Vorlesung

Methodische Grundlagen des Software-Engineering

im Sommersemester 2014

Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

Teil 2.4: Prozessextraktion

v. 02.06.2014

2.4 Prozessextraktion

**[mit freundlicher Genehmigung basierend
auf einem englischen Foliensatz von
Prof. Dr. Wil van der Aalst (TU Eindhoven)]
[inkl. Beiträge von Jutta Mülle und Dr. Silvia von Stackelberg,
LS Prof. Böhm, Karlsruher Institut für Technologie]**

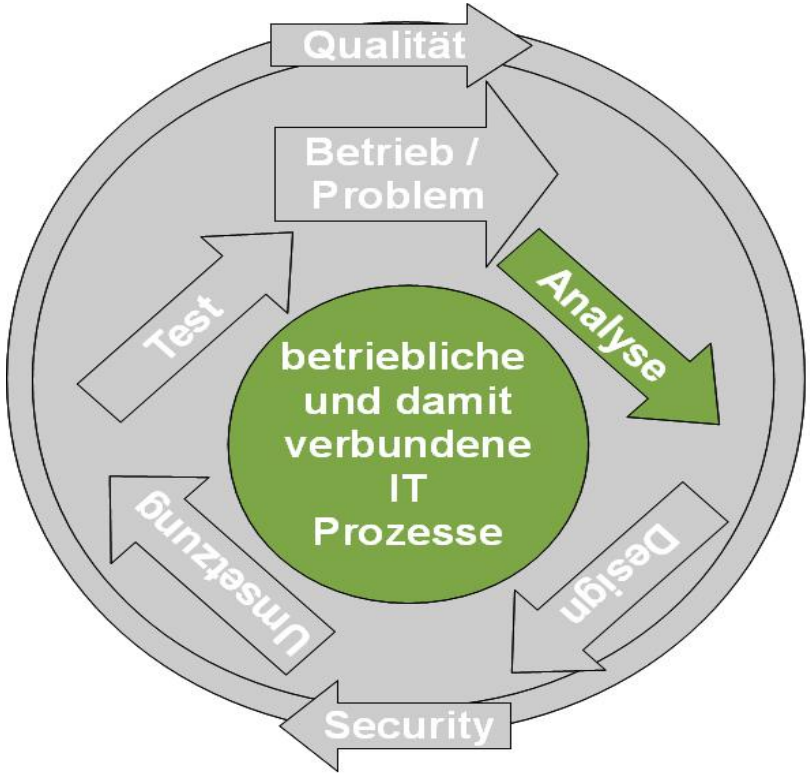
Literatur:

[vdA11] Wil van der Aalst: **Process Mining: Discovery, Conformance and Enhancement of Business Processes**, Springer-Verlag. 2011.
Unibibliothek (6 Exemplare): http://www.ub.tu-dortmund.de/katalog/titel/1332248
(Bei Engpässen kann eine **Kopiervorlage** der relevanten Ausschnitte zur Verfügung gestellt werden.)

- **Kapitel 5**

Einordnung
2.4: Prozessextraktion

- Geschäftsprozessmodellierung
- Process-Mining
  - Einführung: Process-Mining
  - Petrinetze
  - Data-Mining
  - Datenbeschaffung
  - Prozessextraktion
  - Konformanzanalyse
  - Mining: Zusätzliche Perspektiven
  - Betriebsunterstützung
  - Werkzeugunterstützung
  - Analysiere „Lasagne Prozesse“
  - Analysiere „Spaghetti Prozesse“
  - Kartographie und Navigation
  - Epilog
- Modellbasierte Entwicklung sicherer Software



Einleitung Prozessextraktion

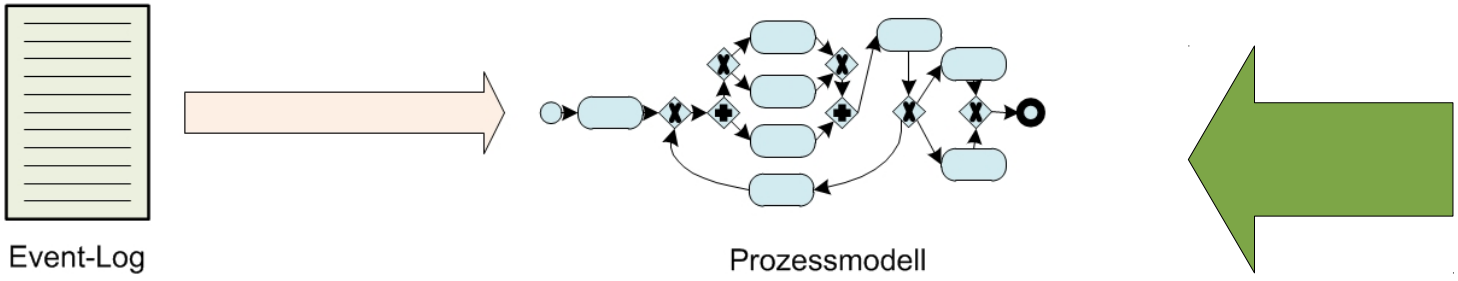
- **Vorheriger Abschnitt:** Datenbeschaffung und Datenspeicherformat XES.
- **Dieser Abschnitt:** „Prozessextraktion“:
  - Einführung und Beispiele
  - **α-Algorithmus:** Vorstellung und Beispiele
  - Prozessmodell: Ein schweres Problem?

Überblick Prozessextraktion

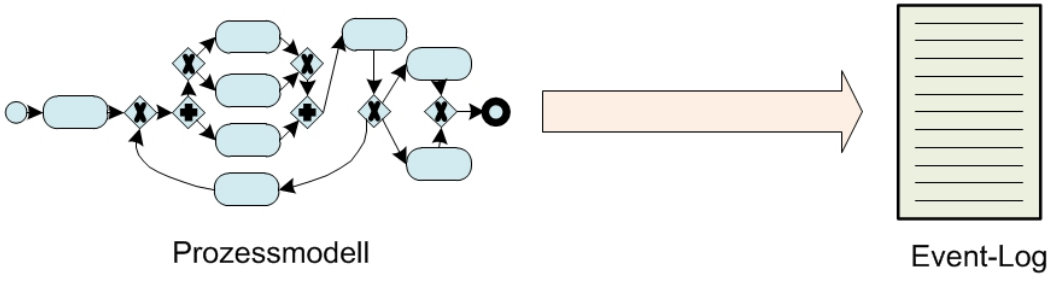
- **Einführung und Beispiel**
- α-Algorithmus
  - Idee und Vorbereitungen
  - Formalisierung
  - Beispiel
  - Weitere Beispiele
  - Einschränkungen
- Allgemeine Herausforderungen beim Process-Mining

Prozessextraktion = Play-In

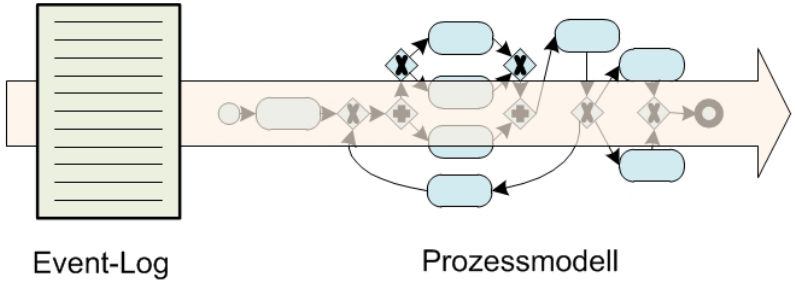
Play-In



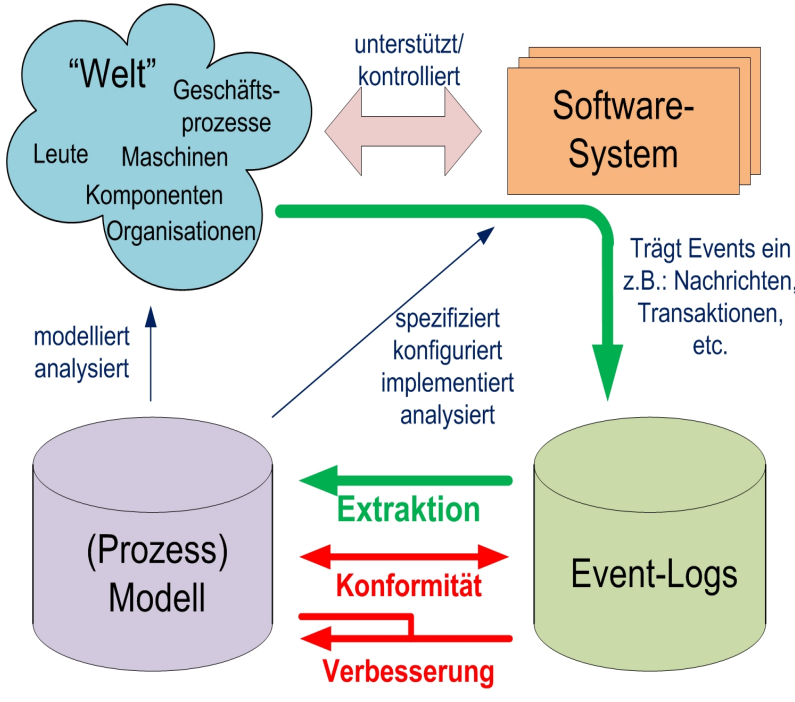
Play-Out



Replay

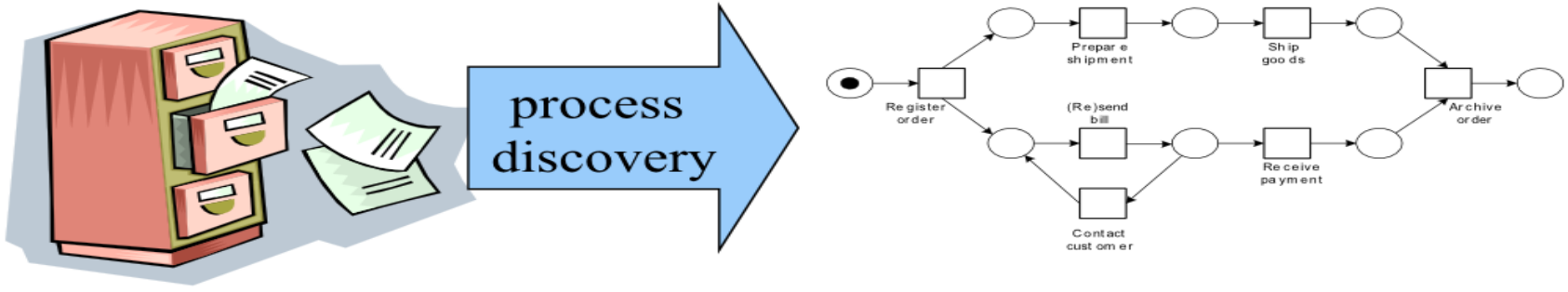


- Erweitertes Modell zeigt: Zeiten, Frequenzen, etc.
- Diagnosen
- Vorhersagen
- Empfehlungen



Prozessextraktion: Idee und Anforderungen

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 irrelevante Daten wegzufiltern.
- **Vollständig**: Relevante Daten für verschiedene GP-Instanzen vorhanden.

Ergebnis: **Workflow-Netz.**

Beispiel-Logfile

Abgebildete Logdatei enthält folgende Folgen von Logdaten für verschiedene GP-Instanzen:

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

Abgebildete Logdatei enthält folgende Folgen von Logdaten für verschiedene GP-Instanzen:

- ABCD (case 1,

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

Abgebildete Logdatei enthält folgende Folgen von Logdaten für verschiedene GP-Instanzen:

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

Zugehöriges Petrinetz ?

