Willkommen zur Vorlesung

Methodische Grundlagen des Software-Engineering

im Sommersemester 2012

Prof. Dr. Jan Jürjens

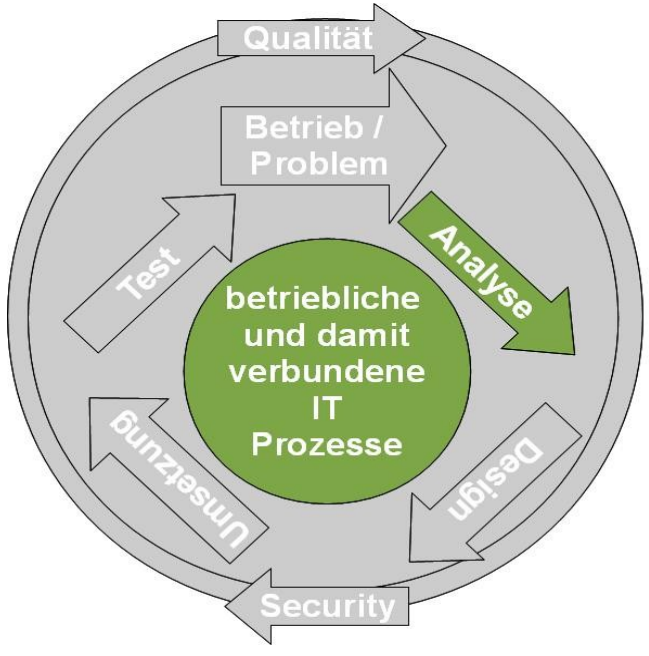
TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

2.4 Business-Process-Mining

[inkl. Beiträge von Jutta Mülle und Dr. Silvia von Stackelberg, LS Prof. Böhm, Karlsruher Institut für Technologie]

Einordnung
Business-Process-Mining

- Anwendungsbeispiel Finanz- und Versicherungsdomäne
- Geschäfts-Prozesse
  - Grundlagen Geschäfts-Prozesse
  - Einführung in die BPMN
  - Elektronische Prozessketten
  - Grundlagen der GP-Modellierung: Petri-Netze
  - Business-Process-Mining
  - Workflow-Management-Systeme
  - Workflow-Automatisierung
- Qualitätsmanagement
- Testen
- Sicherheit
- Sicheres Software Design



Überblick Business-Process-Mining

- Einführung und Beispiel
- α-Algorithmus
  - Idee und Vorbereitungen
  - Formalisierung und Beispiel
- Workflow-Diagnose und Graph-Mining

Problem

- Geschäftsprozessmodellierung oft nicht vorhanden oder nicht vollständig bzw. aktuell.
- Wird die GP-Dokumentation in der Praxis überhaupt befolgt ?

Lösung: Business-Process-Mining

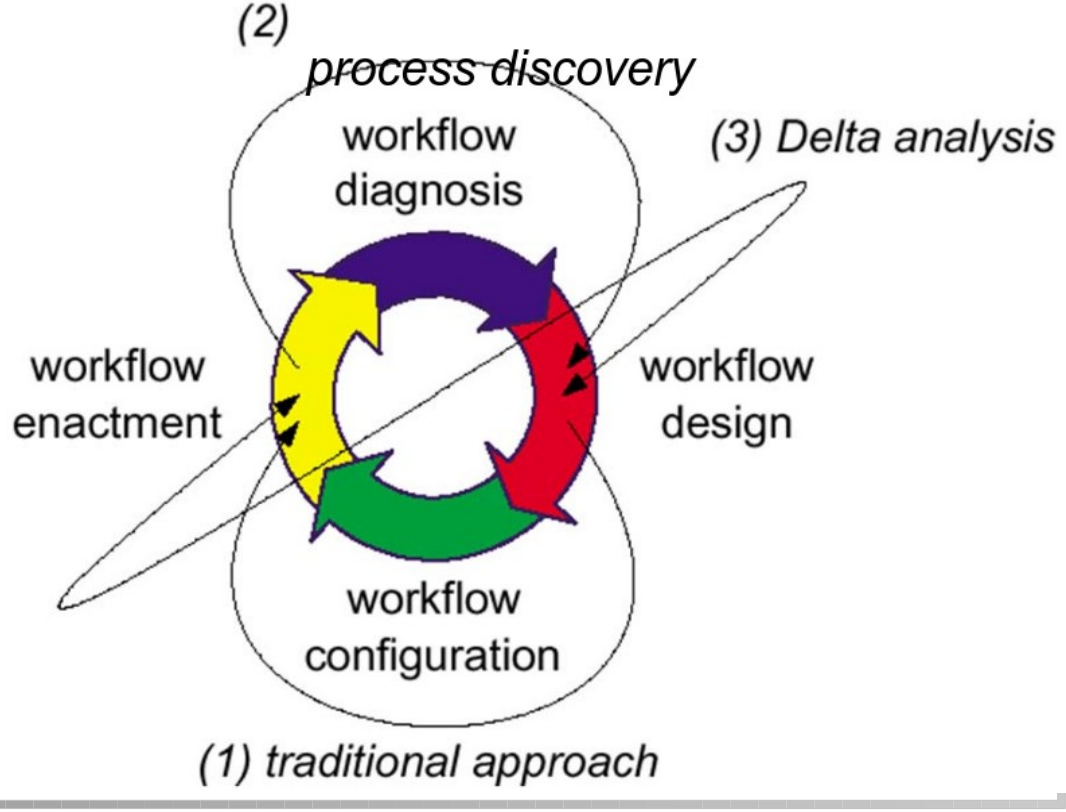
- Ansatz für (teil-)automatisierte Extraktion von GP-Modellen aus Laufzeitdaten (z.B. Log-Daten).
- Kann als Input für Business Process Reengineering (BPR) und Continuous Process Improvement (CPI) genutzt werden.
- Synonym: Workflow-Mining.

Workflow-Lebenszyklus

Klassischer „Lebenszyklus“ von Workflows:

- Workflow design: Entwurf des Workflows
- Workflow configuration: Konfiguration des Workflows
- Workflow enactment: Ausführung des Workflows
- Workflow diagnosis: Diagnose des ausgeführten Workflows

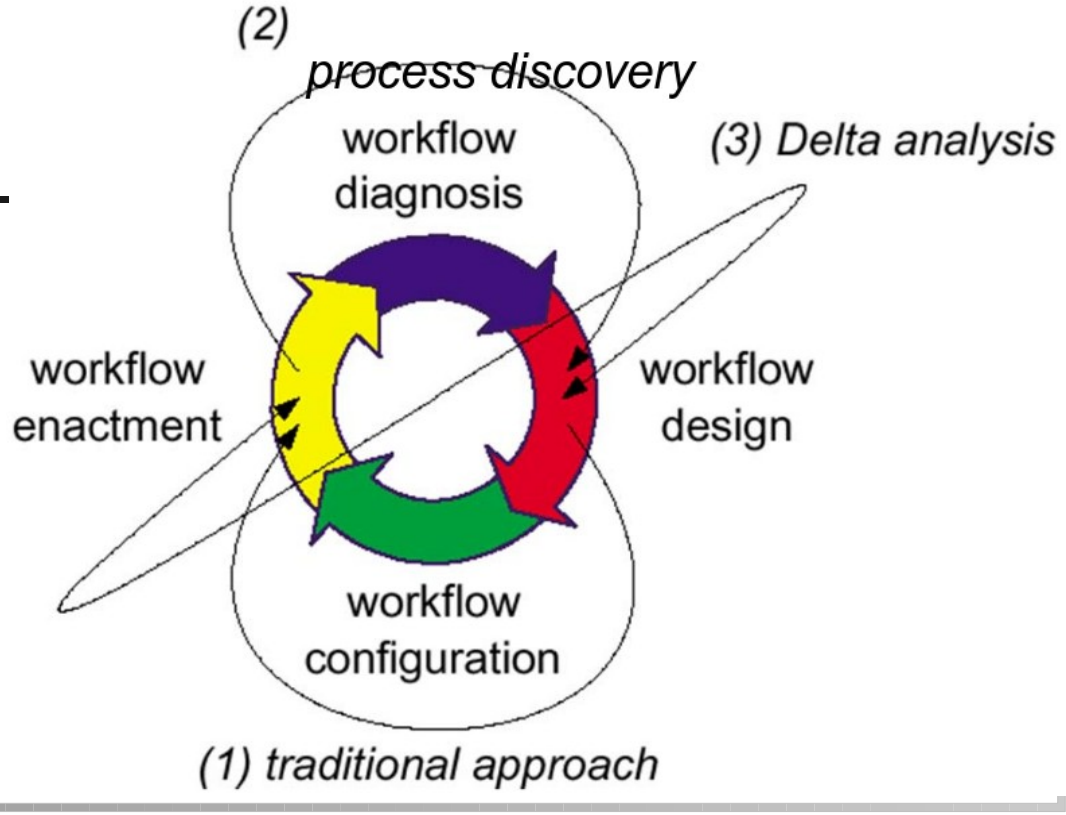
Klassisches Vorgehen (1): Workflow-Ausführung auf Basis des Workflow-Designs.



Workflow-Mining

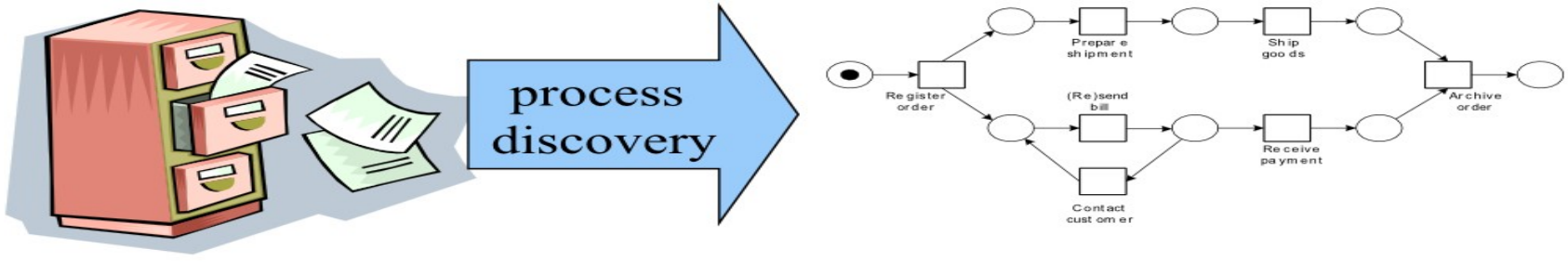
Ansätze auf Basis von Workflow-Mining:

- Process Discovery (2): Extraktion des Workflow-Designs für den tatsächlich ausgeführten Workflow. (Wie sieht der Prozess (wirklich) aus ?)
- Delta Analysis (3): Vergleich des ursprünglichen Workflow-Designs mit dem tatsächlich ausgeführten Workflow. (Passiert das, was spezifiziert wurde ?)



Process-Discovery

Idee: Umdrehen des traditionellen Ansatzes



Ausgangspunkt: Logfile (= Folge von Log-Daten)

- Minimal: Nummer der GP-Instanz („case“ / Fall), GP-Aktivität, zeitliche Ordnung
- Optional: Genaue Zeit, Nutzer, assoziierte Daten etc.
- „Rauschfrei“: GP-Instanznr. erlaubt es, irrelevante Daten wegzufiltern.
- Vollständig: Alle relevante Daten für die verschiedenen GP-Instanzen sind vorhanden.

