1 for \( c \) and \( d \)

Ordering relations:
- \( a \rightarrow b \)

Net constructed by presented algorithm

Net after post-processing
α++ - Algorithm

α+ - algorithm fails to rediscover indirect dependencies imposed by non-free choice constructs (excluded by the notion of SWF-nets)

Example:
- Dependencies between A and D, and B and E
- But: ...AD... and ...BE... are never observed to discover these dependencies
- Complete log: ACD, BCE
Direct vs. indirect dependencies:

- So far, consider only direct dependency in terms of successorship relation of transitions a and b
  - Yields: output place of a is input place of b

- Next: consider indirect dependency between transitions a and b
  - Output place of a is input place of b
  - No reachable marking, such that firing of a in this marking enables b
  - Reachable marking for which holds: firing of a *can lead to marking*, in which b is enabled
α++ - Algorithm

Additional behavioural relations

- XOR-split relation
  \[ a \lessmid b \iff a \neq b \text{ and there exists a } c \text{ such that } c \rightarrow a \text{ and } c \rightarrow b \]

- XOR-join relation
  \[ a \mid> b \iff a \neq b \text{ and there exists a } c \text{ such that } a \rightarrow c \text{ and } b \rightarrow c \]

- Indirect successor
  \[ a \gg b \iff \neg a > b \text{ and there exists a reachable firing sequence in which } a \text{ happens before } b, \text{ such that there is no XOR-split or -join between } a \text{ and } b \]

Example

- Complete log: ACD, BCE
- A |> B, D <| E, A >> D, B >> E
α++ - Algorithm

Approach
• Based on additional relations, identify implicit dependencies
• These implicit dependencies lead to additional places and flow arcs in the net system
• These places may redundant in the sense that they do not change the behaviour of the net system
• If so, they are removed as part of post-processing

Result
• Many non-free-choice constructs are discovered
• But: no formal result on completeness for this net class
Take Away

Family of $\alpha$ – algorithms provide a set of basic discovery algorithms

Focus particularly on the detection of concurrency

Formal proof on rediscoverability

Not robust against noise, and thus, not really relevant from a practical point of view