Syntaktische und Statistische Mustererkennung

VO 1.0 840.040
(UE 1.0 840.041)

Bernhard Jung

bernhard@jung.name
http://bernhard.jung.name/VUSSME/

Rückblick

- PGMs

Markov Field

auch Markov Network oder Markov Random Field genannt.

Graph ähnlich Bayes'sche Netzwerk, aber mit ungerichteten Kanten, die eine Form von probabilistischer Interaktion zwischen Benachbarten Variablen darstellen.

Pairwise Markov Network

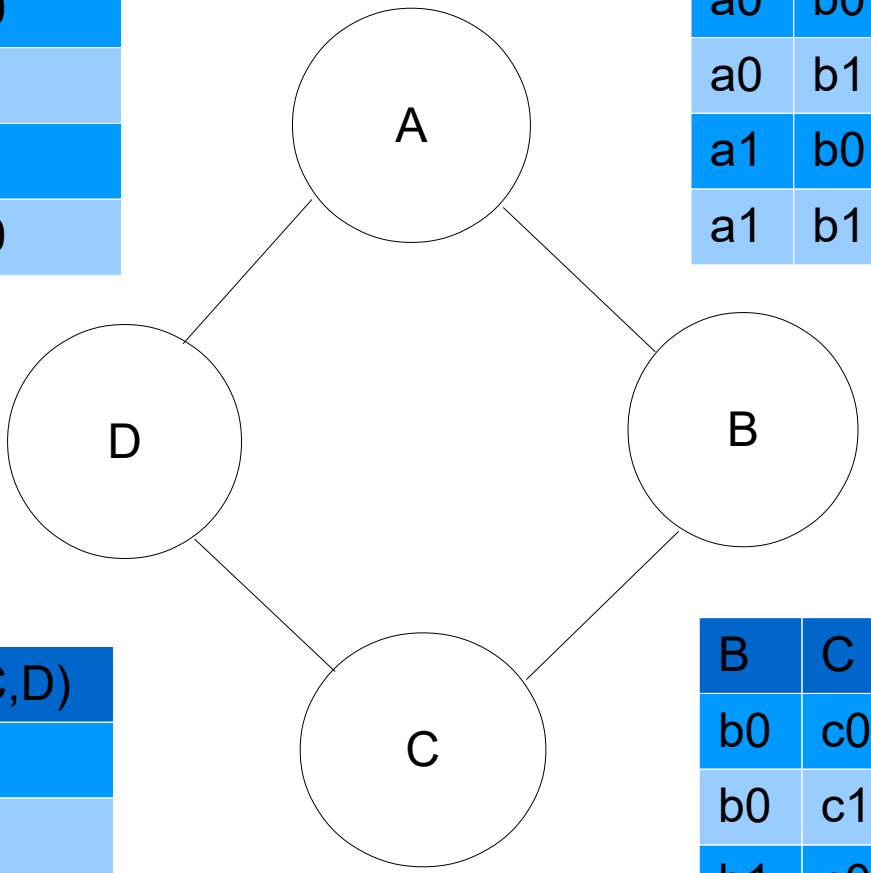
Ein paarweises Markov Netzwerk ist ein ungerichteter Graph, dessen Knoten die Zufallsvariablen $X_1, \ldots, X_n$ sind und jede Kante X_i-X_j ist mit einem Faktor $\Phi(X_i,X_j)$

Pairwise Markov Network

D	A	Φ(D,A)
d0	a0	100
d0	a1	1
d1	a0	1
d1	a1	100

A	B	Φ(A,B)
a0	b0	30
a0	b1	5
a1	b0	1
a1	b1	10

Affinität
Kompatabilität
Soft Constraint



C	D	Φ(C,D)
c0	d0	1
c0	d1	100
c1	d0	100
c1	d1	1

B	C	Φ(B,C)
b0	c0	100
b0	c1	1
b1	c0	1
b1	c1	100

Pairwise Markov Network

$\tilde{P}(A,B,C,D) = \Phi_1(A,B) \times \Phi_2(B,C) \times \Phi_3(C,D) \times \Phi_4(A,D)$ …. unnormalized measure

$P(A,B,C,D) = 1 / Z * \tilde{P}(A,B,C,D)$ Z … partition function

a0	b0	c0	d0	300000
a0	b0	c0	d1	300000
a0	b0	c1	d0	300000
a0	b0	c1	d1	30
a0	b1	c0	d0	500
a0	b1	c0	d1	500
a0	b1	c1	d0	5000000
a0	b1	c1	d1	500
a1	b0	c0	d0	100
a1	b0	c0	d1	1000000
a1	b0	c1	d0	100
a1	b0	c1	d1	100
a1	b1	c0	d0	10
a1	b1	c0	d1	100000
a1	b1	c1	d0	100000
a1	b1	c1	d1	100000

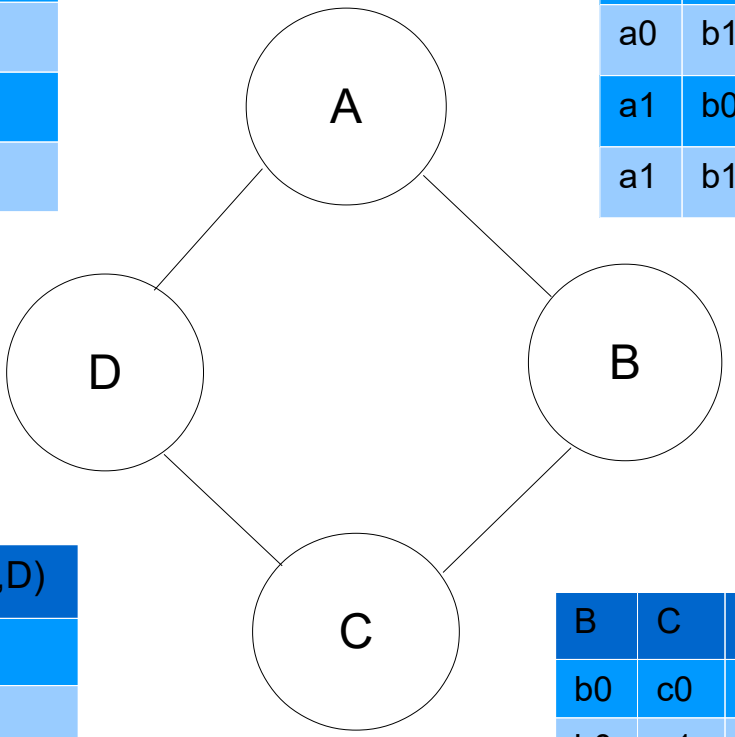
Pairwise Markov Network

P(A, B)