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
```

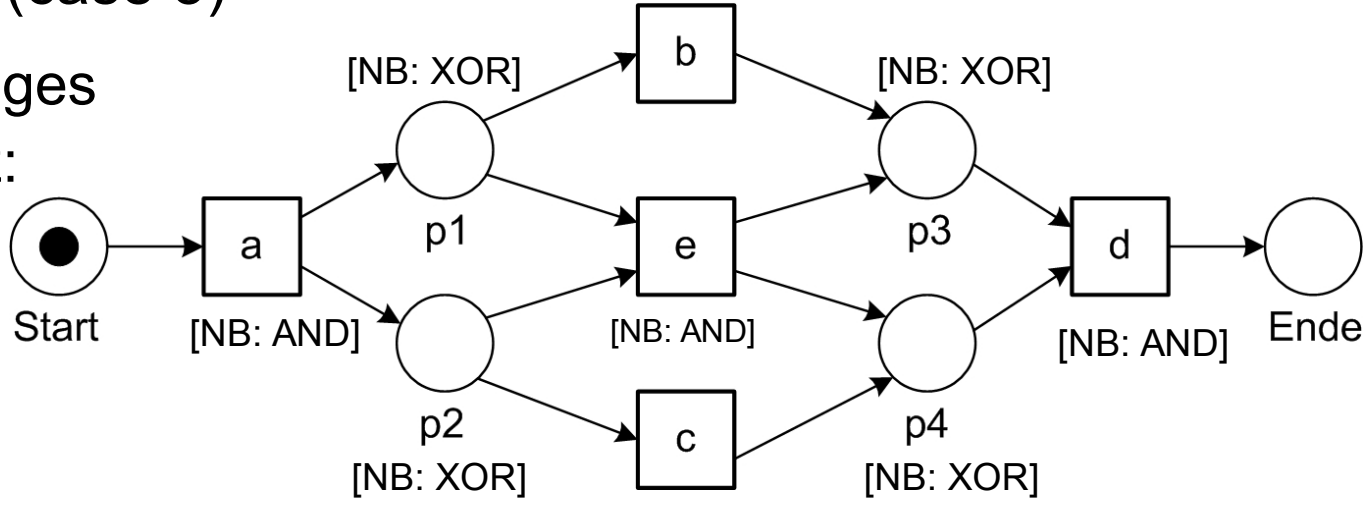
Abstrahiert von „cases“ (insbes. von mehreren gleichen Abläufen).
Jedes Ereignis nur einmal enthalten => nicht einfach o.g. Folgen durch OR-Verzweigungen auflisten.

Beispiel-Logfile

Abgebildete Logdatei enthält folgende Folgen von Logdaten für verschiedene GP-Instanzen:

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

Zugehöriges Petrinetz:

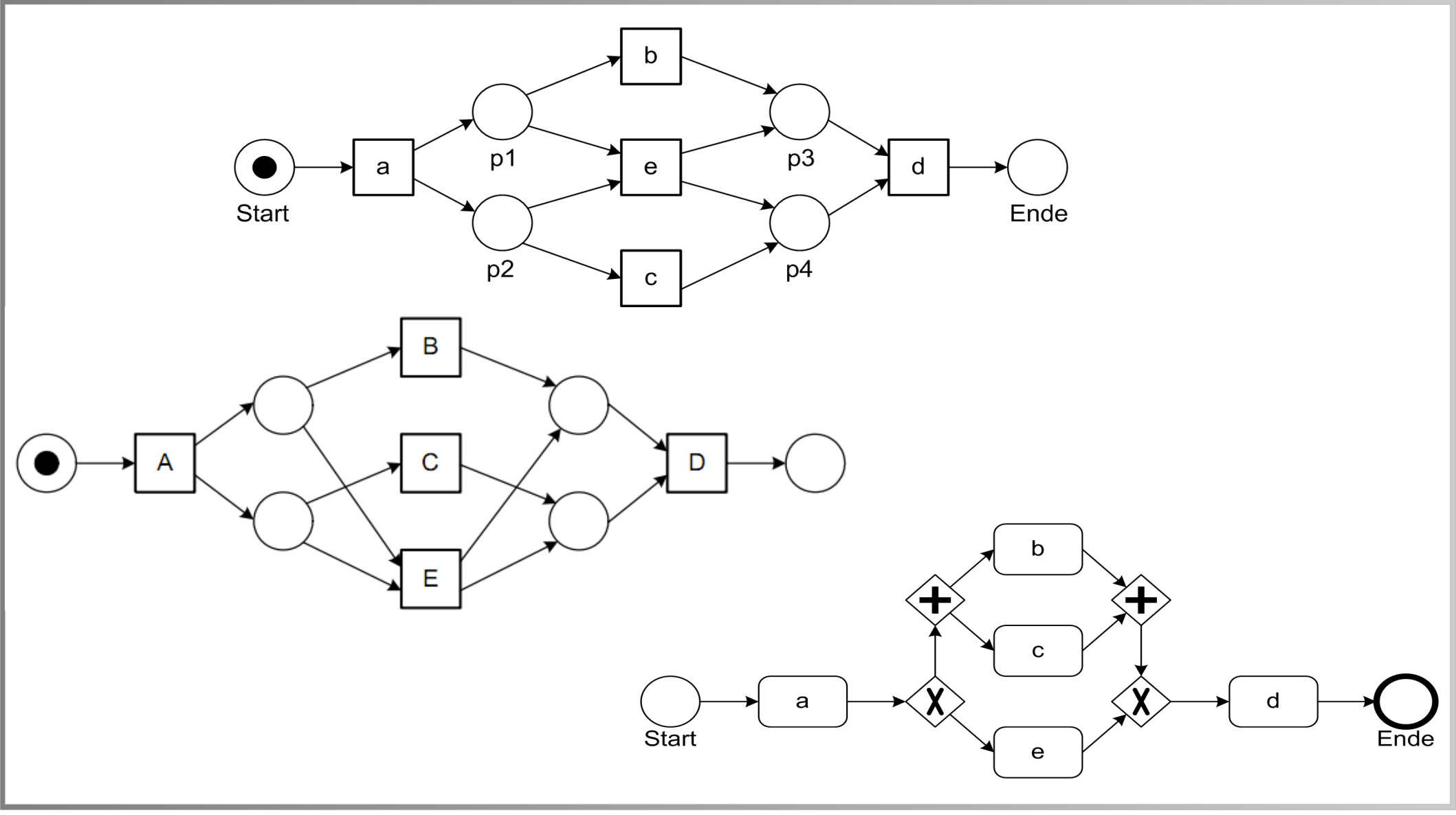


Log-Datei:

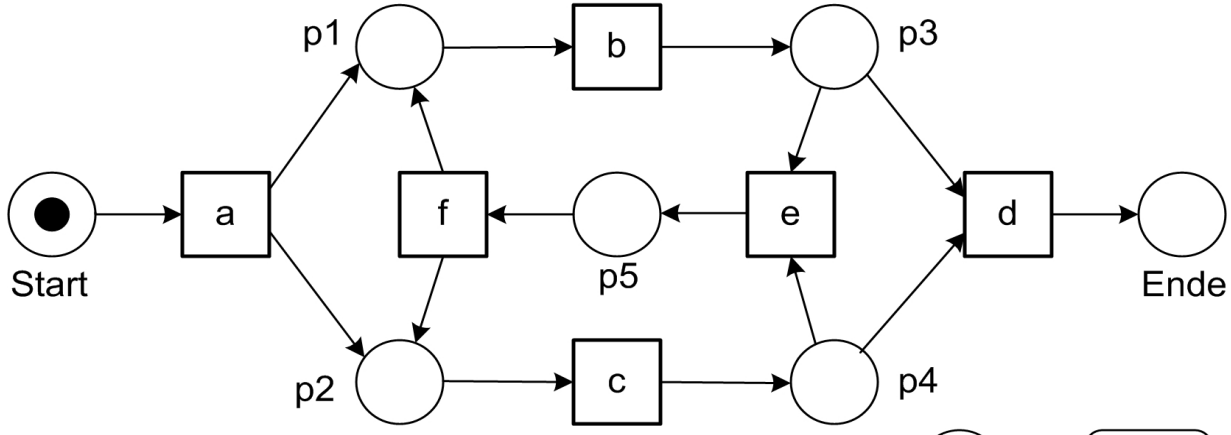
```
case 1, task A
case 2, task A
case 3, task A
case 3, task B
case 1, task B
case 1, task C
case 2, task C
case 4, task A
case 2, task B
case 2, task D
case 5, task A
case 4, task C
case 1, task D
case 3, task C
case 3, task D
case 4, task B
case 5, task E
case 5, task D
case 4, task D
```

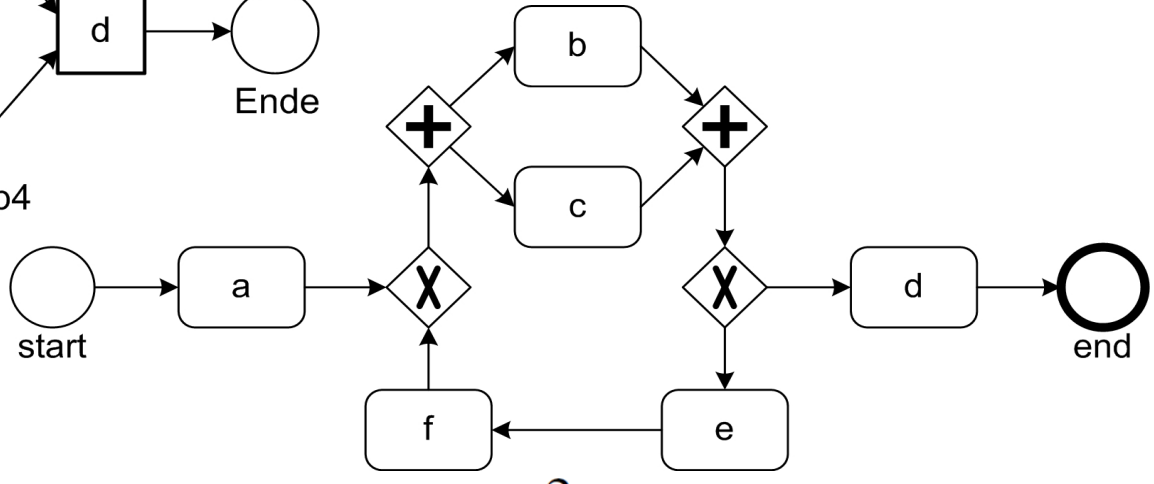
Abstrahiert von „cases“ (insbes. von mehreren gleichen Abläufen).
Jedes Ereignis nur einmal enthalten => nicht einfach o.g. Folgen durch OR-Verzweigungen auflisten.
Erzeugt keine weiteren Eventfolgen (nicht immer der Fall !)

Zugehöriges Petrinetz: Alternative Darstellungen



Weiteres Beispiel (vs. BPMN)

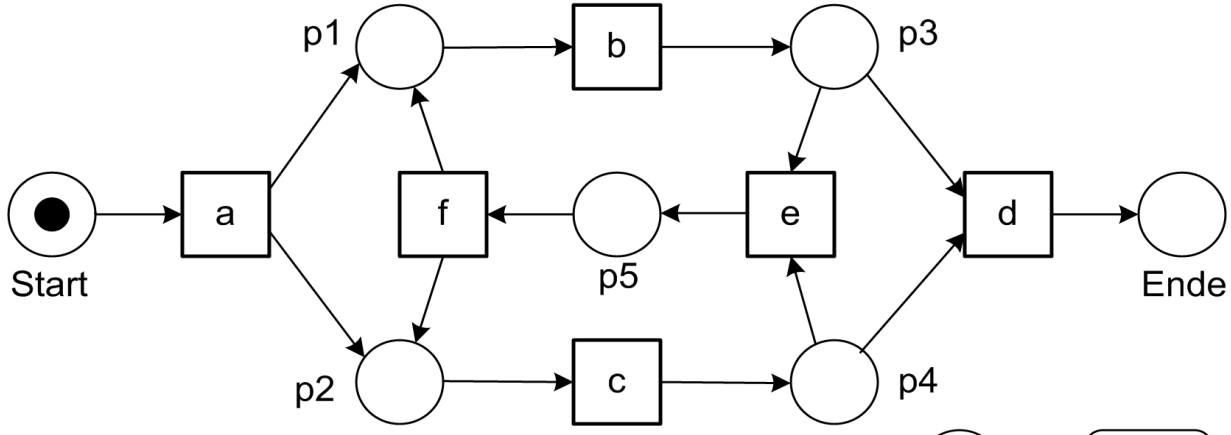


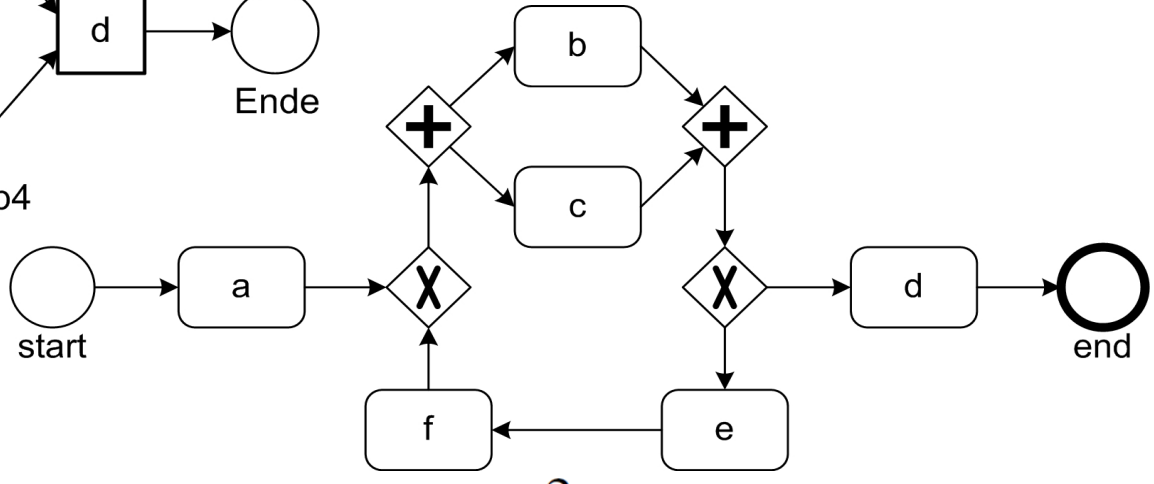


$$L_2 = [\langle a,b,c,d\rangle^3, \langle a,c,b,d\rangle^4, \langle a,b,c,e,f,b,c,d\rangle^2, \langle a,b,c,e,f,c,b,d\rangle,$$
$$\langle a,c,b,e,f,b,c,d\rangle^2, \langle a,c,b,e,f,b,c,e,f,c,b,d\rangle]$$

Superscript von **Event-Folge:** Häufigkeit des Auftretens.
Hier: Event-Log enthält **Teilmenge von möglichen Traces** des Modells
D.h. Modell **Verallgemeinerung der Log-Daten**, lässt weiteres Verhalten zu.
(Welches z.B. ?)

Weiteres Beispiel (vs. BPMN)





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

Superscript von **Event-Folge:** Häufigkeit des Auftretens.
Hier: Event-Log enthält **Teilmenge von möglichen Traces** des Modells
D.h. Modell **Verallgemeinerung der Log-Daten**, lässt weiteres Verhalten zu.
(Welches z.B. ?) => Beliebige Anzahl Iterationen der Schleife !

Anforderungen an extrahiertes Modell

Trade-off zwischen Qualitätskriterien: **Entdecktes Modell soll** ...

- ***Angemessenheit*** *(Fitness):*
  ... Verhalten des Event-Logs zulassen.
- ***Genauigkeit*** *(Precision) - vermeide Unteranpassung (underfitting):*
  ... kein Verhalten ohne Bezug zu Inhalt des Event-Logs zulassen.
- ***Verallgemeinerung*** *(Generalization) - vermeide Überanpassung (overfitting):*
  … Verallgemeinerung des Beispiel-Verhaltens im Event-Log sein.
- ***Einfachheit*** *(Simplicity):*
  … so einfach wie möglich sein.

Überblick
2.4 Prozessextraktion

- Einführung und Beispiel
- **α-Algorithmus**
  - **Idee und Vorbereitungen**
  - Formalisierung
  - Beispiel
  - Weitere Beispiele
  - Einschränkungen
- Allgemeine Herausforderungen beim Process-Mining

α-Algorithmus: Idee

α-Algorithmus: aus Menge von Folgen von Log-Daten GP-Modell als Petrinetz extrahieren.

Idee:

- Informationen über **Abfolge der Aktivitäten** durch Verwendung geeigneter **Ordnungs-Relationen** sammeln.
  - Ob Aktivitäten in verschiedenen Folgen in **Kausal-Beziehung** stehen, oder **parallel** bzw. **unabhängig** voneinander sind.
- Damit Petrinetz konstruieren.

Korrektheit des Algorithmus und benötigte Eigenschaften der Ordnungsrelationen nicht-trivial; hier nicht betrachtet.

α-Algorithmus: Annahmen

Annahmen an zu erzeugende Petrinetze:

- **Ausführung:** jede **Transition erzeugt** jeweils ein **Ereignis** in **Log-Datei** mit zugehöriger Bezeichnung.
- Verschiedene Transitionen haben verschiedene Bezeichnungen.
- Keine Bogen-Vielfachheiten.

Annahmen an Folge von Log-Daten:

- Log-Daten alle vom relevanten GP erzeugt.
- GP durch Petrinetz modellierbar.
- Aufteilung der Log-Daten auf Folgen in gegebener Menge entsprechen verschiedenen GP-Instanzen.
- Abstrahieren von mehrfachen GP-Instanzen mit demselben Verlauf (und von GP-Instanznummern).

Ordnungs-Relationen

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

- **Direkte Nachfolge**: $x >_w y$

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

  $x >_w y$ und nicht $y >_w x$
- **Parallelität**: $x ||_w y$

  $x >_w y$ und $y >_w x$
- **Unabhängigkeit**: $x \#_w y$

  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.)
- A#D, B#E, C#E (und symm.)

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

Anmerkungen

I) (antisymmetrische) **Kausalität**, sowie (symmetrische) **Parallelität** und **Unabhängigkeit** definieren die 4 Möglichkeiten, wie 2 Elemente durch > in Relation stehen können.

II) **Korrektheit** des α-Algorithmus: Unter gegebenen Annahmen an Log-Daten und Petrinetz sind...

1) ... Folgen von Log-Daten durch oben definierte Ordnungs-Relationen **charakterisierbar** (ggf. inkl. Verallgemeinerung).

2) … aus Ordnungs-Relationen Petrinetz **ableitbar**, sodass die vom Petrinetz zur Laufzeit gefeuerten Folgen von Transitionen ursprünglich gegebenen Folgen von Log-Daten enthalten (evtl. auch andere !).

(Verzichten auf (nicht-trivialen) Beweis...)

Patterns für Erzeugung eines Petrinetzes: Kausalität

Ist aus diesen Relationen Petrinetz ableitbar, das diese Aussagen bestätigt ?

Einfachster Fall: **Kausalität** **x** → **y** („sequence pattern“).

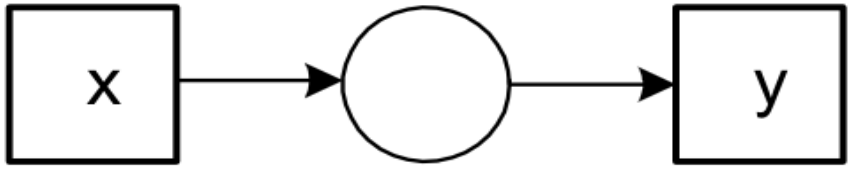
Welches („minimale“) (Teil-)Petrinetz erzeugt bei Ausführung Folgen von Transitionen, die durch diese eine Relation charakterisiert sind ?