Ergebnis: Workflow-Modell als Petri-Netz

Beispiel-Logfile

Die abgebildete Logdatei enthält folgende Folgen von Logdaten für die verschiedene GP-Instanzen (cases):

Log-Datei:

```
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 A
case 4, task C
case 1, task D
case 3, task C
case 3, task D
case 4, task B
case 5, task E
case 5, task D
case 4, task D
```

Zugehöriges Petri-Netz:

- Abstrahiert von „cases“ (insbes. von mehreren gleichen Abläufen).
- (Relativ) „Minimal“: nicht einfach o.g. Folgen durch OR-Verzweigungen auflisten.

Beispiel-Logfile

Die abgebildete Logdatei enthält folgende Folgen von Logdaten für die verschiedene GP-Instanzen (cases):

- ABCD (case 1,

Zugehöriges Petri-Netz:

- Abstrahiert von „cases“ (insbes. von mehreren gleichen Abläufen).
- (Relativ) „Minimal“: nicht einfach o.g. Folgen durch OR-Verzweigungen auflisten.

Log-Datei:

```
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 A
case 4, task C
case 1, task D
case 3, task C
case 3, task D
case 4, task B
case 5, task E
case 5, task D
case 4, task D
```

Beispiel-Logfile

Die abgebildete Logdatei enthält folgende Folgen von Logdaten für die verschiedene GP-Instanzen (cases):

- ABCD (case 1, case 3)
- ACBD (case 2, case 4)
- AED (case 5)

Zugehöriges Petri-Netz:

Log-Datei:

```
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 A
case 4, task C
case 1, task D
case 3, task C
case 3, task D
case 4, task B
case 5, task E
case 5, task D
case 4, task D
```

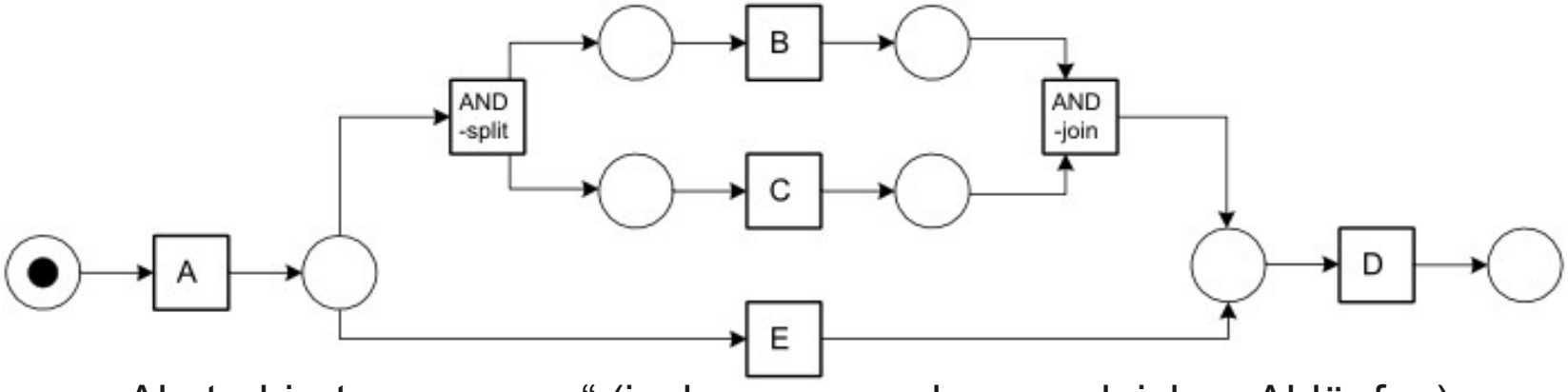
- Abstrahiert von „cases“ (insbes. von mehreren gleichen Abläufen).
- (Relativ) „Minimal“: nicht einfach o.g. Folgen durch OR-Verzweigungen auflisten.

Beispiel-Logfile

Die abgebildete Logdatei enthält folgende Folgen von Logdaten für die verschiedene GP-Instanzen (cases):

- ABCD (case 1, case 3)
- ACBD (case 2, case 4)
- AED (case 5)

Zugehöriges Petri-Netz:



- Abstrahiert von „cases“ (insbes. von mehreren gleichen Abläufen).
- (Relativ) „Minimal“: nicht einfach o.g. Folgen durch OR-Verzweigungen auflisten.

Log-Datei:

```
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 A
case 4, task C
case 1, task D
case 3, task C
case 3, task D
case 4, task B
case 5, task E
case 5, task D
case 4, task D
```

Überblick Business-Process-Mining

- Einführung und Beispiel
- α-Algorithmus
  - Idee und Vorbereitungen
  - Formalisierung und Beispiel
- Workflow-Diagnose und Graph-Mining

α-Algorithmus: Idee

Im Folgenden betrachten wir einen Algorithmus, mit dem aus einer Menge von Folgen von Log-Daten (zu einem gegebenen GP) ein GP-Modell als Petri-Netz extrahiert werden kann („α-Algorithmus“[1]).

Die Idee dabei ist, zunächst einmal die benötigten Informationen über die Abfolge der Aktivitäten (über die gegebene Menge von Folgen von Log-Daten hinweg) durch Verwendung geeignet definierter Ordnungs-Relationen zu sammeln, und auf Basis dieser Relationen dann das Petri-Netz zu konstruieren. Dies betrifft insbesondere die Frage, ob Aktivitäten in verschiedenen Folgen (= verschiedenen Ausführungen des GPs) in einer Kausal-Beziehung stehen, oder parallel bzw. unabhängig voneinander sind.

Dass dieser Algorithmus wirklich wie zu erwarten funktioniert, ist nicht-trivial (sowohl die ursprüngliche Idee zu den verwendeten Ordnungsrelationen, also auch der Korrektheitsbeweis), wird hier aber nicht betrachtet.

[1]van der Aalst, W M P and Weijter, A J M M and Maruster, L (2003). "Workflow Mining: Discovering process models from event logs", IEEE Transactions on Knowledge and Data Engineering, vol 16

α-Algorithmus: Annahmen

Im Folgenden machen wir dazu folgende Annahmen.

Annahmen an die zu erzeugenden Petri-Netze:

- Bezeichnungen der Transitionen unterschiedlich (immer der Fall in Petri-Netzen; erzwingt o.g. „Minimalität“).
- Keine Bogen-Vielfachheiten, Iterationen, Bedingungen an Verzweigungen.
- Verzweigungen nicht beobachtbar.

Annahmen an die Folge von Log-Daten:

- Die Log-Daten wurden alle von dem relevanten GP erzeugt. Die Aufteilung der Log-Daten auf die Folgen in der gegebenen Menge entsprechen den verschiedenen GP-Instanzen (wobei wir von mehrfachen GP-Instanzen mit demselben Verlauf abstrahieren können und abstrahieren daher von den GP-Instanznummern).

Ordnungs-Relationen

Für eine Menge W von Folgen σ von Log-Daten (Folge entspricht einer GP-Instanz) und Elemente x,y in diesen Folgen (mit x≠y) definieren wir:

- Direkte Nachfolge: $\mathbf{x >_w y}$

  $\exists\ \sigma \in W$ worin y direkt auf x folgt
- Kausalität: $\mathbf{x \rightarrow_w y}$

  $x >_w y$ und nicht $y >_w x$
- Parallelität: $\mathbf{x \|_w y}$

  $x >_w y$ und $y >_w x$
- Unabhängigkeit: $\mathbf{x \#_w y}$

  nicht $x >_w y$ und nicht $y >_w x$

- Im Beispiel {ABCD, ACBD, AED}:

[Index W kann weggelassen werden, wenn klar ist, welche Log-Datei gemeint ist.]

Ordnungs-Relationen

Für eine Menge W von Folgen σ von Log-Daten (Folge entspricht einer GP-Instanz) und Elemente x,y in diesen Folgen (mit x≠y) definieren wir:

- Direkte Nachfolge: $\mathbf{x >_w y}$

  $\exists\ \sigma \in W$ worin y direkt auf x folgt
- Kausalität: $\mathbf{x \rightarrow_w y}$

  $x >_w y$ und nicht $y >_w x$
- Parallelität: $\mathbf{x \|_w y}$

  $x >_w y$ und $y >_w x$
- Unabhängigkeit: $\mathbf{x \#_w y}$

  nicht $x >_w y$ und nicht $y >_w x$

- Im Beispiel {ABCD, ACBD, AED}:
- A>B, B>C, C>D, A>C, C>B, B>D, A>E, E>D

[Index W kann weggelassen werden, wenn klar ist, welche Log-Datei gemeint ist.]

Ordnungs-Relationen

Für eine Menge W von Folgen σ von Log-Daten (Folge entspricht einer GP-Instanz) und Elemente x,y in diesen Folgen (mit x≠y) definieren wir:

- Direkte Nachfolge: $\mathbf{x >_w y}$

  $\exists\ \sigma \in W$ worin y direkt auf x folgt
- Kausalität: $\mathbf{x \rightarrow_w y}$

  $x >_w y$ und nicht $y >_w x$
- Parallelität: $\mathbf{x \|_w y}$

  $x >_w y$ und $y >_w x$
- Unabhängigkeit: $\mathbf{x \#_w y}$

  nicht $x >_w y$ und nicht $y >_w x$

- Im Beispiel
  {ABCD, ACBD, AED}:
- A>B, B>C, C>D, A>C, C>B, B>D, A>E, E>D
- A→B, A→C, A→E, B→D, C→D, E→D

[Index W kann weggelassen werden, wenn klar ist, welche Log-Datei gemeint ist.]

Ordnungs-Relationen

Für eine Menge W von Folgen σ von Log-Daten (Folge entspricht einer GP-Instanz) und Elemente x,y in diesen Folgen (mit x≠y) definieren wir:

- Direkte Nachfolge: $\mathbf{x >_w y}$

  $\exists\ \sigma \in W$ worin y direkt auf x folgt
- Kausalität: $\mathbf{x \rightarrow_w y}$

  $x >_w y$ und nicht $y >_w x$
- Parallelität: $\mathbf{x \|_w y}$

  $x >_w y$ und $y >_w x$
- Unabhängigkeit: $\mathbf{x \#_w y}$

  nicht $x >_w y$ und nicht $y >_w x$

- Im Beispiel
  {ABCD, ACBD, AED}:
- A>B, B>C, C>D, A>C, C>B, B>D, A>E, E>D
- A→B, A→C, A→E, B→D, C→D, E→D
- B||C (und symm.)

[Index W kann weggelassen werden, wenn klar ist, welche Log-Datei gemeint ist.]

Ordnungs-Relationen

Für eine Menge W von Folgen σ von Log-Daten (Folge entspricht einer GP-Instanz) und Elemente x,y in diesen Folgen (mit x≠y) definieren wir:

- Direkte Nachfolge: $\mathbf{x} >_w \mathbf{y}$

  $\exists\ \sigma \in W$ worin y direkt auf x folgt
- Kausalität: $\mathbf{x} \rightarrow_w \mathbf{y}$

  $x >_w y$ und nicht $y >_w x$
- Parallelität: $\mathbf{x} \|_w \mathbf{y}$

  $x >_w y$ und $y >_w x$
- Unabhängigkeit: $\mathbf{x} \#_w \mathbf{y}$

  nicht $x >_w y$ und nicht $y >_w x$

- Im Beispiel {ABCD, ACBD, AED}:
- A>B, B>C, C>D, A>C, C>B, B>D, A>E, E>D
- A→B, A→C, A→E, B→D, C→D, E→D
- B||C (und symm.)
- A#D, B#E, C#E (und symm.)

[Index W kann weggelassen werden, wenn klar ist, welche Log-Datei gemeint ist.]

Anmerkungen

I) Kausalität, Parallelität und Unabhängigkeit definieren die vier Möglichkeiten, wie zwei Elemente durch > in Relation stehen können.

