Process Mining – SoSe 2019

Inhalt

[Motivation und Überblick 3](#_Toc8492055)

[Tools 3](#_Toc8492056)

[Process Mining Fragestellungen 3](#_Toc8492057)

[Mathematische Grundlagen 3](#_Toc8492058)

[Data Mining 4](#_Toc8492059)

[Petrinetze 5](#_Toc8492060)

[Andere Systemmodelle 6](#_Toc8492061)

[Eingabeseite/Ereignis-Log 7](#_Toc8492062)

[Alpha-Algorithmus (Filtert Noise nicht) 8](#_Toc8492063)

[Heuristik Miner (filtert Noise) 9](#_Toc8492064)

[Synthese-basierte Verfahren 9](#_Toc8492065)

[Conformance Checking – Qualitätsüberprüfung 9](#_Toc8492066)

[Enhancement – Daten + Modell = besseres Modell 9](#_Toc8492067)

[Erweiterte Überblicksgrafik und Ausblick 9](#_Toc8492068)

# Motivation und Überblick

## Tools

PROM-Toolbox (PromLite)

Ceronis (keine Nebenläufigkeit)

Disco

## Process Mining Fragestellungen

Welchem Arbeitsablauf folgen die Teilnehmer (eines Prozesses) wirklich?

* Abweichungen zur erwarteten Praxis
* Gründe für Abweichungen
* Sind Probleme zu erwarten?
* Was sind mögliche Gegenmaßnahmen

Gibt es Engpässe im Prozess?

* Wie kann ich diese vermeiden?

Was sind die meistbenutzten Pfade?

Können Probleme vorausgesagt werden?

* Abweichungen, Kostenrisiken, Terminrisiken, Qualitätsrisiken

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Pat.Nr | Untersuchung | Zeit | Arzt | Alter | Diagnose | Kosten | Name |
| 1 | Bluttest | 31.1 |  |  |  |  |  |
| 2 | Röntgen | 31.1 |  |  |  |  |  |
| .. | .. |  |  |  |  |  |  |
| 1 | x | 2.2 |  |  |  |  |  |

Jede Zeile wird als Aktivität / Ereignis in Arbeitsablauf betrachtet. Beispielsweise könnte man Untersuchungen analysieren: Zu jedem Patienten gehört ein Behandlungsverlauf. Die Menge der Behandlungsverläufe können wir durch ein Prozessmodell darstellen und analysieren.

# Mathematische Grundlagen





## Formale Sprachen

Alphabet = Endliche Menge T von Zeichen

Wort = Endliche Folge w = von Zeichen aus T

Leeres Wort = 

Länge eines Wortes w ist die Anzahl seiner Zeichen |w|

### Beispiel:

T = {a, b} u = ab a als Aktivität

 = 0 v = bbb

| a b a | = 3 uv = abbbb

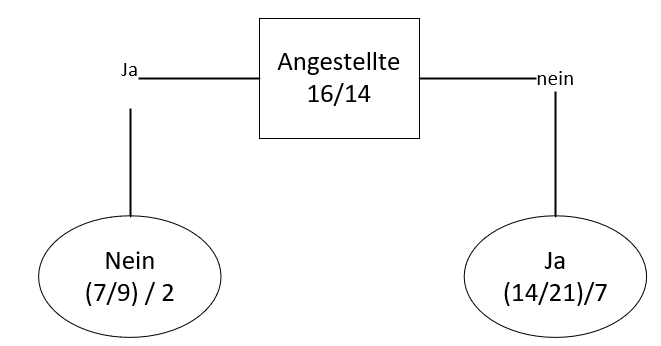
 = Menge aller Wörter über T mit Länge >= 1

* = 
* = Menge aller Wörter über T der Länge n für ein 

# Data Mining

Übungsaufgabe zu Entscheidungsbäumen

|  |  |
| --- | --- |
| Angestellte Ja | Angestellte Nein |
| C(ja) = 2  C(nein) = 7 | C(ja) = 14  C(nein) = 7 |



= 0,76

🡪 <0.37 letzte Vorlesung

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Vorhergesagte Klasse | |
| Ja | Nein |
| Tatsächliche Klasse | Ja | 14 | 2 |
| Nein | 7 | 7 |

Fehlerrate =

Genauigkeit =

Trefferquote =

# Petrinetze

Ein einfaches Petrinetz

besteht aus

* Einer endlichen Menge von Stellen P
* Einer endlichen Menge von Transitionen T
* Die Flussrelation

Interpretation:

* Transitionen repräsentieren Aktionen
* Stellen repräsentieren lokale Zustände

°

Eine Markierung ist eine Abbildung . Diese repräsentiert den Zustand

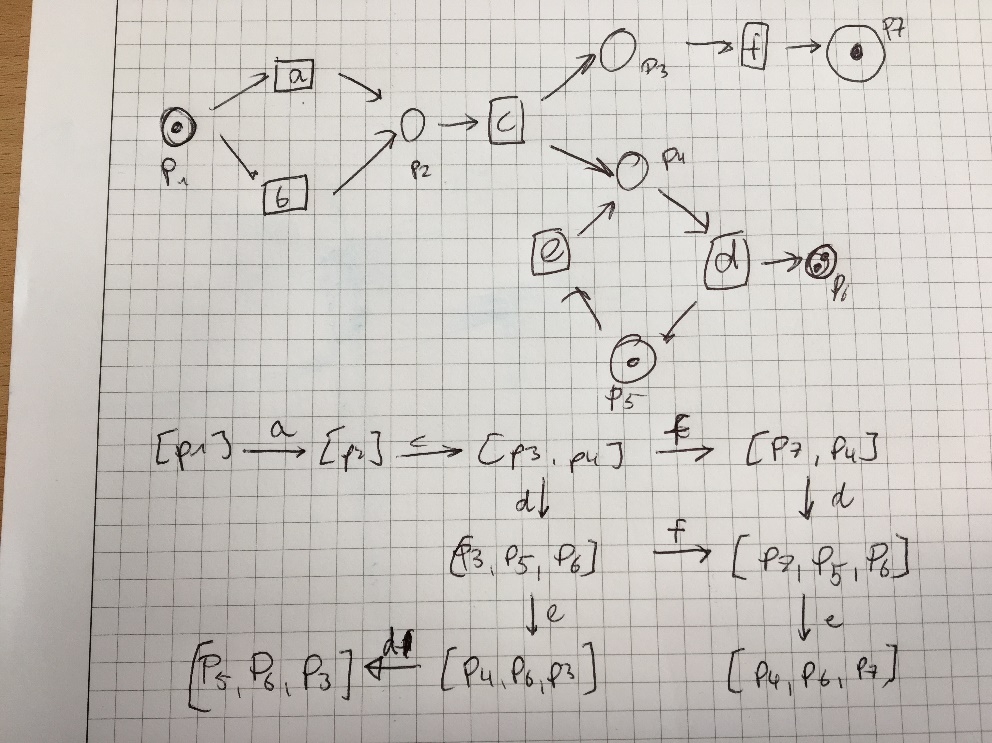
Ein markiertes Netz besteht aus einem einfachen Netz und einer Anfangsmarkierung m0.

Jetzt definieren wir das Verhalten von markierten Petrinetzen. In welchem Zustand ist welche Aktion möglich und in welchen Folgezustand führt eine mögliche Aktion.

Schaltregel

Eine Transition t ist aktiviert in einer Markierung m, falls m >= °t

In allen Vorbereichstellen von t liegt eine Marke.

Eine aktivierte Transition t kann schalten ( = Ausführung einer Aktion). Das Schalten von t führt in den Folgezustand . Dafür schreiben wir 

Eine Schaltfolge ist ein Wort mit . Dafür schreiben wir .

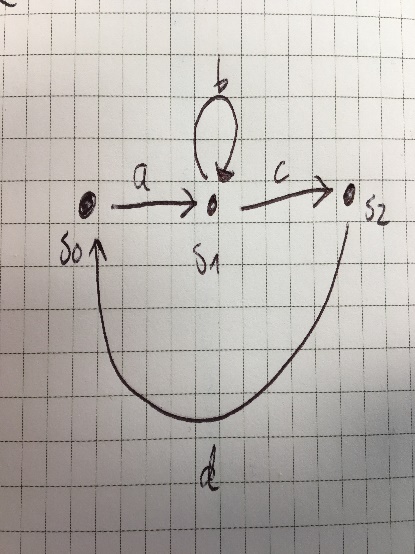
Die Sprache von N=(P, T, F, m0) ist definiert als

Die Menge der erreichbaren Markierungen ist

## Exkurs Transitionssysteme

Ein Transitionssystem besteht aus

* S Menge der (globalen) Zustände
* Anfangszustand
* T endliche Menge von Transitionen
* Zustandsübergangsrelation



Für

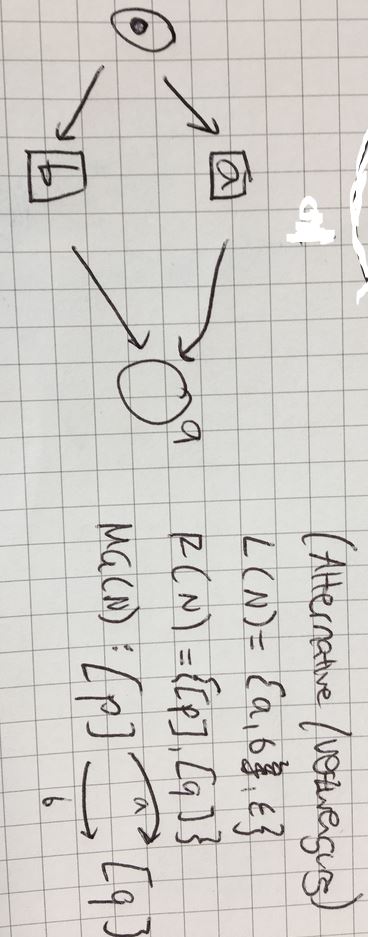
Interpratation:

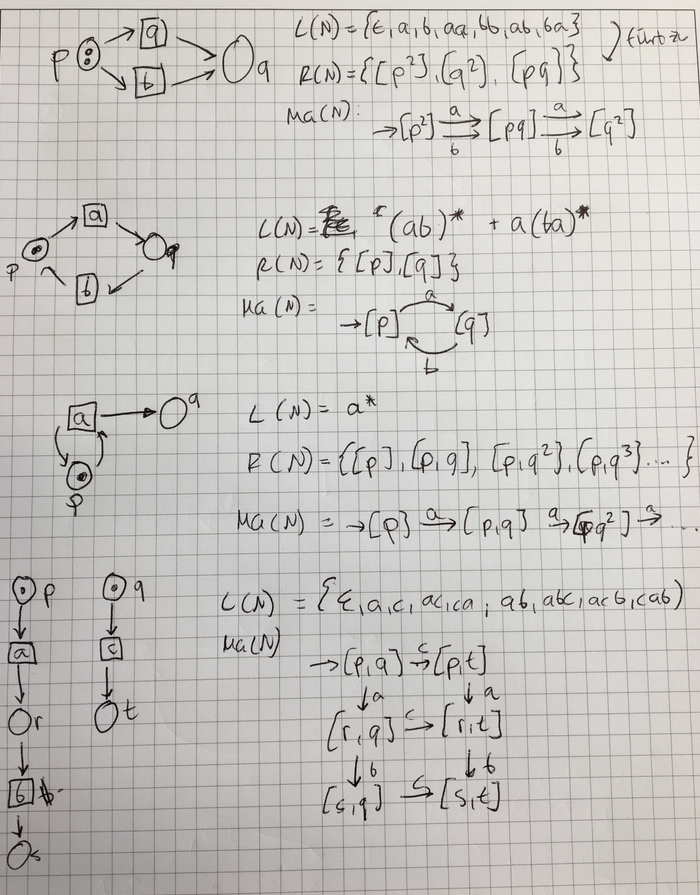
T kann in s schalten und führt in den Folgezustand s‘