Patterns für Erzeugung eines Petrinetzes: Kausalität

Ist aus diesen Relationen Petrinetz ableitbar, das diese Aussagen bestätigt ?

Einfachster Fall: **Kausalität** **x** → **y** („sequence pattern“)

Welches („minimale“) (Teil-)Petrinetz erzeugt bei Ausführung Folgen von Transitionen, die durch diese eine Relation charakterisiert sind ?



Patterns für Erzeugung eines Petrinetzes (2)

Welches („minimale“) (Teil-)Petrinetz erzeugt Folgen von Transitionen, die jeweils durch folgende Relationen charakterisiert sind ?

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

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

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

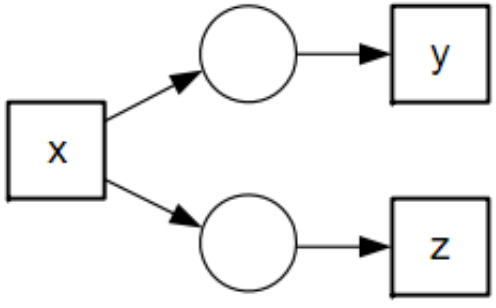
$x \rightarrow z,\ y \rightarrow z,\ x \# y$

Patterns für Erzeugung eines Petrinetzes (2)

Welches („minimale“) (Teil-)Petrinetz erzeugt Folgen von Transitionen, die jeweils durch folgende Relationen charakterisiert sind ?

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

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



AND-split pattern

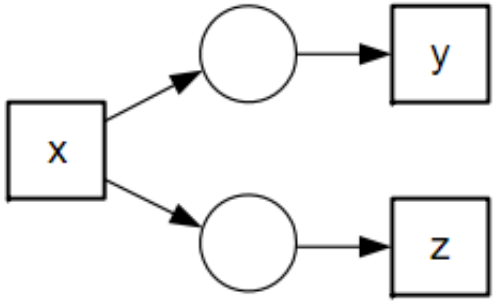
$x \rightarrow z,\ y \rightarrow z,\ x||y$

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

Patterns für Erzeugung eines Petrinetzes (2)

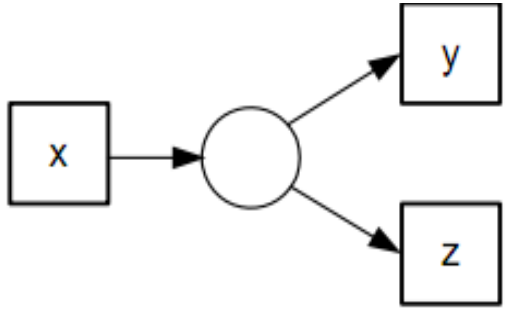
Welches („minimale“) (Teil-)Petrinetz erzeugt Folgen von Transitionen, die jeweils durch folgende Relationen charakterisiert sind ?

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



AND-split pattern

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



XOR-split pattern

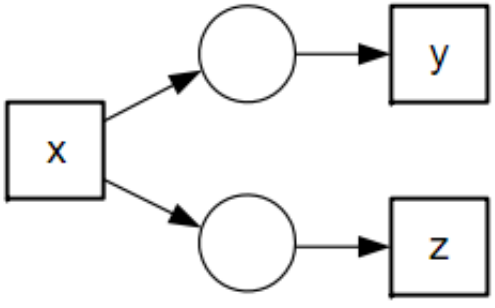
$x \rightarrow z,\ y \rightarrow z,\ x \,||\, y$

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

Patterns für Erzeugung eines Petrinetzes (2)

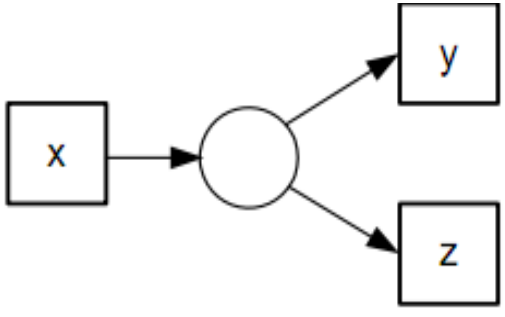
Welches („minimale“) (Teil-)Petrinetz erzeugt Folgen von Transitionen, die jeweils durch folgende Relationen charakterisiert sind ?

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



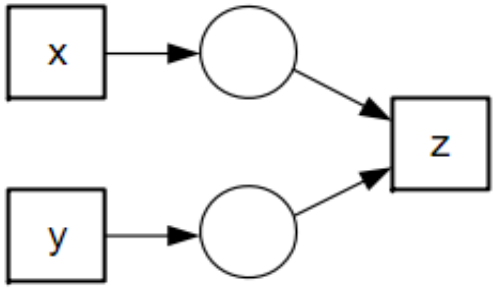
AND-split pattern

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



XOR-split pattern

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



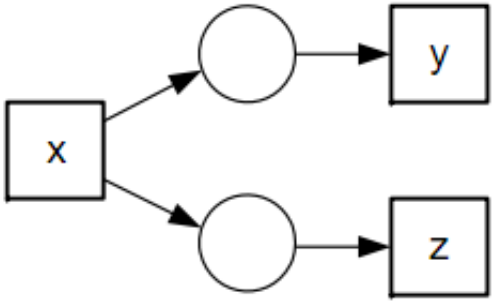
AND-join pattern

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

Patterns für Erzeugung eines Petrinetzes (2)

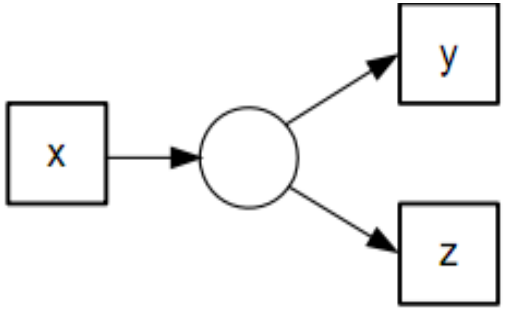
Welches („minimale“) (Teil-)Petrinetz erzeugt Folgen von Transitionen, die jeweils durch folgende Relationen charakterisiert sind ?

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



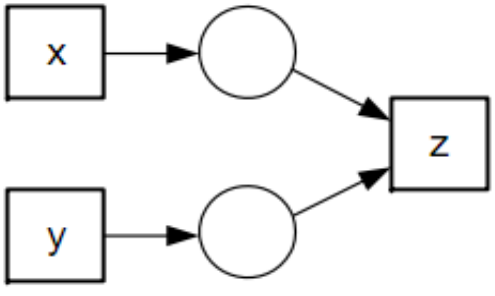
AND-split pattern

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



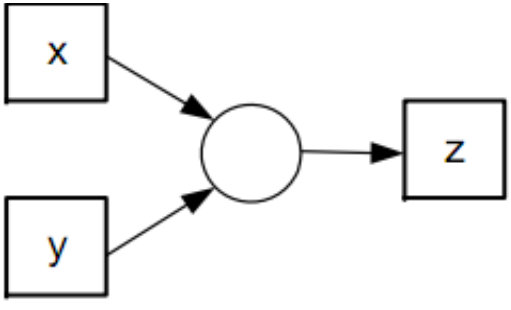
XOR-split pattern

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



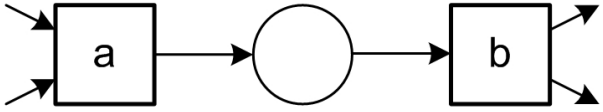
AND-join pattern

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

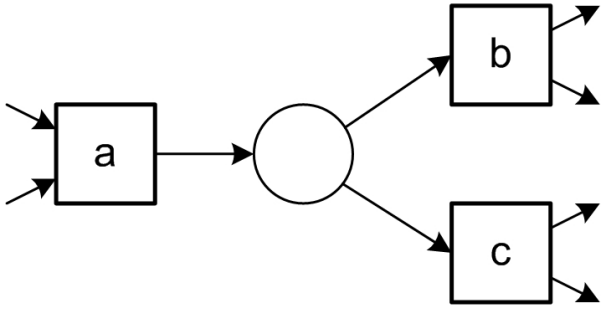


XOR-join pattern

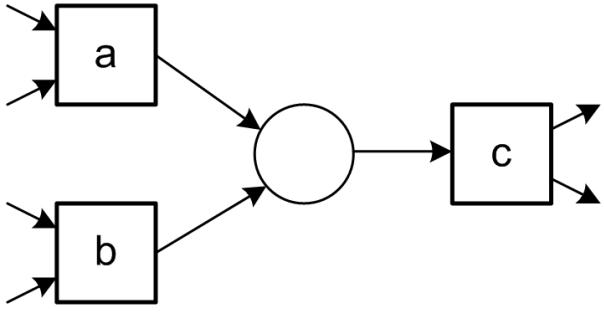
Überblick: Patterns für Erzeugung eines Petrinetzes



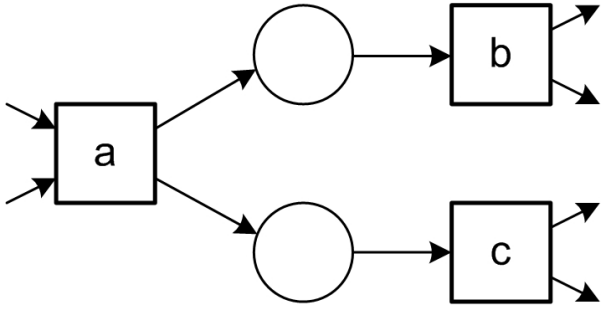
(a) sequence pattern: a→b



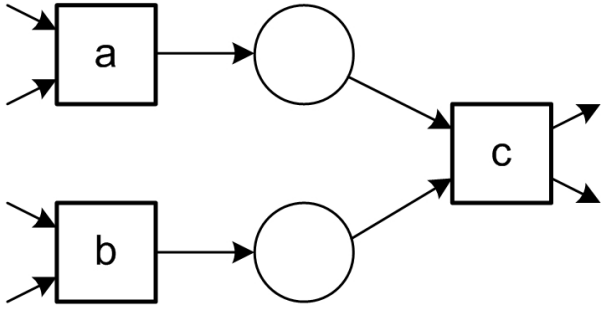
(b) XOR-split pattern:
a→b, a→c, and b#c



(c) XOR-join pattern:
a→c, b→c, and a#b



(d) AND-split pattern:
a→b, a→c, and b||c



(e) AND-join pattern:
a→c, b→c, and a||b

Überblick
2.4 Prozessextraktion

- Einführung und Beispiel
- **α-Algorithmus**
  - Idee und Vorbereitungen
  - **Formalisierung**
  - Beispiel
  - Weitere Beispiele
  - Einschränkungen
- Allgemeine Herausforderungen beim Process-Mining

Der α-Algorithmus Schritt 1-3: Transitionen

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

Gesucht: Workflow-Schema als **Petrinetz**

α(W) = (Stellen P_W, Transitionen T_W, Verbindungen F_W)

Schritte 1-3 (Transitionen)

Annahme (zur Erinnerung):

- Ausführung: jede Transition erzeugt ein Ereignis in Log-Datei; Bezeichnung wie im Petrinetz.

→ Wie demnach Menge der Transitionen im Petrinetz definieren ?

Der α-Algorithmus Schritt 1-3: Transitionen

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

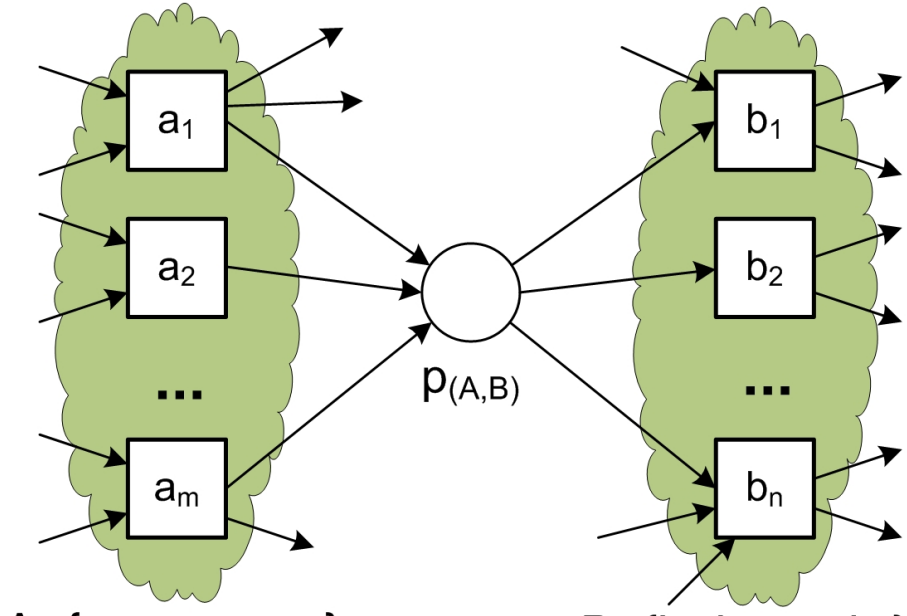
Gesucht: Workflow-Schema als **Petrinetz**
α(W) = (Stellen P_W, Transitionen T_W, Verbindungen F_W)

Schritte 1-3 (Transitionen):

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**

Der α-Algorithmus Schritt 4: Kandidaten für Stellen (Idee)

LEHRSTUHL 14
SOFTWARE ENGINEERING

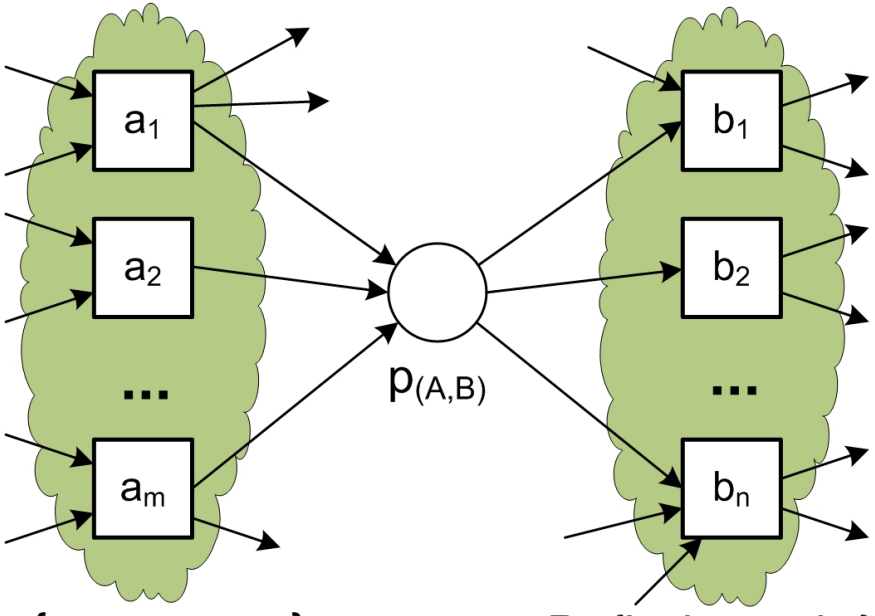


	a_1	a_2	…	a_m	b_1	b_2	…	b_n
a_1								
a_2								
…								
a_m								
b_1								
b_2								
…								
b_n								

Welche Relationen (wie vorher definiert) zwischen Transitionen $a_1,...,a_m$, $b_1,...,b_n$ sind in dieser Situation in der Logdatei zu erwarten ?

Wie könnte man dann die Stelle $p_{(A,B)}$ aus der Logdatei ableiten ?

Der α-Algorithmus Schritt 4: Kandidaten für Stellen (Idee)



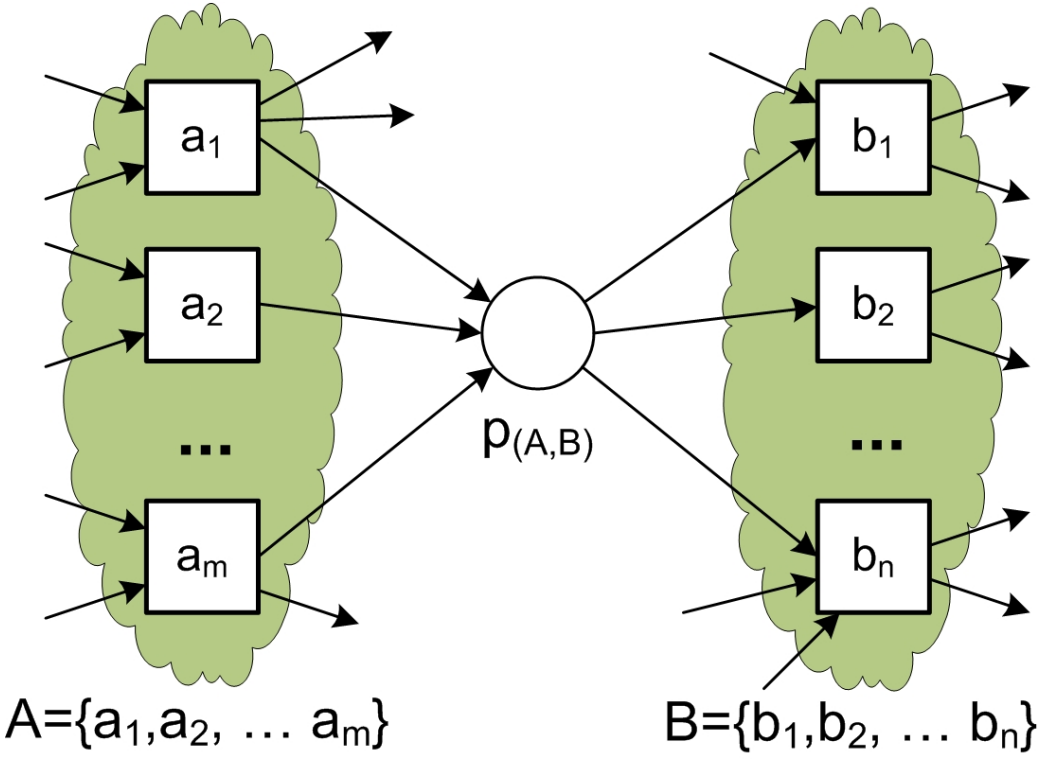
→ $p_{(A,B)}$=(A,B) Repräsentant für **Stellen**, die Paare von **kausal aufeinander-folgenden Transitionen** verbinden (Unabhängigkeit: gleichzeitig möglich).

Via Transitionsmengen (A,B), weil Stelle nicht direkt in Logdaten beobachtbar.

Wie allgemein definieren ?

	a_1	a_2	…	a_m	b_1	b_2	…	b_n
a_1	#	#	…	#	→	→	…	→
a_2	#	#	…	#	→	→	…	→
…	…	…	…	…	…	…	…	…
