Inhaltsverzeichnis

Distanztransformation 2
Vorwärtslauf 2
Rückwärtslauf 2
Weitere Masken 2
Morphologie 3
Erosion 3
Dilatation 3
Co-ocurrence Matrix 3
Filteroperation 5
Randbehandlung 5
Glättungsfilter 6
Box/ Mittelwertfilter (Tiefpassfilter) 6
Binomialfilter (gewichtete Mittelung) 6
Medianfilter 6
Freeman Chain Code 7
Jordan´sches Kurventheorem 7
RAG und Split and Merge 8
RAG 8
Dualer Graph 8
Pyramide 9
Laplace Pyramide 9
Waveletpyramide 10
Region Growing 11
Integral Image und Predictive Code 12
B-Matix 12
I-Matrix 12
B-Matrix -1 +1 12
Array Grammatik 13
Beispiel: Rechtwinkeliges Dreieck, mit einem rechten Winkel in der linken unteren Ecke! 13
Beispiel: Aufgestelltes Quadrat! 13
Beispiel: Freeman Chain Code zum letzten Beispiel, also Umfang. 14
Beispiel: Rechteck mit Seitenverhältnis 1:2 15

Distanztransformation

Zu Beginn ist alles gefüllt mit 1 und unendlich (leeres Kastl).

Vorwärtslauf

Links oben wird gestartet. Das rote Feld im Raster ist das, das als erstes berechnet wird. Dabei wird der Nuller von der Maske über das neu zu berechnende Pixel gelegt. Der blaue Pixel vom Raster wird mit dem blauen Einser der Maske addiert. Dasselbe geschieht mit dem lila Feld und dem lila Einser.

Raster Bildecke

0	0	0
0		

Maske

	1
1	0

→ Summe blau = 1
→ Summe lila = 1
=> Die **kleinste** Summe ist 1, das neue Pixel in rot ist 1.

Achtung: blau und lila ändern nicht die Werte – die Summe werden nur für das neue Pixel benötigt!

=>

0	0	0
0	1	
0		

Rückwärtslauf

Genauso, nur von rechts unten starten. Wenn der Pixel vom roten Pixel schon kleiner ist als die Summe, die raus kommt, wird der kleinere, alte Pixel genommen.

Maske

0	1
1	

Wenn der Pixel vom roten schon kleiner ist, als die Summe, die raus kommt, wird der kleinere, alte Pixel genommen.

Weitere Masken

	d4	d8 = Schachbrettdistanz	d23
vorwärts			
	1	1 1 1	3 2 3
	1 0	1 0	2 0
rückwärts			
	0 1	0 1	0 2
	1	1 1 1	3 2 3

Morphologie

Morphologische Bildverarbeitung: durch Erosion und Dilatation => Öffnung, Schließung, morphologische Glättung. Weil nur 1&0 → was nicht 0 ist, ist 1.

S … Strukturelement, auf die erste Matrix anwenden → in 2. eintragen.

Erosion

Objekt wird kleiner. Passt S vollständig => Bezugspixel stehen lassen (gehört zur erodierten Menge), sonst Referenzpixel aus Region entfernen.

Dilatation

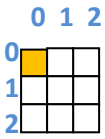
Strukturelement auf Region legen, schaun, ob Bezugspixel 0 oder 1 ist, also auf dem Objekt liegt, oder nicht.
0 => Nachbarschaft um S erweitern
1 => nicht erweitern

Co-ocurrence Matrix

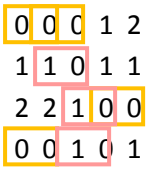
delta (1,0) = 1 Kasterl in x-Richtung, 0 in y-Richtung

In einer immer quadratischen Matrix wird eingetragen, wie oft die Ziffern in gegebener Stellung zu finden sind. Methode der Texturanalyse. Gibt an, wie oft in einem Bild 2 Grauwerte (2 Symbole, 2 Ziffern,…) in bestimmter Anordnung (delta) in Nachbarschaft auftreten.