Der Markierungsgrad eines markierten Netzes N ist das Transitionssystem

)

Beispiele:





Ein Schritt ist eine Multimenge über T. Er kann schalten in m, falls

Wenn schaltet, so führt das zu der Folgemarkierung

Dafür schreiben wir

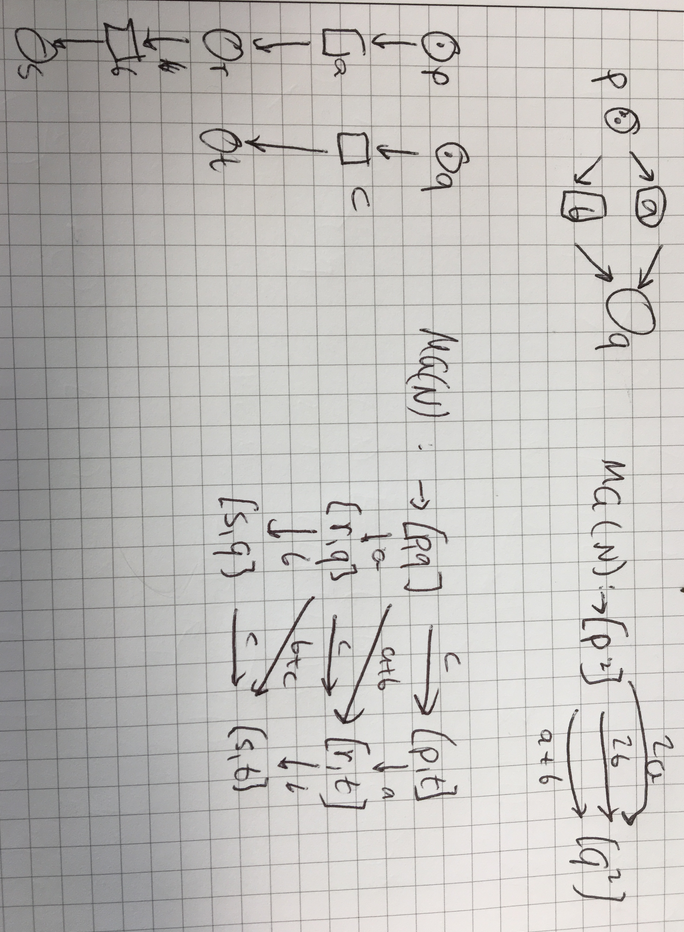
Die Transitionen innerhalb eines Schritts sind nebenläufig mit

Ein Wort heißt Schrittschaltfolge, falls Markierungen m1,.., mn existieren mit

Wir schreiben auch . Die Menge alle Schrittschaltfolgen heißt Schrittsprache Ls(N) von N.

Der Schrittmarkierungsgraph ist wie folgt definiert:

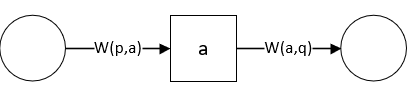
Im Schrittmarkierungsgraphen kann man Nebenläufigkeit ablesen im Gegensatz zum Markierungsgraphen

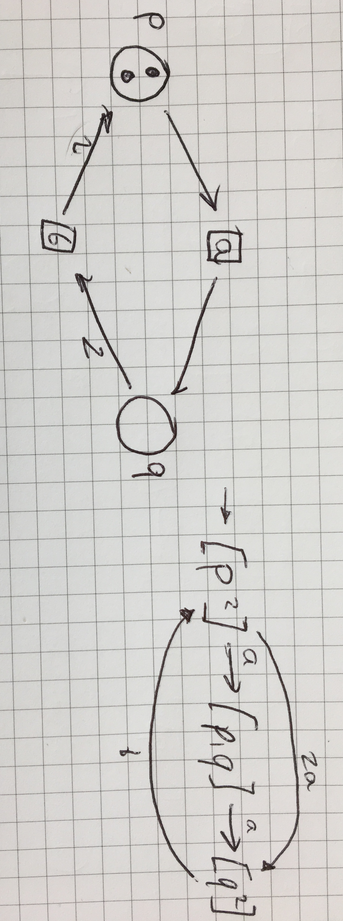


## Stellen / Transitionsnetze (S/T-Netze)

Ein S/T-Netz N=(P, T, F, W) besteht aus

* Einem einfachen Netz (P, T, F)
* Einer Kantengewichtsfunktion W





Wir erweitern die Definitionen °t und t° wie folgt:

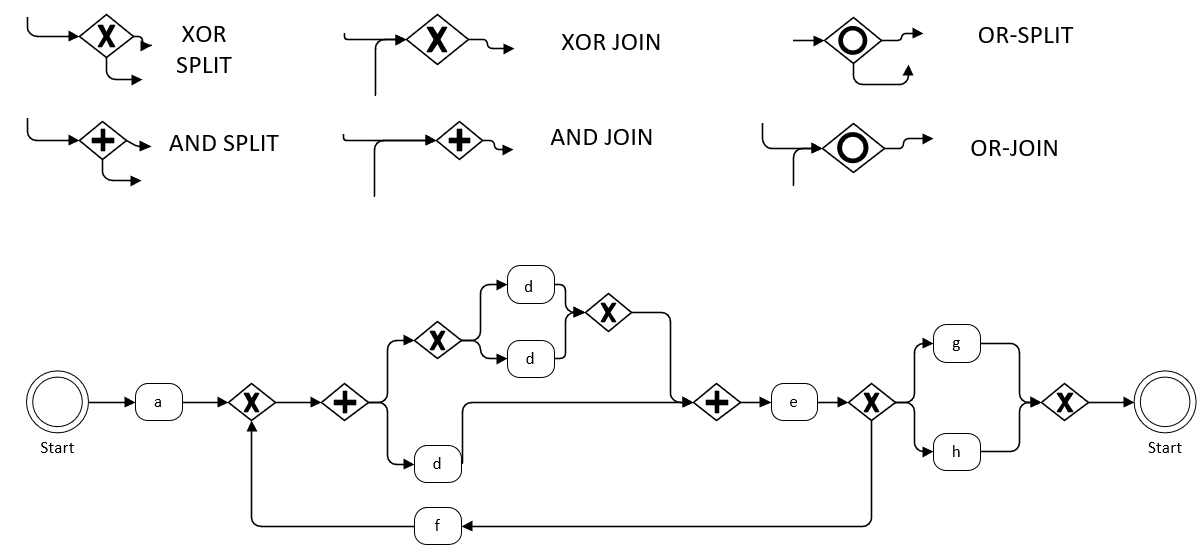
Folgende Definition können wir unverändert von einfachen Netzen übernehmen:

* Schaltregel
* Schaltfolge
* Sprache
* Markierungsgraph
* Und dasselbe für Schritte
* Erreichbare Markierungen

## BPMN

(Business Process Modeling Notation)

Wir besprechen nur einige Basiskomponenten.



Dargestellt ist das Beispiel N3 von vorher.

Das Verhalten eines BPMN istdurch seine Schaltsequenzen gegeben.

Transitionen und Kontroll werden dabei durch Marker auf Kanten aktiviert.

Wenn eine Transition t schaltet, wandert die Marke von den eingehenden auf die ausgehende Kante.

* XOR-SPLIT: Die Marke wandert von der eingehenden auf genau eine der ausgehenden Kanten. Mehrere Transitionen streiten sich um eine Marke.
* AND-Split: Alle ausgehenden Kanten werden markiert -> Nebenläufigkeit
* XOR-Join: Genau eine eingehende Kante muss eine Marke tragen
* AND-Join: Alle eingehenden Kanten müssen markiert sein
* OR-Split: Eine beliebige Teilmenge der ausgehenden Kanten wird markiert (nicht ausschließendes Oder)
* OR-Join: Eine beliebige Teilmenge der eingehenden Kanten muss markiert sein. Er wartet, bis alles getan ist. „Alle Marken, die noch irgendwo rumliegen, müssen eingesammelt werden.“

## Dependency Graph

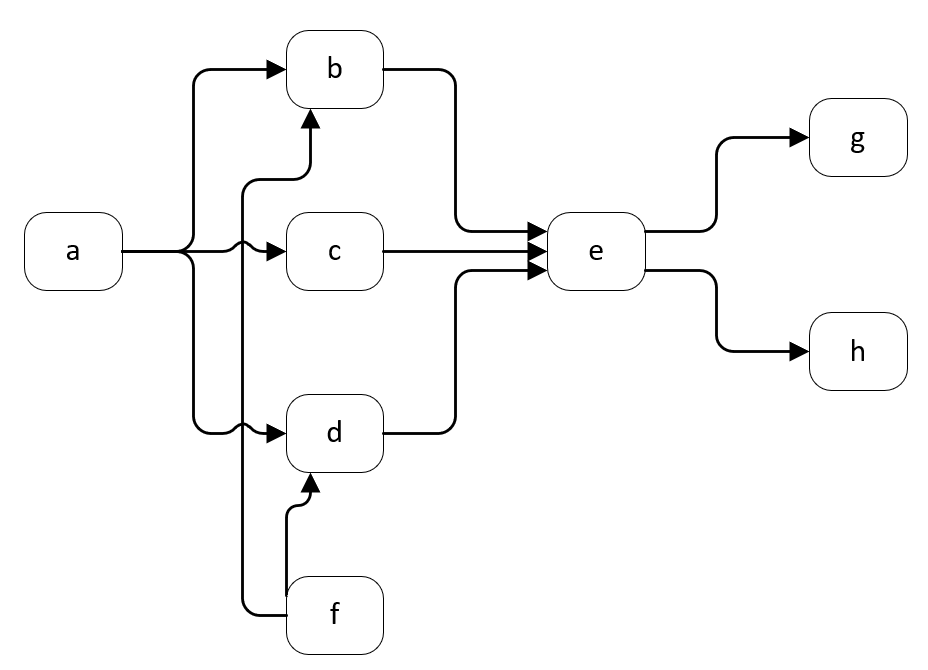


Abbildung 1: Dependency Graph zu oben dargestellter Abfolge

In einem DG werden die Transitionen direkt mit Pfeilen verbunden, es gibt keine weitere Knotenart.