a_m	#	#	…	#	→	→	…	→
b_1	←	←	…	←	#	#	…	#
b_2	←	←	…	←	#	#	…	#
…	…	…	…	…	…	…	…	…
b_n	←	←	…	←	#	#	…	#

Der α-Algorithmus Schritt 4: Kandidaten für Stellen



X_W: „**verallgemeinerte Kausalitätsrelationen**“:

Kandidaten für **Stellen**, um Paare von **kausal aufeinanderfolgenden Transitionen** zu verbinden.

Beispiele:

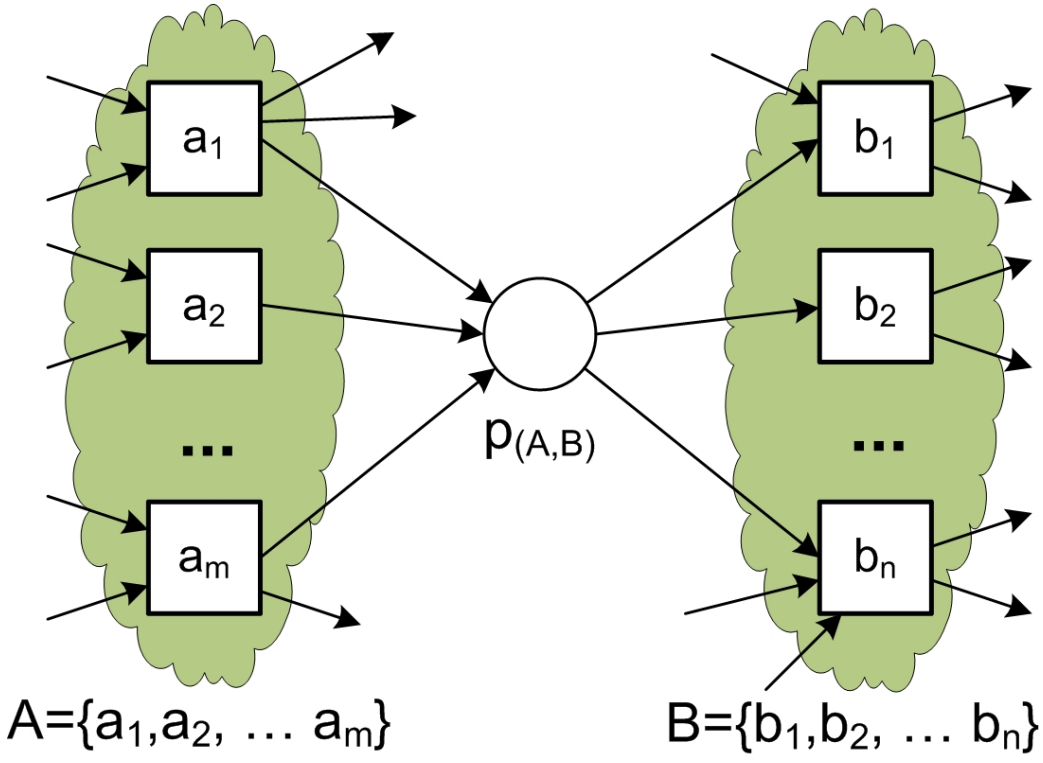
$(\{a\},\{b\}) \in X_W$ für alle $a \rightarrow b$

„Kandidaten“ weil teilweise gegenseitig redundant (Teilmengen (A',B') auch dabei).

Wie könnte man das reparieren ?

$$4.\ X_W = \{ (A,B) \mid A \subseteq T_W \wedge A \neq \emptyset \wedge B \subseteq T_W \wedge B \neq \emptyset \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a1,a2 \in A} a_1 \#_w a_2 \wedge \forall_{b1,b2 \in B} b_1 \#_W b_2 \}$$

Der α-Algorithmus Schritt 5-6: Stellen



Y_W: „**Maximale Kausalitätsrelationen**“.

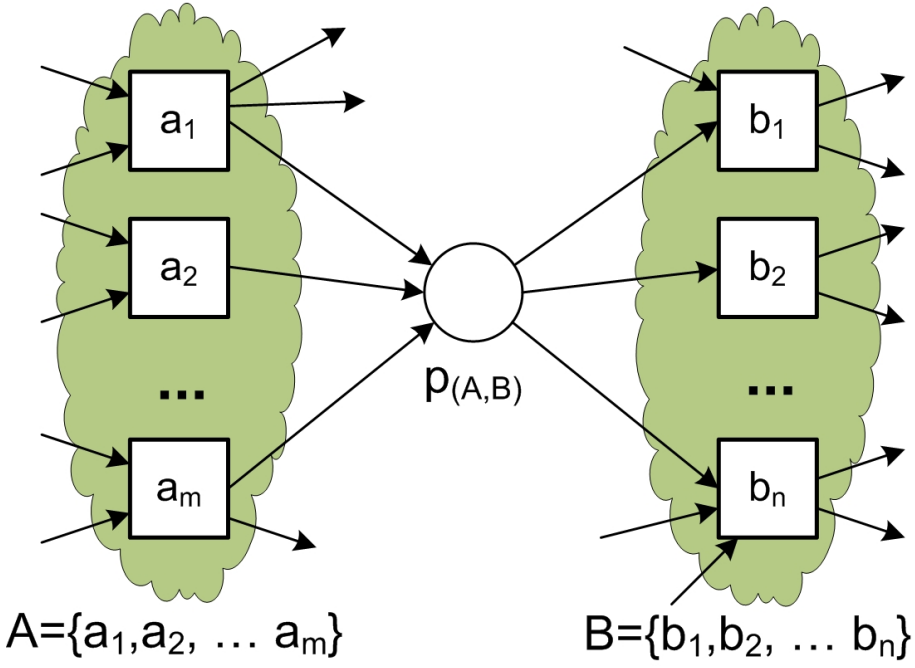
Maximalität: soweit möglich keine unnötigen Stellen erzeugen, um Paare von kausal aufeinander folgenden Transitionen zu verbinden.

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

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

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

Der α-Algorithmus: Abschluss



Haben Transitionen und Stellen.

Was fehlt für Petrinetz und wie könnte man das aus Logdatei ableiten ?

	a_1	a_2	…	a_m	b_1	b_2	…	b_n
a_1	#	#	…	#	→	→	…	→
a_2	#	#	…	#	→	→	…	→
…	…	…	…	…	…	…	…	…
a_m	#	#	…	#	→	→	…	→
b_1	←	←	…	←	#	#	…	#
b_2	←	←	…	←	#	#	…	#
…	…	…	…	…	…	…	…	…
b_n	←	←	…	←	#	#	…	#

Der α-Algorithmus Schritt 7-8: Stellen und Verbindungen

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

Verbindungen: Durch Kausalitätsrelationen definierte Verbindungen. Zusätzlich: Verbindungen zwischen Start- bzw. Endstellen und Start- bzw. Endtransitionen.

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

.

α-Algorithmus: Diskussion

Frage:

Wie würde man **Korrektheit** des Algorithmus zeigen ?

α-Algorithmus: Diskussion

Frage:

Wie würde man **Korrektheit** des Algorithmus zeigen ?

Antwort:

Z.z.: erhaltenes Petrinetz gibt bei Ausführung **ursprüngliche Folgen von Logdaten** aus.

Gilt i.A. nur unter o.g. Annahmen (z.B. Logdaten von einem Petrinetz generiert).

Petrinetz kann i.A. noch andere Folgen von Logdaten ausgeben !

Überblick
2.4 Prozessextraktion

- Einführung und Beispiel
- **α-Algorithmus**
  - Idee und Vorbereitungen
  - Formalisierung
  - **Beispiel**
  - Weitere Beispiele
  - Einschränkungen
- Allgemeine Herausforderungen beim Process-Mining

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 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 einzelnen Logs

α-Algorithmus – Beispiel

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

Ursprungs-Log:
{ABCD, ACBD, AED}

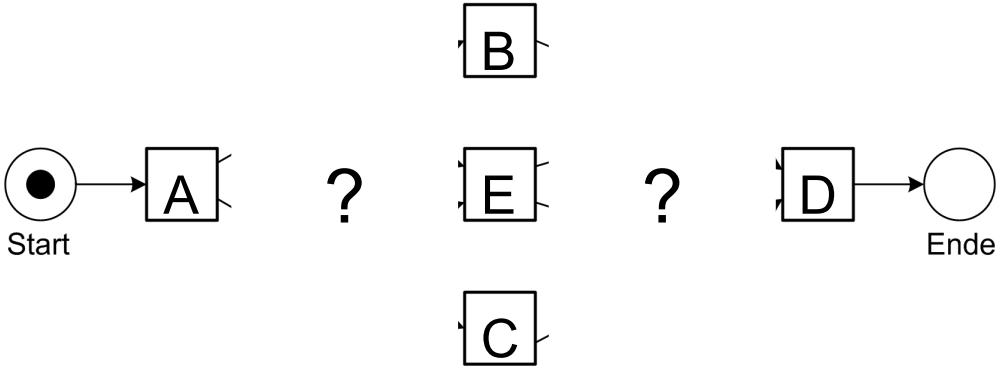
$T_I = \{t \in T \mid \exists_{\sigma \in W}\, t = last(\sigma)\}$
Sammle Endtransitionen aus einzelnen Logs

α-Algorithmus – Beispiel

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

Ursprungs-Log:
{ABCD, ACBD, AED}

$T_I = \{t \in T \mid \exists_{\sigma \in W}\, t = last(\sigma)\}$
Sammle Endtransitionen aus einzelnen Logs



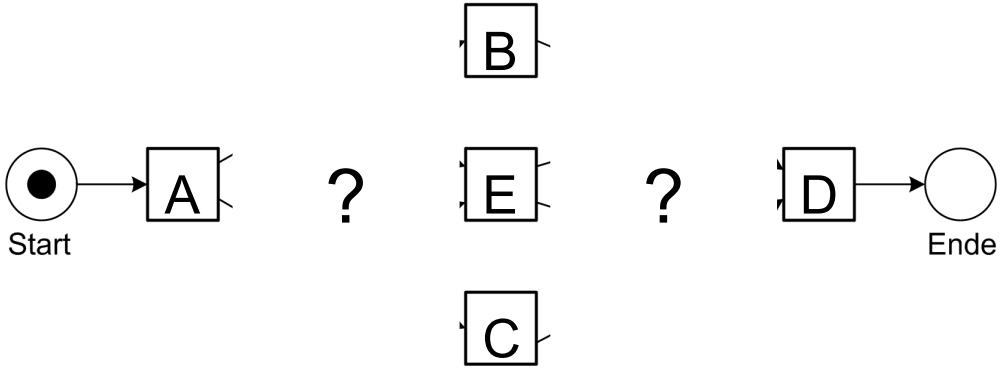
α-Algorithmus – Beispiel

1. T_W =
2. T_I = {
3. T_O =

Bilde Kandidaten für Stellen.

Ursprungs-Log:
{ABCD, ACBD, AED}

4. X_W = {



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

NB:

A: Event / Transition

$\mathcal{A}$: Menge von Events / Transitionen

α-Algorithmus – Beispiel

Bilde Kandidaten für Stellen:
Bilde Teilmengen mit unabhängigen Elementen
als Kandidaten für Mengen ***A*** und ***B***

Ursprungs-Log:
{ABCD, ACBD, AED}

4. X_W = {

a>b a>c a>e b>c b>d c>b c>d e>d	a→b a→c a→e b→d c→d e→d	b||c c||b	b#e e#b c#e a#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 Kandidaten für Stellen:
Bilde Teilmengen mit unabhängigen Elementen
als Kandidaten für Mengen ***A*** und ***B***
{A},{A,D}

Ursprungs-Log:
{ABCD, ACBD, AED}

Alle andere Transitionen abhängig von A, bis auf D

3. $T_O = \{D\}$

4. $X_W = \{$

a>b a>c a>e b>c b>d c>b c>d e>d	a→b a→c a→e b→d c→d e→d	b||c c||b	b#e e#b c#e a#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 Kandidaten für Stellen:
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 von B
E unabhängig

3. T_O = {D}

4. X_W = {

a>b a>c a>e b>c b>d c>b c>d e>d	a→b a→c a→e b→d c→d e→d	b||c c||b	b#e e#b c#e a#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 Kandidaten für Stellen:
Bilde Teilmengen mit unabhängigen Elementen
als Kandidaten für Mengen ***A*** und ***B***
{A},{A,D},{B},{B,E},{C},{C,E},{D},{E}

Ursprungs-Log:
{ABCD, ACBD, AED}

3. T_O = {D}

4. X_W = {

a>b a>c a>e b>c b>d c>b c>d e>d	a→b a→c a→e b→d c→d e→d	b||c c||b	b#e e#b c#e a#d …

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

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

α-Algorithmus – Beispiel

Bilde Kandidaten für Stellen:
Bilde Paare (***A***,***B***), 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. $T_O = \{D\}$

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

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

a>b	a→b	b‖c	b#e
a>c	a→c	c‖b	e#b
a>e	a→e		c#e
b>c	b→d		a#d
b>d	c→d		…
c>b	e→d		
c>d			
e>d			

α-Algorithmus – Beispiel

Bilde Kandidaten für Stellen:
Bilde Paare (A,B), 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. $X_W = \{(\{A\}, \{B\}), (\{A\}, \{C\}), (\{A\}, \{E\}), (\{B\}, \{D\}),$