II) Die Korrektheit des α-Algorithmus beruht darauf, dass unter den gegebenen Annahmen an die vorliegenden Log-Daten und das zu erzeugende Petri-Netz:

1) Die Folgen von Log-Daten sich eindeutig durch die oben definierten Ordnungs-Relationen charakterisieren lassen.
2) Aus den Ordnungs-Relationen ein („minimales“) Petri-Netz abgeleitet werden kann, sodass die von dem Petri-Netz zur Laufzeit gefeuerten Folgen von Transitionen den ursprünglich gegebenen Folgen von Log-Daten entspricht.

Wir verzichten hier auf einen Beweis dieser beiden (nicht-trivialen) Aussagen.

Idee: Erzeugung eines Petri-Netzes (1)

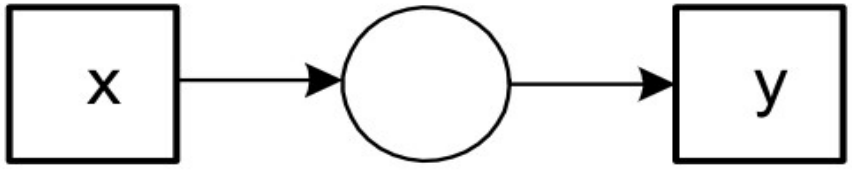
(Wie) kann aus diesen Relationen nun ein Petri-Netz abgeleitet werden, dass diese Aussagen bestätigt ?

Einfachster Fall: Kausalität **x** → **y**
Welches („minimale“) (Teil-)Petri-Netz erzeugt bei Ausführung Folgen von Transitionen, die durch (genau) diese eine Relation charakterisiert sind ?

Idee: Erzeugung eines Petri-Netzes (1)

(Wie) kann aus diesen Relationen nun ein Petri-Netz abgeleitet werden, dass diese Aussagen bestätigt ?

Einfachster Fall: Kausalität **x** → **y**
Welches („minimale“) (Teil-)Petri-Netz erzeugt bei Ausführung Folgen von Transitionen, die durch (genau) diese eine Relation charakterisiert sind ?



Wie würde man nachweisen, dass dieses tatsächlich so ist ?

Idee: Erzeugung eines Petri-Netzes (2)

Welches („minimale“) (Teil-)Petri-Netz erzeugt Folgen von Transitionen, die jeweils durch (genau) die folgenden Relation charakterisiert sind ?

$x \rightarrow y,\ x \rightarrow z,\ y \| z$

$x \rightarrow y,\ x \rightarrow z,\ y \# z$

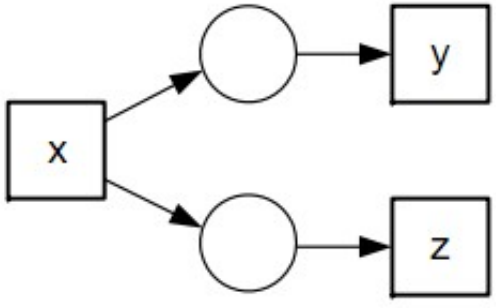
$x \rightarrow z,\ y \rightarrow z,\ x \| y$

$x \rightarrow z,\ y \rightarrow z,\ x \# y$

Idee: Erzeugung eines Petri-Netzes (2)

Welches („minimale“) (Teil-)Petri-Netz erzeugt Folgen von Transitionen, die jeweils durch (genau) die folgenden Relation charakterisiert sind ?

$x \rightarrow y,\ x \rightarrow z,\ y \| z$



$x \rightarrow y,\ x \rightarrow z,\ y \# z$

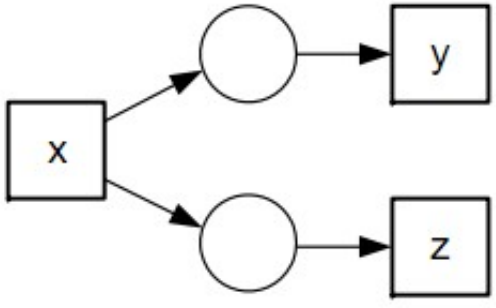
$x \rightarrow z,\ y \rightarrow z,\ x \| y$

$x \rightarrow z,\ y \rightarrow z,\ x \# y$

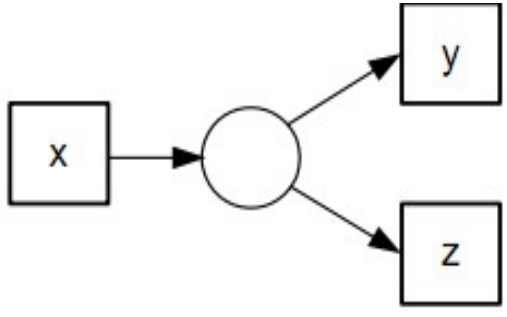
Idee: Erzeugung eines Petri-Netzes (2)

Welches („minimale“) (Teil-)Petri-Netz erzeugt Folgen von Transitionen, die jeweils durch (genau) die folgenden Relation charakterisiert sind ?

$x \rightarrow y,\ x \rightarrow z,\ y \| z$



$x \rightarrow y,\ x \rightarrow z,\ y \# z$



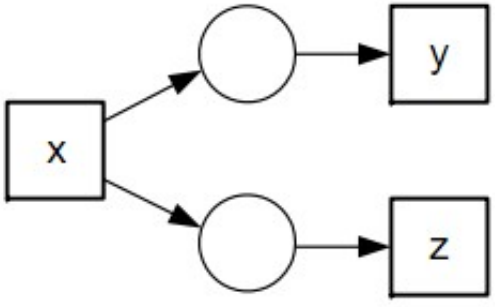
$x \rightarrow z,\ y \rightarrow z,\ x \| y$

$x \rightarrow z,\ y \rightarrow z,\ x \# y$

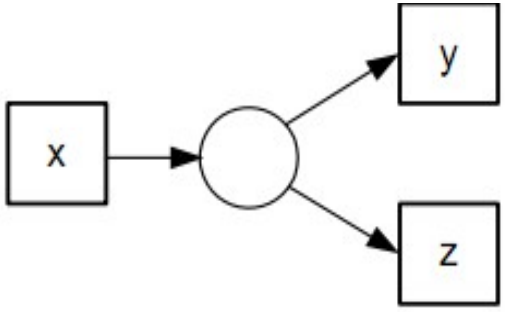
Idee: Erzeugung eines Petri-Netzes (2)

Welches („minimale“) (Teil-)Petri-Netz erzeugt Folgen von Transitionen, die jeweils durch (genau) die folgenden Relation charakterisiert sind ?

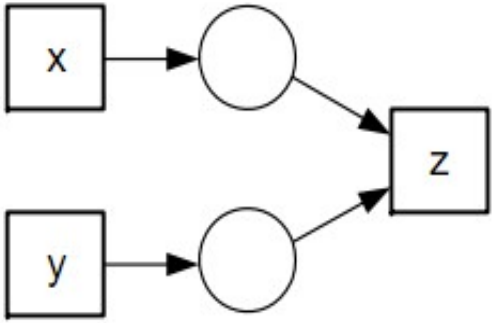
$x \rightarrow y,\ x \rightarrow z,\ y \| z$



$x \rightarrow y,\ x \rightarrow z,\ y \# z$



$x \rightarrow z,\ y \rightarrow z,\ x \| y$

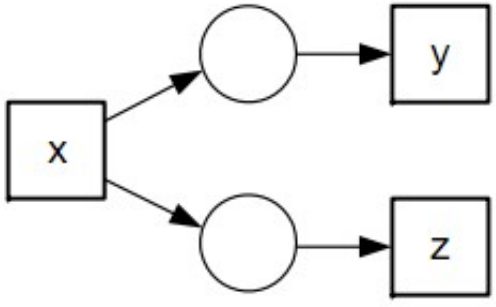


$x \rightarrow z,\ y \rightarrow z,\ x \# y$

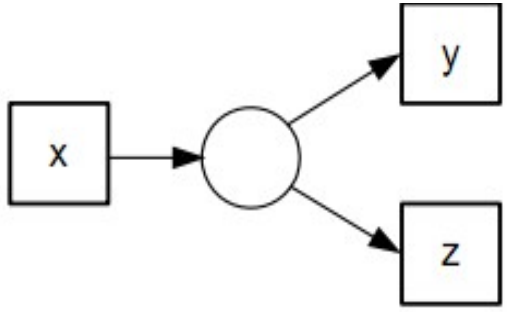
Idee: Erzeugung eines Petri-Netzes (2)

Welches („minimale“) (Teil-)Petri-Netz erzeugt Folgen von Transitionen, die jeweils durch (genau) die folgenden Relation charakterisiert sind ?

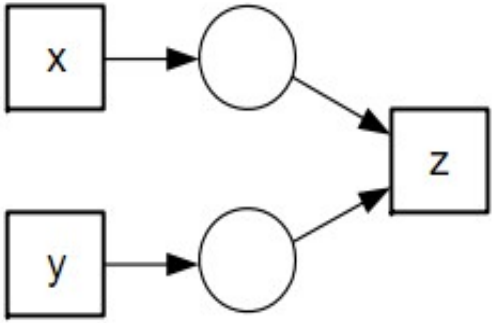
$x \rightarrow y,\ x \rightarrow z,\ y \| z$



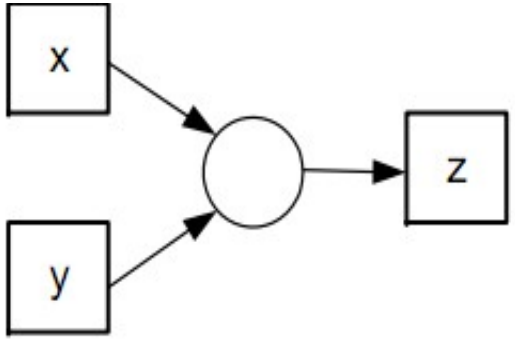
$x \rightarrow y,\ x \rightarrow z,\ y \# z$



$x \rightarrow z,\ y \rightarrow z,\ x \| y$



$x \rightarrow z,\ y \rightarrow z,\ x \# y$



Überblick Business-Process-Mining

- Einführung und Beispiel
- α-Algorithmus
  - Idee und Vorbereitungen
  - Formalisierung und Beispiel
- Workflow-Diagnose und Graph-Mining

Formalisierung: Der α-Algorithmus

Geg.: Workflow-Log W = Menge von Folgen σ von Log-Daten (z.B. ABCD).

Gesucht: Workflow-Schema als Petri-Netz
α(W) = (Stellen P_W, Transitionen T_W, Verbindungen F_W)

1. $T_W = \{t \in T \mid \exists_{\sigma \in W}\, t \in \sigma\}$ — Transitionen (= Vereinigung der Transitionsmengen der Log-Daten)
2. $T_I = \{t \in T \mid \exists_{\sigma \in W}\, t = first(\sigma)\}$ — Start-Transitionen
3. $T_O = \{t \in T \mid \exists_{\sigma \in W}\, t = last(\sigma)\}$ — End-Transitionen
4. $X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B}\, a \rightarrow_W b \wedge \forall_{a_1 \neq a_2 \in A}\, a_1 \#_W a_2 \wedge \forall_{b_1 \neq b_2 \in B}\, b_1 \#_W b_2\}$ — „verallgemeinerte Kausalitätsrelationen“ [NB: ({a},{b}) ∈ X_W für alle a→b]

Kandidaten für Stellen, um Paare von kausal aufeinanderfolgenden Transitionen zu verbinden (geht gleichzeitig wegen Unabhängigkeit).