|t°| > 1 : entspricht OR-Split (vorwärts verzweigte Transition)

|°t| > 1: entspricht OR-Join

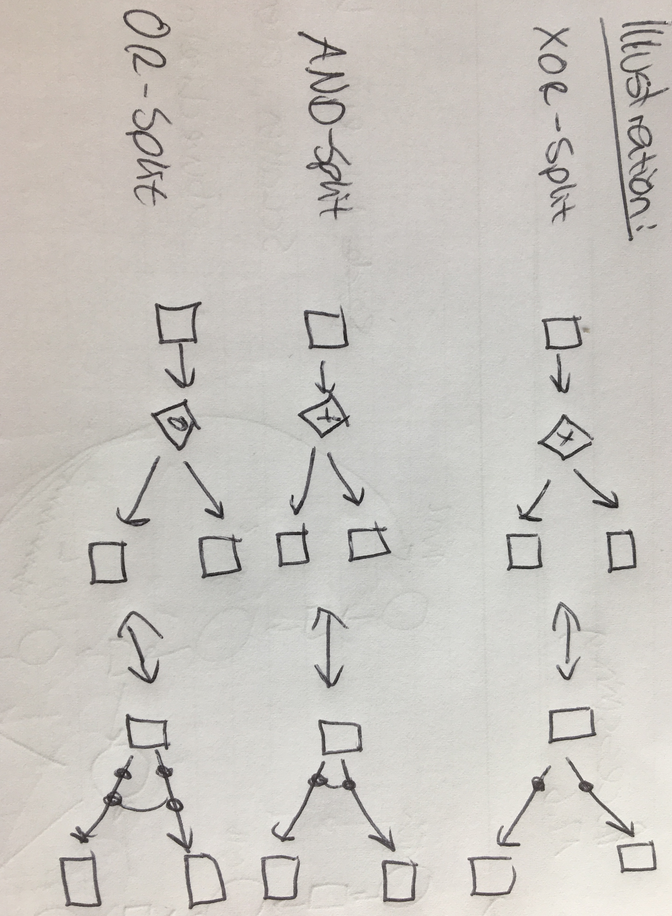
Man kann nicht mehr zwischen Alternativen und Nebenläufigkeiten unterscheiden. DGs sind Ausgabemodell einiger Processdiscovery-Algorithmen. Sie sind Basismodell für Causal Nets. Die Kanten in einem DG können mit Wahrscheinlichkeiten oder Häufigkeiten annotiert werden (als Ergebnis eines Process Discovery Algorithmus). Man sieht dann sofort, was die meistbenutzten Pfade sind.

## Causal Net

## 

Ein Causal Net ist ein DG zusammen mit einer Menge von Eingabe- und Ausgabebindungen für jede Transition.

Eine Bindung ist hierbei immer eine Kombination von eingehenden oder ausgehenden Kanten.



Schaltregel:

Eine Transition ist aktiviert, wenn die Verteilung der Marken auf eingehende Kante einer Eingabe-Bindung entspricht. Wenn die Transition schaltet, produziert sie Marken auf den ausgehenden Kanten gemäß einen der Ausgabe-Bindungen.

Es gibt genau eine Starttransition. Diese ist genau einmal zu Beginn aktiviert. Es eine End-Transition, die genau einmal am Ende schaltet. Es dürfen keine Marken übrigbleiben.

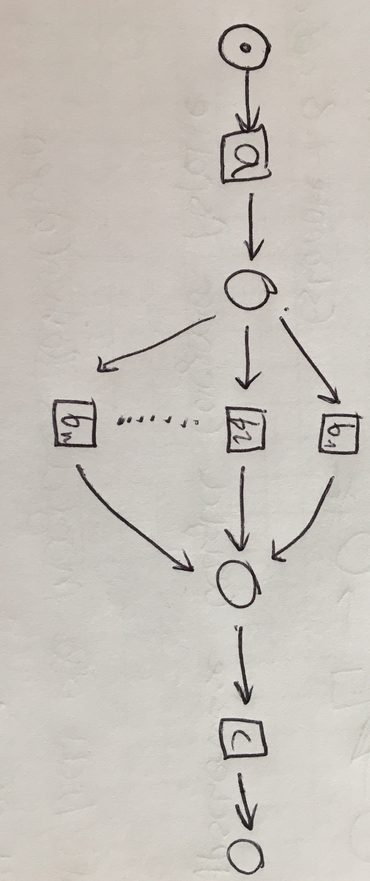
Die Semantik von CN ist nicht ausführbar, das heißt die Schaltsequenzen können nicht durch iterative Anwendung der Schaltregel erstellt werden (da bei einer Entscheidung für ein „falsches“ Bindung später Marken übrigbleiben können). Stattdessen kann man für eine vorgegebene Transitionsfolge testen, ob sie eine Schaltfolge ist (Replay-Semantik).