A	B	Prob
a0	b0	0.13
a0	b1	0.69
a1	b0	0.14
a1	b1	0.04

D	A	Φ(D,A)
d0	a0	100
d0	a1	1
d1	a0	1
d1	a1	100

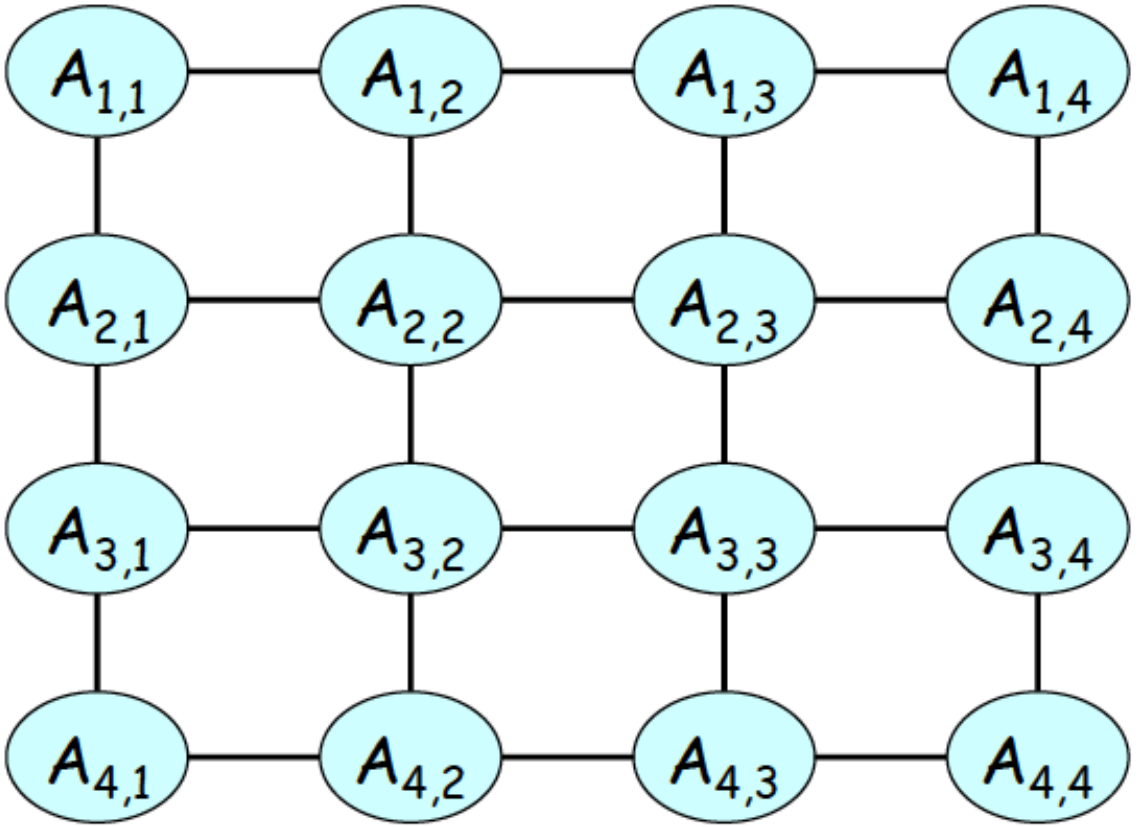
A	B	Φ(A,B)
a0	b0	30
a0	b1	5
a1	b0	1
a1	b1	10



C	D	Φ(C,D)
c0	d0	1
c0	d1	100
c1	d0	100
c1	d1	1

B	C	Φ(B,C)
b0	c0	100
b0	c1	1
b1	c0	1
b1	c1	100

Markov Network



Gibbs Distribution

Menge an Faktoren $\Phi = \{\Phi_1(D_1), \ldots, \Phi_k(D_k)\}$

$\tilde{P}_\Phi(X_1,...,X_n) = \Pi\, \Phi_i(D_i)$ … factor product

$Z_\Phi = \Sigma\, \tilde{P}_\Phi(X_1,...,X_n)$ … partition function

$P_\Phi(X_1,...,X_n) = 1 / Z_\Phi * \tilde{P}_\Phi(X_1,...,X_n)$

Definiertes Markov Netzwerk

z.B. $\Phi_1(A,B,C)$, $\Phi_2(B,C,D)$

Netzwerk H_Φ hat eine Kante X_i-$X_{j,}$ wenn ein Φ_m existiert für das gilt $x_i, x_j \in D_m$

Faktorisierung

P faktorisiert über H wenn ein Φ existiert mit

$\Phi = \{\Phi_1(D_1), \ldots, \Phi_k(D_k)\}$

so dass gilt: $P = P_\Phi$

Zusammenfassung

- Gibbs distribution repräsentiert Verteilungen als Produkt von Faktoren
- Das erzeugte Markov Netzwerk verbindet Paare von Knoten, die im selben Faktor sind
- Die Markov Netzwerkstruktur spezifiziert nicht vollständig die Faktorisierung von P
- Aktive Pfade hängen nur von der Graphenstruktur ab

Schlussfolgern in PGMs

- Bestimmung der bestimmten Wahrscheinlichkeit einer Variable P(Y | E = e)
- Suche nach wahrscheinlichsten Erklärung
  most probable explanation (MPE)
- Bestimmung der maximum a posteriori (MAP) Wahrscheinlichkeit
- Algorithmen: Variable Elimination, Belief Propagation, MAP Estimation, Sampling Methods (MCMC, Gibbs, Metropolis Hastings), ...

Syntaktische, strukturelle ME

Syntaktische Mustererkennung

- Struktur der Muster im Vordergrund
- Analogie zur Syntax der Sprache
- nicht Zahlen, sondern Symbole und Symbolketten

- Voraussetzungen:
  - Formalismus zur Beschreibung/Generierung von Mustern (z.B. formale Grammatik G)
  - Algorithmus zur Entscheidung welcher Klasse ein Muster zuzuordnen ist (z.B. Parser für die Sprachen L(G))

Merkmale in der syntaktischen ME

- Muster besitzen **einfache Bestandteile** (elementare/atomare Teilmuster, Terminalsymbole, Primitiva, etc.), welche **zueinander in Beziehung** stehen
- Ähnliche/Zusammengehörende Muster haben eine bestimmte Struktur, die nach bestimmten Regeln gestaltet ist
- Keine numerischen Merkmalsvektoren, sondern Symbolgraphen oder Symbolketten

Musterprimitiva

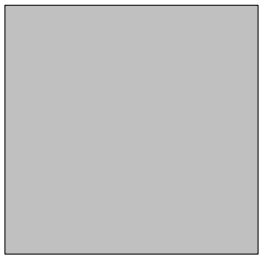
- Textone (Bestandteile einer Textur)
- Phoneme (in der gesprochenen Sprache)
- Grapheme (in der geschriebenen Sprache)
- Linien und Kurvensegmente, definiert durch:
  - Anfangs- und Endpunkt
  - Anstieg
  - Krümmung
- Regionen
- etc.

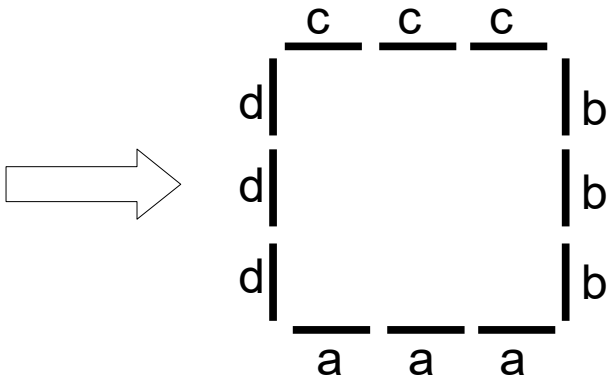
Voraussetzungen/Eigenschaften

- Muster müssen in Musterprimitive zerlegt werden können
- Primitiva und die definierten strukturellen Relationen ermöglichen eine kompakte und ausreichend genaue Beschreibung des Musters
- Primitiva sind mittels nicht-syntaktischer Methoden extrahierbar, d.h. die Extraktion ist unabhängig von der Struktur durchführbar