Formalisierung: Der α-Algorithmus

5. $Y_W = \{(A,B) \in X_W \mid \forall_{(A',B') \in X_W} A \subseteq A' \wedge B \subseteq B' \Rightarrow (A,B) = (A',B')\}$

   „Maximale Kausalitätsrelationen“. Bringen Minimalität des Petri-Netzes: Minimale Anzahl von Stellen, die notwendig sind, um Paare von kausal aufeinanderfolgenden Transitionen zu verbinden.

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

   Stellen: Jede maximale Kausalitätsrelation definiert genau eine Stelle. Zusätzlich: i_W, o_W: eine Start- und eine Endstelle

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

   Verbindungen: Die durch die Kausalitäts-relationen definierten Verbindungen. Zusätzlich Verbindungen zwischen Start- bzw. Endstellen und Start- bzw. Endtransitionen.

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

   Petri-Netz

Frage: Wie würde man zeigen, dass der Algorithmus korrekt ist ?

α-Algorithmus – Beispiel

1. T_W = {

Ursprungs-Log:
{ABCD, ACBD, AED}

$$T_W = \{t \in T \mid \exists_{\sigma \in W}\, t \in \sigma\}$$

Extrahiere alle Transitionen
aus dem Log

α-Algorithmus – Beispiel

1. T_W = {A,B,C,D

Ursprungs-Log:
{ABCD, ACBD, AED}

$$T_W = \{t \in T \mid \exists_{\sigma \in W}\, t \in \sigma\}$$

Extrahiere alle Transitionen
aus dem Log

α-Algorithmus – Beispiel

1. T_W = {A,B,C,D,E}

Ursprungs-Log:
{ABCD, ACBD, AED}

$$T_W = \{t \in T \mid \exists_{\sigma \in W}\, t \in \sigma\}$$

Extrahiere alle Transitionen
aus dem Log

α-Algorithmus – Beispiel

1. $T_W = \{A,B,C,D,E\}$
2. $T_I = \{$

Ursprungs-Log:
{ABCD, ACBD, AED}

$$T_I = \{ t \in T \mid \exists_{\sigma \in W}\, t = first(\sigma) \}$$

Sammle Starttransitionen aus den einzelnen Logs

α-Algorithmus – Beispiel

1. $T_W = \{A,B,C,D,E\}$
2. $T_I = \{A\}$

Ursprungs-Log:
{ABCD, ACBD, AED}

$$T_I = \{t \in T \mid \exists_{\sigma \in W}\, t = first(\sigma)\}$$

Sammle Starttransitionen aus den einzelnen Logs

α-Algorithmus – Beispiel

1. $T_W = \{A,B,C,D,E\}$
2. $T_I = \{A\}$
3. $T_O = \{$

$$T_O = \{t \in T \mid \exists_{\sigma \in W}\, t = last(\sigma)\}$$

Sammle Endtransitionen aus den einzelnen Logs

Ursprungs-Log:
{ABCD, ACBD, AED}

α-Algorithmus – Beispiel

1. T_W = {A,B,C,D,E}
2. T_I = {A}
3. T_O = {D}

$$T_O = \{t \in T \mid \exists_{\sigma \in W}\, t = last(\sigma)\}$$

Sammle Endtransitionen aus den einzelnen Logs

Ursprungs-Log:
{ABCD, ACBD, AED}

α-Algorithmus – Beispiel

1. T_W = {A,B
2. T_I = {A}
3. T_O = {D}
4. X_W = {

Bilde Tupel möglicher Flüsse.

Ursprungs-Log:
{ABCD, ACBD, AED}

$$X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2\}$$

α-Algorithmus – Beispiel

1. $T_W = \{A, B, C, D, E\}$
2. $T_I = \{A\}$
3. $T_O = \{D\}$
4. $X_W = \{$

Bilde Tupel möglicher Flüsse:
Bilde Teilmengen mit unabhängigen Elementen
als Kandidaten für Mengen *A* und *B*
(in Def. rechts unten auf dieser Folie)

Ursprungs-Log:
{ABCD, ACBD, AED}

$$X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2\}$$

α-Algorithmus – Beispiel

Bilde Tupel möglicher Flüsse:
Bilde Teilmengen mit unabhängigen Elementen
als Kandidaten für Mengen *A* und *B*
{A},{A,D}

Ursprungs-Log:
{ABCD, ACBD, AED}

Alle andere Transitionen
abhängig bis auf D

$$X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2\}$$

α-Algorithmus – Beispiel

Bilde Tupel möglicher Flüsse:
Bilde Teilmengen mit unabhängigen Elementen
als Kandidaten für Mengen *A* und *B*
{A},{A,D},{B},{B, E}

Ursprungs-Log:
{ABCD, ACBD, AED}

A,C,D abhängig
E unabhängig

$$X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2\}$$

α-Algorithmus – Beispiel

Bilde Tupel möglicher Flüsse:
Bilde Teilmengen mit unabhängigen Elementen
als Kandidaten für Mengen *A* und *B*
{A},{A,D},{B},{B, E},{C},{C,E},{D},{E}

{A,D} nicht relevant
obwohl unabhängig,
weil kein gemeinsamer
Vorgänger / Nachfolger

Ursprungs-Log:
{ABCD, ACBD, AED}

2.

3. T_O = {D}

4. X_W = {

$$X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2\}$$

α-Algorithmus – Beispiel

Bilde Tupel möglicher Flüsse:
Bilde Tupel der Teilmengen, sodass Menge *A* nur kausale Vorgänger der Elemente in Menge *B* enthält
{A},{A,D},{B},{B, E},{C},{C,E},{D},{E}

Ursprungs-Log:
{ABCD, ACBD, AED}

3. T_O = {B}

4. X_W = {({A}, {B}), ({A}, {C}), ({A}, {E}),
({A}, {B, E}), ({A}, {C, E}),

$$X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2\}$$

α-Algorithmus – Beispiel

Bilde Tupel möglicher Flüsse:
Bilde Tupel der Teilmengen, sodass Menge *A* nur kausale Vorgänger der Elemente in Menge *B* enthält
{A},{A,D},{B},{B, E},{C},{C, E},{D}

Ursprungs-Log:
{ABCD, ACBD, AED}

{C} nicht wegen BC und CB: Abfolge nicht eindeutig

3. T_O = {D}

4. X_W = {({A}, {B}), ({A}, {C}), ({A}, {E}), ({B}, {D}),
({A}, {B, E}), ({A}, {C, E}),

$$X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2\}$$

α-Algorithmus – Beispiel

1.

2.

3. T_O = {D}

Bilde Tupel möglicher Flüsse

Ursprungs-Log:
{ABCD, ACBD, AED}

4. X_W = {({A}, {B}), ({A}, {C}), ({A}, {E}), ({B}, {D}), ({C},{D}), ({E}, {D}), ({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E}, {D}) }

$$X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2\}$$

α-Algorithmus – Beispiel

1. $T_W = \{A,B,C,D,E\}$
2. $T_I = \{A\}$
3. $T_O = \{D\}$
4. $X_W = \{(\{A\}, \{B\}), (\{A\}, \{C\}), (\{A\}, \{E\}), (\{B\}, \{D\}), (\{C\},\{D\}), (\{E\}, \{D\}), (\{A\}, \{B, E\}), (\{A\}, \{C, E\}), (\{B, E\}, \{D\}), (\{C, E\}, \{D\}) \}$
5. $Y_W = \{$

Ursprungs-Log:
{ABCD, ACBD, AED}

$$Y_W = \{(A,B) \in X_W \mid \forall_{(A',B') \in X_W} A \subseteq A' \wedge B \subseteq B' \Rightarrow (A,B) = (A',B')\}$$

Entferne alle Tupel aus X_W, die Teilmenge eines anderen Tupels sind.

α-Algorithmus – Beispiel

1. $T_W = \{A,B,C,D,E\}$
2. $T_I = \{A\}$
3. $T_O = \{D\}$
4. $X_W = \{$~~$(\{A\}, \{B\})$~~, $(\{A\}, \{C\})$, ~~$(\{A\}, \{E\})$~~, $(\{B\}, \{D\})$, $(\{C\},\{D\})$, $(\{E\}, \{D\})$, $(\{A\}, \{B, E\})$, $(\{A\}, \{C, E\})$, $(\{B, E\}, \{D\})$, $(\{C, E\}, \{D\})$ $\}$
5. $Y_W = \{(\{A\}, \{B, E\}),$

Ursprungs-Log:
{ABCD, ACBD, AED}

$$Y_W = \{(A,B) \in X_W \mid \forall_{(A',B') \in X_W} A \subseteq A' \wedge B \subseteq B' \Rightarrow (A,B) = (A',B')\}$$

Entferne alle Tupel aus X_W, die Teilmenge eines anderen Tupels sind.

α-Algorithmus – Beispiel

Ursprungs-Log:
{ABCD, ACBD, AED}

1. T_W = {A,B,C,D,E}
2. T_I = {A}
3. T_O = {D}
4. X_W = {~~({A}, {B}), ({A}, {C}), ({A}, {E}), ({B}, {D}), ({C},{D}),~~
~~({E}, {D})~~, ({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}),
({C, E}, {D}) }
5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}),
({C, E},{D})}

$$Y_W = \{(A,B) \in X_W \mid \forall_{(A',B') \in X_W} A \subseteq A' \wedge B \subseteq B' \Rightarrow (A,B) = (A',B')\}$$

Entferne alle Tupel aus X_W, die Teilmenge eines anderen Tupels sind.

α-Algorithmus – Beispiel

1. $T_W = \{A,B,C,D,E\}$
2. $T_I = \{A\}$
3. $T_O = \{D\}$
4. $X_W = \{(\{A\}, \{B\}), (\{A\}, \{C\}), (\{A\}, \{E\}), (\{B\}, \{D\}), (\{C\},\{D\}), (\{E\}, \{D\}), (\{A\}, \{B, E\}), (\{A\}, \{C, E\}), (\{B, E\}, \{D\}), (\{C, E\}, \{D\})\}$
5. $Y_W = \{(\{A\}, \{B, E\}), (\{A\}, \{C, E\}), (\{B, E\}, \{D\}), (\{C, E\},\{D\})\}$

Ursprungs-Log:
{ABCD, ACBD, AED}

α-Algorithmus – Beispiel

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}),
({C, E},{D})}
6. P_W = {

$$P_W = \{(p_{(A,B)}) | (A,B) \in Y_W\} \cup \{i_W, o_W\}$$

Füge für jedes Tupel eine Stelle, sowie eine Start- und Endstelle ein.

α-Algorithmus – Beispiel

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}),
({C, E},{D})}

6. P_W = {i_W, o_W,

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

Füge für jedes Tupel eine Stelle, sowie eine Start- und Endstelle ein.

α-Algorithmus – Beispiel

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}),
({C, E},{D})}

6. P_W = {i_W, o_W, p({A},{B,E}),

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

Füge für jedes Tupel eine Stelle, sowie eine Start- und Endstelle ein.

α-Algorithmus – Beispiel

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

6. P_W = {i_W, o_W, p({A},{B,E}), p({A},{C,E}), p({B,E},{D}), p({C,E},{D})},

$$P_W = \{(p_{(A,B)}) | (A,B) \in Y_W\} \cup \{i_W, o_W\}$$

Füge für jedes Tupel eine Stelle, sowie eine Start- und Endstelle ein.

α-Algorithmus – Beispiel

1. T_W = {A,B,C,D,E}

2. T_I = {A} 3. T_O = {D}

$$F_W = \{(a, p_{(A,B)}) | (A,B) \in Y_W \wedge a \in A\} \cup \{p_{(A,B)}, b | (A,B) \in Y_W \wedge b \in B\} \cup \{(i_W, t) | t \in T_I\} \cup \{(t, o_W) | t \in T_O\}$$

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

6. P_W = {i_W, o_W, p({A},{B,E}), p({A},{C,E}), p({B,E},{D}), p({C,E},{D})},

7. F_W = {

Füge die zugehörigen Bögen ein.