$(\{A\}, \{B, E\}), (\{A\}, \{C, E\}),$

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

a>b	a→b	b‖c	b#e
a>c	a→c	c‖b	e#b
a>e	a→e		c#e
b>c	b→d		a#d
b>d	c→d		…
c>b	e→d		
c>d			
e>d			

α-Algorithmus – Beispiel

X_W: Kandidaten für Stellen.

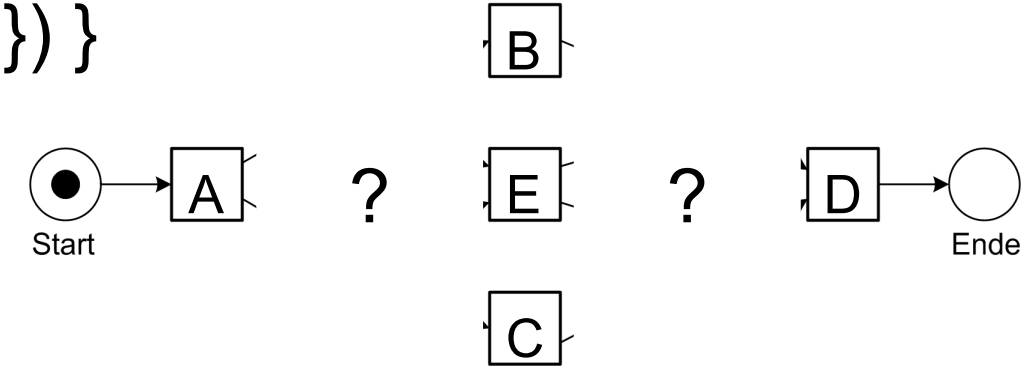
Ursprungs-Log:
{ABCD, ACBD, AED}

1.

2.

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}) }



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

α-Algorithmus – Beispiel

Ursprungs-Log:
{ABCD, ACBD, AED}

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

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

Entferne alle Paare (A,B) aus X_W, die Teilmenge eines anderen Paares sind.

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 = {

α-Algorithmus – Beispiel

Ursprungs-Log:
{ABCD, ACBD, AED}

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

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

Entferne alle Paare (A,B) aus X_W, die Teilmenge eines anderen Paares sind.

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}),

α-Algorithmus – Beispiel

Ursprungs-Log:
{ABCD, ACBD, AED}

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

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

Entferne alle Paare (A,B) aus X_W, die Teilmenge eines anderen Paares sind.

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})}

α-Algorithmus – Beispiel

Ursprungs-Log:
{ABCD, ACBD, AED}

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

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

Entferne alle Paare (A,B) aus X_W, die Teilmenge eines anderen Paares sind.

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})}

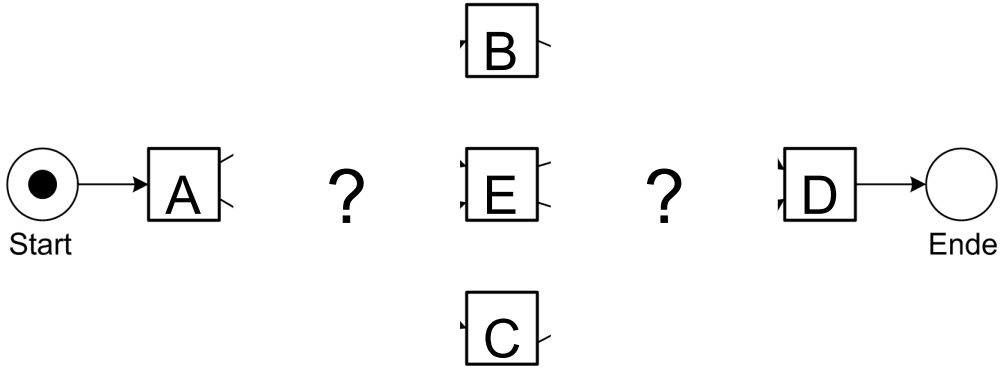
α-Algorithmus – Beispiel

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

6. P_W = {

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

Füge für jedes Paar (A,B) eine Stelle, sowie eine Start- und Endstelle ein.



α-Algorithmus – Beispiel

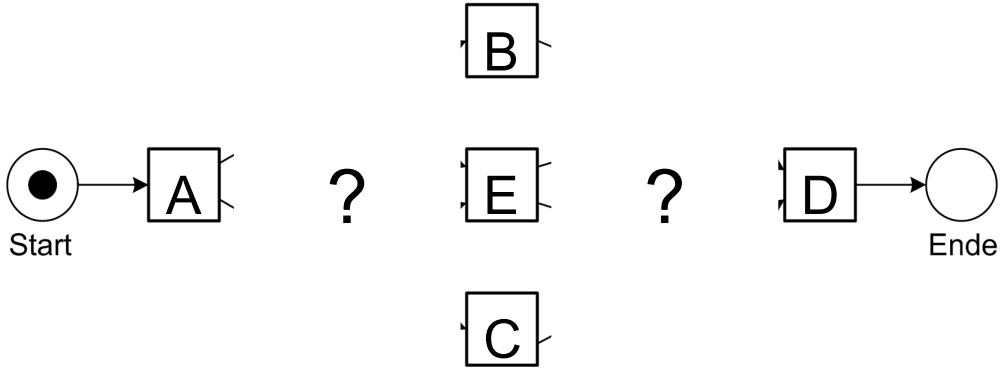
5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}),

({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 Paar (A,B) eine Stelle, sowie eine Start- und Endstelle ein.



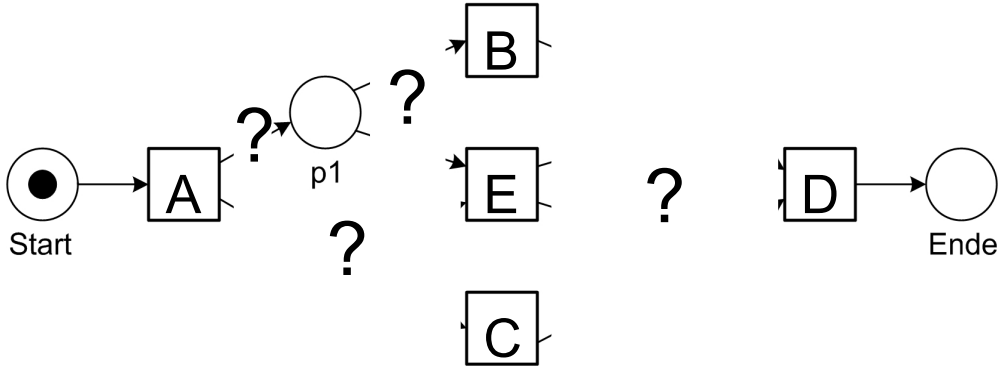
α-Algorithmus – Beispiel

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 Paar (*A*,*B*) eine Stelle, sowie eine Start- und Endstelle ein.

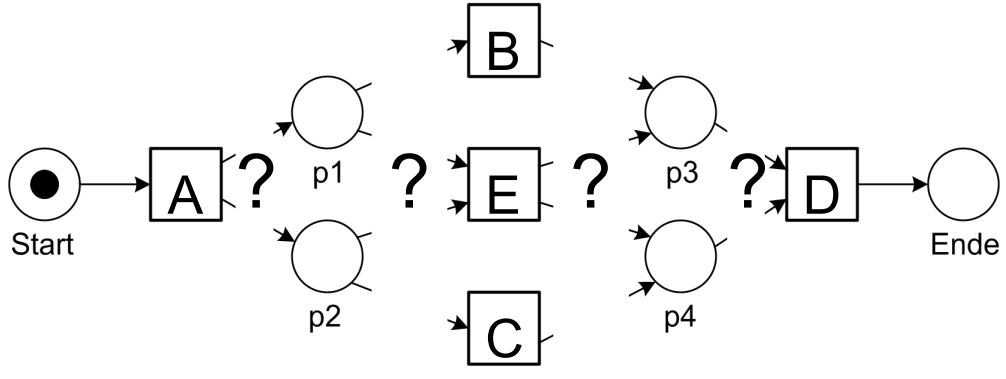


α-Algorithmus – Beispiel

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})},

$P_W = \{(p_{(A,B)}) \mid (A,B) \in Y_W\} \cup \{i_W, o_W\}$
Füge für jedes Paar (A,B) eine Stelle, sowie eine Start- und Endstelle ein.



α-Algorithmus – Beispiel

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

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

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

$(\{C, E\},\{D\})\}$

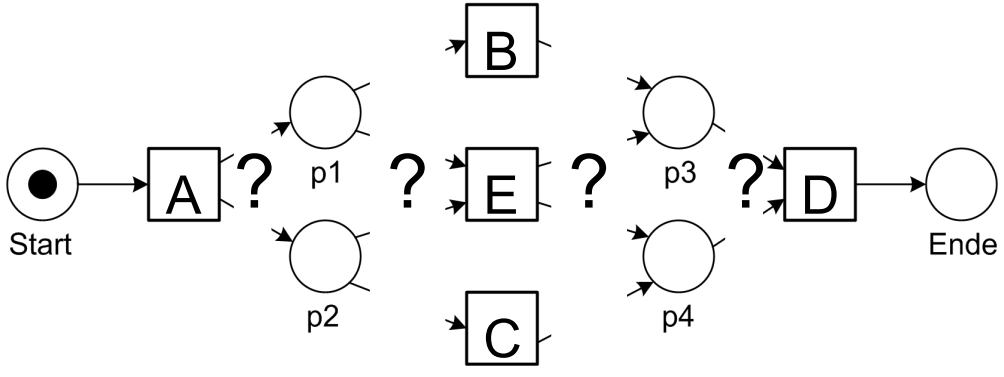
6. $P_W = \{i_W, o_W, p(\{A\},\{B,E\}), p(\{A\},\{C,E\}), p(\{B,E\},\{D\}),$

$p(\{C,E\},\{D\})\},$

7. $F_W = \{$

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

Füge die zugehörigen Bögen ein.



α-Algorithmus – Beispiel

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

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

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

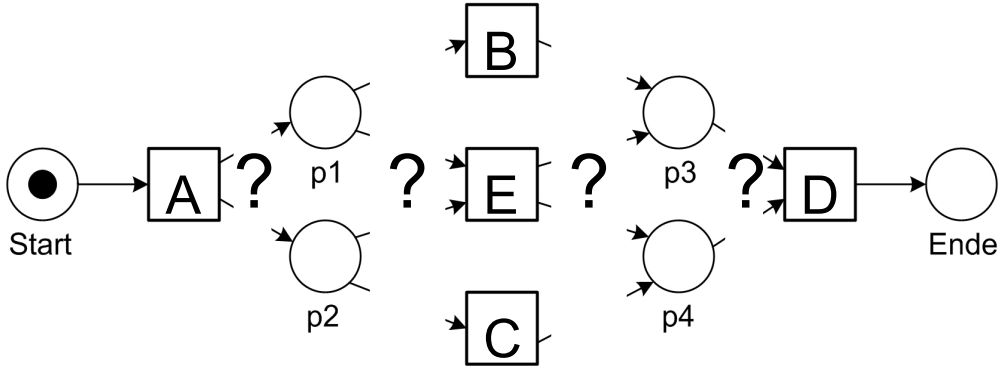
({C, E},{D})}

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),

..., (D, o_W)



α-Algorithmus – Beispiel

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

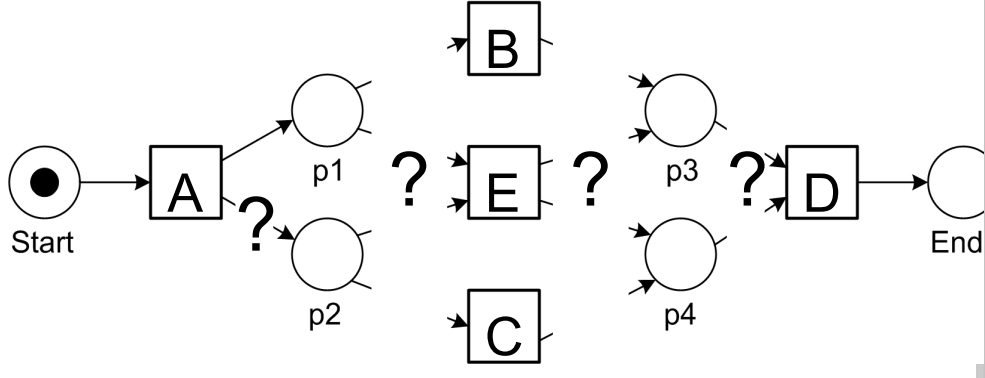
2. $T_I = \{A\}$ 3. $T_O = \{D\}$

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

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

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)



α-Algorithmus – Beispiel

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

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

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

({B, E}, {D}),

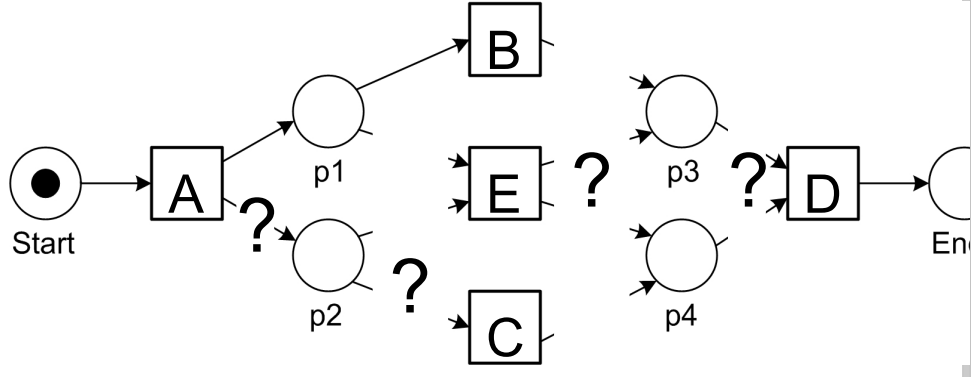
Füge die zugehörigen Bögen ein:
Verbinde zugehörigen Stelle eines Y_W – Tupels mit den Elementen aus B

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)



α-Algorithmus – Beispiel

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

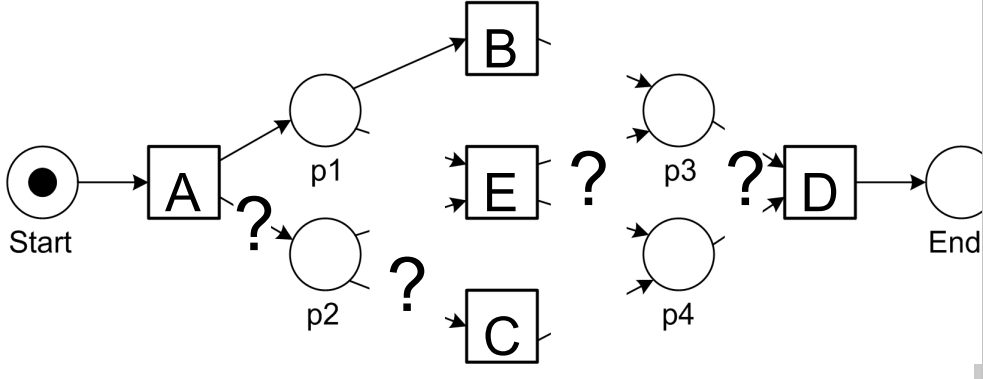
2. $T_I = \{A\}$ 3. $T_O = \{D\}$

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

Füge die zugehörigen Bögen ein: Etc.

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

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



α-Algorithmus – Beispiel

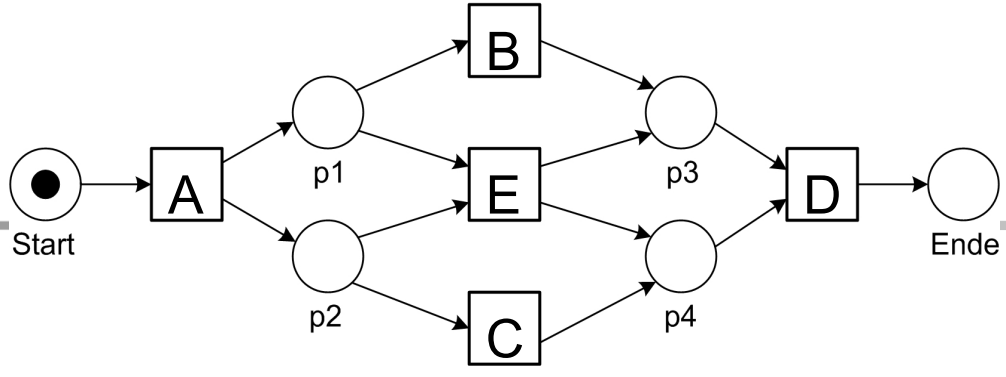
5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}),
({C, E},{D})}

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

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



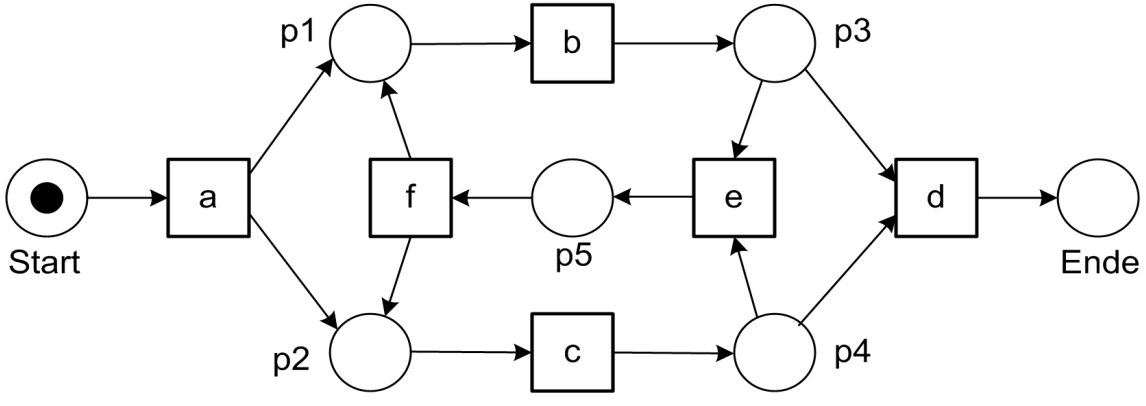
Überblick
2.4 Prozessextraktion

- Einführung und Beispiel
- **α-Algorithmus**
  - Idee und Vorbereitungen
  - Formalisierung
  - Beispiel
  - **Weitere Beispiele**
  - Einschränkungen
- Allgemeine Herausforderungen beim Process-Mining

2. Beispiel: L_2

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

	a	b	c	d	e	f
a	#	$\rightarrow$	$\rightarrow$	#	#	#
b	$\leftarrow$	#	$\parallel$	$\rightarrow$	$\rightarrow$	$\leftarrow$
c	$\leftarrow$	$\parallel$	#	$\rightarrow$	$\rightarrow$	$\leftarrow$
d	#	$\leftarrow$	$\leftarrow$	#	#	#
e	#	$\leftarrow$	$\leftarrow$	#	#	$\rightarrow$