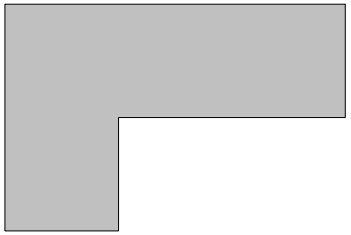
Merkmale in der syntaktischen ME

- Konturkodierung





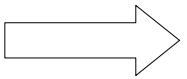
$a^3b^3c^3d^3$



Linienprimitive

a= b= c=

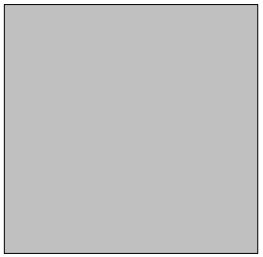
d= e= f=



aeeebfceeafcedff
= $ae^3bfce^2afcedf^2$

Merkmale in der syntaktischen ME

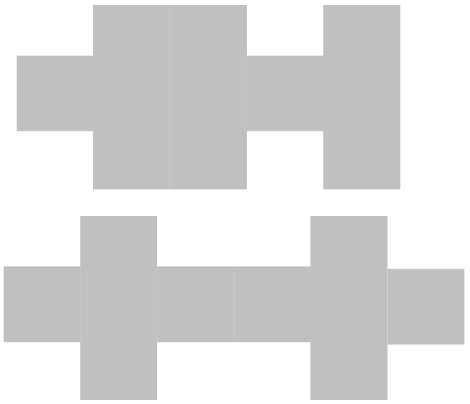
- Feldkodierung von Objekten



d	h	h	h	c
v	i	i	i	v
v	i	i	i	v
v	i	i	i	v
a	h	h	h	b

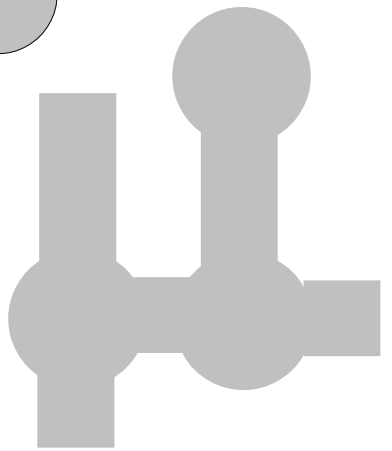
- Codierung mittels Flächenprimitiva

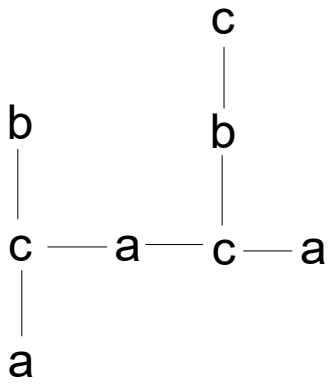
Flächenprimitive a= b= c=



ab^2ab

aba^2ba





Strukturbeschreibung

- Definition der Struktur der Muster in Form einer **Symbolkette**

- **Beziehungsgraphen**
  repräsentiert die strukturellen Eigenschaften. Die Knoten stellen die Primitiva dar, die Kanten die Beziehungen dazwischen.

Anwendungen der syntaktischen Mustererkennung

- Szenenanalyse
- Fingerabdruck- und Gesichtserkennung
- Sprach-/Textanalyse
- Schriftzeichenerkennung
- EKG/EEG-Analyse
- Chemische Strukturen
- Chromosomenanalyse (genomics)
- Proteinanalyse (proteomics)

Erkennen/Klassifizieren von Symbolketten

Pattern Matching: Ähnlichkeit mit Prototypen

- Hamming-Distanz
  - Ketten gleicher Länge
  - Anzahl der Stellen, an denen die Symbol in den Ketten unterschiedlich sind
- Levenshtein-Distanz (Editing-Distanz)
  - Minimale Anzahl an Operationen Einfügen, Löschen und Ersetzen um eine Symbolkette in die andere überzuführen
  - Sonderform der Dynamic-Time-Warping-Distanz
  - Definiert eine Metrik auf dem Raum der Symbolsequenzen

Levenshtein-Distanz

Berechnung:

- Tabelle D $(n_1+1) \times (n_2+1)$
  wobei n_i die Länge der Zeichenketten bezeichnet
- $D_{0,0} = 0$
- $D_{i,j} = \min$

$$\begin{cases} D_{i-1,\,j-1} + 0 & \text{(gleicher Buchstabe)} \\ D_{i-1,\,j-1} + 1 & \text{(Ersetzung)} \\ D_{i-1,\,j} + 1 & \text{(Einfügung)} \\ D_{i,\,j-1} + 1 & \text{(Löschung)} \end{cases}$$

	ε	T	o	r
ε	0	1	2	3
T	1	0	1	2
i	2	1	1	2
e	3	2	2	2
r	4	3	3	2

Erkennen/Klassifizieren von Symbolketten

Parsing: Test, ob eine Struktur von einer bestimmten Grammatik erzeugt werden kann.

- Mustergrammatiken:
  Für jede Musterklasse wird eine Grammatik konstruiert, deren Sätze den Mustern der Klasse entsprechen
  - Am häufigsten werden reguläre und kontextfreie Sprachen verwendet, weil dafür effiziente Methoden existieren
    - Reguläre Sprachen mit endliche Automaten (FSM)
    - kontextfreie Sprachen mit Stack/Kellermaschinen und andere Parsing-Algorithmen