α-Algorithmus – Beispiel

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

2. $T_I = \{A\}$ 3. $T_O = \{D\}$

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

6. $P_W = \{i_W, o_W, p(\{A\},\{B,E\}), p(\{A\},\{C,E\}), p(\{B,E\},\{D\}),$
$p(\{C,E\},\{D\})\},$

7. $F_W = \{(i_W, A),$
$\ldots, (D, o_W)$

$$F_W = \{(a, p_{(A,B)}) | (A,B) \in Y_W \wedge a \in A\} \cup$$
$$\{p_{(A,B)}, b | (A,B) \in Y_W \wedge b \in B\} \cup$$
$$\{(i_W, t) | t \in T_I\} \cup$$
$$\{(t, o_W) | t \in T_O\}$$

Füge die zugehörigen Bögen ein
Verbinde Startstelle mit Starttransitionen
Verbinde Endstelle mit Starttransitionen

α-Algorithmus – Beispiel

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

1. T_W = {A,B,C,D,E}

2. T_I = {A} 3. T_O = {D}

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

6. P_W = {i_W, o_W, p({A},{B,E}), p({A},{C,E}), p({B,E},{D}), p({C,E},{D})},

7. F_W = {(i_W, A), (A, p({A},{B,E}))

. . . , (D, o_W)

Füge die zugehörigen Bögen ein
Verbinde Elemente aus A eines Y_W – Tupels mit der zugehörigen Stelle

α-Algorithmus – Beispiel

$$F_W = \{(a, p_{(A,B)}) | (A,B) \in Y_W \wedge a \in A\} \cup$$
$$\{p_{(A,B)}, b | (A,B) \in Y_W \wedge b \in B\} \cup$$
$$\{(i_W, t) | t \in T_I\} \cup$$
$$\{(t, o_W) | t \in T_O\}$$

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

2. $T_I = \{A\}$ 3. $T_O = \{D\}$

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

6. $P_W = \{i_W, o_W,$ p({A},{B,E}), p({A},{C,E}), p({B,E},{D}),
p({C,E},{D})},

7. $F_W = \{(i_W, A), (A,$ p({A},{B,E})), (p({A},{B,E}), B),
. . . , (D, o_W)

Füge die zugehörigen Bögen ein
Verbinde zugehörigen Stelle eines Y_W – Tupels mit den Elementen aus B

α-Algorithmus – Beispiel

1. T_W = {A,B,C,D,E}

2. T_I = {A} 3. T_O = {D}

$$F_W = \{(a, p_{(A,B)}) | (A,B) \in Y_W \wedge a \in A\} \cup \{p_{(A,B)}, b | (A,B) \in Y_W \wedge b \in B\} \cup \{(i_W, t) | t \in T_I\} \cup \{(t, o_W) | t \in T_O\}$$

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

6. P_W = {i_W, o_W, p({A},{B,E}), p({A},{C,E}), p({B,E},{D}), p({C,E},{D})},

7. F_W = {(i_W, A), (A, p({A},{B,E})), (p({A},{B,E}), B), …, (D, o_W)}

Füge die zugehörigen Bögen ein

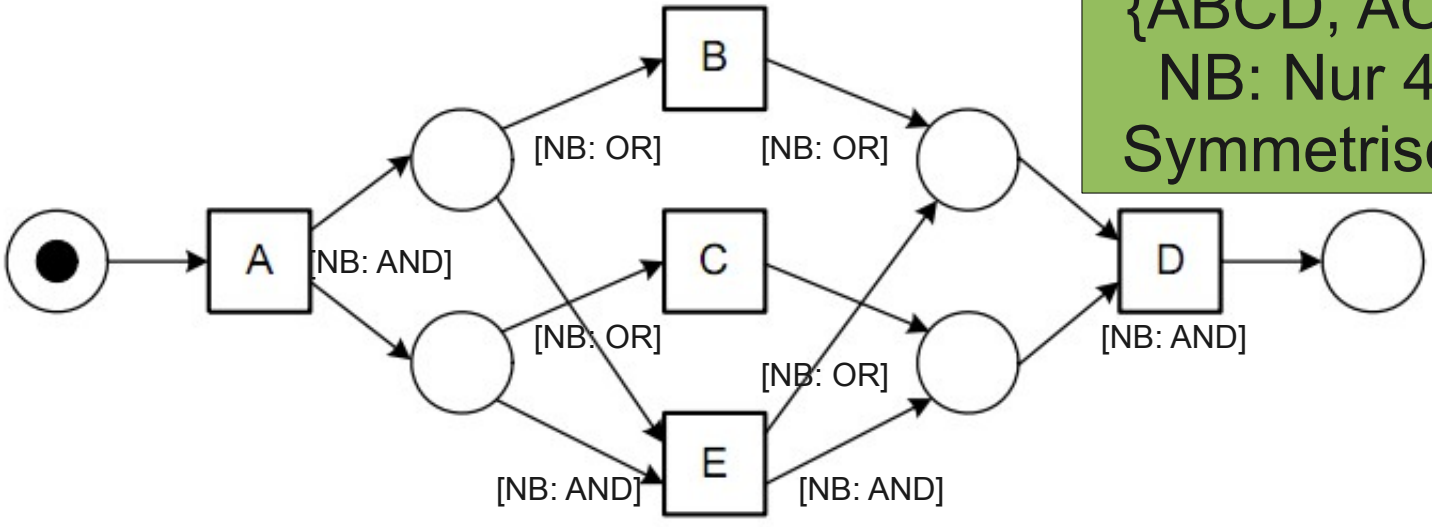
α-Algorithmus – Beispiel

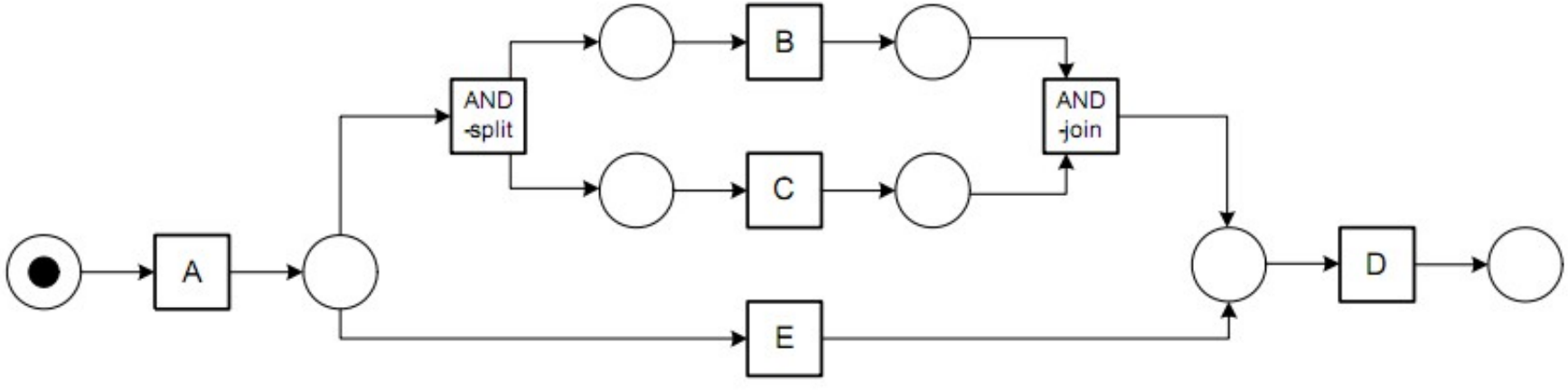
5. Y_w = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}
6. P_w = {i_w, o_w, p({A},{B,E}), p({A},{C,E}), p({B,E},{D}), p({C,E},{D})},
7. F_w = {(i_w, A), (A, p({A},{B,E})), (p({A},{B,E}), B), …, (D, o_w)}
8. a(W) = (P_w, T_w, F_w)

$\alpha(W) = (P_W, T_W, F_W)$
Das extrahierte Petri Netz ist nun definiert.

α-Algorithmus – Beispiel-Netz

Ursprungs-Log:
{ABCD, ACBD, AED}
NB: Nur 4 Stellen !
Symmetrisch in AED.



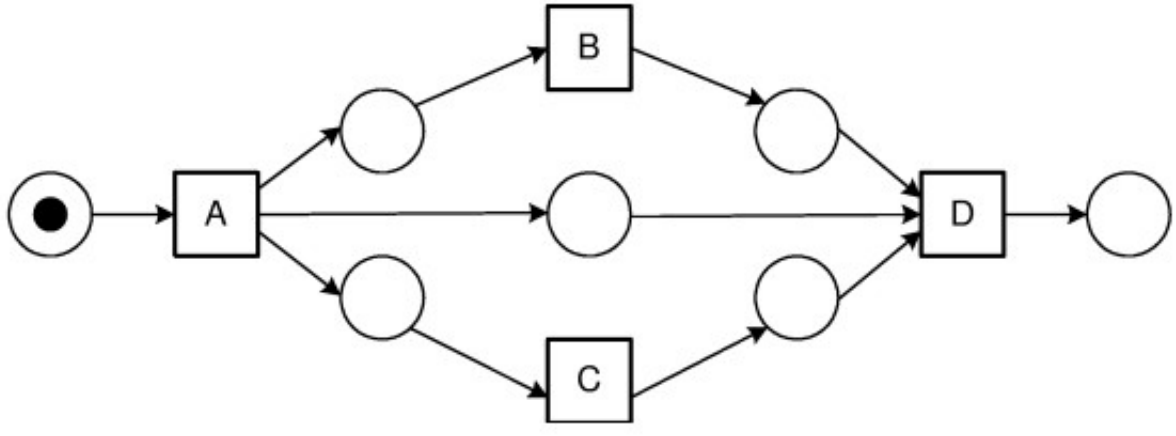


α-Algorithmus Unzulänglichkeiten (1)

- Wie auf der vorherigen Folie gesehen, können Log-Daten im Allgemeinen von mehr als einem Petri-Netz erzeugt werden. Der α-Algorithmus erzeugt das Petri-Netz mit der minimalen Anzahl Stellen. Wie wir ebenfalls gesehen haben, ist es nicht unbedingt das am leichtesten verständliche Petri-Netz.
- Es können nur „beobachtbare“ Netzwerke erzeugt werden.
  - Splits und Joins nicht immer direkt beobachtbar.
  - Beobachtetes Verhalten wird aber korrekt wiedergegeben.