Beispiel:

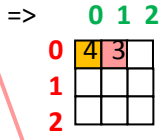


0, 1, 2 ist hier kein Index, sondern die Grauwerte.
C (1, 0) heißt, dass die Anzahl der Kombinationen von 2 Grauwerten horizontal nebeneinander



Es gibt 3 Grauwerte (0, 1, 2) → 3x3 Co-occurence Matrix
=> berechne die drei C (k, l)
k = Abstand horizontal
l = Abstand vertikal

1. Wir suchen alle 0-0 Paare, die horizontal nebeneinander liegen.



0, 1, 2 … der linke Grauwert
0, 1, 2 … der rechte Grauwert

2. Wir suchen alle 1-0 Paare, die horizontal nebeneinander liegen.
usw.

Dasselbe geht auch mit vertikalen Paaren mit k=0, l=1

Filteroperation

Mit Faltung → Filter möglich (Hoch- und Tiefpassfilter)

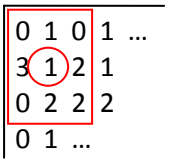
Lineare Faltung berechnet Werte des Ausgangsbildes aus rechteckiger Umgebung des Eingangsbildes. Zeilen und Spalten der Matrix sind in der Bildverarbeitung immer kleiner als das Eingangsbildes.

Anwendung: für jeden Pixel Bestimmung des neuen Wertes als Summe der Produkte der Faltungsmatrix (Maske) im Bild.

Bsp:

Eingabebild B		Faltungsmatrix F		Ausgabebild
0 1 0 1 1 3 1 2 1 1 0 2 2 2 0 0 1 2 0 0 4 0 0 0 0	*	0 1 0 1 1 1 0 1 0	=	9 6 7 6 10 5 5 5 4

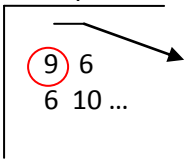
Vorgehen:



Pixel ◯ der Multiplikation mit Faltungsmatrix berechnen

=> *F

0*0 + 1*1 + 0*0 + 3*1 + 1*1 + 2*1 + 0*0 + 2*1 + 2*0 = 9



Manchmal wird das Bild dadurch kleiner, es gibt aber auch andere Möglichkeiten

Randbehandlung

- Ignorieren => Bild wird dann mit jeder Faltung kleiner
- Auffüllen mit Nullen => starke Randeffekte
- Auffüllen mit Randwerten (Neumannsche Randbedingung, 1.Ableitung = 0)
- Zirkuläre Fortsetzung (Bild als periodische Struktur, wenns rechts aus ist, einfach links weiterlesen, selbiges mit oben und unten)

Glättungsfilter

um Störungen zu reduzieren, Kanten verwischen, Bild wird unscharf

Box/ Mittelwertfilter (Tiefpassfilter)

3x3 Matrix => Faltungsmatrix

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

5x5 Matrix => lauter 1/25

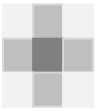
Verstärkt tiefe Ortsfrequenzen, unterdrückt hohe → kleine Grauwertspitzen

Binomialfilter (gewichtete Mittelung)

Filterwerte aus Binomialkoeffizienten

3x3:

1	2	1
2	4	2
1	2	1



5x5:

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1

← innen:

1 4 6 4
4 ↓
6 → 4*6

3 →
1
1 1
1 2 1
1 3 3 1

5 →
1 4 6 4 1

Medianfilter

Jedes Pixel ersetzt durch Median der Nachbarschaft. Zuerst der Größe nach ordnen, dann die Mitte der Reihe wählen. Rechnung: n*m/2, bei einem geraden Zähler +1

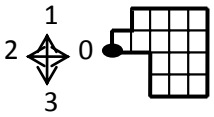
=> Ausreißer haben keinen Einfluss
=> keine neuen Grauwerte
=> keine Kantenglättung, obwohl Störungen weg sind
=> schlecht bei Schrift, weil dünne, gerade Linien schnell weg sind

Freeman Chain Code

Bild muss aus Chaincode erstellt werden. Raster und Startpunkt sind vorgegeben.

Zur Beschreibung der Konturen/ des Randes einer Region. Eine Folge von Punkten in 8er-Nachbarschaft kann durch Angabe des Startpixels, gefolgt von den Codes, die den Ort des folgenden Pixels relativ zum vorhergehenden Pixel angeben.

Crack-Codes

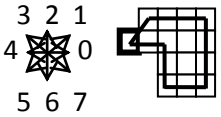


1, 0, 1, 0, 0, 0, 3, 3, 3,

3, 2, 2, 2, 1, 1, 2, 2

Zeigt nicht auf Pixel, sondern gibt die Richtung der äußeren Kante der Randpixel an.