Typ-n Grammatiken

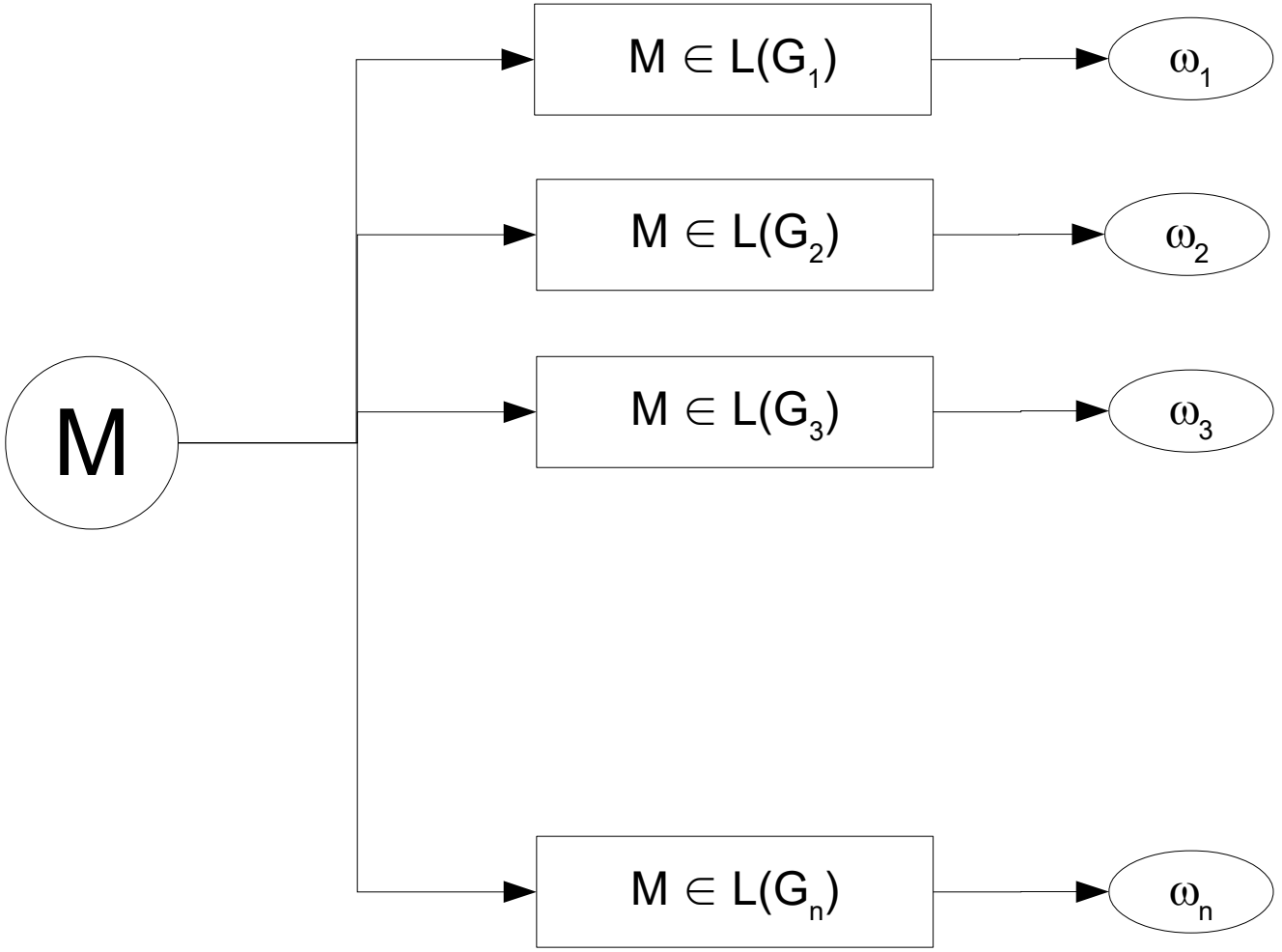
Einschränkung der Mächtigkeit einer Grammatik durch Beschränkungen der zulässigen Produktionsregeln

Typ	Grammatik	Regeln	(minimaler) Automat
0	unbeschränkt	$\alpha \rightarrow \beta$	Turingmaschine
1	kontextsensitiv	$\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2 \quad (\beta \neq \varepsilon)$	nicht-deterministische, linear beschränkte Turing-Maschine
2	kontextfrei	$A \rightarrow \beta$	nicht-deterministischer Kellerautomat
3	regulär	$A \rightarrow \alpha B$ oder $A \rightarrow \alpha$	endlicher Automat

$\alpha,\beta,...$ Terminal- oder Nichterminalsymbol (inkl. Leerwort ε, wenn nicht anders angegeben)

A,B,... Nichtterminalsymbol

Klassifikation



Formale Grammatiken

- Terminalsymbole (i.e. Musterprimitiva)
- Nonterminalsymbole
- Verknüpfungsregeln:
  - einfachste Form: Konkatenation (Hintereinanderreihung)
  - allgemeinste Form: beliebige Vernetzung von Symbolen in einem Graphen
  - Definition in Form von Produktionsregeln
    {links} → {rechts}
    Wenn in einer Satzform der linke Teil der Regel vorkommt, dann darf dieser Teil durch die rechte Seite der Regel ersetzt werden

Chomsky Grammatik

- Quadrupel G = (V_n,V_t,S,R)
  - V_n ... Menge der Nicht-terminalsymbole
  - V_t ... Menge der Terminalsymbole
  - S ... (Menge der) Startsymbole
  - R ... Menge der (Produktions)regeln

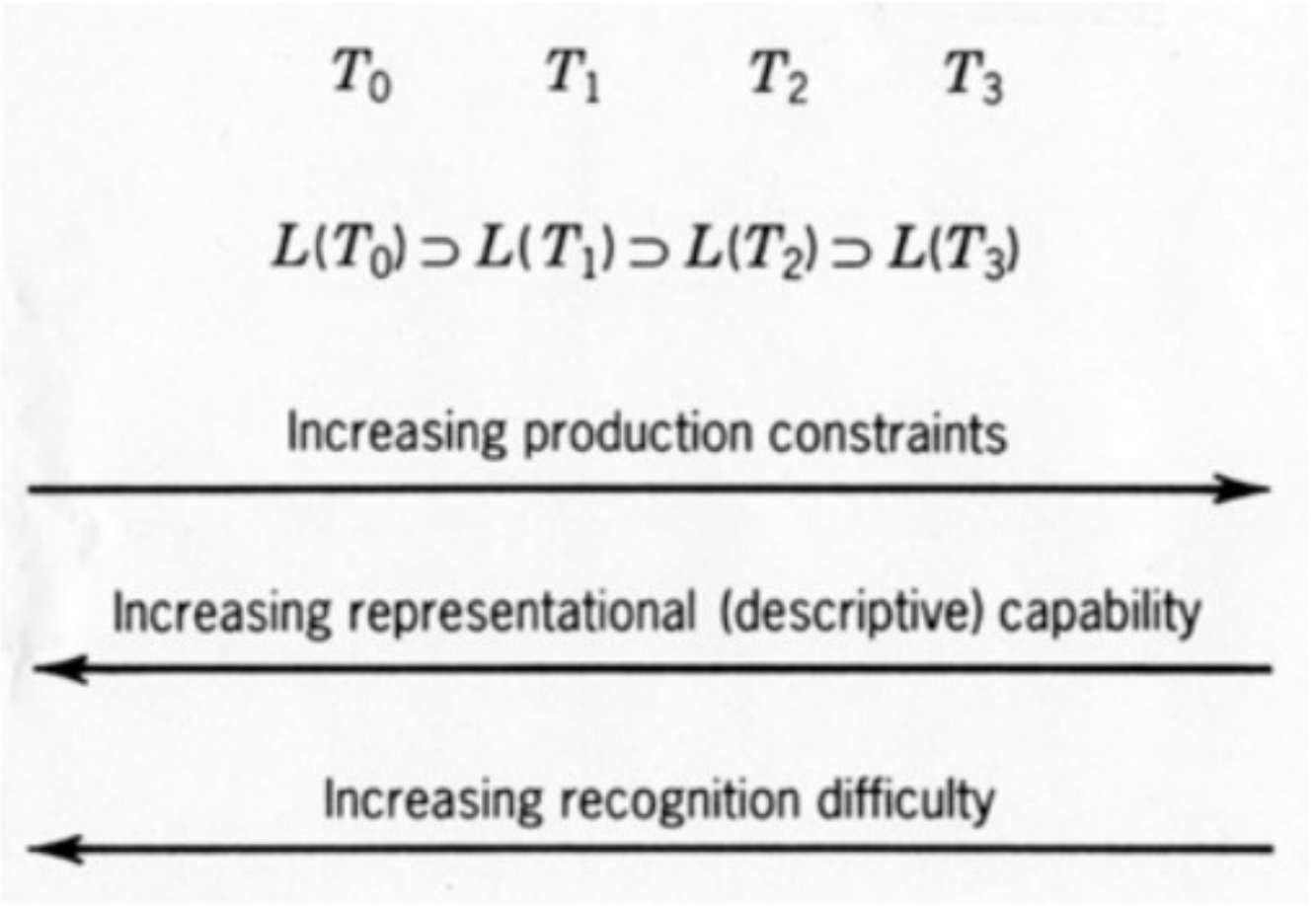
$V_n \cap V_t = \varnothing$

$V = V_n \cup V_t$

V* ... Menge aller erzeugbaren Symbolketten

$V^+ = V^* \setminus \varnothing$

Chomsky Hierarchie



modifizierte Grammatiken

durch weitere Einschränkungen auf Produktionsregeln

- **Programmierte** Grammatiken
  Anwendungsreihenfolge der Regeln ist vorgeschrieben
- **Stochastische** Grammatiken
  Jeder Regel wird eine Wahrscheinlichkeit zugeordnet. Jeder Satz erhält somit eine Auftrittswahrscheinlichkeit (Anwendung z.B. bei verrauschten/mehrdeutigen Primitiven)
- **Attributierte** Grammatiken
  Symbole erhalten zusätzliche Attribute. Regeln enthalten nicht nur Anweisungen zur Transformation von Symbolen, sondern auch zur Behandlung der Attribute (Anwendungsbeispiel: Armlänge bei Chromosomen)