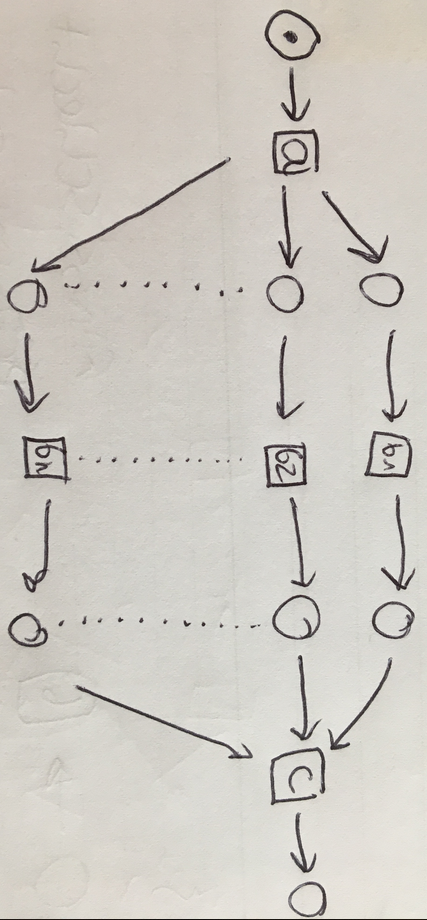
## Zusammenfassung

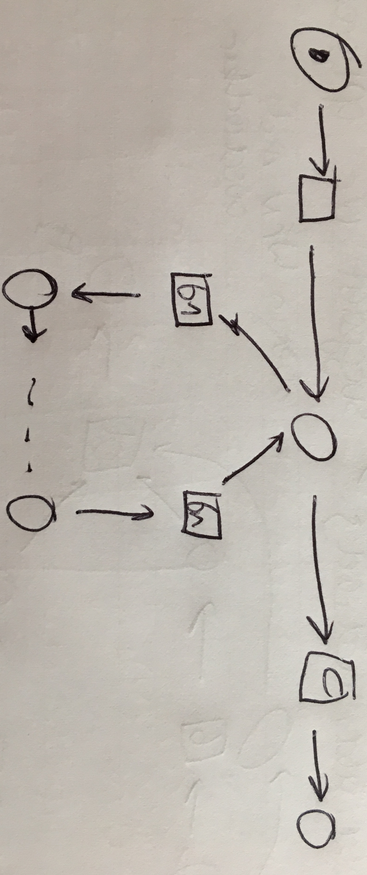
1. kann implizite Stellen enthalten (kann im Post-Processing behoben werden).
2. Alpha-Algorithmus kann keine Zyklen der Länge 1 und 2 entdecken (behoben von a+-Alg.).
3. Kann keine nicht-lokalen Abhängigkeiten entdecken (es existiert keine Verbesserung, andere Algorithmen notwendig).
4. ist in der Regel nicht sound.
5. Berücksichtigt keinen Noise.
6. Representational Bias (Verzerrung durch verwendete Modellierungssprache) bzgl. WFNs.

(keine -Transitionen, keine doppelte Transition, keine Kantengewichte, max. 1 Marke/Stelle, keine Selbstnebenläufigkeit, …)

1. Robustheit bzgl. Unvollständigkeit des Logs:

a)





Wie viele und welche Beobachtungen benötigt der Alpha-Algorithmus, um diese WFNs wiederzuentdecken? -> Übung

1. Ist N ein strukturiertes WFN und vollständig bzgl. N, dann gilt alpha(L)=N

(ohne Details)

# Heuristic Miner

* Berücksichtigt Frequencies (Noise)
* Wird verwendet, um das Hauptverhalten zu analysieren (seltenes Verhalten bleibt als Noise unberücksichtigt)
* 1. Publikation Log -> WFN
* Aktuelle Version: PROM – Log – CN

Beispiel-Log:

L ={ abcd^9, acbd^9, aed^9, abced^1, aecbd^1, ad^1}

(Wie oft ist x direkt vor y in einem Trace im Log, Paare zählen und eintragen)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| > | A | B | C | D | E |
| A |  | 10 | 9 | 1 | 10 |
| B |  |  |  | 10 |  |
| C |  | 10 |  | 9 | 1 |
| D |  |  |  |  |  |
| E |  |  | 1 | 10 |  |

-> Wie sicher ist y kausal abhängig von x

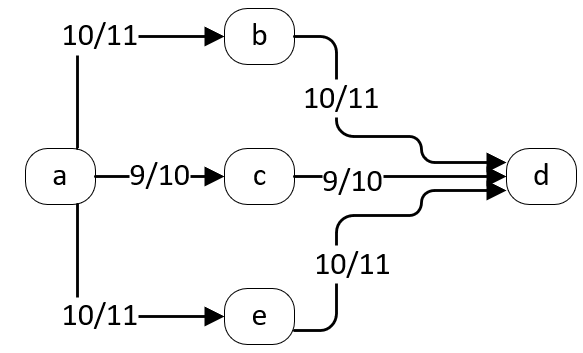
-> Wahrscheinlichkeit/Sicherheit, dass A vor B kommt beträgt 10/11

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| -> | A | B | C | D | E |
| A |  | 10/11 | 9/10 | ½ | 10/11 |
| B | -10/11 |  | 0/21 | 10/11 |  |
| C | -9/10 | 0/21 |  | 9/10 | 0/3 |
| D | -1/2 | -10/11 | -9/10 |  | -10/11 |
| E | -10/11 |  | 0/3 | 10/11 |  |

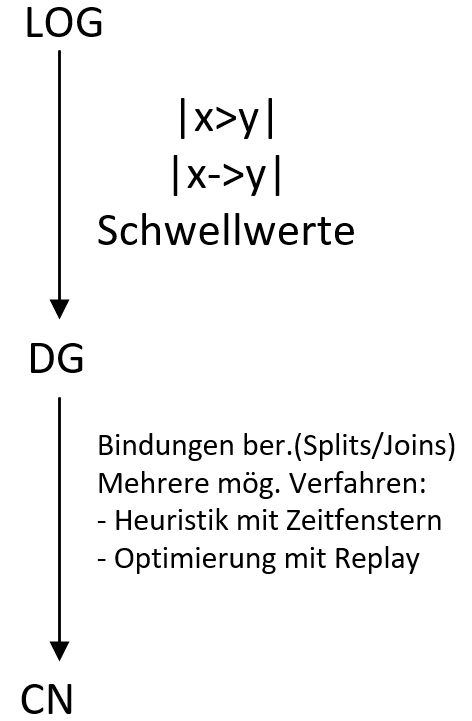
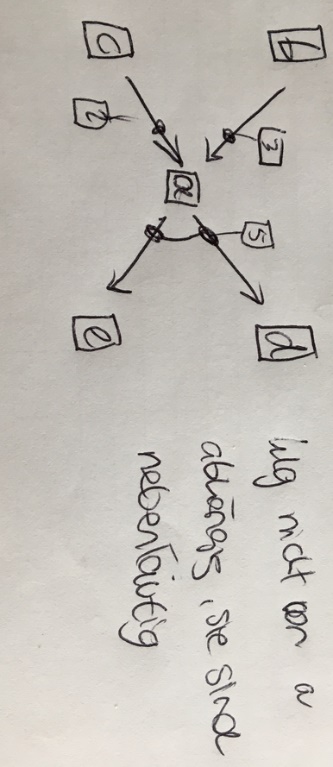
* Alles unter Schwellwert ist noise:
* **Positive observation Treshold** (Schwellwert) legt eine untere Grenze für |x>y | fest (z.B. 3)
* **Causal dependency Treshold** legt eine untere Grenze für || x->y|| fest (zB. 0.8)