f	#	$\rightarrow$	$\rightarrow$	#	$\leftarrow$	#



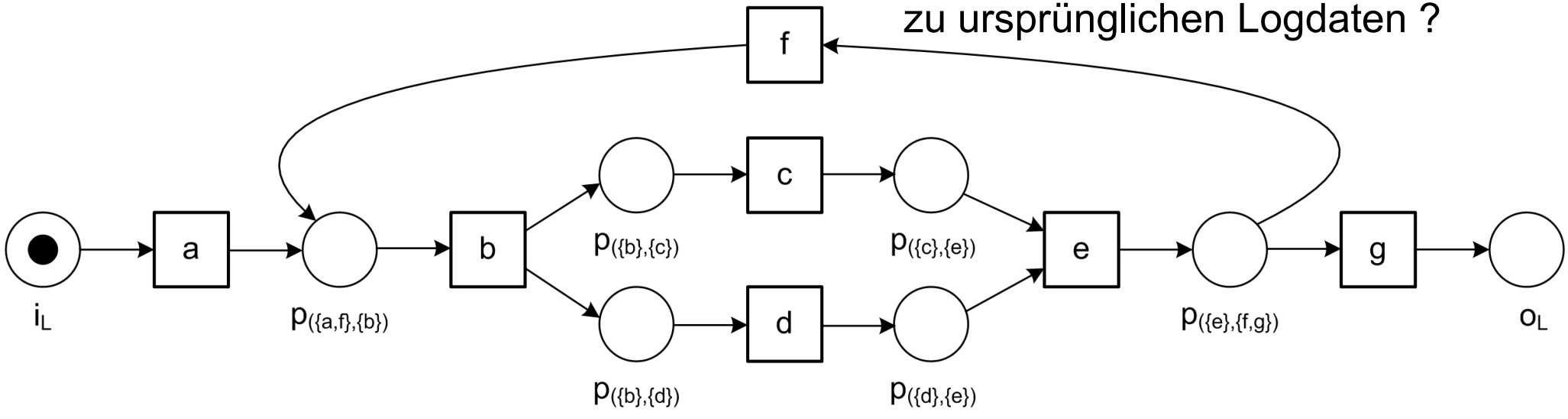
Welches **zusätzliche Verhalten** ergibt sich aus Petrinetz im Vergleich zu ursprünglichen Logdaten ?

3. Beispiel: L_3

$$L_3 = [\langle a,b,c,d,e,f,b,d,c,e,g\rangle, \langle a,b,d,c,e,g\rangle^2, \langle a,b,c,d,e,f,b,c,d,e,f,b,d,c,e,g\rangle]$$

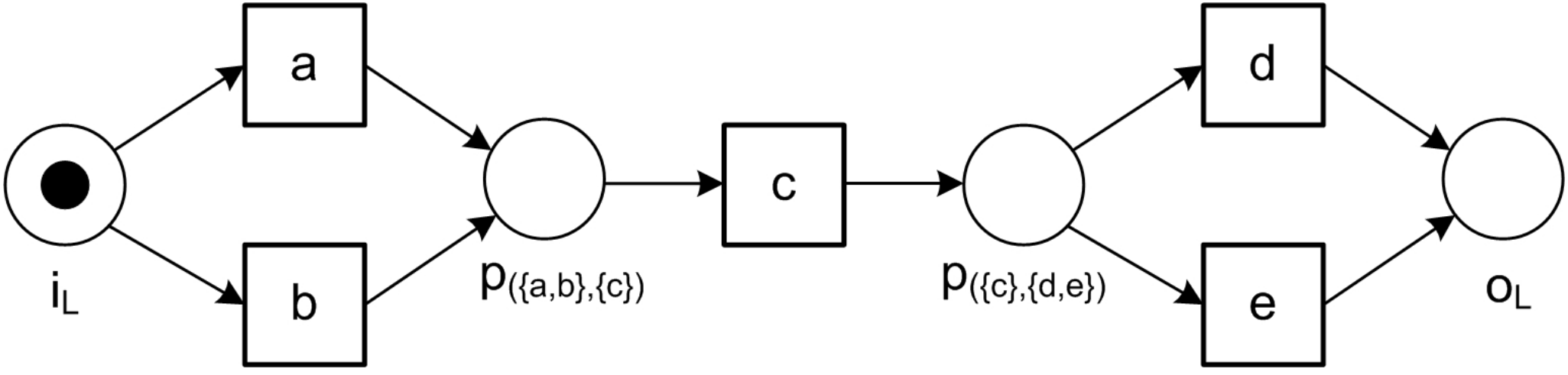
	a	b	c	d	e	f	g
a	#	$\rightarrow$	#	#	#	#	#
b	$\leftarrow$	#	$\rightarrow$	$\rightarrow$	#	$\leftarrow$	#
c	#	$\leftarrow$	#	$\|\|$	$\rightarrow$	#	#
d	#	$\leftarrow$	$\|\|$	#	$\rightarrow$	#	#
e	#	#	$\leftarrow$	$\leftarrow$	#	$\rightarrow$	$\rightarrow$
f	#	$\rightarrow$	#	#	$\leftarrow$	#	#
g	#	#	#	#	$\leftarrow$	#	#

Welches **zusätzliche Verhalten** ergibt sich aus Petrinetz im Vergleich zu ursprünglichen Logdaten ?



4. Beispiel: L_4

$$L_4 = [\langle a,c,d\rangle^{45}, \langle b,c,d\rangle^{42}, \langle a,c,e\rangle^{38}, \langle b,c,e\rangle^{22}]$$



5. Beispiel L_5: Event-Log und Relationen

$$L_5 = [\langle a,b,e,f\rangle^2, \langle a,b,e,c,d,b,f\rangle^3, \langle a,b,c,e,d,b,f\rangle^2, \langle a,b,c,d,e,b,f\rangle^4, \langle a,e,b,c,d,b,f\rangle^3]$$

	a	b	c	d	e	f
a	#	$\rightarrow$	#	#	$\rightarrow$	#
b	$\leftarrow$	#	$\rightarrow$	$\leftarrow$	$\|\|$	$\rightarrow$
c	#	$\leftarrow$	#	$\rightarrow$	$\|\|$	#
d	#	$\rightarrow$	$\leftarrow$	#	$\|\|$	#
e	$\leftarrow$	$\|\|$	$\|\|$	$\|\|$	#	$\rightarrow$
f	#	$\leftarrow$	#	#	$\leftarrow$	#

5. Beispiel L_5: Durchführung α-Algorithmus

$$T_L = \{a, b, c, d, e, f\}$$

$$T_I = \{a\}$$

$$T_I = \{f\}$$

$$X_L = \{(\{a\},\{b\}),(\{a\},\{e\}),(\{b\},\{c\}),(\{b\},\{f\}),(\{c\},\{d\}),$$
$$(\{d\},\{b\}),(\{e\},\{f\}),(\{a,d\},\{b\}),(\{b\},\{c,f\})\}$$

$$Y_L = \{(\{a\},\{e\}),(\{c\},\{d\}),(\{e\},\{f\}),(\{a,d\},\{b\}),(\{b\},\{c,f\})\}$$

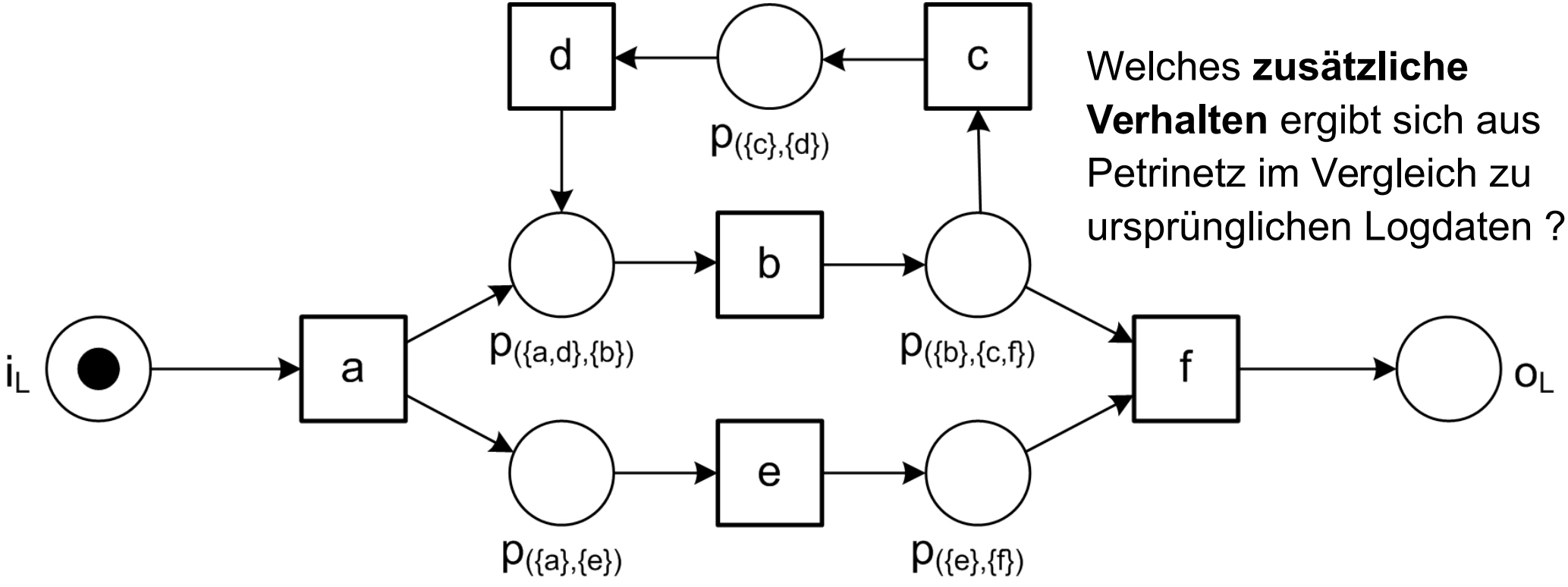
$$P_L = \{p_{(\{a\},\{e\})}, p_{(\{c\},\{d\})}, p_{(\{e\},\{f\})}, p_{(\{a,d\},\{b\})}, p_{(\{b\},\{c,f\})}, i_L, o_L\}$$

$$F_L = \{(a, p_{(\{a\},\{e\})}), (p_{(\{a\},\{e\})}, e), (c, p_{(\{c\},\{d\})}), (p_{(\{c\},\{d\})}, d),$$
$$(e, p_{(\{e\},\{f\})}), (p_{(\{e\},\{f\})}, f), (a, p_{(\{a,d\},\{b\})}), (d, p_{(\{a,d\},\{b\})}),$$
$$(p_{(\{a,d\},\{b\})}, b), (b, p_{(\{b\},\{c,f\})}), (p_{(\{b\},\{c,f\})}, c), (p_{(\{b\},\{c,f\})}, f),$$
$$(i_L, a), (f, o_L)\}$$

$$\alpha(L) = (P_L, T_L, F_L)$$

5. Beispiel L_5: Extrahiertes Petrinetz

$$L_5 = [\langle a,b,e,f\rangle^2, \langle a,b,e,c,d,b,f\rangle^3, \langle a,b,c,e,d,b,f\rangle^2, \langle a,b,c,d,e,b,f\rangle^4, \langle a,e,b,c,d,b,f\rangle^3]$$

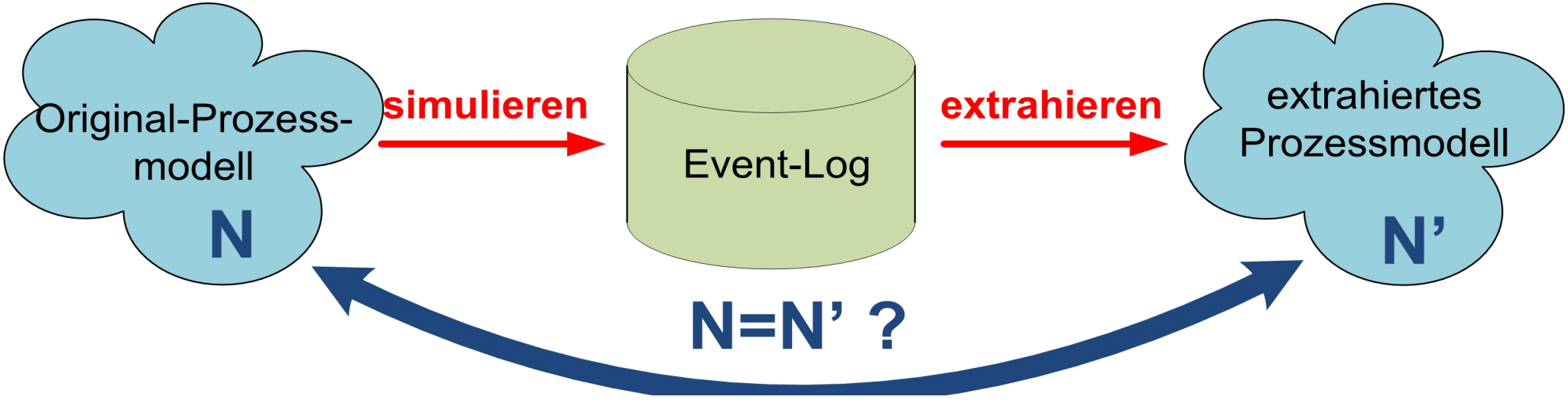


Welches **zusätzliche Verhalten** ergibt sich aus Petrinetz im Vergleich zu ursprünglichen Logdaten ?

Überblick
2.4 Prozessextraktion

- Einführung und Beispiel
- **α-Algorithmus**
  - Idee und Vorbereitungen
  - Formalisierung
  - Beispiel
  - Weitere Beispiele
  - **Einschränkungen**
- Allgemeine Herausforderungen beim Process-Mining

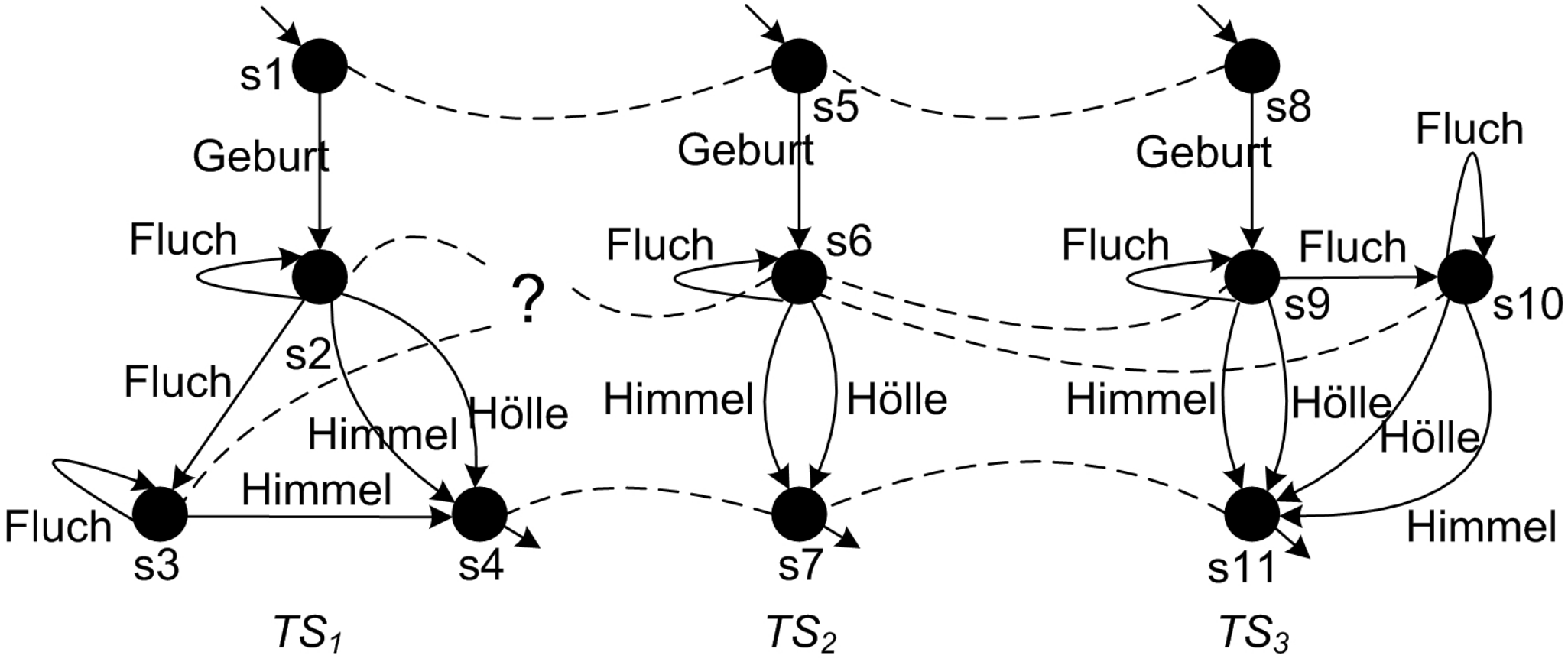
Re-Extraktion von Prozessmodellen



Re-Extraktionsproblem:

Entdeckte Modell N' äquivalent zum Original-Modell N?

Log-Daten vs. Kontrollflussverzweigungen



Allgemeines Problem: **Log-Daten abstrahieren** von **Kontrollflussverzweigungen**.

Beispiel: TS_1,...,TS_3 **spuräquivalent** (trace equivalent). TS_2 und TS_3 auch bisimilar.

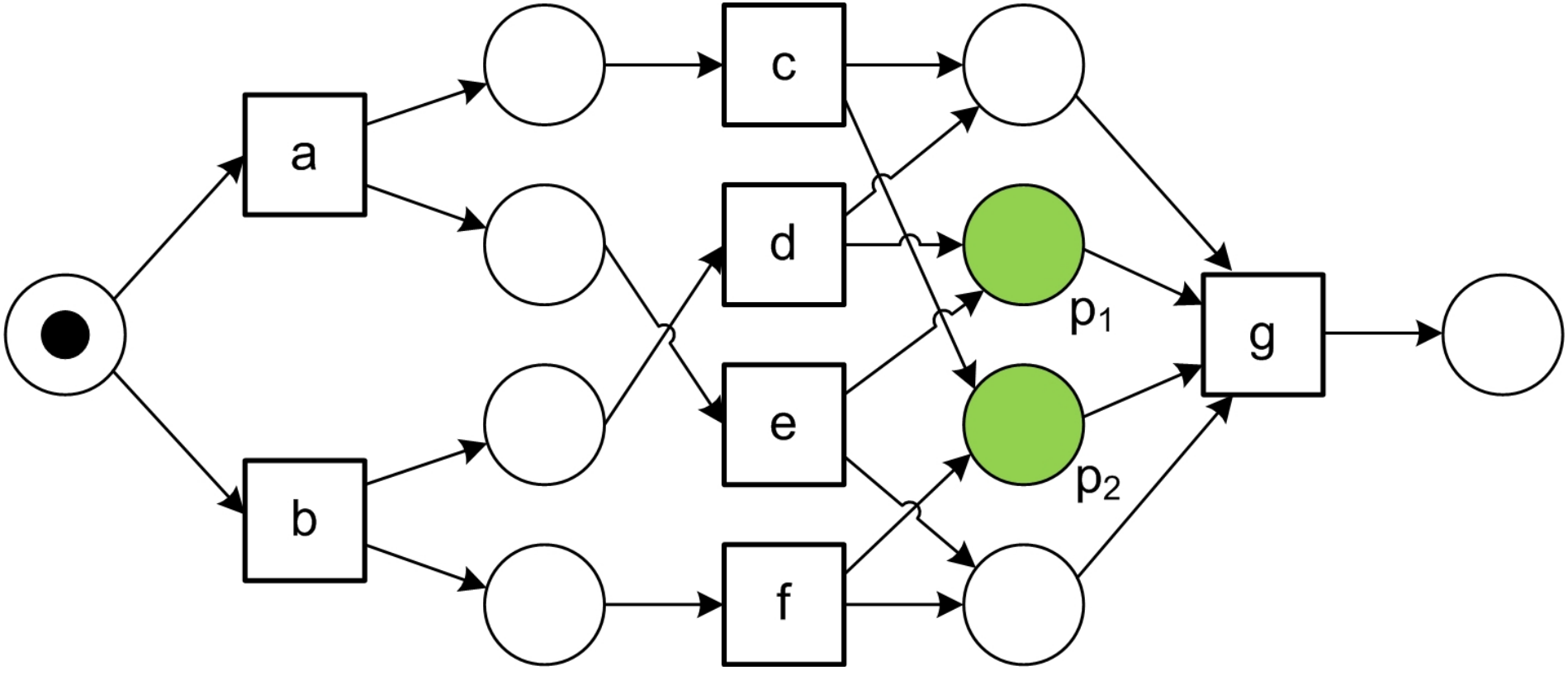
Aber: TS_1 und TS_2 **nicht bisimilar**: Fehlende Transition Hölle:s3->s4.

Eindeutigkeit der Re-Extraktion

- → i.A. Log-Daten von **mehr als einem Petrinetz** erzeugbar.
- Nur „**beobachtbare**“ Netzwerke werden erzeugt.
  - **Splits** und **Joins** nicht immer direkt beobachtbar.
  - Beobachtetes Verhalten korrekt wiedergegeben.

Implizite Stellen

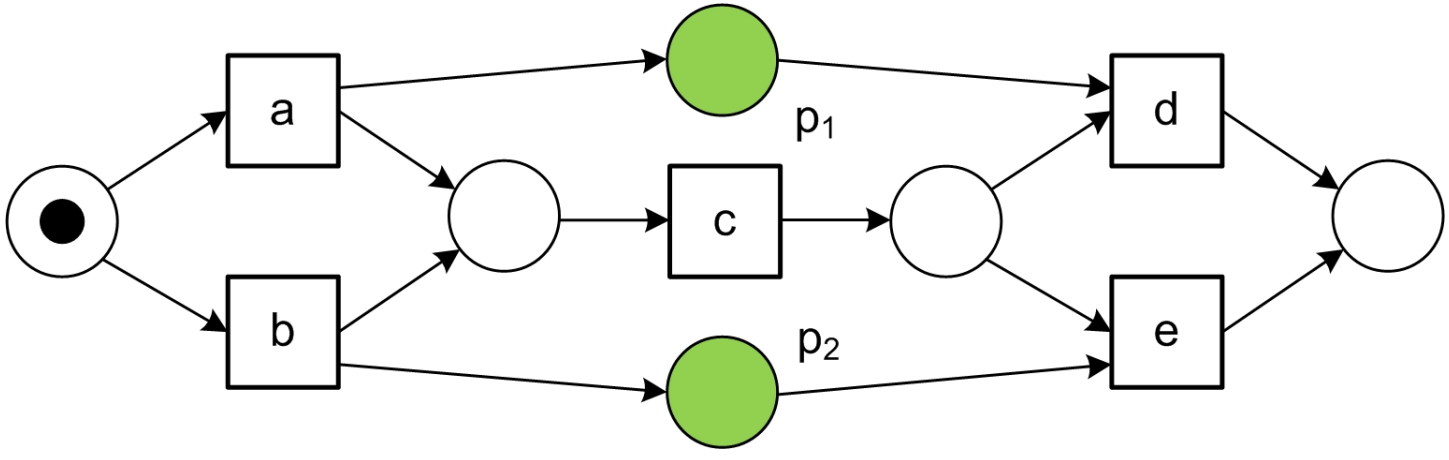
$$L_6 = [\langle a,c,e,g\rangle^2, \langle a,e,c,g\rangle^3, \langle b,d,f,g\rangle^2, \langle b,f,d,g\rangle^4]$$



Grüne Stellen **implizit**: Werden im Petrinetz nicht benötigt, aufgrund Symmetrie der Logdaten erzeugt.

Nicht-lokale Abhängigkeiten

$$L_9 = [\langle a, c, d \rangle^{45}, \langle b, c, e \rangle^{42}]$$



Grüne Stellen werden **nicht entdeckt** !
Extrahiertes Petrinetz also dasselbe wie für:

$$L_4 = [\langle a, c, d \rangle^{45}, \langle b, c, d \rangle^{42}, \langle a, c, e \rangle^{38}, \langle b, c, e \rangle^{22}]$$

Eindeutigkeit der Transitionsbezeichner

- **Eindeutigkeit der Bezeichner** nach o.g. Annahme:
  Jeder Transitionsname taucht nur einmal im Petrinetz auf.

→ **Nicht jede Folge** von Transitionen aus wohlgeformten Petrinetz **erzeugbar** !

NB: Ohne o.g. Annahme Petrinetz immer als XOR-Verknüpfung über Transitionen, die je eine Sequenz erzeugen, darstellbar.

Mehrfach auftretende Aktivitäten:

- Schwierig; α-Algorithmus konstruiert nur eine Transition pro Aktivität.

Eindeutigkeit der Transitionen: Beispiel

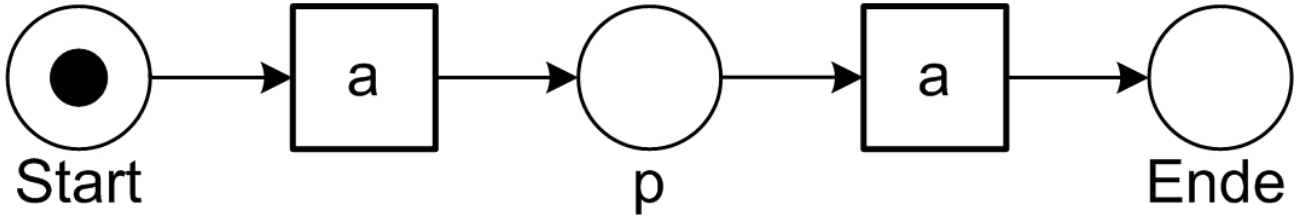
$$L_{10} = [\langle a, a \rangle^{55}]$$

Wie durch Petrinetz darstellen, das jeden Transitionsbezeichner jeweils **nur einmal** enthält ?

Eindeutigkeit der Transitionen: Beispiel

$$L_{10} = [\langle a, a \rangle^{55}]$$

Naheliegend (nicht mit eindeutigen Transitionsbezeichnungen):



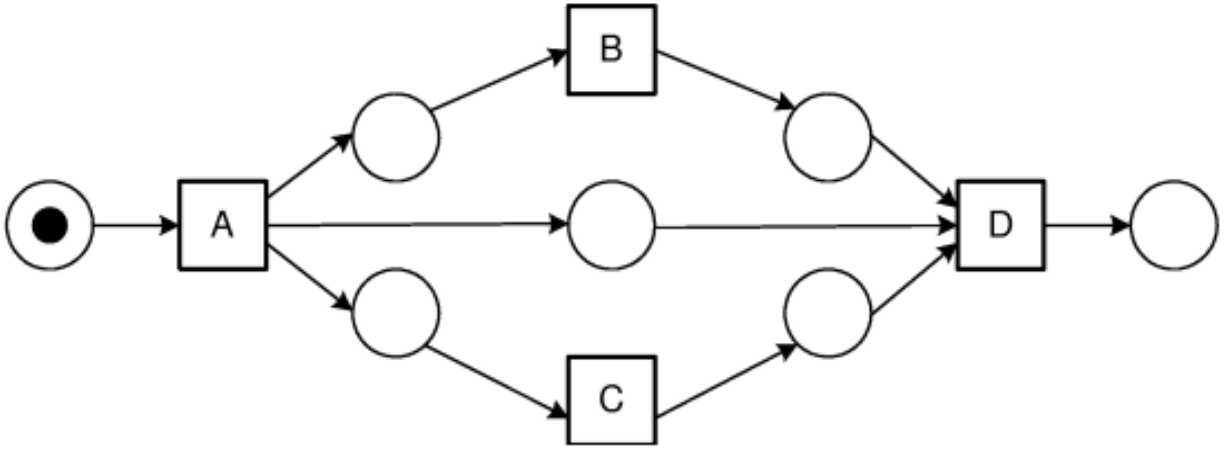
Gibt **kein WF-net** mit eindeutigen Transitionsbezeichnungen, welches (exakt) dieses Verhalten darstellt.

Beobachtbarkeit der Transitionen

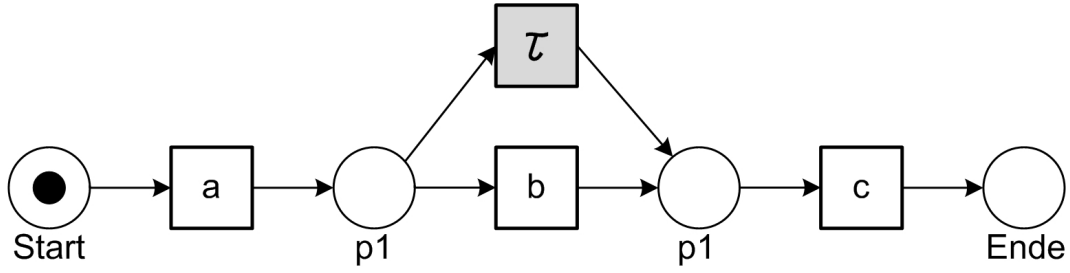