Mehrdimensionale Grammatiken

- **Baum-Grammatiken**

  Regeln in der Form $b_i \rightarrow b_j$, wobei b_i, b_j Bäume sind, deren Blätter Terminal- oder Nonterminalsymbole sind.
  Dadurch ist die Verknüpfung von Symbolen nicht mehr nur auf linken und rechten Nachbarn beschränkt.
  (Anwendung: Klassifizierung von Fingerabdrücken)

- **Feld-Grammatiken**

  Regeln in der Form $F_i \rightarrow F_j$, wobei F_i, F_j 2-D Felder von Terminal- oder Nonterminalsymbole sind.

- **Graph-Grammatik** (allgemeinste Form)

  Regeln in der Form $G_i \rightarrow G_j$; E_i ,
  wobei G_k (Teil-)Graphen sind und E_i eine Einbettungsvorschrift beschreibt, die angibt, wie ein Teilgraph durch einen anderen zu ersetzen ist.

Erkennen/Klassifizieren von Symbolgraphen

Pattern Matching: Ähnlichkeit mit Prototypen

- Graph Matching
  - Sind zwei Graphen gleich?
  - Wie ähnlich sind sich zwei Graphen?
  - Wie stabil ist das Ähnlichkeitsmaß?

Grammatiklernen

Automatische Generierung einer Grammatik aufgrund einer gegebenen Stichprobe von Mustern
Ansätze:

- Grammar Induction:
  Inferenz mittels trial-and-error (hypothesis testing)
  Generierung von Regeln für positive Beispiele, Testen, Verwerfen von Regeln bei zu vielen negativen Beispielen
- Inferenz mittels Genetischer Algorithmen
  Kodierung der Grammatik als Genom

Grammar Induction

- Im Normalfall keine optimale Lösung vorhanden
- Mehrere Grammatiken für die meisten Sprachen möglich
- Suche nach der einfachsten Grammatik
  (Occam's razor)

Grammar Induction

- Vorbereitung:
  - Positive (D+) und negative (D-) Trainingsbeispiele festlegen
  - Grammatiktyp festlegen bzw. Kriterium für „einfache“ Grammatik festlegen
- Ziel:
  - Finde eine Grammatik, die alle Strings aus D+ (und ähnliche) parsed, aber keine aus D-

Grammar Induction

- Algorithmus:
  - Starte mit leeren Menge an Produktionsregeln
  - Wähle einzeln positive Trainingsbeispiele und teste, ob ein x_i^+ von der aktuellen Grammatik geparsed werden kann. Falls nicht:
  - Füge neue Produktionsregeln hinzu, die es ermöglichen dass x_i^+ geparsed wird, aber kein String aus D-

Grammar Induction - Beispiel

- D+ = {bbaab, caab, bbab, cab, bbb, cb}
- Festlegung auf Reguläre Grammatik, d.h.
  A → a und A → aB
- Mögliche Regeln:

S → bA
A → bB
B → b
B →aC
S → cD
C → b
C → aE
E → b
D → b
D → aF
F → aG
F → b
G → b

- Zusammenfassen von Regeln:

S → bA1 | cA2
A1 → bA2
A2 → aA2 | b

Factoring Finite Inductive Strings

- Zeichenketten mit:
  - Begrenztem Alphabet
  - Finiter Länge
  - Ist induktiv beschreibbar, d.h. spezifisches Auftreten von Symbolen und Teilstrings besitzt gewisse Regelmäßigkeiten

Beispiel: Faktorisierung

- Gegebene Zeichenkette: aacacgacgt
- Startsymbol anhängen: Saacacgacgt
- Definiere für jedes Zeichen eine eindeutige Struktur/Regel

  S → a
  Sa → a
  aa → c
  aac → a
  ca → c
  cac → g
  cacg → a
  ga → c
  gac → g
  gacg → t

Beispiel: Faktorisierung

- Beschränke Regeln/Präfix auf bestimmte Anzahl an Zeichen, z.B. 2, und streiche zu lange Regeln:
  S → a
  Sa → a
  aa → c
  aac → a
  ca → c
  cac → g
  cacg → a
  ga → c
  gac → g
  gacg → t
  (Level 0 Regeln)
- Saacacgacgt → Sagagt

Beispiel: Faktorisierung

- Faktorisiere reduzierten String: Sagagt
  S → a
  a → g
  Sag → a
  gag → t
  (Level 1 Regeln)
- Beschränke wieder auf 2 Zeichen, reduziere String auf: Sat
- Faktorisiere erneut:
  S → a
  a → t

  (Level 2 Regeln)

Beispiel: Faktorisierung

- Vergleich String: aacacgaccat
  Startsymbol anhängen und Level 0 Regeln anwenden
- S → a
  Sa → a
  aa → c
  ca → c
  ga → c
- Saacacgaccat → Saacacgaccat
- Reduktion auf Sagacat