Liegt ein Wert unter dem Schwellwert, so wird Noise angenommen und die zugehörige kausale Abhängigkeit nicht berücksichtigt.

Aus allen übrig gebliebenen kausalen Abhängigkeiten wird ein Dependency Graph erstellt:



**Ablauf Heuristik Mining:**



## Extrabeispiel für Zeitfenster

**Traces im Log:**

1. …klbgadhek…

2. …lkgcahedl…

3. …kblgaehdk…

4. …klgbadehk…

5. …[klkc]a[dkeh]…

Zeitfenster der Länge 4 vor und nach a

Welche der direkt mit A verbundenen Aktivitäten kommen vor bzw nach a im Zeitfenster vor?

Vor a: {b},{c} (Weil klg nebenläufig zu a(Graph))

Nach a : {d,e}

Zu jeder der dabei erstellten Teilmenge von Aktivitäten vor | nach a wird eine Bindung eingefügt und festgehalten, wie viele Traces dieser Bindung zugrunde liegen (hier können nochmals Bindungen mit einem Schwellwert gefiltert werden).

Durch Einschränkung des Zeitfensters geht Information verloren.

# Eingabeseite/Ereignis-Log

# Alpha-Algorithmus (Filtert Noise nicht)

Wie kann man die Qualität eines Process Discovery Algorithmus testen?

1. Kann er Modelle „wiederentdecken“?

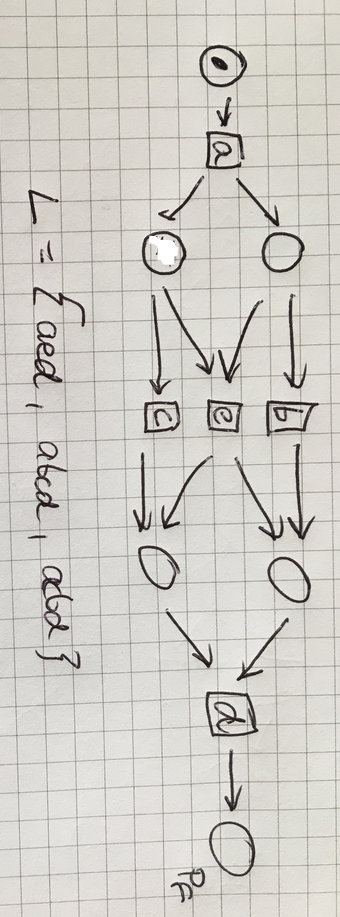
* Modell vorgeben
* Modell simulieren (Log aus Modell erstellen)
* Algorithmus auf Log anwenden
* Ergebnis vergleichen mit Modell aus a).

1. Ergebnis mit anderen Algorithmen vergleichen
2. Ergebnismodell mit Log vergleichen

Finde ich Schaltsequenzen im Log wieder? Gibt es zusätzliche Sequenzen? Ist alles was ich im Log sehe im Modell möglich? Conformance Checking

Input:

Output: WFN N=(P,T,F,m0)



## Grundideen:

* Der alpha-Algorithmus berücksichtigt keine Häufigkeiten (Frequencies)
* Berücksichtigung ausschließlich direkte Nachbarschaften von Ereignissen m Traces
* Er berechnet zuerst die folgenden 4 Relationen zwischen Aktivitäten aus T

1. X > Y, falls in einer Trace y direkt auf x folgt (direkte Nachbarschaft)

Im Beispiel:

A > e a-> e a#d

E > d e -> d e#d

A > b a -> b e#c

B > c a#a

C > d c -> d b#b

A > c a -> c c#c

C > b d#d

B > d b -> d e#e

1. X -> y, falls x>y, abier nicht y>x (x muss immer vor y sein, kausale Abhängigkeit)
2. , falls x > y und y > x (Nebenläufigkeit)
3. , falls weder x>y, noch y>x (alle Paare, die nicht unter 1-3 fallen)

Zugehöriger „Footprint“ (Raute weglassen)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | a | b | c | D | E |
| A |  |  |  |  |  |
| B |  |  |  |  |  |
| C |  |  |  |  |  |
| D |  |  |  |  |  |
| e |  |  |  |  |  |

Dann berechnet der a – Algorithmus zum Footprint ein WFN:

1. T = { t | } T = {a,b,c,d,e}

Ti = { t | } Ti = {a}

To= { t | } To = {d}

1. X = X= {({a},{e}),({e},[d}),({a},{b}),({c},{d}),({a},{c}),({b},{d}),({a},{b,e}),({a},{c,e}),({b,e},{d}),({c,e},{d})}

X= Paare, aus denen Y eine Marke nimmt, die X liefert

Y = (maximale Tupel}

Y= {({a},{b,e}),({a},{c,e}),({b,e},{d}),({c,e},{d})}

S = {Y =

# Heuristik Miner (filtert Noise)

# Synthese-basierte Verfahren

## Regionen-basierte Synthese

Berechnet wird zu einem präfix-abg. Log L ein Netz N mit

* L<=L(N)
* L(N) minimal

Die Stellen von N können als nicht-negative, ganzzahlige Lösungen eines homogenen linearen Ungleichungssystems

berechnet werden.