Abstraktion („unsichtbar machen“) von Transitionen:

Wegabstrahieren der Transition E in **Log-Daten** im obigen Beispiel führt zu **nicht** durch **Petrinetz** darstellbaren Log-Daten.

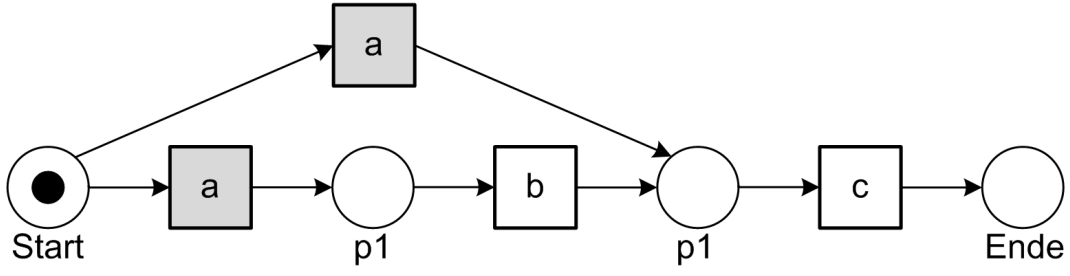
Z.B. erzeugt folgendes Petrinetz nur [<ABCD>,<ACBD>] (nicht <AD>, wie durch Abstrahieren von E aus <AED> erhalten):



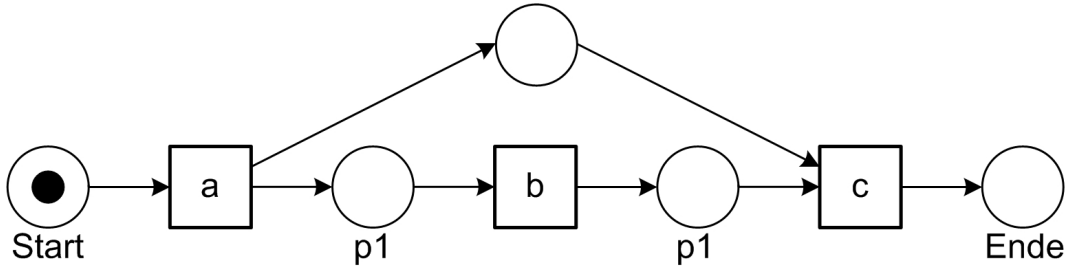
Eindeutigkeit / Beobachtbarkeit der Transitionen: Beispiel



(a)



(b)



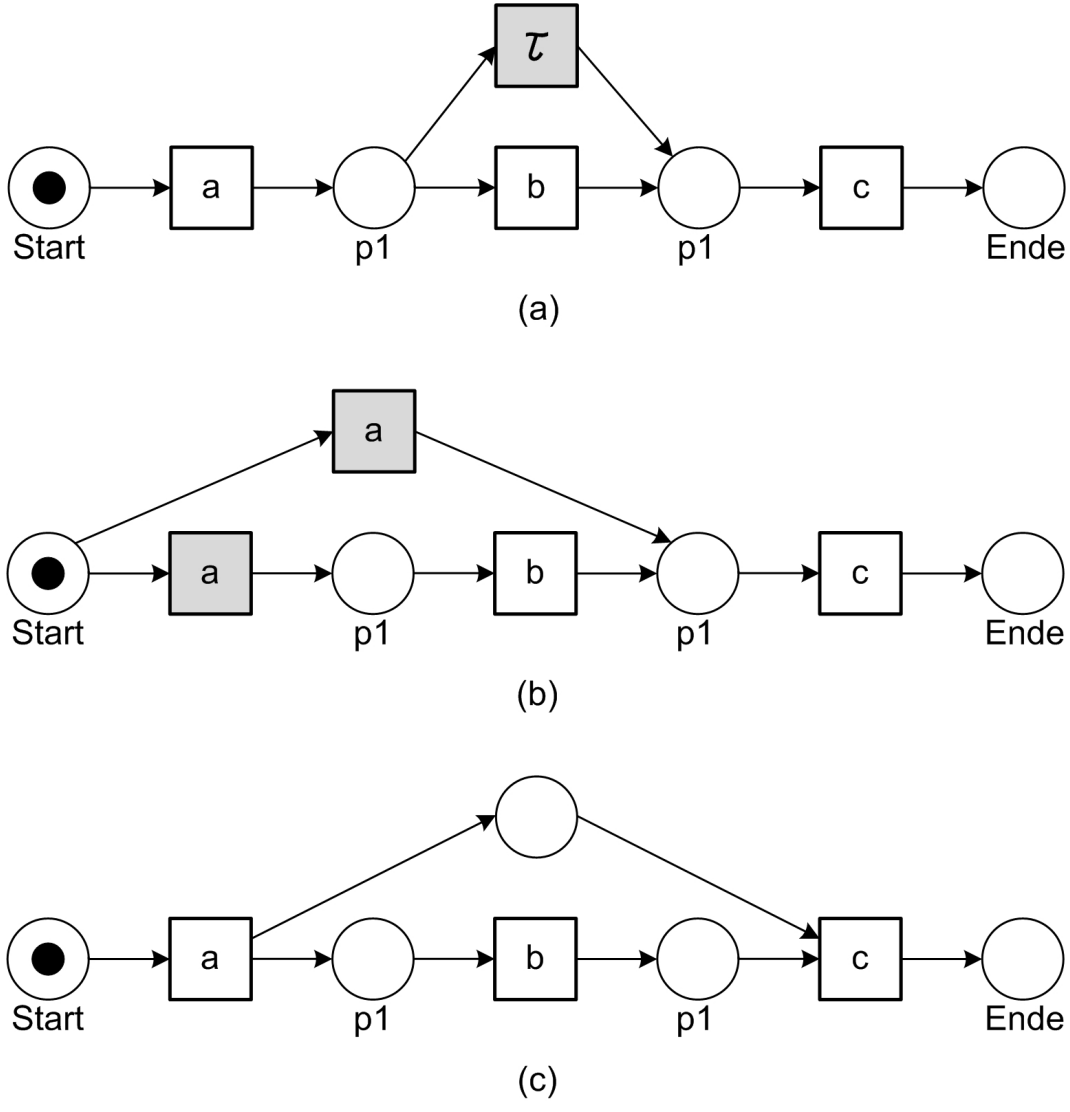
(c)

$$L_{11} = [\langle a, b, c \rangle^{20}, \langle a, c \rangle^{30}]$$

Welches Petrinetz extrahiert **α-Algorithmus** aus $\mathbf{L_{11}}$?

Welche Logdaten erzeugt das Petrinetz bei Ausführung ?

Eindeutigkeit / Beobachtbarkeit der Transitionen: Beispiel



$$L_{11} = [\langle a,b,c\rangle^{20}, \langle a,c\rangle^{30}]$$

Gibt **kein WF-net** mit **eindeutigen** und **sichtbaren** Bezeichnungen, welches (genau) dieses Verhalten darstellt.

(a) **Unsichtbares** Ereignis τ

(b) Ereignis a **mehrfach**

(c) Extrahiert mit **α-Algorithmus** aus $\mathbf{L_{11}}$.

Erzeugt nur [**<a,b,c>**].

Einschränkungen: Weiteres Beispiel

$$L_7 = [\langle a, c\rangle^2, \langle a, b, c\rangle^3, \langle a, b, b, c\rangle^2, \langle a, b, b, b, b, c\rangle^1]$$

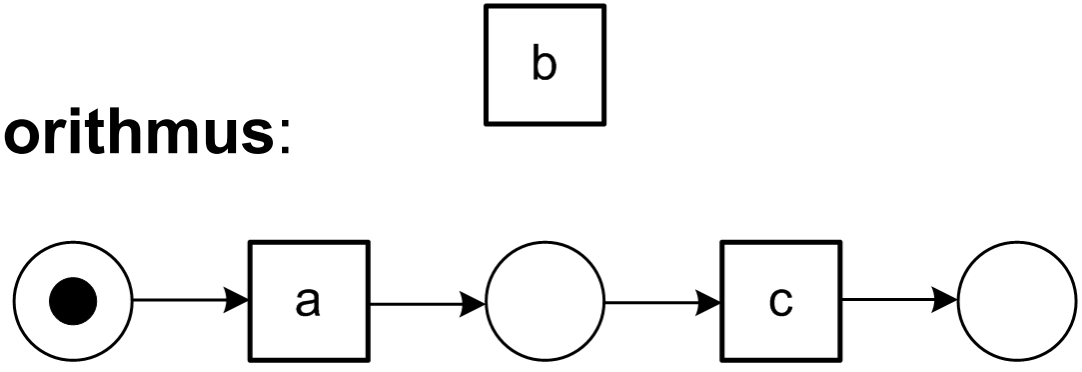
Welches Petrinetz extrahiert **α-Algorithmus** aus o.g. Logdaten ?

Welches **Petrinetz** generiert o.g. Logdaten ?

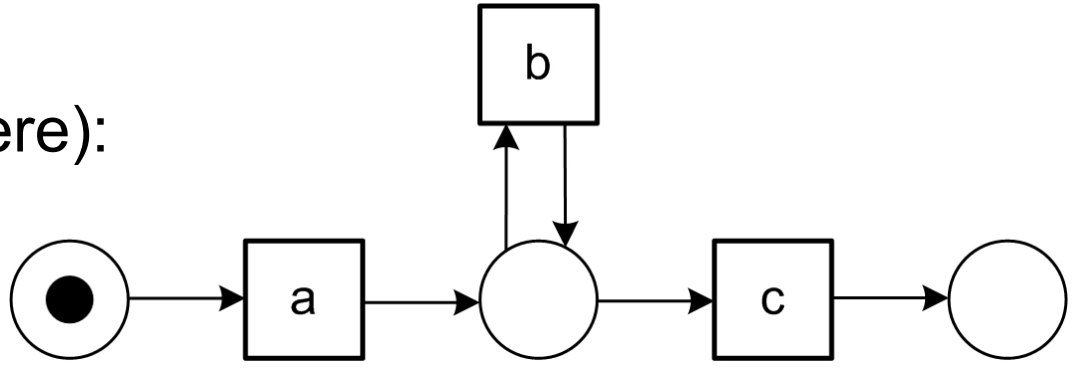
Schleifen der Länge 1: Beispiel

$$L_7 = [\langle a,c\rangle^2, \langle a,b,c\rangle^3, \langle a,b,b,c\rangle^2, \langle a,b,b,b,b,c\rangle^1]$$

Petrinetz **extrahiert** von **α-Algorithmus**:



Petrinetz, das tatsächlich o.g. **Logdaten generiert** (und weitere):



Iterationen

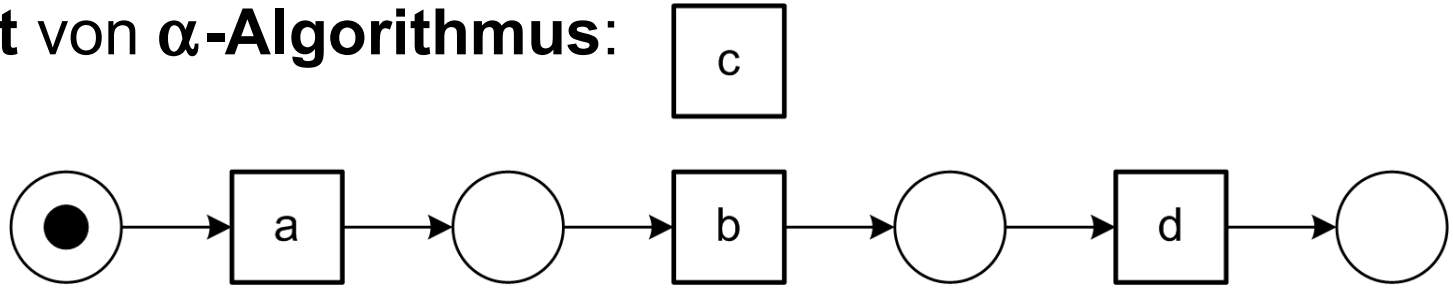
Iterationen:

- Führen zu mehrfach auftretenden Aktivitäten.
- Müssen ggf. extra erkannt werden.
  - Mitunter sehr schwierig !
- Keine Beschränkung des Algorithmus; **Schleifen** durch **endliche Logdatenmengen** nur **unvollständig** charakterisierbar.

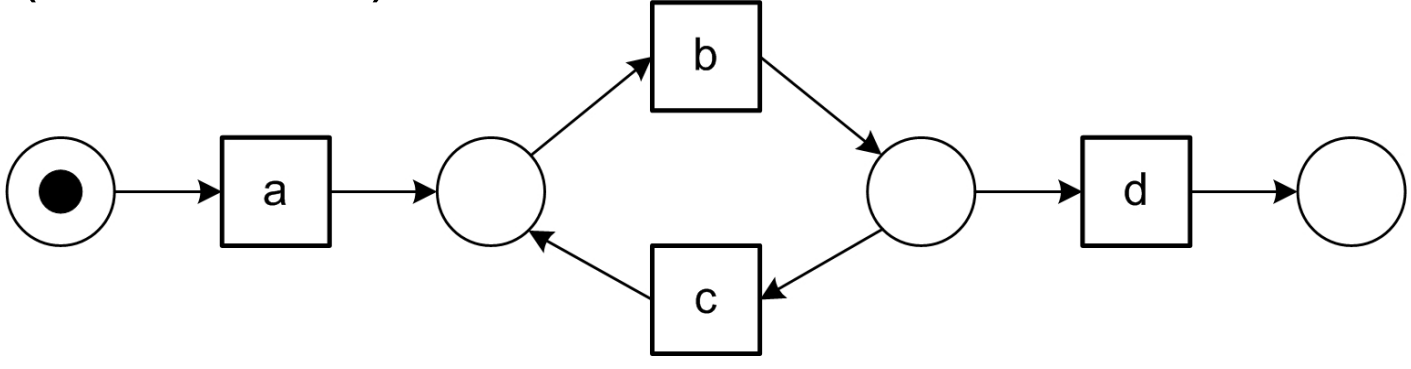
Schleifen der Länge 2: Beispiel

$$L_8 = [\langle a,b,d\rangle^3, \langle a,b,c,b,d\rangle^2, \langle a,b,c,b,c,b,d\rangle]$$

Petrinetz **extrahiert** von **α-Algorithmus**:



Petrinetz, das tatsächlich o.g.
Logdaten generiert (und weitere):



Überblick
2.4 Prozessextraktion

- Einführung und Beispiel
- α-Algorithmus
  - Idee und Vorbereitungen
  - Formalisierung
  - Beispiel
  - Weitere Beispiele
  - Einschränkungen
- **Allgemeine Herausforderungen beim Process-Mining**

Herausforderung: Rauschen und Unvollständigkeit

Annahme passendes Prozessmodell zu finden: Event-Log enthält **repräsentative Verhaltensmenge**.

Zwei verwandte Phänomene:

- Rauschen: Event-Log enthält seltenes und unregelmäßiges Verhalten. → **Nicht typisches Verhalten** des Prozesses.
  - Lösung z.B. durch Betrachtung der **Häufigkeit**.
- Unvollständigkeit: Event-Log enthält **zu wenig Events**, um alle Kontrollflüsse zu erfassen.
  - Siehe auch Teil 2.2 (cross-validation, precision, recall, etc.).

Mehr zu Unvollständigkeit

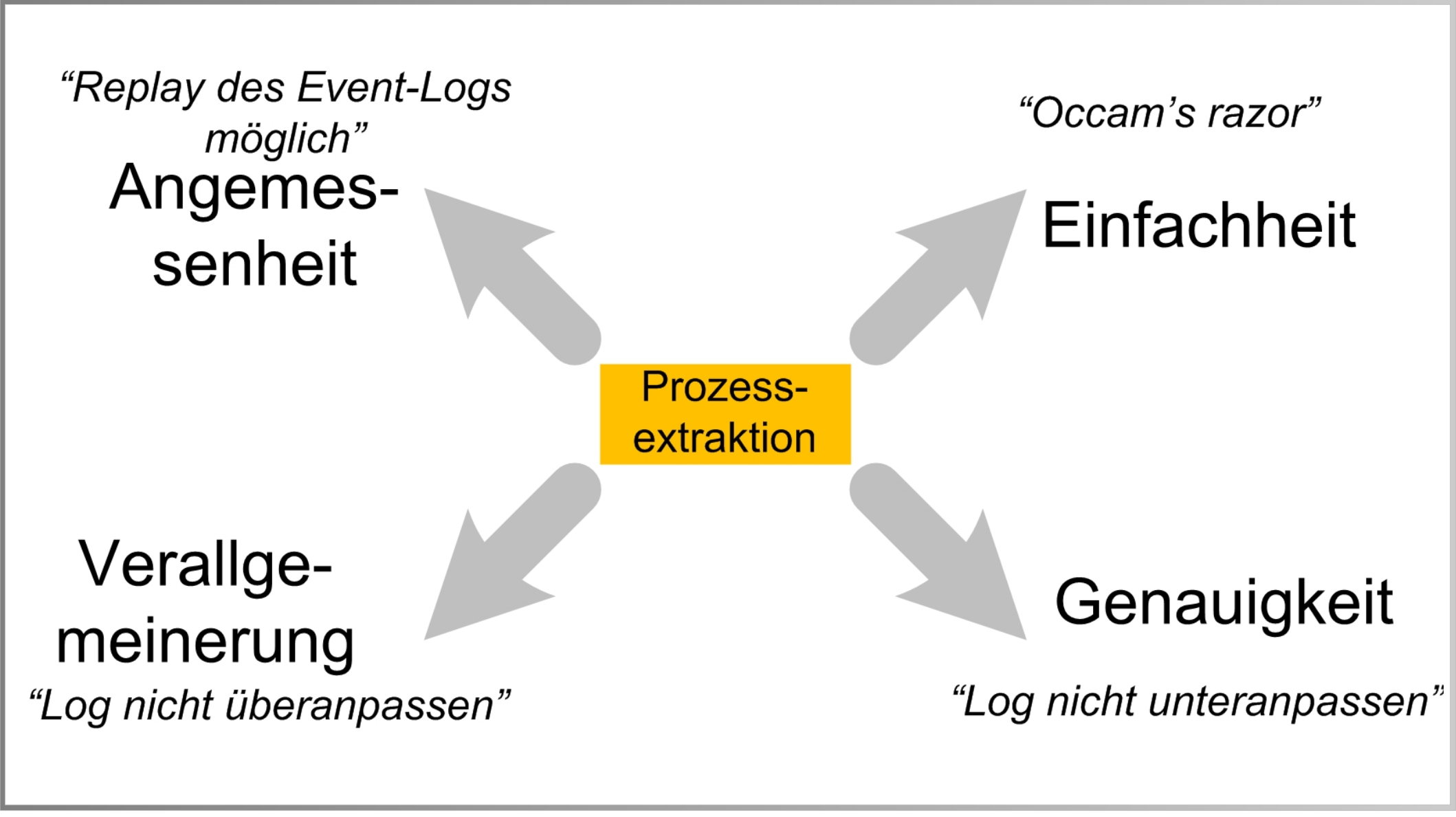
Bsp. zu Relevanz von **Vollständigkeit**: Betrachte Prozess aus 10 Aktivitäten.

- Annahme: Parallel ausführbar.

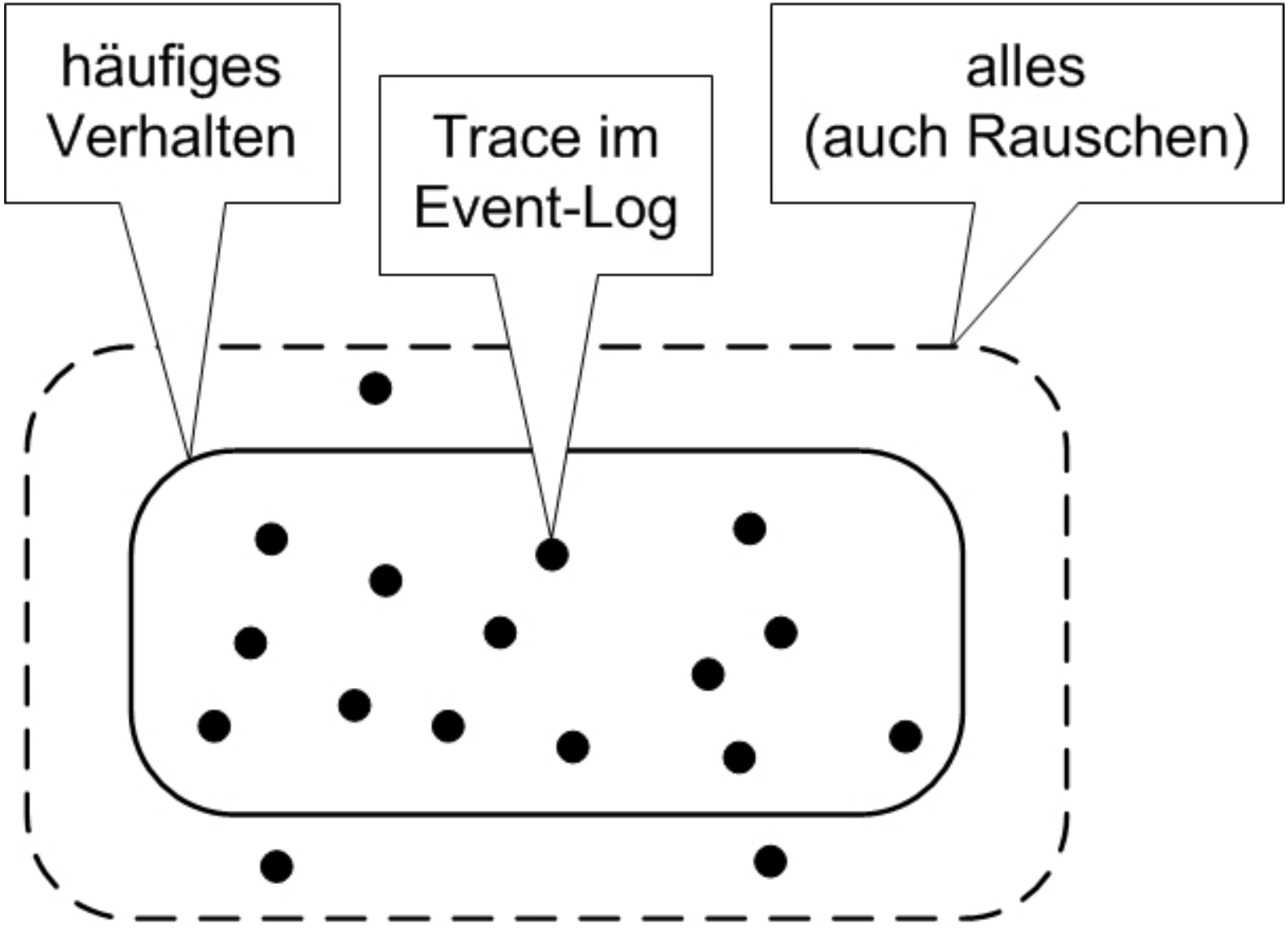
Gegeben: **Log mit Informationen** über 10.000 Fälle.

- Anzahl mögliche Kombinationen (10 nebenläufige Aktivitäten): 10! = 3.628.800.
- Unmöglich jede Kombination im Log enthalten:
  - Im Log weniger Fälle (10.000) als mögliche Traces (3.628.800).
- Selbst bei 3.628.800 Fällen im Log:
  - **Alle Variationen vorhanden → Unwahrscheinlich**
  - Analogie: 365 Menschen, jeder hat am anderen Tag Geburtstag.
    → Unwahrscheinlich
    W'keit bei $365!/365^{365} \approx 1.454955 \times 10^{-157}$.
  - Anzahl von Atomen im Universum ca. 10^{79}.

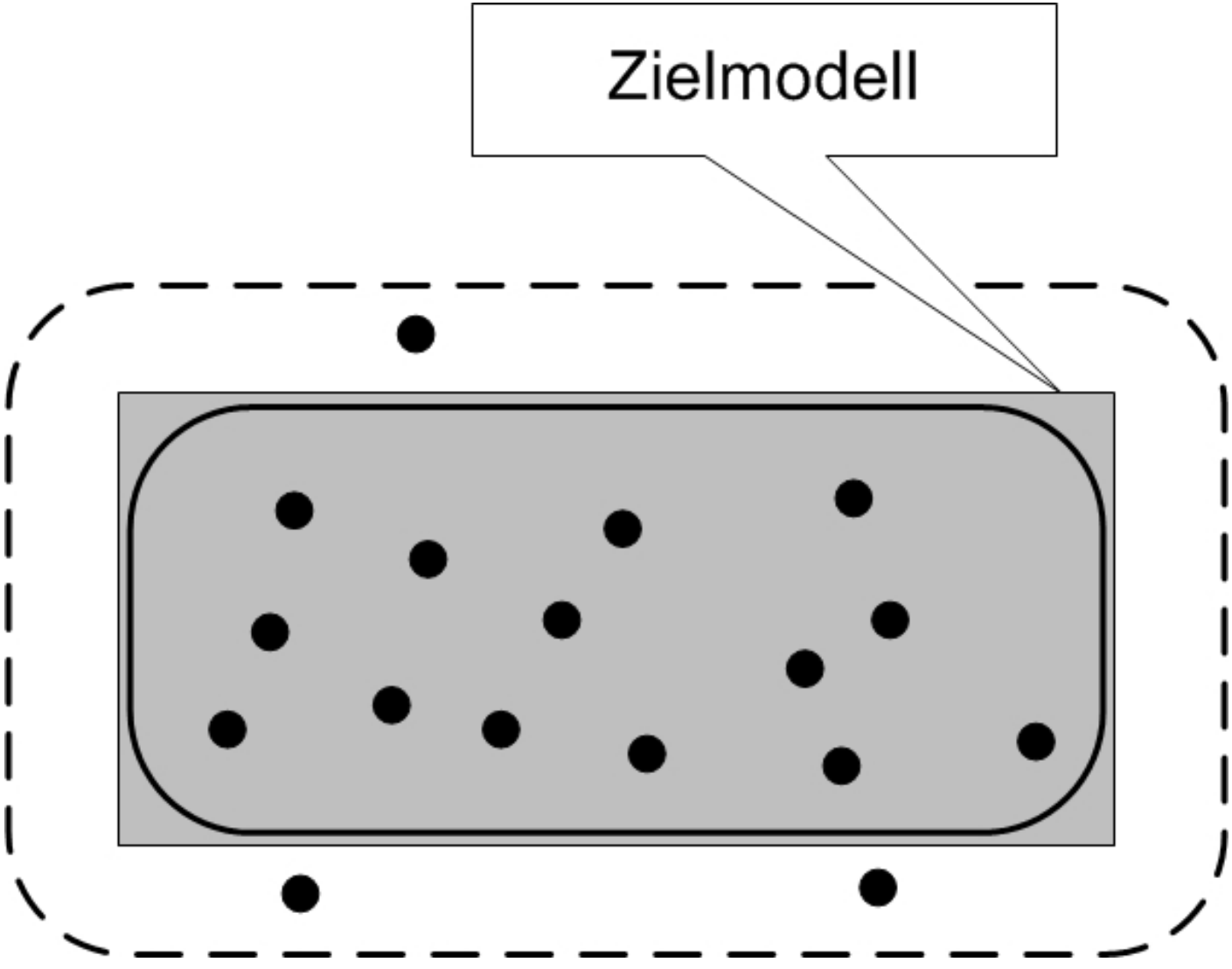
Herausforderung: Vier divergierende Qualitätskriterien



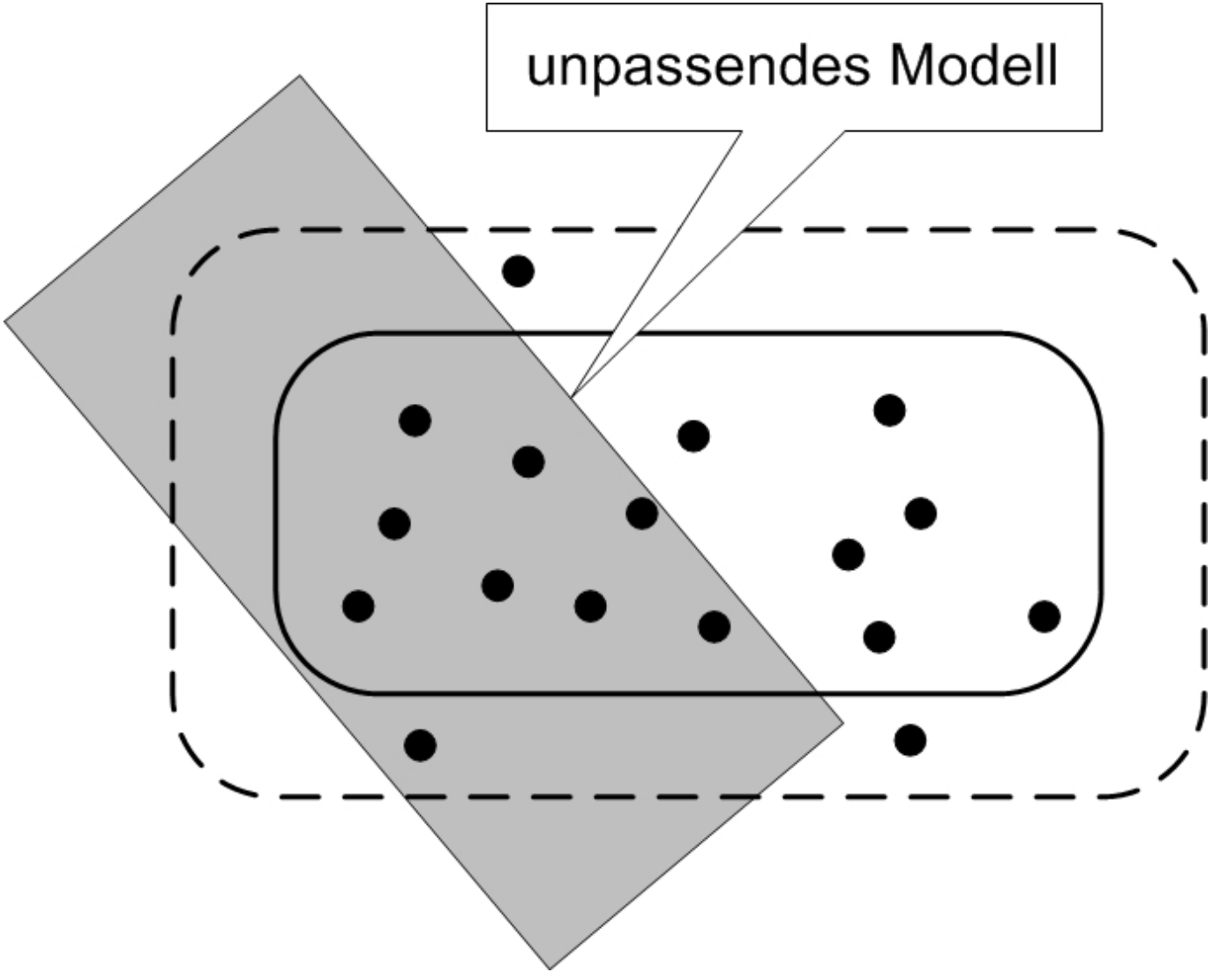
Endlose Überwachung eines stabilen Prozesses

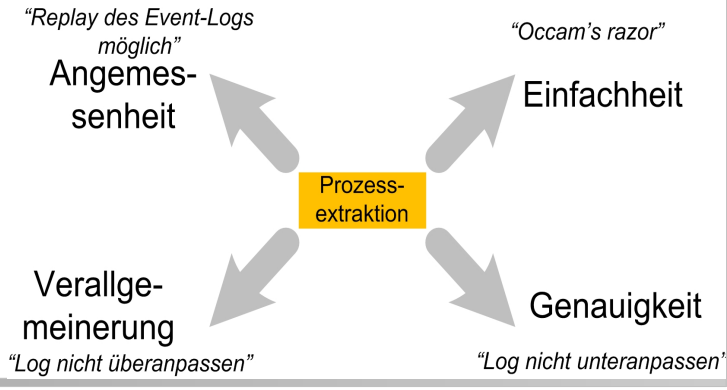


Zielmodell



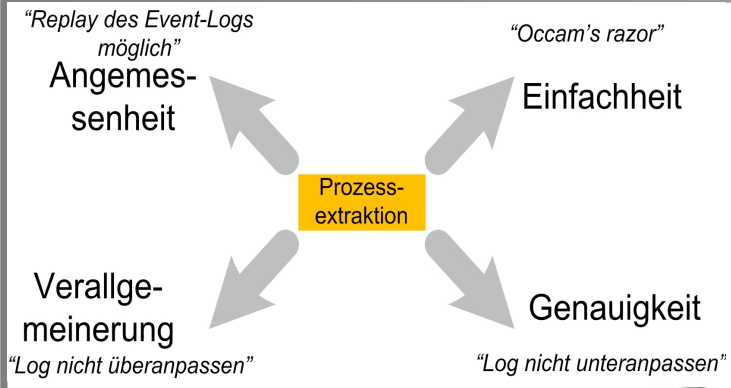
Unpassendes Modell

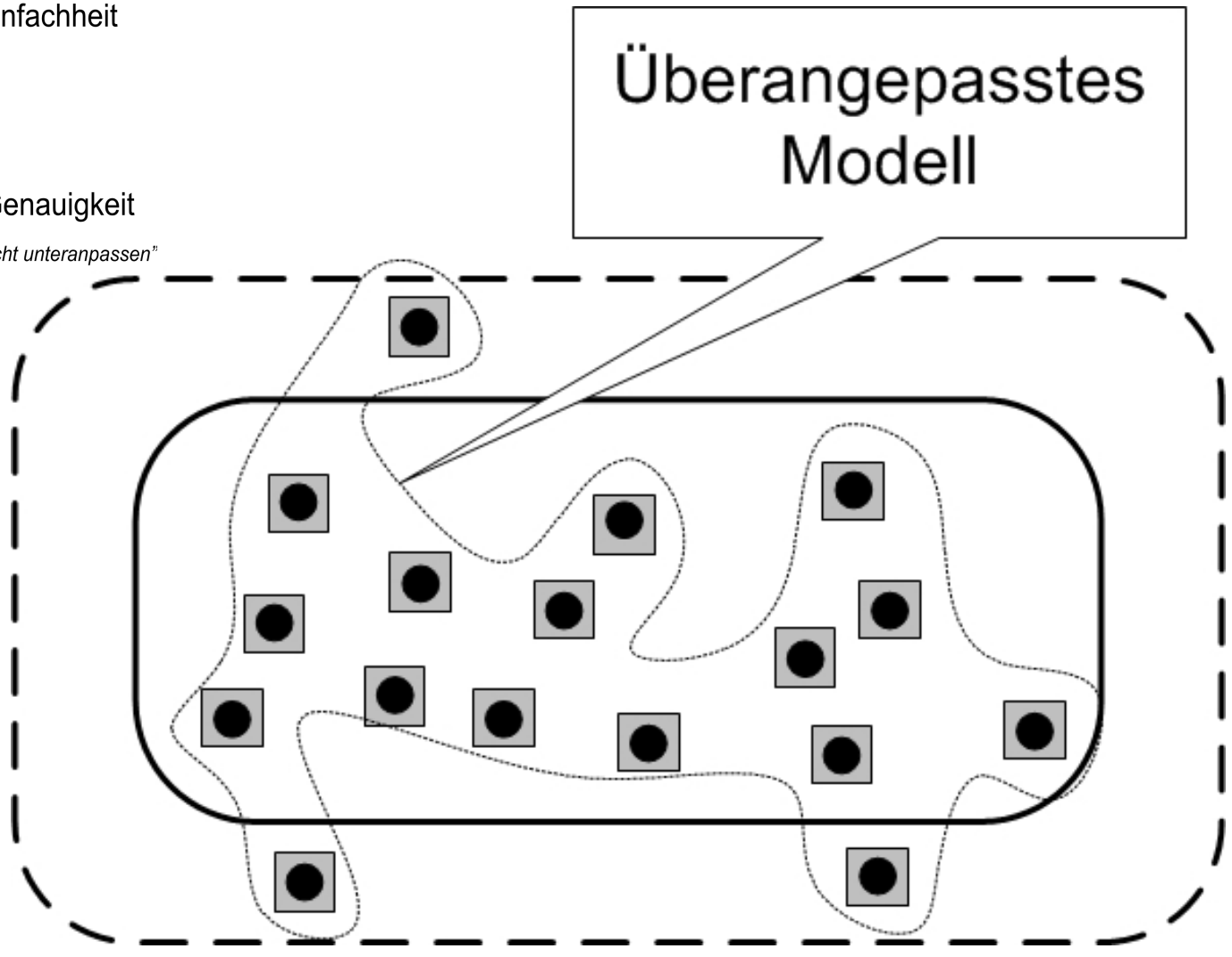




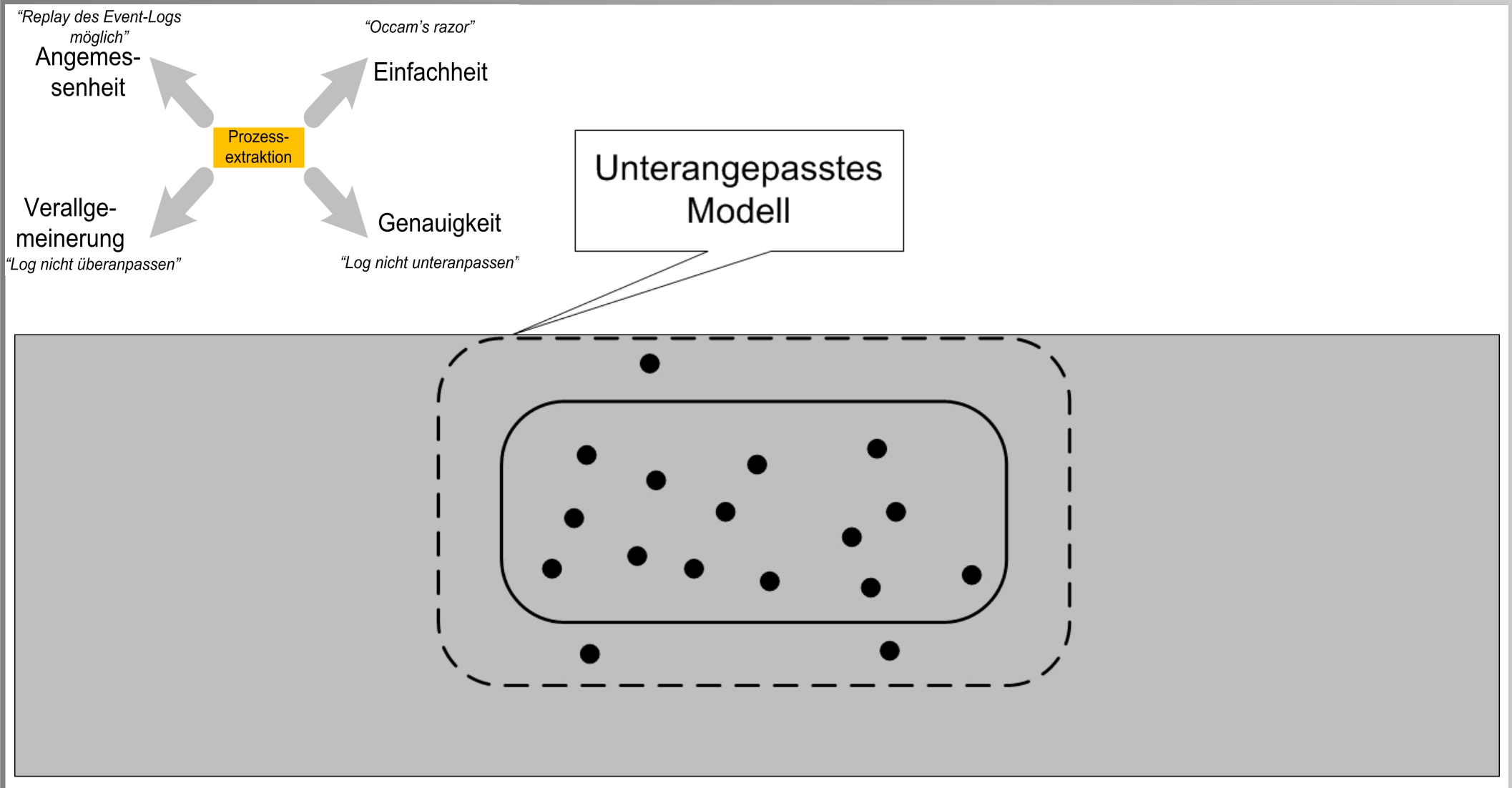
Besondere Herausforderung: Balance zwischen Unteranpassung und Überanpassung

Überangepasstes Modell

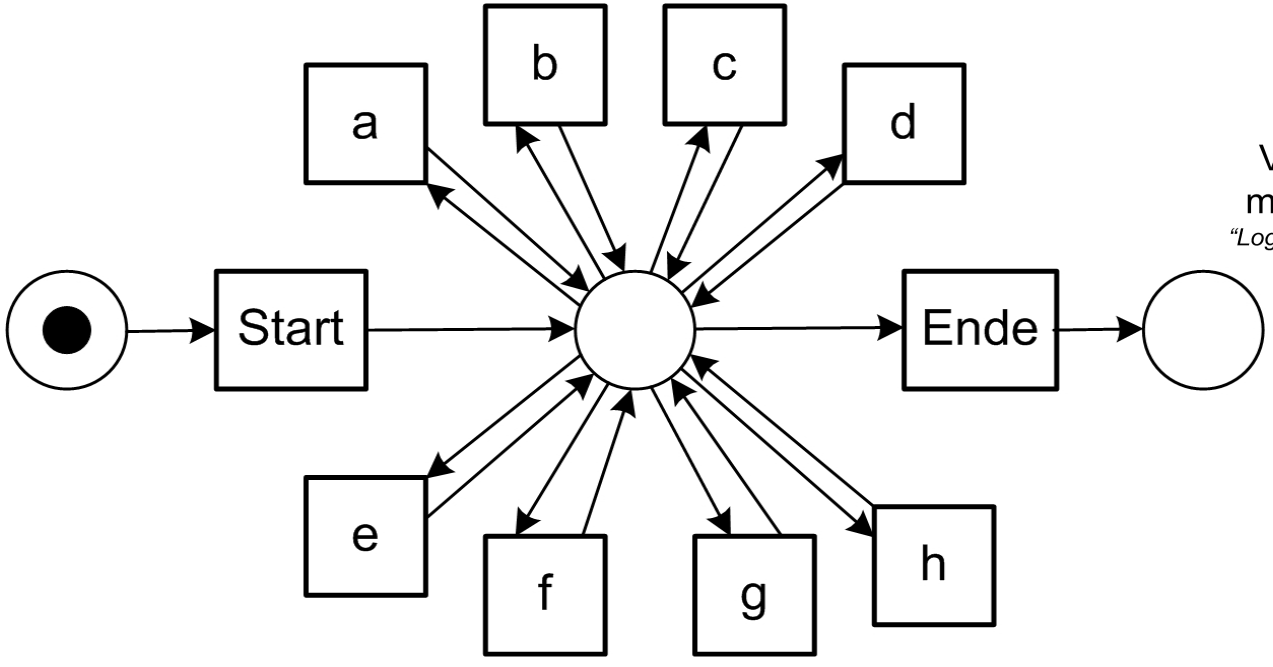


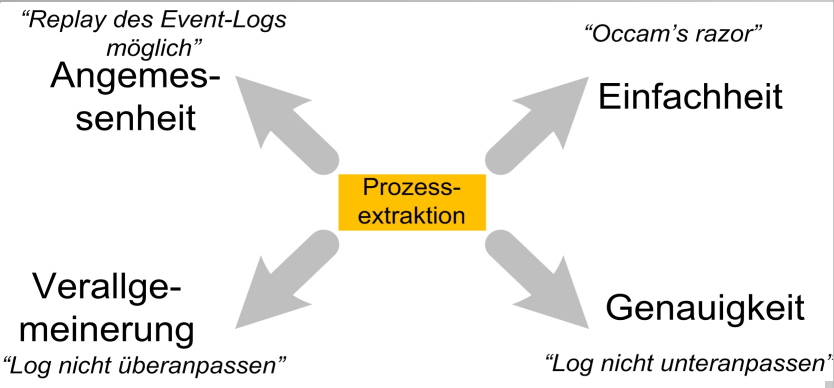


Unterangepasstes Modell



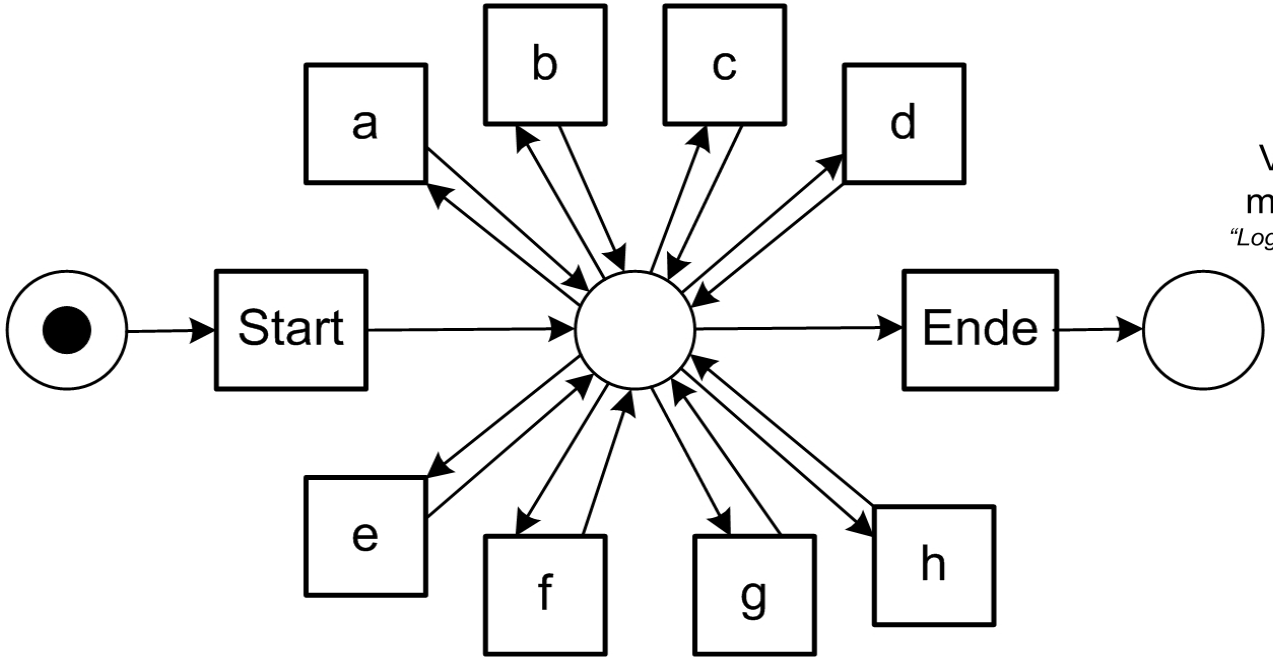
„Blumen-Modell“

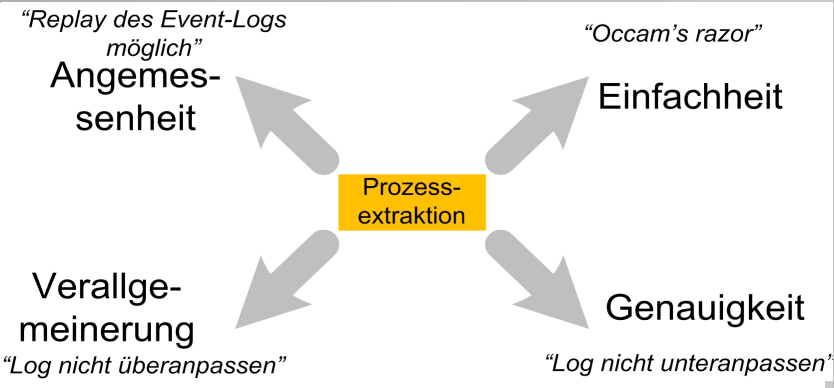




Welche **Qualitätskriterien** erfüllt dieses Modell in besonderer Weise ?

„Blumen-Modell“



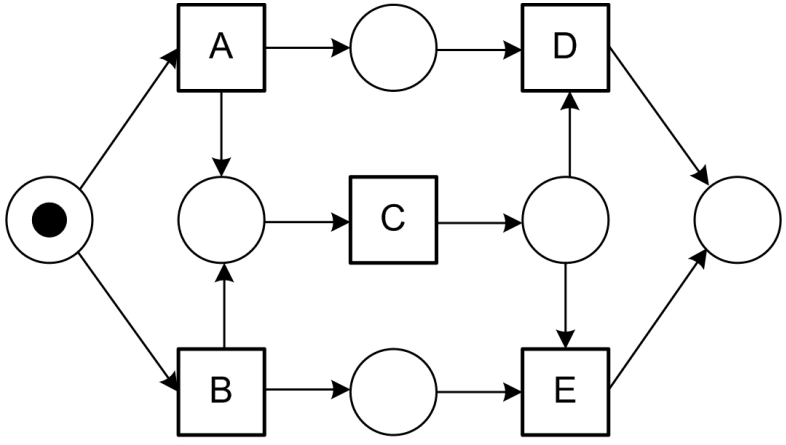


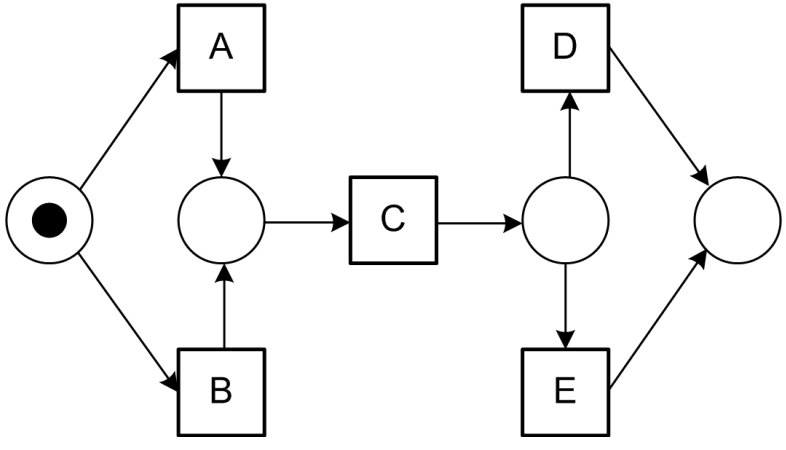
Welche **Qualitätskriterien** erfüllt dieses Modell in besonderer Weise ?

Paradebeispiel für **unterangepasstes Modell:**
jedes Verhalten auf Basis der Ereignisse a, …, h zulässig.

Welches ist das beste Modell?

ACD	99
ACE	0
BCE	85
BCD	0

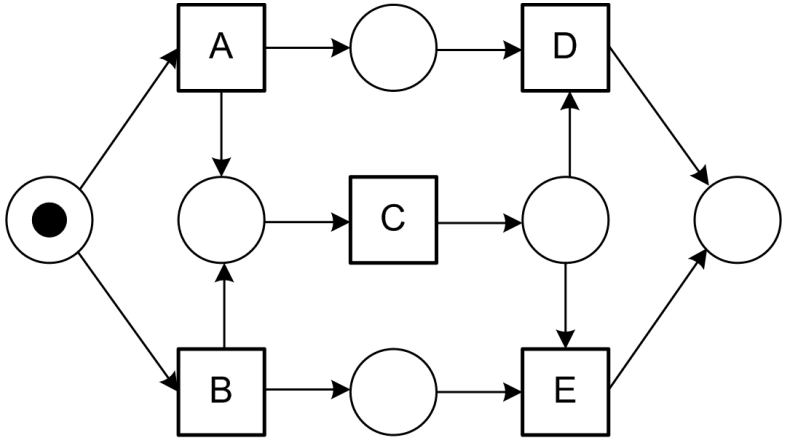


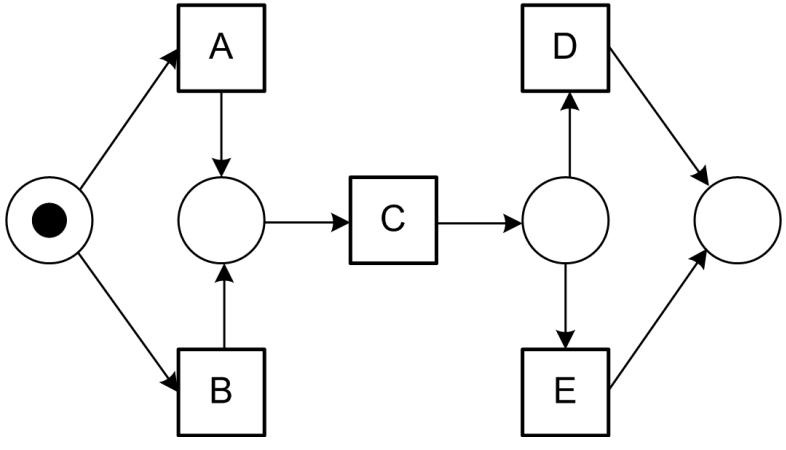


?

Welches ist das beste Modell?

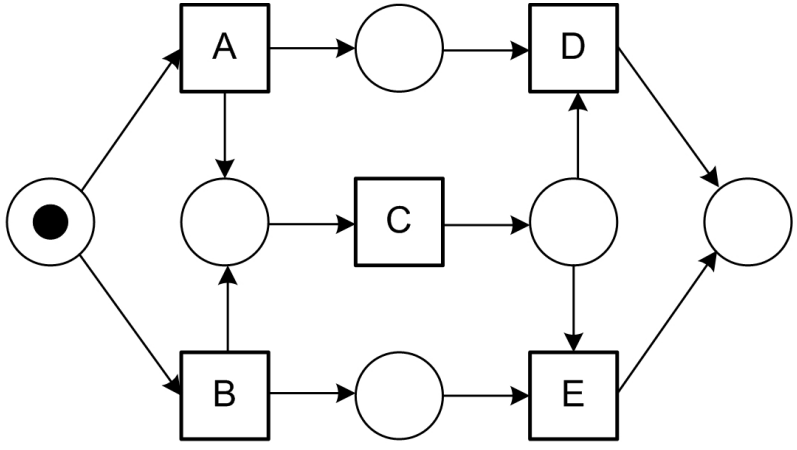
ACD	99
ACE	0
BCE	85
BCD	0

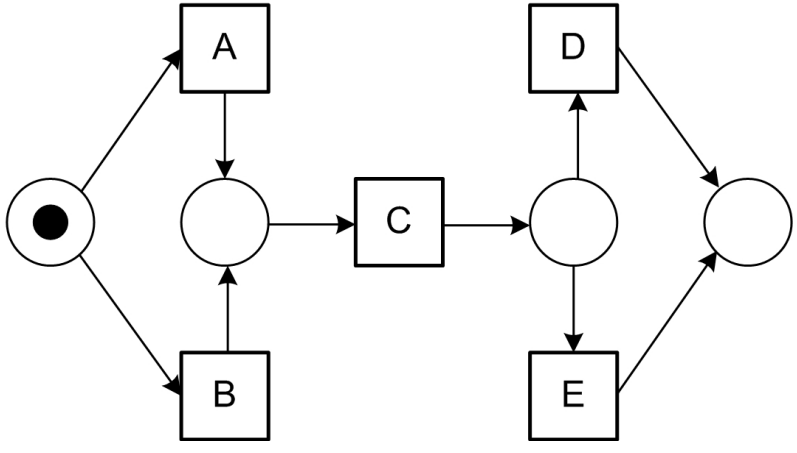




Welches ist das beste Modell?

ACD	99
ACE	88
BCE	85
BCD	78

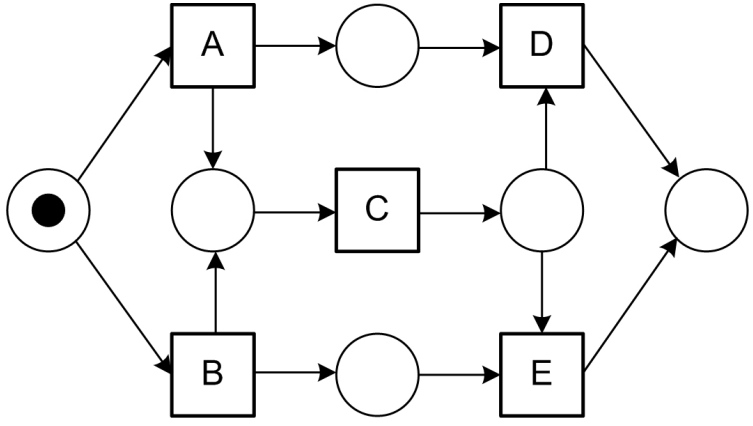


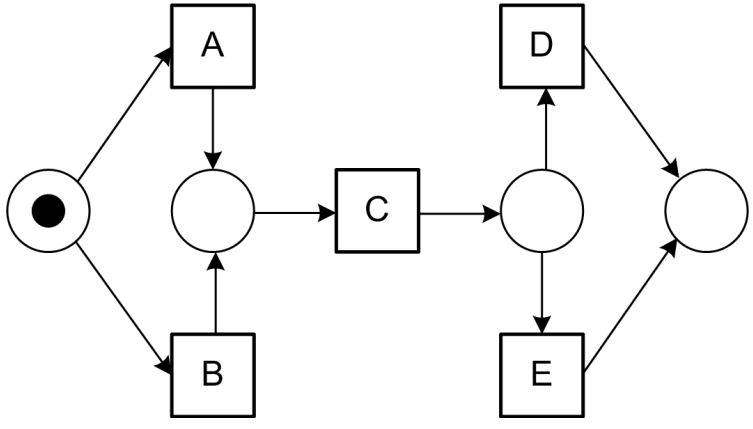


?

Welches ist das beste Modell?