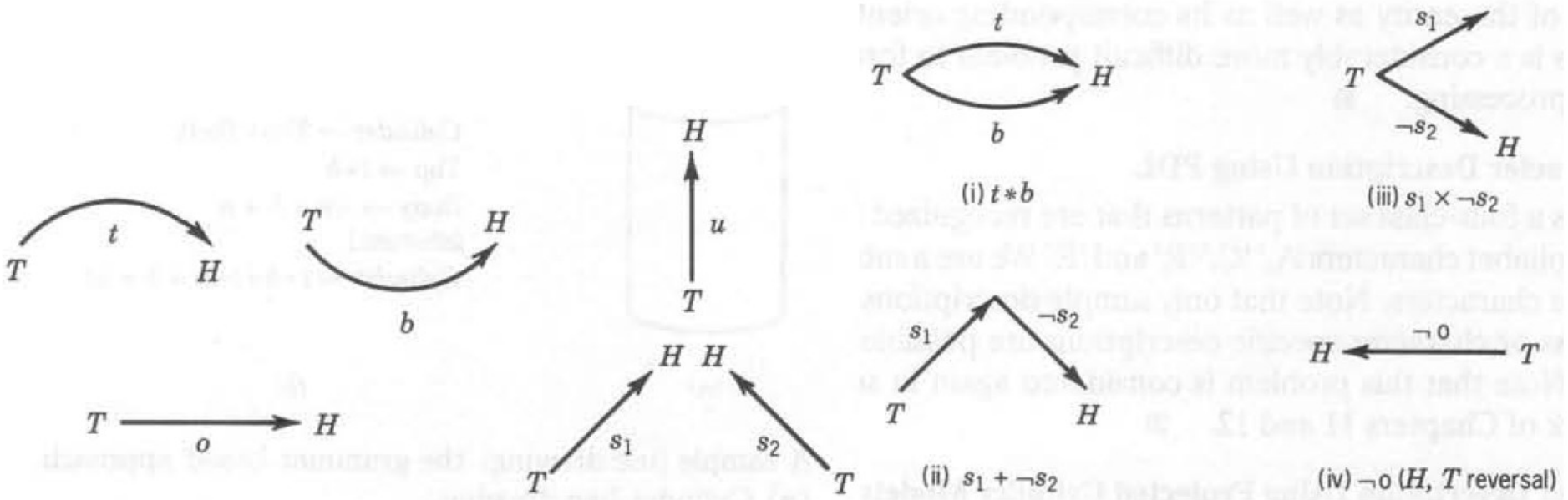
Beispiel: Faktorisierung

- Sagacat
  Anwenden der Level 1 Regeln
- S → a
  a → g
- Sagacat → Sagacat
- Reduktion auf Sacat
- Anwenden der Level 2 Regel (S → a, a → t)
- Nicht anwendbar, d.h. Residuum „ca“ nicht weiter erkennbar

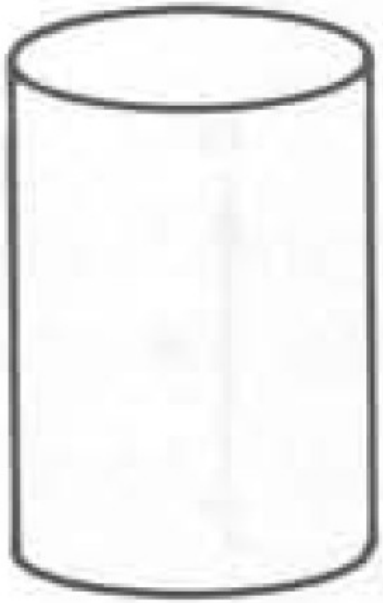
Fehlerbehandlung in der syntaktischen Mustererkennung

- Erfolg der syntaktischen ME hängt von der Qualität der Musterprimitiva ab. In der Praxis treten Fehler bei Vorverarbeitung und Extraktion auf.
  - Falsche Symbole, Überschüssige Symbole, Fehlende Symbole
- Methoden zur Fehlerkorrektur
  - Einbeziehung von fehlerhaften Mustern in die Musterbeschreibung
  - Fehlerfreie Musterbeschreibung, aber Maß für Übereinstimmung
  - Problem: nicht-disjunkte Grammatiken
    -> Verwendung von probabilistischen Grammatiken und A-Posteriori Klassifikation

Beispiel: Kontourbasierte Geometrien

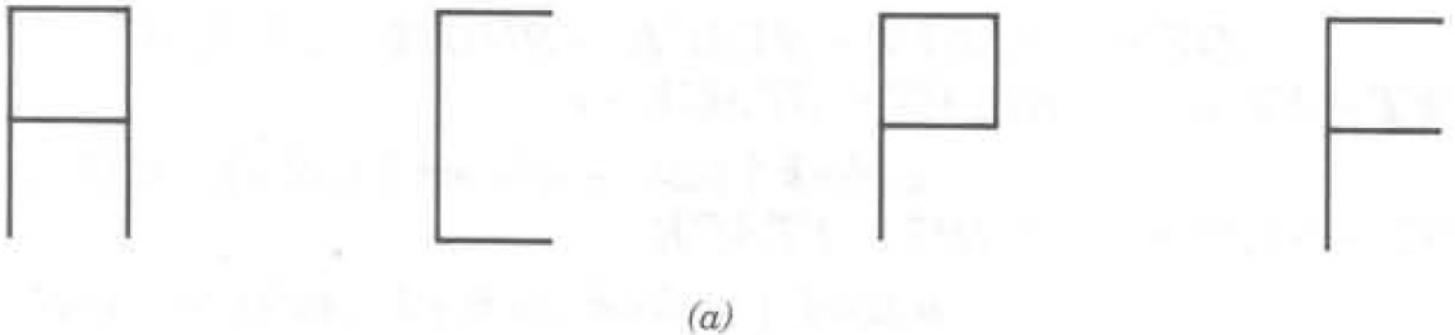


Beispiel: Kontourbasierte Geometrien



$$Cylinder \rightarrow Top * Body$$
$$Top \rightarrow t * b$$
$$Body \rightarrow \neg u + b + u$$
(alternate:)
$$Cylinder \rightarrow t * b * (\neg u + b + u)$$

Beispiel: Kontourbasierte Geometrien



(a)

(b)

$A = u + ((u + 0 + \neg u) \cdot 0) + \neg u$

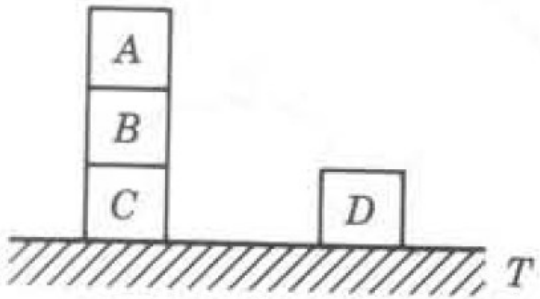
$C = \neg 0 + u + u + 0$

$P = u + ((u + 0 + \neg u) \cdot 0)$

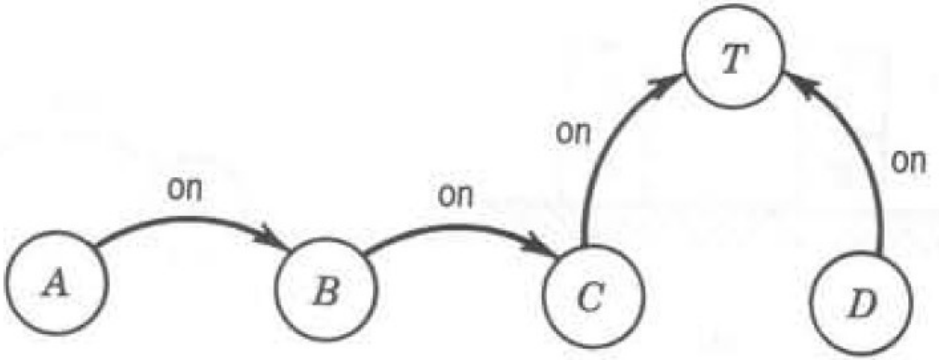
$F = u + (0 \times u) + 0$

(c)

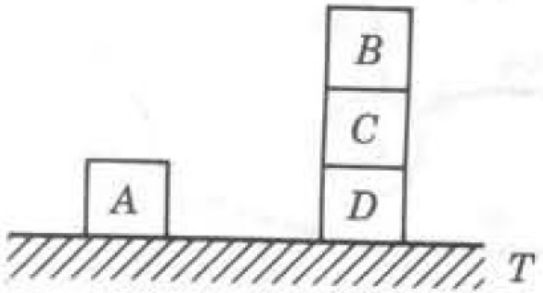
Logische Strukturen (Blocksworld)