α-Algorithmus Unzulänglichkeiten (1a)

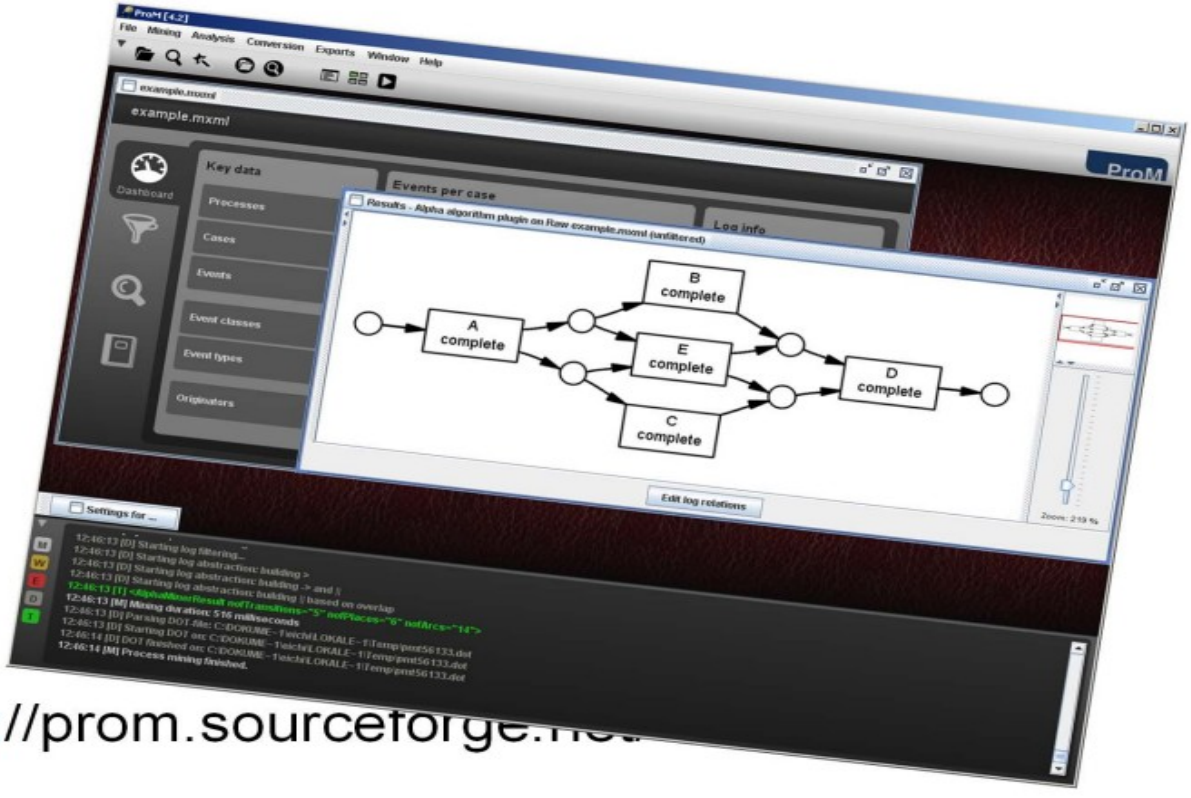
- Eindeutigkeit der Bezeichner: Bei der getroffenen Annahme, dass jeder Transitionsname nur einmal im Petri-Netz auftaucht, ist nicht jede Folge von Transitionen aus einem wohlgeformten Petri-Netz erzeugbar !
  [NB: Falls die Annahme nicht gemacht wird, gibt es immer die triviale Möglichkeit, das Petri-Netz als OR-Verknüpfung über Transitionen, die je eine Sequenz erzeugen, darzustellen.]
- Insbesondere hinsichtlich Abstraktion („unsichtbar machen“) von Transitionen: Wegabstrahieren der Transition E in den Log-Daten im obigen Beispiel für zu Log-Daten, die (unter o.g. Annahme) nicht durch ein Petri-Netz darstellbar sind:



α-Algorithmus Unzulänglichkeiten (2)

- Mehrfach auftretende Aktivitäten
  - Schwierig, der α-Algorithmus konstruiert nur eine Transition pro Aktivität.
- Iterationen
  - Führen zu mehrfach auftretenden Aktivitäten.
  - Müssen ggf. extra erkannt werden.
    - Mitunter sehr schwierig!
- Verrauschte Daten
  - Lösung z.B. durch Betrachtung der Häufigkeit.

Überblick: Process-Mining-Werkzeuge



http://prom.sourceforge.net

Process Mining TV Episode 1

http://www.promtools.org/pmtv/movies/pmtv01.mov

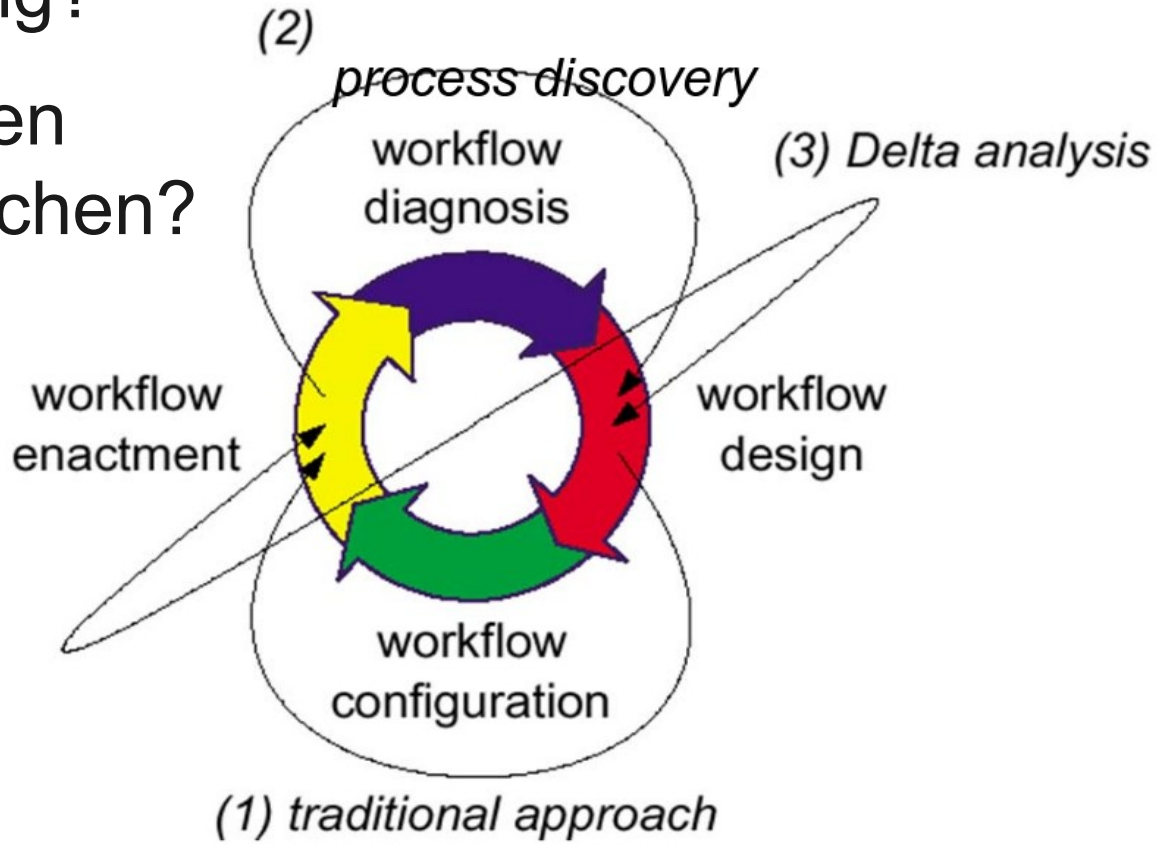
Überblick Business-Process-Mining

- Einführung und Beispiel
- α-Algorithmus
  - Idee und Vorbereitungen
  - Formalisierung und Beispiel
- Workflow-Diagnose und Graph-Mining

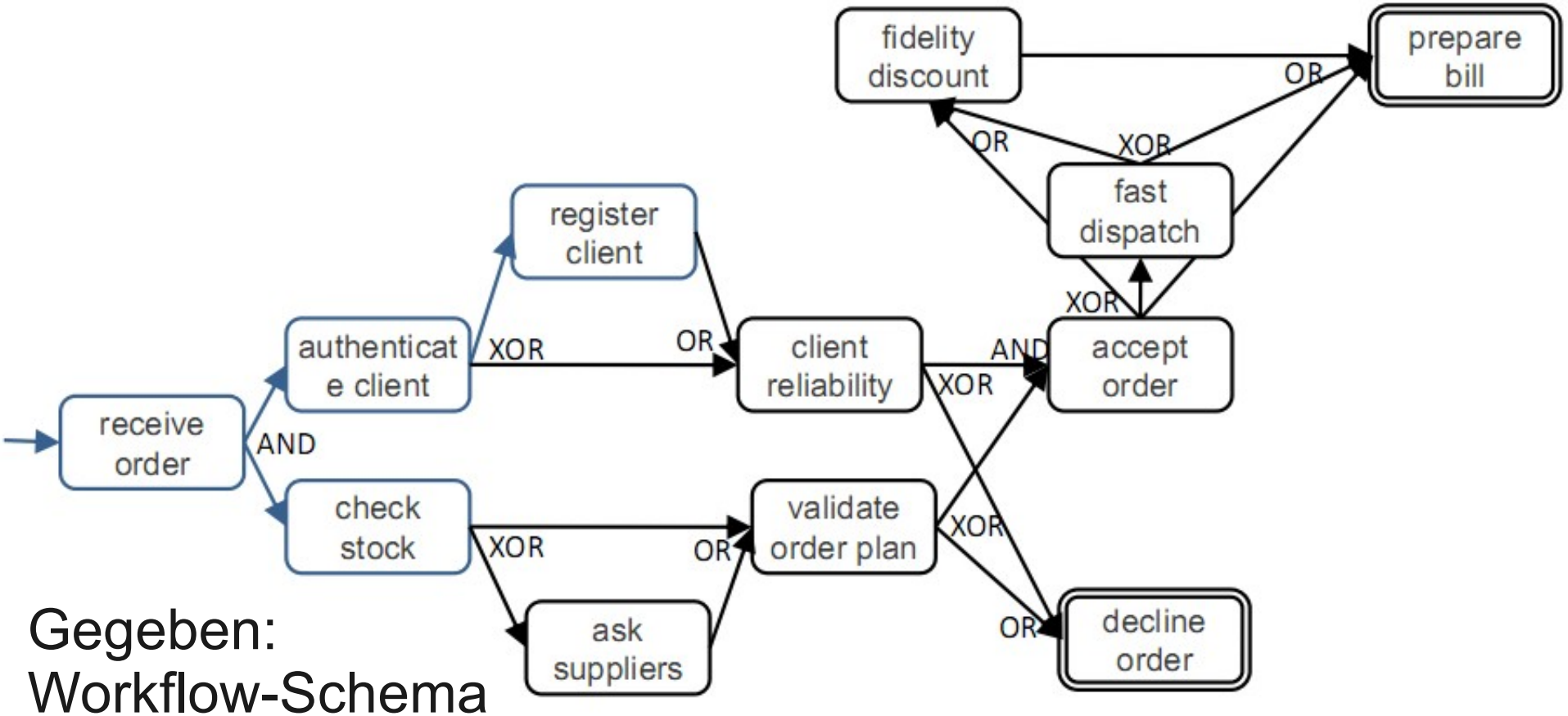
Workflow-Diagnose

Nächster Schritt („workflow diagnosis“):

- Welche Pfade sind häufig?
- Welche Instanzen werden wahrscheinlich abgebrochen?