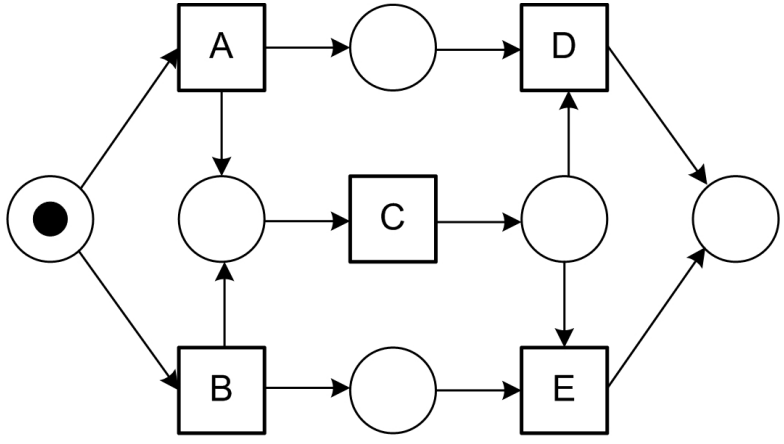
ACD **99**
ACE **88**
BCE **85**
BCD **78**

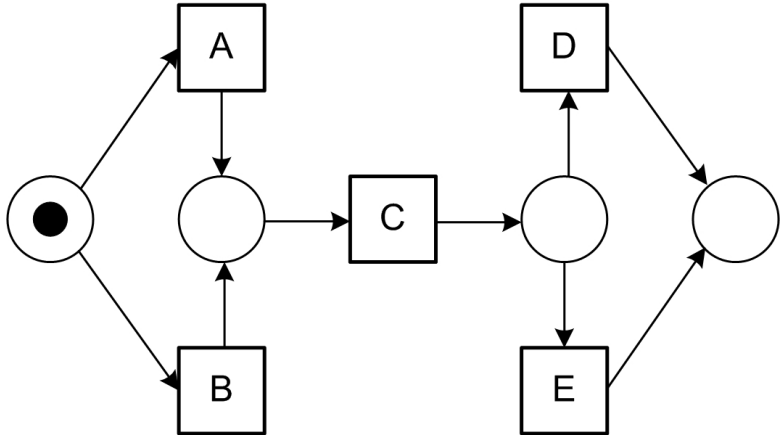




Welches ist das beste Modell?

ACD	99
ACE	2
BCE	85
BCD	3

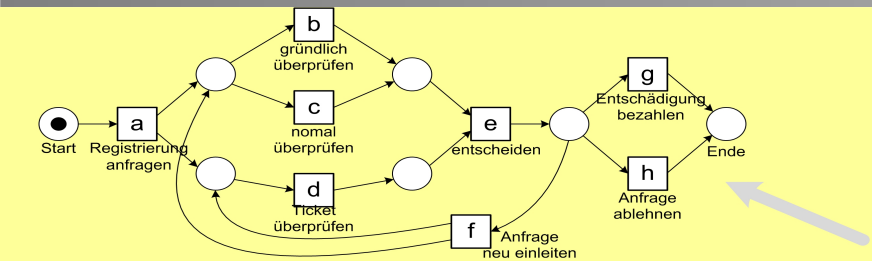




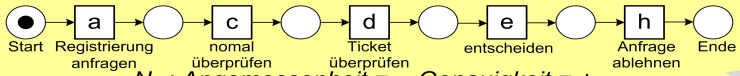
?

Beispiel: Ein Log, vier Modelle

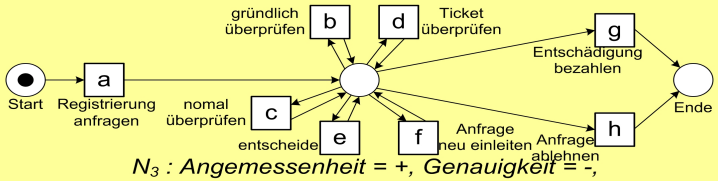
LEHRSTUHL 14
SOFTWARE ENGINEERING



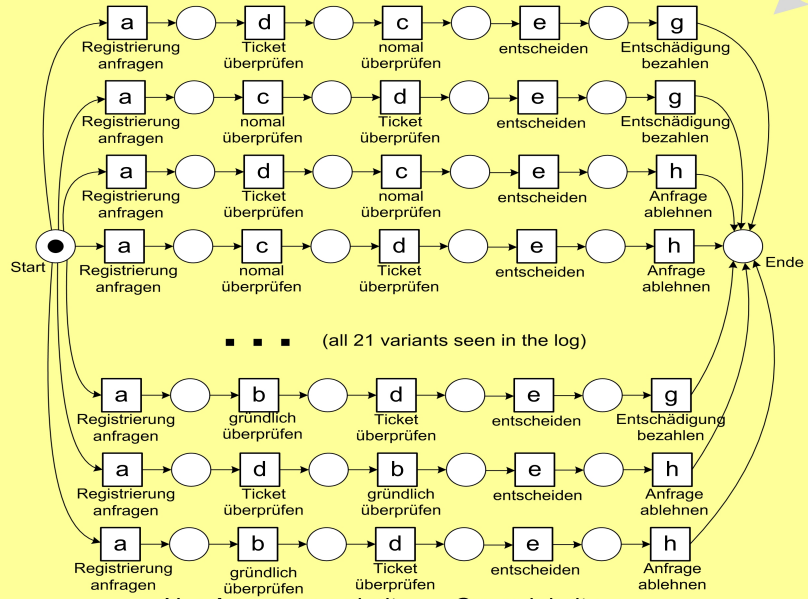
N_1 : Angemessenheit = +, Genauigkeit = +,
Verallgemeinerung = +, Einfachheit = +



N_2 : Angemessenheit = -, Genauigkeit = +,
Verallgemeinerung = -, Einfachheit = +

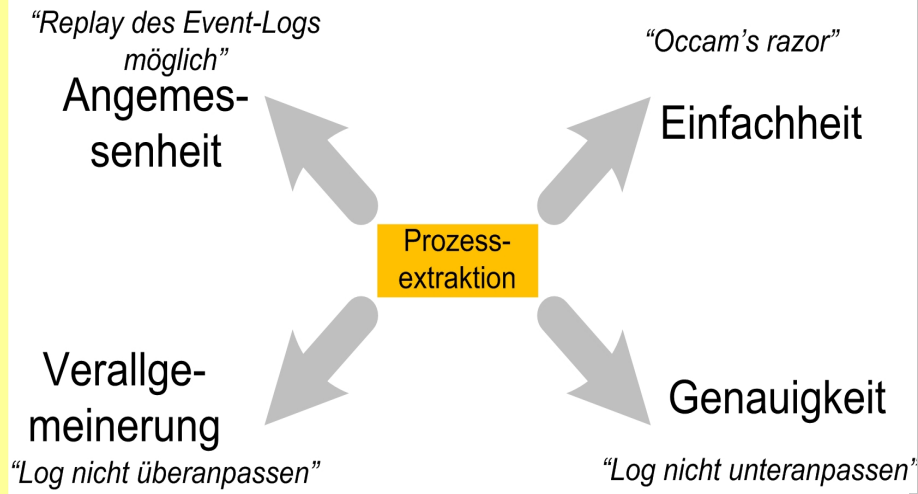


N_3 : Angemessenheit = +, Genauigkeit = -,
Verallgemeinerung = +, Einfachheit = +



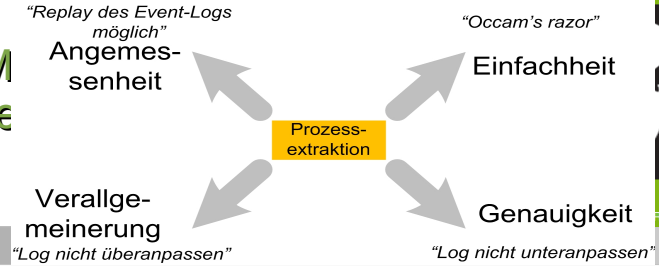
N_4 : Angemessenheit = +, Genauigkeit = +,
Verallgemeinerung = -, Einfachheit = -

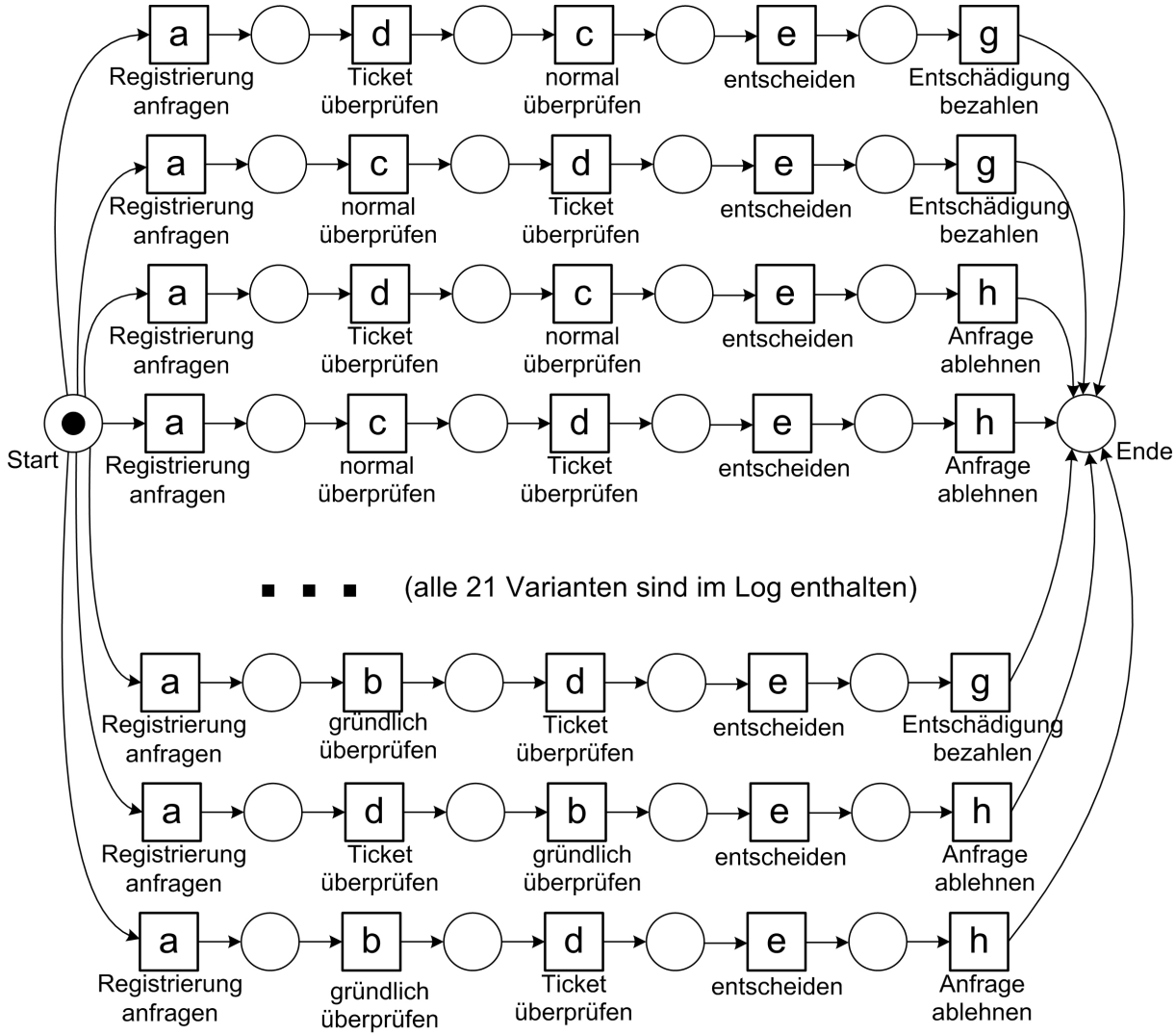
#	Trace
455	acdeh
191	abdeg
177	adceh
144	abdeh
111	acdeg
82	adceg
56	adbeh
47	acdefdbeh
38	adbeg
33	acdefbdeh
14	acdefbdeg
11	acdefdbeg
9	adcefcdeh
8	adcefdbeh
5	adcefbdeg
3	acdefbdefdbeg
2	adcefdbeg
2	adcefbdefbdeg
1	adcefdbefbdeh
1	adbefbdefdbeg
1	adcefdbefcdefdbeg
1391	



#	trace
455	acdeh
191	abdeg
177	adceh
144	abdeh
111	acdeg
82	adceg
56	adbeh
47	acdefdbeh
38	adbeg
33	acdefbdeh
14	acdefbdeg
11	acdefdbeg
9	adcefcdeh
8	adcefdbeh
5	adcefbdeg
3	acdefbdefdbeg
2	adcefdbeg
2	adcefbdefbdeg
1	adcefdbefbdeh
1	adbefbdefdbeg
1	adcefdbefcdefdbeg
1391	

Modell 1

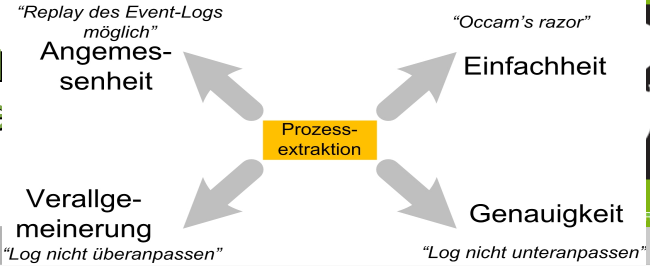


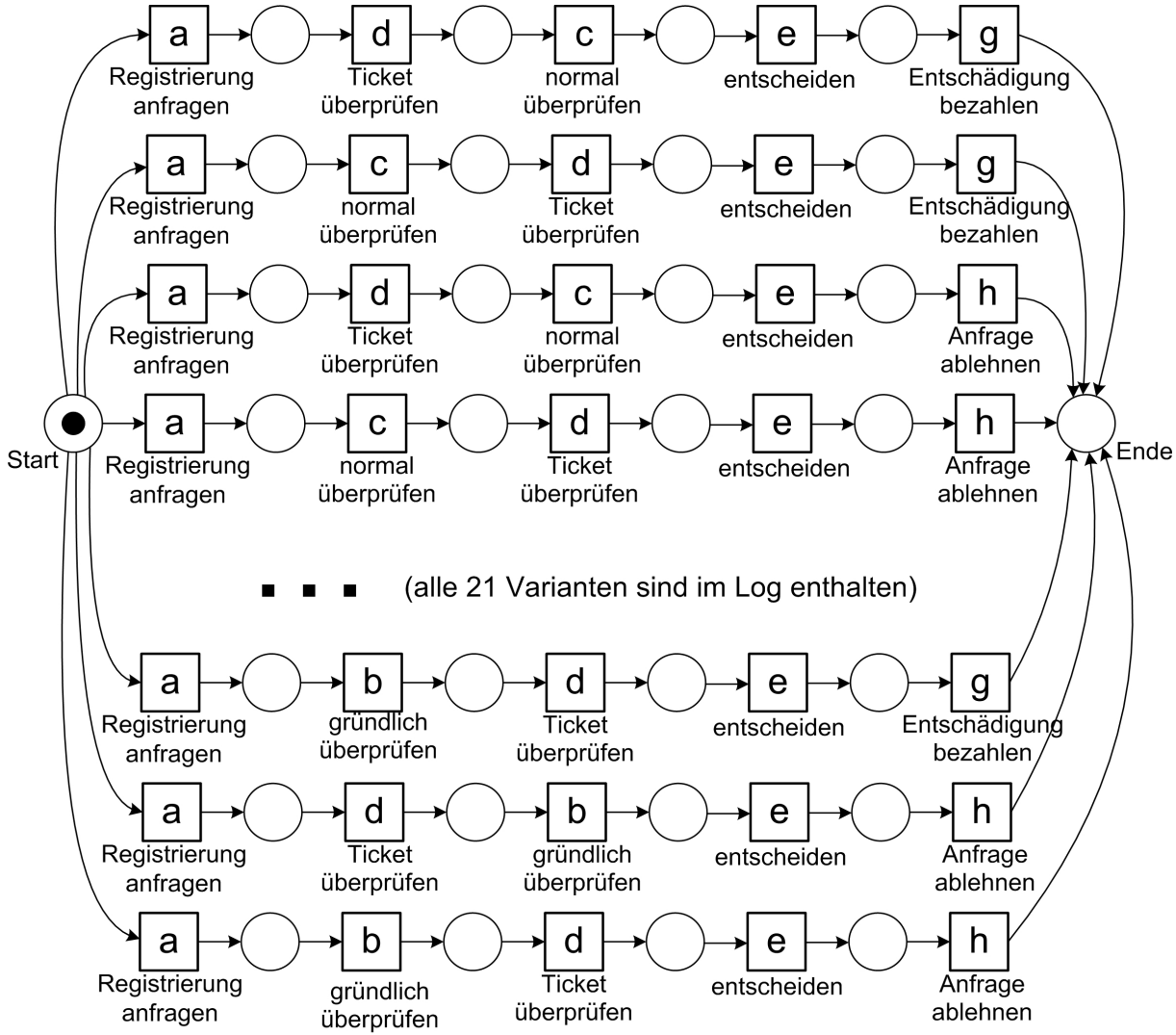


Wie schneidet das Modell bzgl. der 4 Qualitätskriterien ab ?

Modell 1

#	trace
455	acdeh
191	abdeg
177	adceh
144	abdeh
111	acdeg
82	adceg
56	adbeh
47	acdefdbeh
38	adbeg
33	acdefbdeh
14	acdefbdeg
11	acdefdbeg
9	adcefcdeh
8	adcefdbeh
5	adcefbdeg
3	acdefbdefdbeg
2	adcefdbeg
2	adcefbdefbdeg
1	adcefdbefbdeh
1	adbefbdefdbeg
1	adcefdbefcdefdbeg
1391	

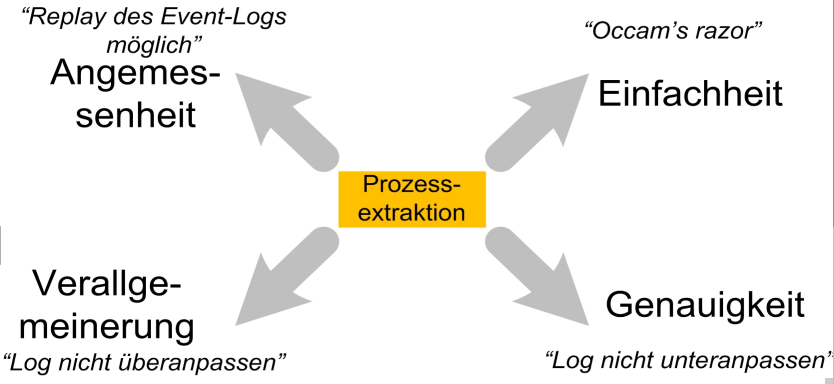




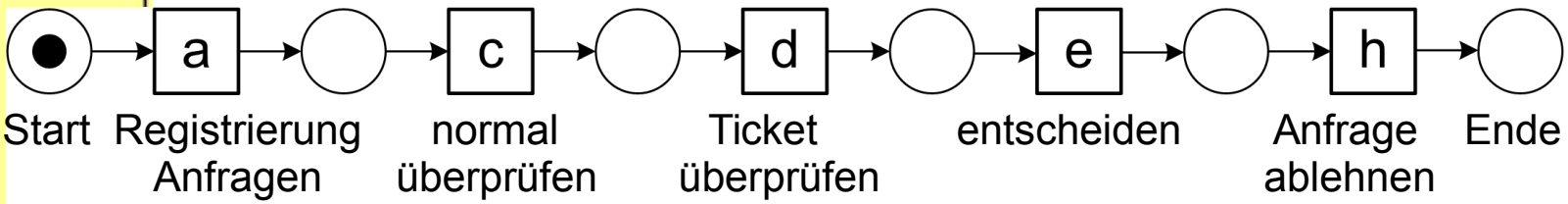
Angemessenheit: + *Genauigkeit: +*

Verallgemeinerung: - *Einfachheit: -*

Modell 2



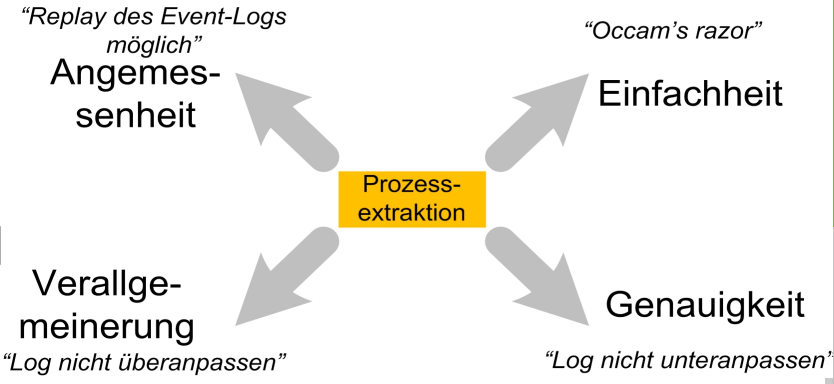
#	trace
455	acdeh
191	abdeg
177	adceh
144	abdeh
111	acdeg
82	adceg
56	adbeh
47	acdefdbeh
38	adbeg
33	acdefbdeh
14	acdefbdeg
11	acdefdbeg
9	adcefcdeh
8	adcefdbeh
5	adcefbdeg
3	acdefbdefdbeg
2	adcefdbeg
2	adcefbdefbdeg
1	adcefdbefbdeh
1	adbefbdefdbeg
1	adcefdbefcdefdbeg
1391	

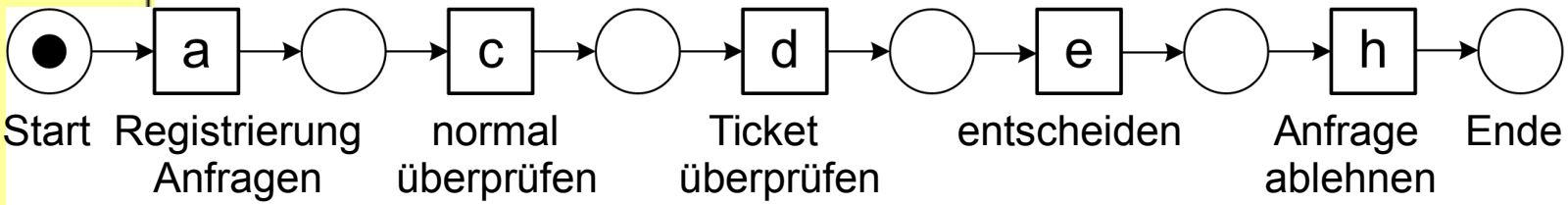


Wie schneidet das Modell bzgl. der 4 Qualitätskriterien ab ?

Modell 2

#	trace
455	acdeh
191	abdeg
177	adceh
144	abdeh
111	acdeg
82	adceg
56	adbeh
47	acdefdbeh
38	adbeg
33	acdefbdeh
14	acdefbdeg
11	acdefdbeg
9	adcefcdeh
8	adcefdbeh
5	adcefbdeg
3	acdefbdefdbeg
2	adcefdbeg
2	adcefbdefbdeg
1	adcefdbefbdeh
1	adbefbdefdbeg
1	adcefdbefcdefdbeg
1391	



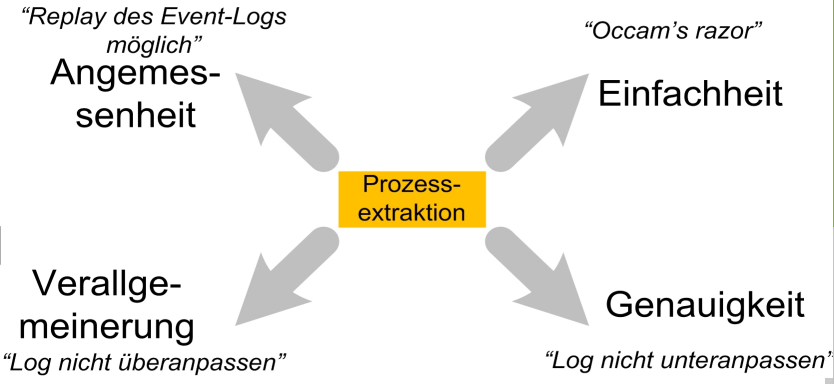


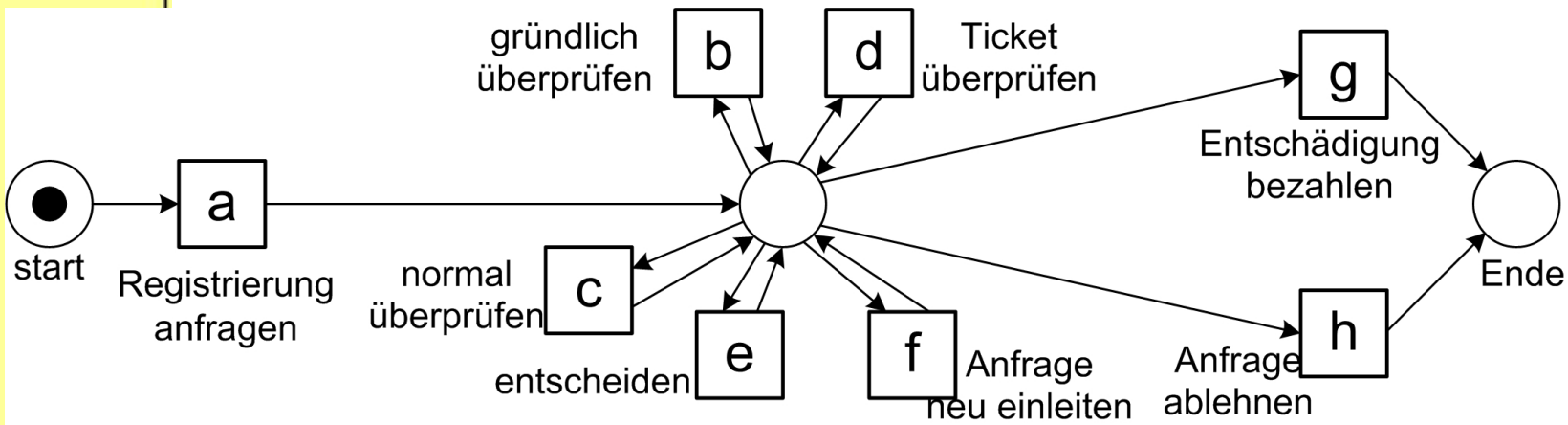
Angemessenheit: - Genauigkeit: +

Verallgemeinerung: - Einfachheit: +

Modell 3

#	trace
455	acdeh
191	abdeg
177	adceh
144	abdeh
111	acdeg
82	adceg
56	adbeh
47	acdefdbeh
38	adbeg
33	acdefbdeh
14	acdefbdeg
11	acdefdbeg
9	adcefcdeh
8	adcefdbeh
5	adcefbdeg
3	acdefbdefdbeg
2	adcefdbeg
2	adcefbdefbdeg
1	adcefdbefbdeh
1	adbefbdefdbeg
1	adcefdbefcdefdbeg
1391	

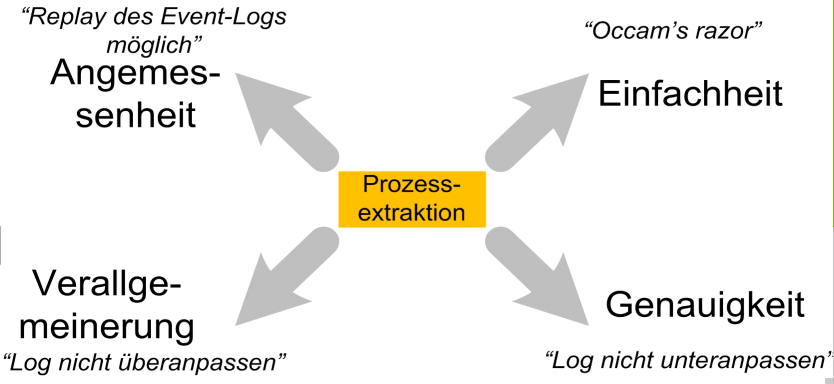


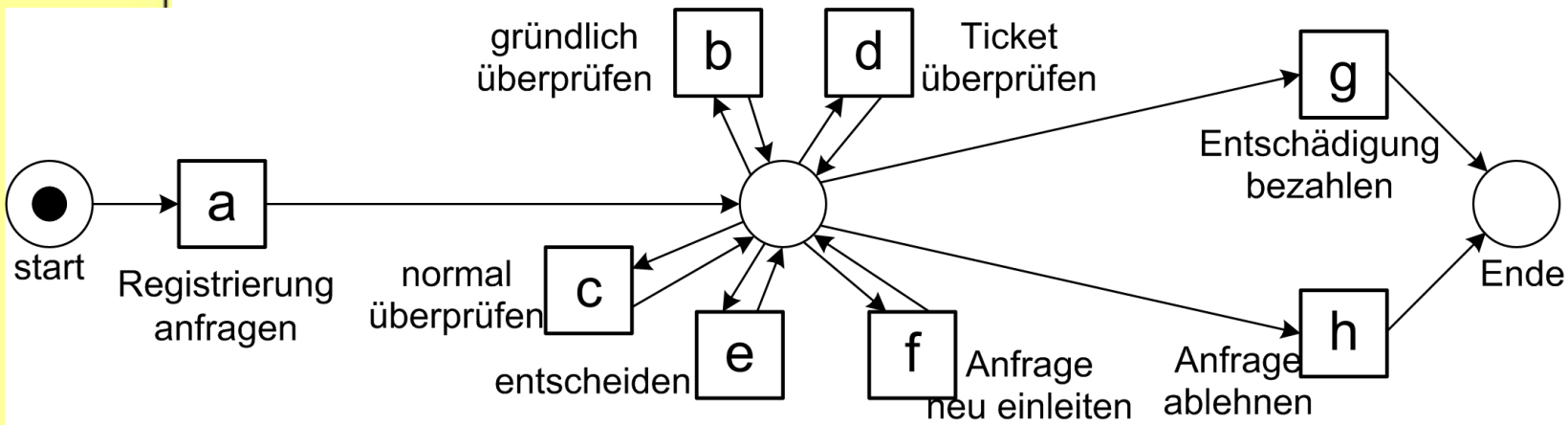


Wie schneidet das Modell bzgl. der 4 Qualitätskriterien ab ?

Modell 3

#	trace
455	acdeh
191	abdeg
177	adceh
144	abdeh
111	acdeg
82	adceg
56	adbeh
47	acdefdbeh
38	adbeg
33	acdefbdeh
14	acdefbdeg
11	acdefdbeg
9	adcefcdeh
8	adcefdbeh
5	adcefbdeg
3	acdefbdefdbeg
2	adcefdbeg
2	adcefbdefbdeg
1	adcefdbefbdeh
1	adbefbdefdbeg
1	adcefdbefcdefdbeg
1391	



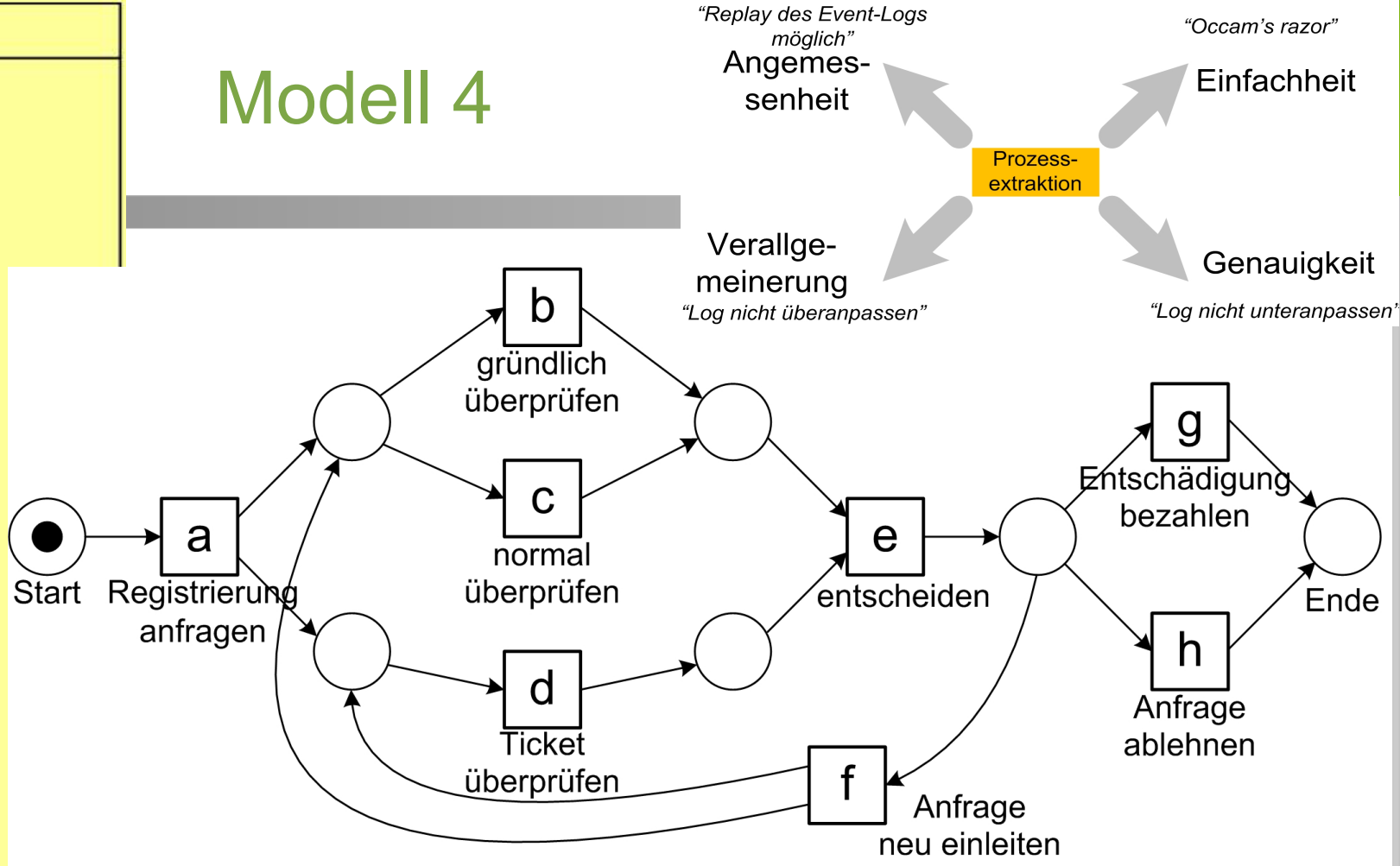


Angemessenheit: + Genauigkeit: -

Verallgemeinerung: + Einfachheit: +

Modell 4

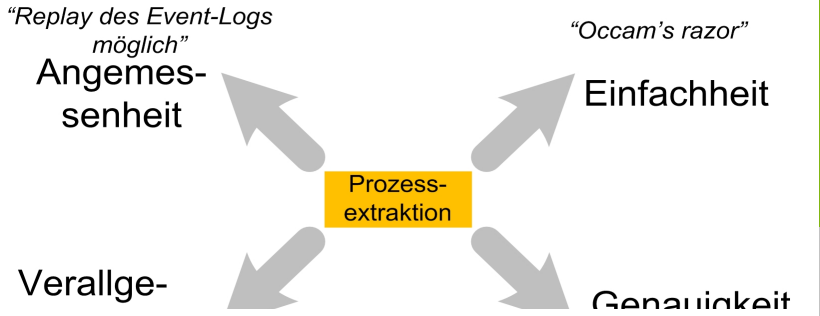
#	trace
455	acdeh
191	abdeg
177	adceh
144	abdeh
111	acdeg
82	adceg
56	adbeh
47	acdefdbeh
38	adbeg
33	acdefbdeh
14	acdefbdeg
11	acdefdbeg
9	adcefcdeh
8	adcefdbeh
5	adcefbdeg
3	acdefbdefdbeg
2	adcefdbeg
2	adcefbdefbdeg
1	adcefdbefbdeh
1	adbefbdefdbeg
1	adcefdbefcdefdbeg
1391	

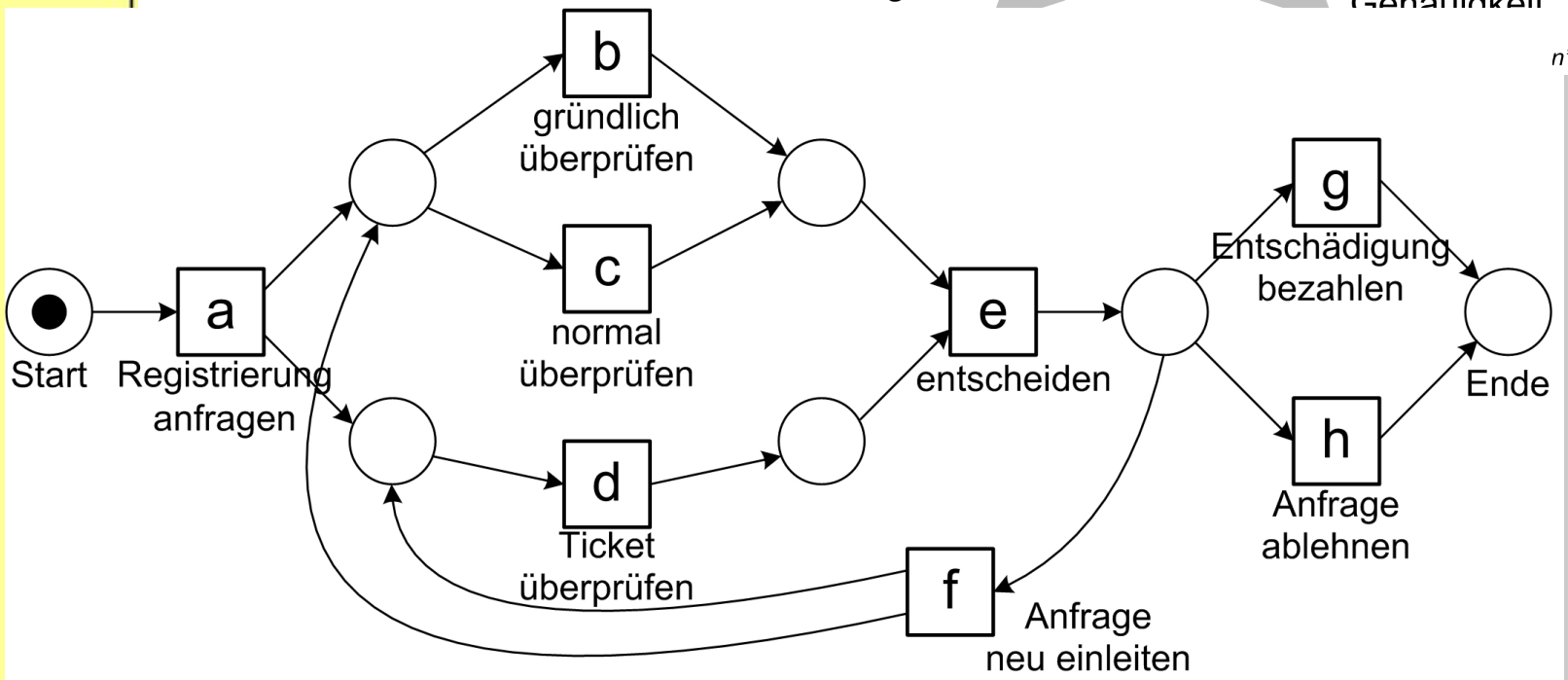


Wie schneidet das Modell bzgl. der 4 Qualitätskriterien ab ?

#	trace
455	acdeh
191	abdeg
177	adceh
144	abdeh
111	acdeg
82	adceg
56	adbeh
47	acdefdbeh
38	adbeg
33	acdefbdeh
14	acdefbdeg
11	acdefdbeg
9	adcefcdeh
8	adcefdbeh
5	adcefbdeg
3	acdefbdefdbeg
2	adcefdbeg
2	adcefbdefbdeg
1	adcefdbefbdeh
1	adbefbdefdbeg
1	adcefdbefcdefdbeg
1391	

Modell 4





Angemessenheit: + *Genauigkeit: +*

Verallgemeinerung: + *Einfachheit: +*

Was macht Process-Mining zu so einem schweren Problem?

- Keine negativen Beispiele:
  - Logs zeigen, was geschah. **Nicht:** was nicht passieren kann.
- Durch Nebenläufigkeit, Schleifen, Verzweigungen:
  - Suchraum hat komplexe Struktur.
  - Log enthält **nur Teil** des möglichen Verhaltens.
- Kein Zusammenhang zwi. **Größe des Modells** und seinem **Verhalten:**
  - Kleines Modell kann mehr/weniger Verhalten generieren.
  - Klassische Analyse- und Evaluations-Methoden erwarten Monotonie-Eigenschaften.

Zusammenfassung
2.4 Prozessextraktion

In diesem Abschnitt:

- Einführung und Beispiel
- α-Algorithmus
  - Idee und Vorbereitungen
  - Formalisierung
  - Beispiel
  - Weitere Beispiele
  - Einschränkungen
- Allgemeine Herausforderungen beim Process-Mining

Im nächsten Abschnitt:

- Konformanzanalyse (mit Hilfe von Replay).