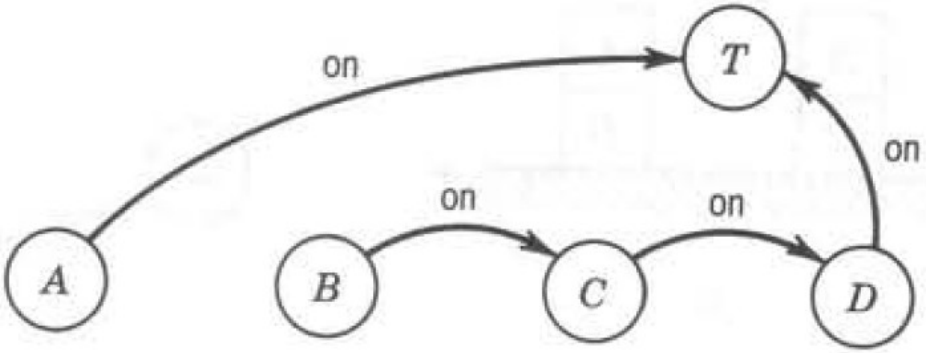
(i)



(ii)

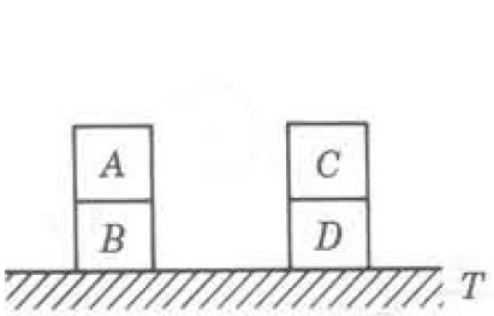


(iii)

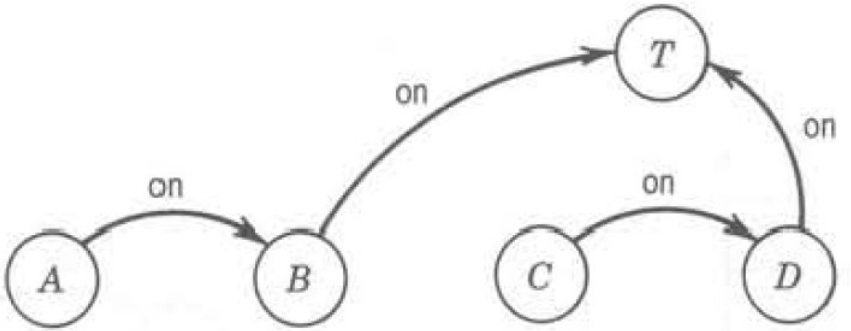


(iv)

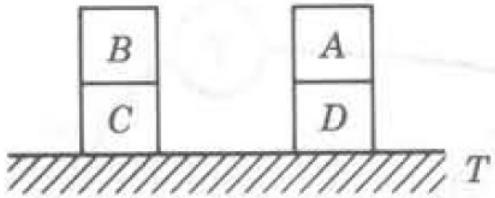
Logische Strukturen (Blocksworld)



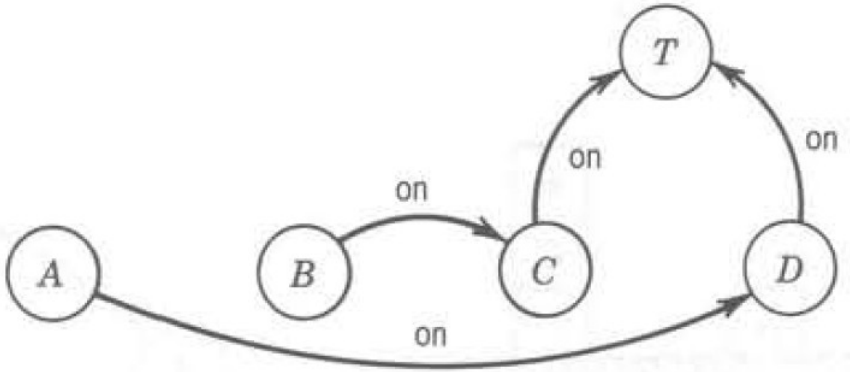
(i)



(ii)



(iii)



(iv)

Logische Strukturen (Blocksworld)

$$V_T = \{table, block, +, \uparrow\} \qquad \text{(terminal symbols)}$$

$$V_N = \{DESC, LEFT\_STACK, RIGHT\_STACK\}$$

(non-terminal symbols)

$$S = DESC \in V_N \qquad \text{(root symbol)}$$

$$P = \{DESC \rightarrow LEFT\_STACK + RIGHT\_STACK$$

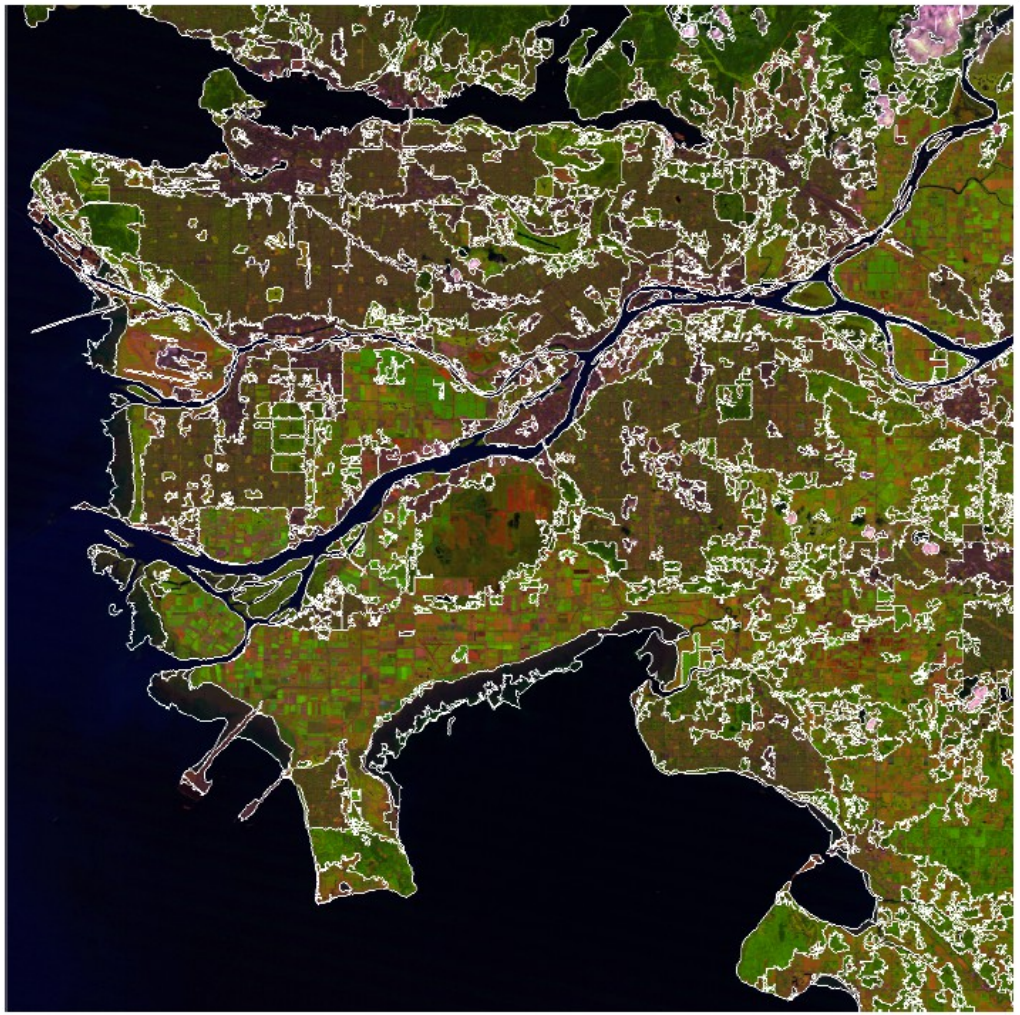
$$LEFT\_STACK + RIGHT\_STACK \rightarrow$$