Freeman

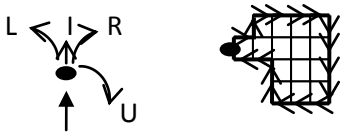


1, 0, 0, 0, 6, 6, 6,

4, 4, 2, 2, 4, 4

Bei folgendem Code wird mittels einer Zahl von 0 bis 7 angegeben, wo der nächste Pixel liegt.

Chain Ruli Chain Code



RLRIIIRIIIRIIRILIR

Rand der äußeren Kanten der Randpixel, immer in Blickrichtung, wie wenn man an der Pfeilspitze sitzt und immer geradeaus schaut.

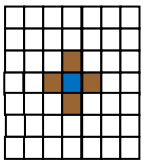
Ein **Weg** ist eine Folge benachbarter Pixel. Wenn alle Pixel durch einen Weg verbunden sind, ist die Region **zusammenhängend**.

4-zusammenhängend: Pixel mit einer gemeinsamen Kante 

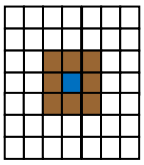
8-zusammenhängend: Pixel mit einem gemeinsamen Punkt 

=> Zn(R) ist die Zusammenhangskomponente Z einer Region R mit der Nachbarschaft N (4 oder 8)

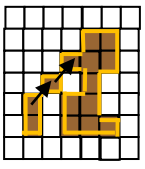
4er-Nachbarschaft
Pixel mit gemeinsamen Kanten



8er-Nachbarschaft
Pixel mit gemeinsamen Kanten **und** Ecken



Zusammenhanfkomponenten



Jordan´sches Kurventheorem

Geschlossene Kurve erzeugt kleine Mengen (Zusammenhangskomponenten) → alles innerhalb der Kurve = Objekt

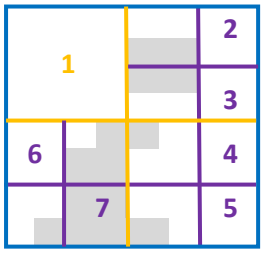
Graphik s. Skript

Wohlgeformt? Wenn in Freeman Chaincode eine 45 Grad Schräge Kante (Schachbrettmuster) vorkommt, dann ist das Bild NICHT wohlgeformt.

RAG und Split and Merge

1. Startregion ist das ganze Bild
2. Split: Zerlege ALLE inhomogenen Regionen in Teilregionen
3. Merge: Vereinige benachbarte Regionen, wenn sie ähnlich/gleich sind
4. Iteriere Split und Merge bis keine Änderung mehr auftritt

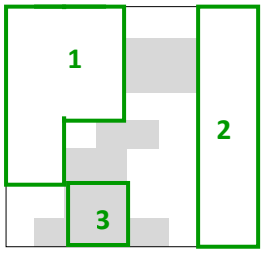
Homogen ist eine Region, wenn alle Pixel denselben Wert haben. Alle schwarz, alle weiß,…



1. Ist die Reihe inhomogen? Nein, also splitten.
2. Splitten der Region in 4 Teilregionen.

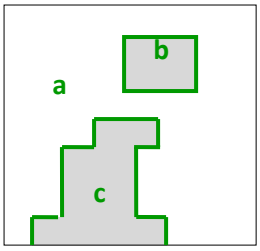
Split 1 Region in 4 Teilregionen: 1 → 4

3. Sind Teilregionen homogen? Ja, eine (links oben). 3 weitere Teilregionen aber nicht. Kann man die homogene mit einer anderen mergen? Nein, weil es keine zweite ähnliche gibt.
4. Teile die übrigen 3 Teilregionen in je 4 Teile



Split 3 Regionen in je 4 Teilregionen: 3 → 12
5. Sind die Teilregionen homogen? Ja, jetzt 7 insgesamt. Kann man diese mit benachbarten mergen? Ja.

Merge 1 mit 6; 2 mit 3, 4, 5: von 7 homogenen Regionen → 3



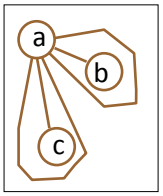
Über bleiben 3 Regionen: **a, b und c**

„Region adjacency“ = „angrenzende Region“. Wenn 2 Regionen durch 1 oder mehrere Pixel verbunden sind, sind sie angrenzend.