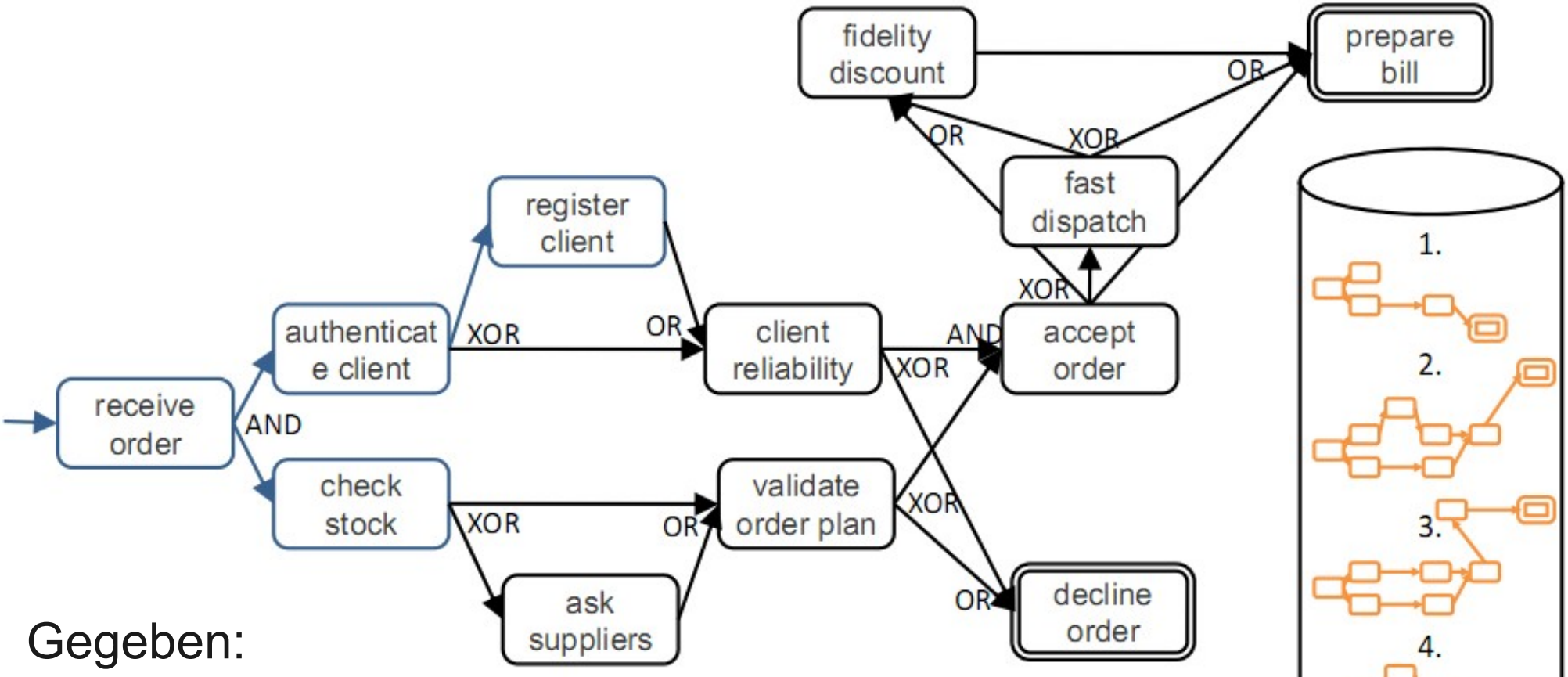
Workflow-Diagnose: Statische Analyse



Gegeben:
Workflow-Schema

Kann diese Teilausführung möglicherweise zu einem erfolgreichen Verkaufsvertrag führen ?

Workflow-Diagnose: Analyse der Laufzeitdaten



Gegeben:
Workflow-Schema, DB mit erfolgreichen Instanzen

Führt diese Teilausführung **wahrscheinlich** zu einem erfolgreichen Verkaufsvertrag?

Workflow-Diagnose

- Weitere interessante Fragestellungen
  - Fehlerfreie Workflow-Ausführung?
  - Workflow-Ausführung mit geringem oder normalem Ressourcenverbrauch?
  - Identifikation kritischer Aktivitäten (besonders hoher Ressourcenverbrauch).
  - Wie sehen häufige / typische Ausführungen aus?
- Verwertung der Information
  - Zur Laufzeit: Scheduling
  - Information für Management / Optimierung (Business Process Reengineering (BPR), Continuous Process Improvement (CPI))
- Ansatz: Ermitteln von häufigen Mustern bei gegebenem Workflow-Schema und Prozess-Logs.

Die extrahierten Modellen müssen oft noch aufbereitet / vereinfacht / vereinheitlicht werden.

Ein Ansatz dafür: Graph-Mining.

Graph-Mining

Finden von **häufigen Untergraphen** in einer Menge von Graphen (Graph-Datenbank).

- **Untergraph**: Teilmenge der im Graphen vorhandenen Kanten (mit den zugehörigen Knoten).
- **Häufig**: Ein Graph G ist Untergraph von mindestens *minSup* Graphen der Graph-Datenbank (genannt „Support“).

Weitere Anwendungen neben Workflow-Modellen:

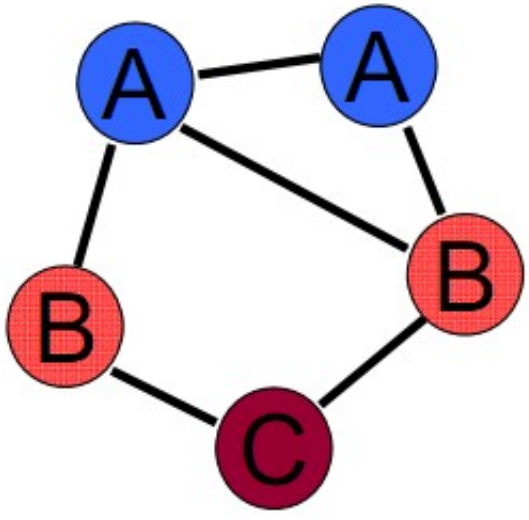
- Chemie und Pharmaforschung (Moleküle)
- Compilertechnik (Call-Graphen)
- Soziale Netzwerke etc.

Beobachtungen:

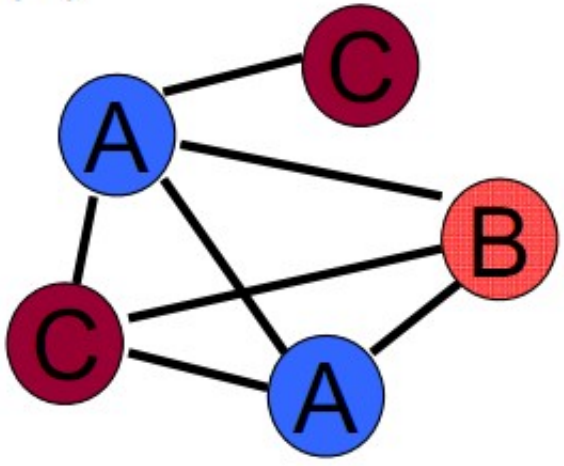
- G häufig => Alle Untergraphen häufig
- G hat Support n+1 => G hat Support n

Graph-Mining: Beispiel

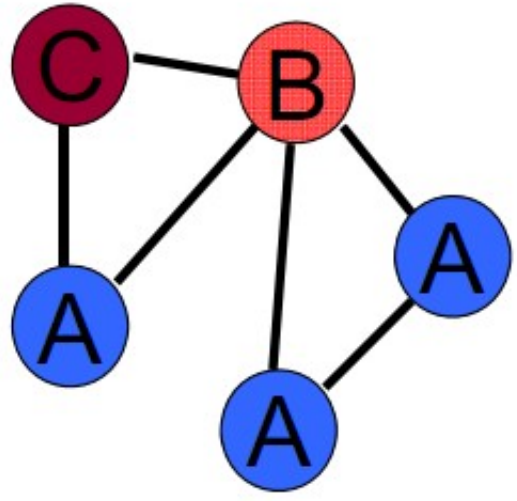
(1)



(2)



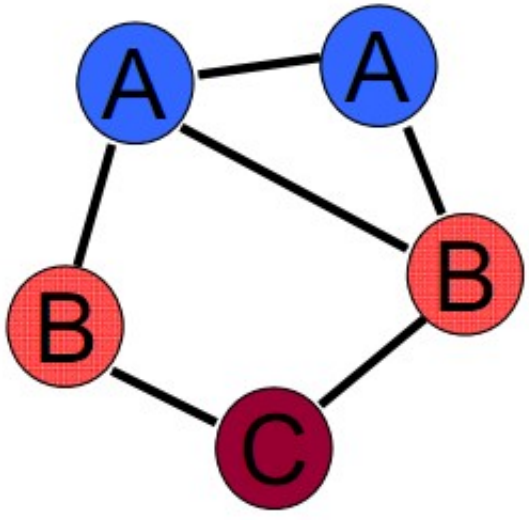
(3)



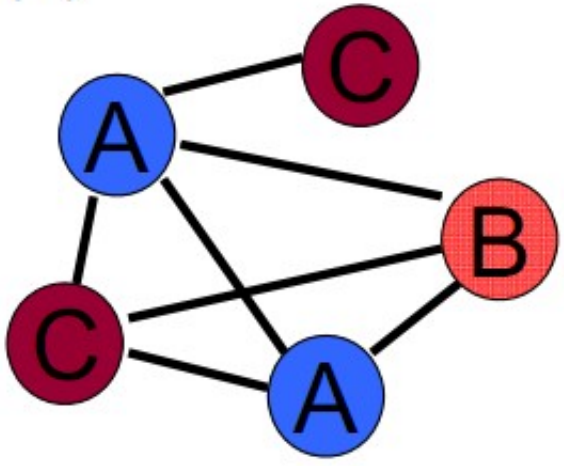
Support	1	2	3
Teilgraph			

Graph-Mining: Beispiel

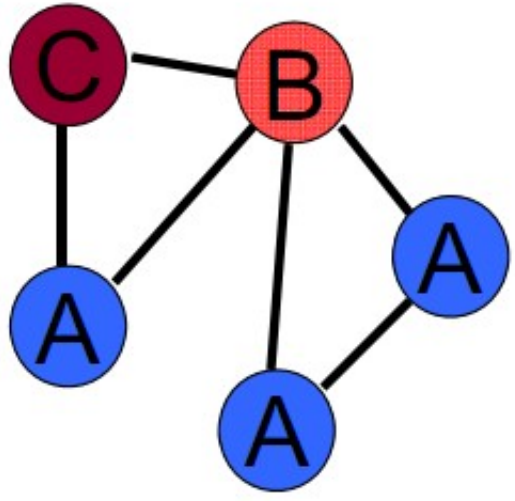
(1)

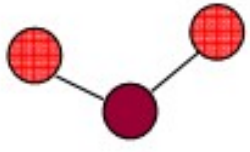


(2)

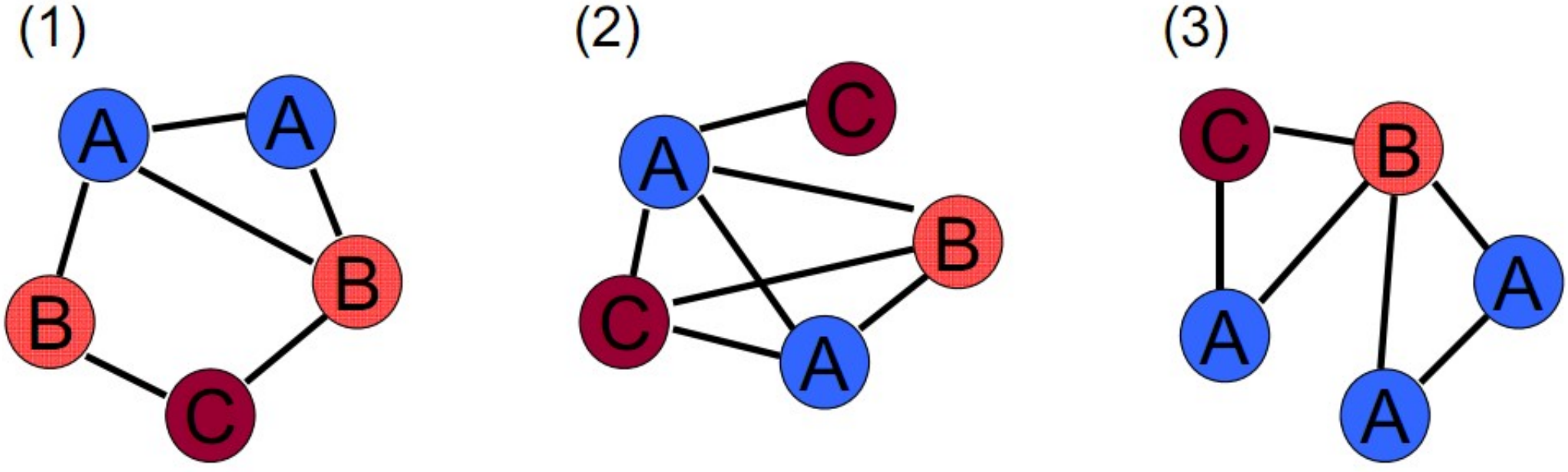


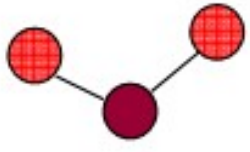
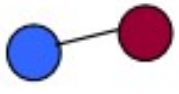
(3)