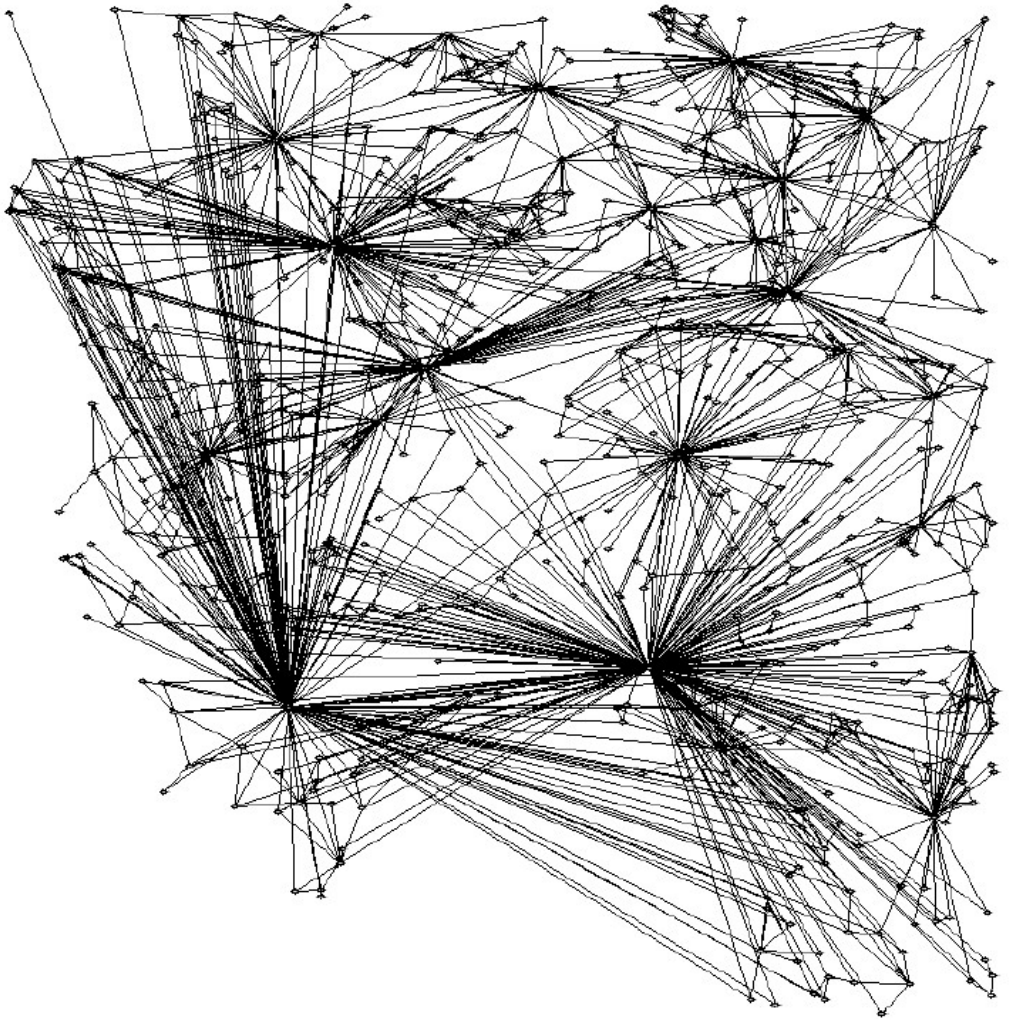
$$block \uparrow table + block \uparrow block \uparrow block \uparrow table$$

$$LEFT\_STACK + RIGHT\_STACK \rightarrow$$

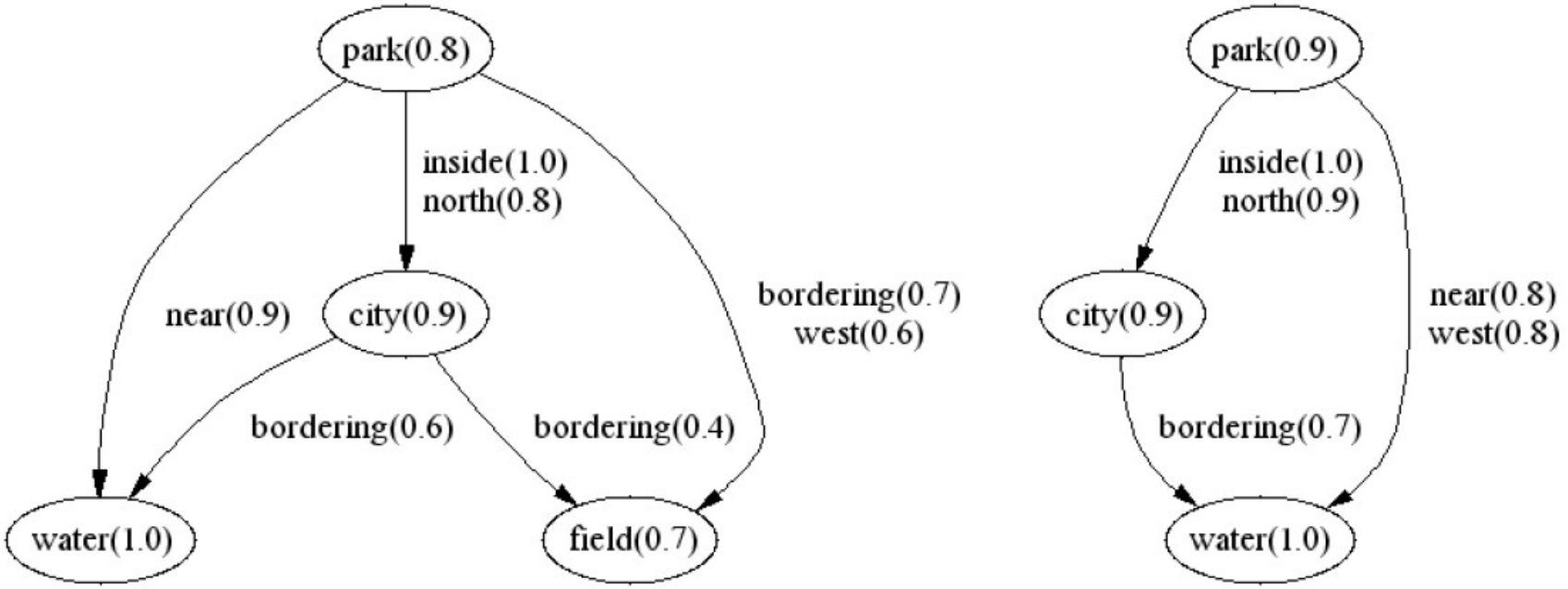
$$block \uparrow block \uparrow block \uparrow table + block \uparrow table\}$$

Beispiel: Maps





Beispiel: Maps



Ausblick

- Nächster Termin:

Donnerstag, 22.12.2016 13.00-15.00 (c.t.)

Übung 2