Lösungen r nennt man Regionen. Aus r lässt sich direkt eine Stelle ableiten.

r = ()

m0() =

W() =

W() =

Das Netz NSat enthält alle zu Lösungen von AL \* r >= 0 gehörenden Stellen – das sind unendlich viele. NSat ist eine gesuchte Lösung von (\*). Ziel: Auswahl einer endlichen Teilmenge von Lösungen mit zugehörigem Netz Nfin, sodass L(Nfin) = L(Nsat).

**Möglichkeit 1:**

Bestimme endliche Menge der sogenannten Basislösungen von mit zugehörigem Netz . Es gilt wie gewünscht: L()=L(. Nachteil: Die Anzahl der Basislösungen ist exponentiell in |L| und n .

**Möglichkeit 2:**

Berechne Lösungen zu allen sogenannten falschen Fortsetzungen von L.

Eine falsche Fortsetzung von L ist ein Wort von der Form v=v’a mit und a, sodass

Beispiel: L = {a, ab, ac, abc}

Falsche Fortsetzungen zu L:

Aa, abb, aca, acb, abca, abcb, abcc, b, c, aba, acc

Es gibt maximal |L|\*n viele falsche Fortsetzungen. Versuche zu jeder falschen Fortsetzung v eine Stelle zu finden, bezüglich der v nicht schalten kann (die v verbietet (sodass es nicht genug Marken zum durchschalten gibt, es entsteht keine Schaltfolge).

~von papier einfügen ~

Falls eine Lösung rv existiert, so verbietet die Stelle prv die falsche Fortsetzung v.

Es gilt: Findet man zu jeder falschen Fortsetzung eine Lösung des erweiterten Systems, so gilt für das zugehörige Netz NSep:

L(NSep)=L(NSat)

Das ist genau dann der Fall, wenn L=L(NSat). Was lässt sich aber über den Fall L < L (NSat) sagen? Dann gibt es falsche Fortsetzungen, die nicht verboten werden können und es gilt dann nicht zwingend, dass das endliche Netz dasselbe Verhalten hat wie das unendliche: L(Nsep) = L(Nsat).

(Wenn man abb nicht verbieten kann muss man trotzdem schauen, ob man vielleicht abbb verbieten kann)

## Anwendung regionenbasierter Synthese auf Process Discovery

### Direkte Anwendungen

* Log als Eingabe
* Synthetisiertes Netz als Ausgabe
* Kein Pre-/Post-Processing

Nachteile:

* Zu zeitaufwändig für große Logs
* Berücksichtigt keinen Noise
* Geht von einer vollständigen Verhaltensbeschreibung aus, Logs aber in der Regel unvollständig => Gefahr des „Overfitting“
* Modell ist zu genau auf Beispieldaten angepasst

Vorteile:

* Ergebnisse sind mathematisch fundiert
* Methode ist variabel anpassbar:
  + Per Konstruktion Lösungen mit gewünschten Eigenschaften (z.B Sound WFN)
  + Zielfunktion
  + Reihenfolge der falschen Fortsetzungen
  + Unterschiedlichste Input- und Outputformate
* Entdeckt auch komplexe Kontrollstukturen (Ausdrucksmächtigkeit des Output- Modells wird voll ausgenutzt)

### Indirekte Anwendungen

1. ILP-Minen (Prom)

Verbesserung des Alpha-Algorithmus – Berechnung von „optimalen“ Stellen für lokale Abhängigkeiten aus Footprint

1. 2-Phasen-Process-Discovery (Prom)
2. Aktuell an unserem Lehrstuhl

LPES ist Ereignisstruktur -> Kompaktestes existierendes Modell

Postprocessing mit Noise(stukturierte Methoden) und Nebenläufigkeit

# Conformance Checking – Qualitätsüberprüfung

Wie gut passt das berechnete Modell zu meinem Log?

Qualitätskriterien: Fitness, Präzision, Einfachheit, Verallgemeinerbarkeit

**Fitness**

Wie gut kann das Modell das Log generieren? Kann es alle Traces aus dem Log generieren, oder nur einen Teil? Wie groß ist dieser Anteil?

Beispiel: Synthesebasierte Methoden generieren Modelle mit optimaler, 100%iger Fitness.

**Precision**

Hat das Modell viel zusätzliches Verhalten, das nicht zum Log gehört? Gibt es sehr viele zusätzliche Traces im Modell, so nennt man das Modell „underfitting“.

Beispiel: Synthesebasierte Methoden generieren Modelle mit optimaler Präzision

**Simplicity**

Ist das Modell möglichst einfach und anschaulich? (keine unnötigen Stellen, möglichst einfache Stellen). Steht oft im Gegensatz zur Precision.

**Generalization**

Wie gut vervollständigt das Modell das Log? (bezüglich fehlender Traces z.B. wegen Nebenläufigkeit der Zyklen.) Vermeidet „Overfitting“, schlecht bei synthesebasierten Verfahren.

Klausur: Wissensfragen, zu verschiedenen Algorithmen, Bereitstellen von Modellen(Wie präzise, simpel etc ist das Modell, das Log).

# 

# Enhancement – Daten + Modell = besseres Modell

# Erweiterte Überblicksgrafik und Ausblick