Support	1	2	3
Teilgraph			

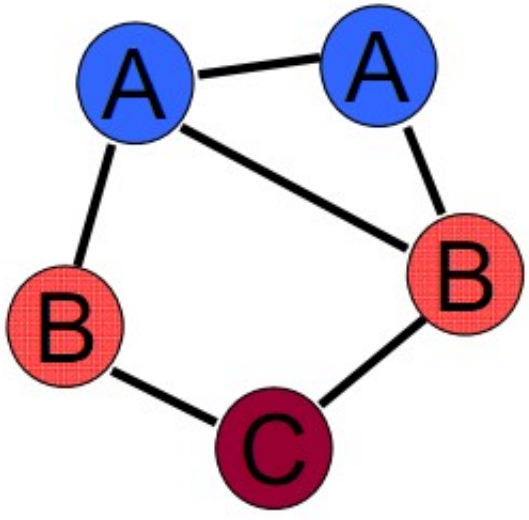
Graph-Mining: Beispiel



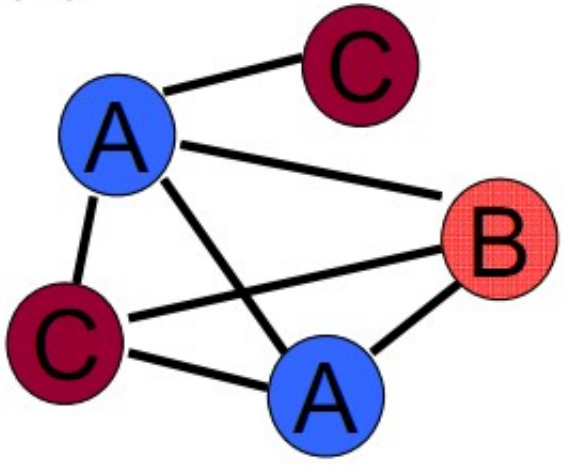
Support	1	2	3
Teilgraph			

Graph-Mining: Beispiel

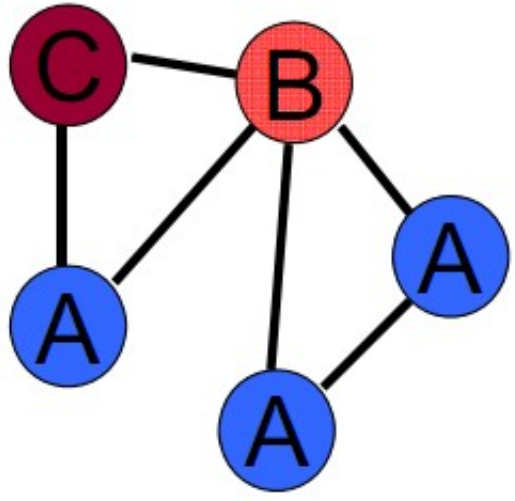
(1)

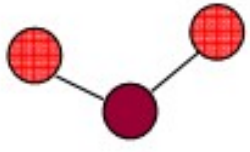
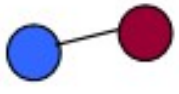
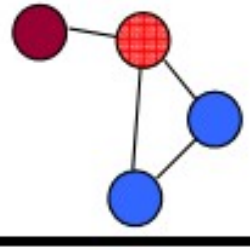


(2)



(3)



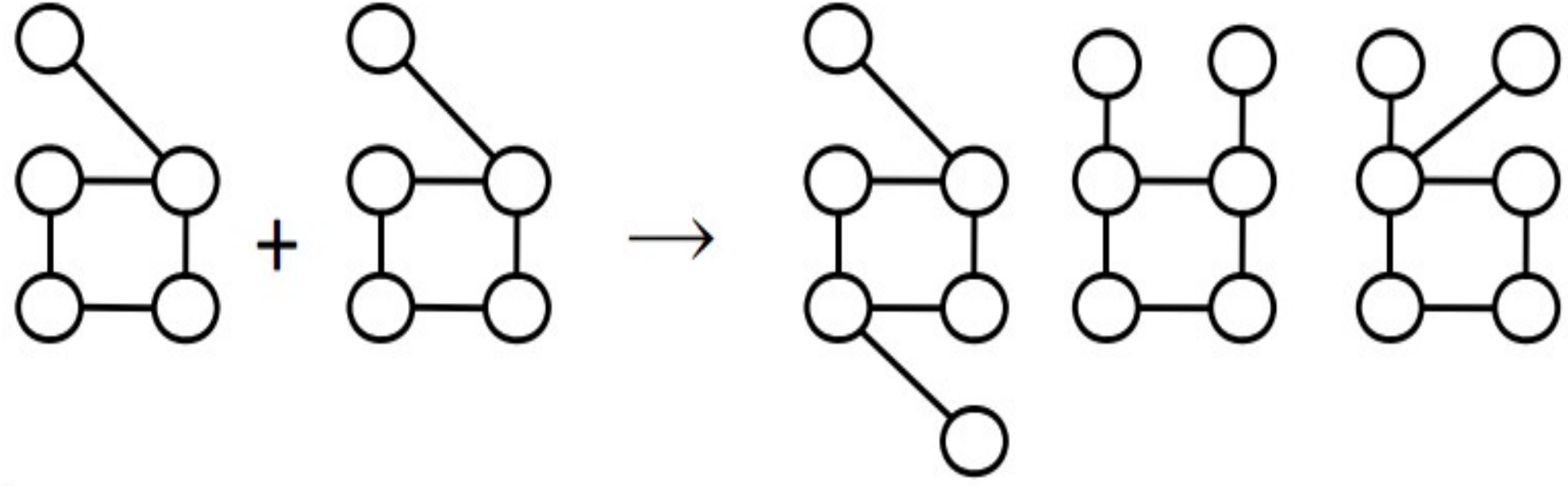
Support	1	2	3
Teilgraph			

Apriori-Graph-Mining Algorithmus

Iteratives Vorgehen (Breitensuche):

1. Suche aller häufigen Kanten in der Graph-Datenbank (n=1).
2. Generieren von Kandidaten der Größe n+1 aus häufigen Untergraphen der Größe n.
3. Überprüfen der Häufigkeit der Kandidaten in der Graph-Datenbank.
4. Wiederhole 2+3 solange häufige Graphen gefunden werden.

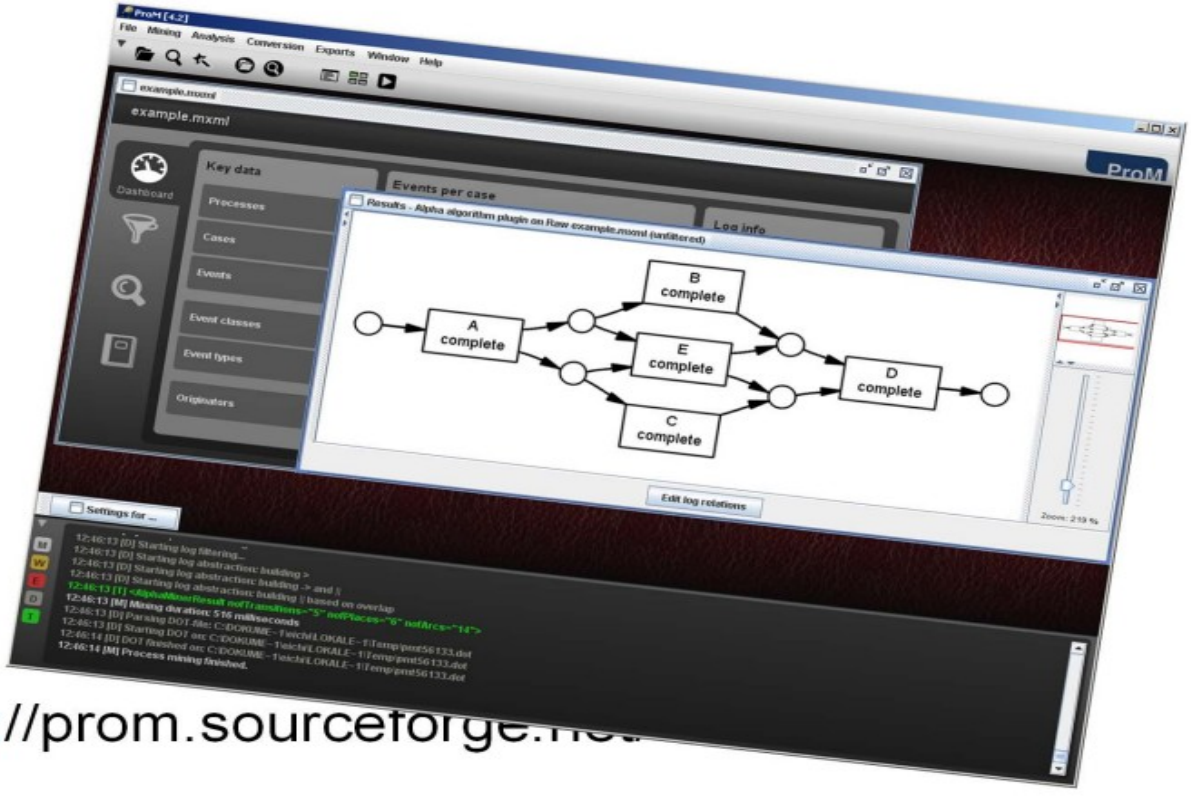
Ein möglicher Ansatz für n+1-Kandidatengenerierung durch Join von n–Kernen:



Nachteile von Apriori-Graph-Mining

- Aufwändige Kandidatengenerierung
  - Jeder Graph der Größe n wird mit jedem Kandidaten verglichen.
  - Finden gemeinsamer Kerne beinhaltet Untergraphisomorphie
  - Mergen der Kerne beinhaltet Suche nach Automorphien.
- Aufwändige Kandidatenüberprüfung
  - Es entstehen im allgemeinen sehr viele Kandidaten.
  - Jede Überprüfung beinhaltet Untergraphisomorphie.
- Untergraphisomorphie ist NP-vollständig.
- **Kein Skalieren für große Workflow-Netze.**
- Es existieren andere Graph-Mining-Algorithmen, die besser skalieren (PatternGrowth-Ansatz).

Demo: ProM



http://prom.sourceforge.net

Process Mining TV Episode 2

http://www.promtools.org/pmtv/movies/pmtv02.mov

Zusammenfassung

In diesem Teil der Vorlesung haben wir uns beschäftigt mit:

- Grundlagen Business-Process-Mining
- α-Algorithmus im Detail
- Grundlagen Graph-Mining

In den nächsten beiden Abschnitt werden wir uns dann ansehen, welche Rolle Geschäftsprozessmodellierung in praktischen Systemen zur Unterstützung des Workflow-Managements spielt, und welche Möglichkeiten es zur Workflow-Automatisierung gibt.