RAG

Durch RAG kann man angeben, welche Regionen angrenzend sind. Regionen entsprechen Knoten, Kanten dazu geben an, welche Regionen zueinander adjazent (also angrenzend) sind.

zu oben:



Kanten a-b, a-c, je diese 2 Regionen sind benachbart.
b-c nicht, haben keine angrenzenden Pixel

Dualer Graph

Für jede gebildete Fläche wird ein Knoten angenommen. Kanten werden so gezogen, dass immer Flächen, die sich in mindestens einer Pixelkante berühren, benachbart sind.

Pyramide

in Ebenen aufgebaute Pyramiden.

Man fasst aus untersten 4 Pixel zusammen, berechnet Durchschnitt => nächsthöhere Ebene, die auf ¼ der vorhergehenden Ebene reduziert ist, den Wert des „zusammengefassten“ Pixels.

Bsp:

Ausgangsmatrix

3 5 3 1	3 4 4 1	5 1 2 8	4 1 2 9
5 1 2 8	3 5 3 1	1 1 1 1	3 1 4 8
6 5 1 4	2 1 3 2	1 3 3 1	1 1 1 5
2 2 1 3	1 1 1 1	1 4 2 1	3 3 3 3

(3+5+3+1)/4 = 3
Die 3 in der nächsten Ebene links oben eintragen usw.

3 3 4 3	4 4 1 4
4 2 2 1	2 2 2 3

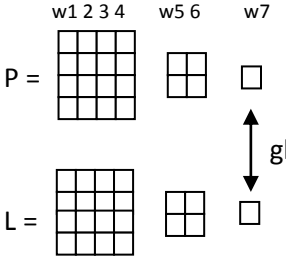
3,25 3,25 2,25 2,25

2,75

Laplace Pyramide

Wi = min {Mi, |6-Mi|} → Es werden die einzelnen Ziffern der Matrikelnummer von 6 abgezogen. Dann werden diese verglichen mit den eigentlichen Ziffern der Matrikelnummer und der kleinere Wert davon genommen. Dadruch liegen alle Werte von Wi zwischen 0 und 2.

Zuerst trag in die gekennzeichneten Pixel der noch leeren Matrizen die berechneten Werte ein. Die anderen Pixel musst du berechnen. Ein Pixel wird berechnet, indem man von den 4 Pixel aus der Region, die dieses Pixel bilden, den Durchschnitt berechnet. Also: | a b
c d | → (a+b+c+d)/4



w7 aufblasen: 2 → | 2 2
2 2 | Die dann minus der w5/w6 Matrix von P abgezogen ergibt die 2x2 Matrix von L

Dasselbe passiert mit der 4x4 Matrix: Zuerst wird die w5/w6 Matrix von P aufgeblasen und von der die w1/w2/w3/w4 (4x4 Matrix von P) abgezogen. Das ergibt dann die 4x4-Matrix von L.

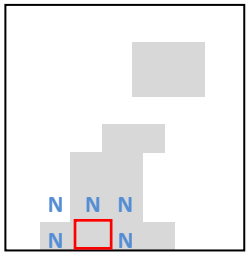
Achtung: Meist sind zu Beginn bestimmte Pixel vorgegeben, welche Werte die haben müssen. Die dürfen dann natürlich nicht verändert werden.

Waveletpyramide

1. In I_0 wie verlangt Ziffern eingtragen.
2. Immer von 2 Werte nebeneinander den Mittelwert berechnen.
3. Die Differenzen ausrechnen von den Werten, die diesen Mittelwert berechnet haben, zu eben diesem. Und zwar erster der beiden Werte minus dem Mittelwert, oder Mittelwert minus des zweiten der beiden, die den Mittelwert gebildet haben.
4. In der nächsten Zeile dasselbe berechnen mit der Hälfte, die durch die Mittelwerte entstanden sind. Dadurch wird die Wurscht in jedem Schritt halbiert.

Achtung: Die Mittelwerte im nächsten I in die Hälfte eintragen, unter der I_0*H steht, die Differenzwerte auf die Hälfte, unter der I_0*G steht.

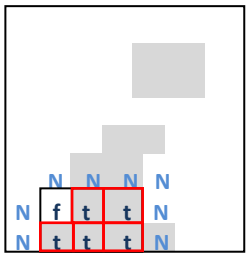
Region Growing



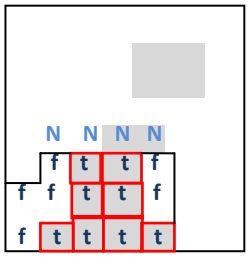
Saatpixel/ Saatregion → Start

→ markiere alle Nachbarpixel der Saatregion → **N**

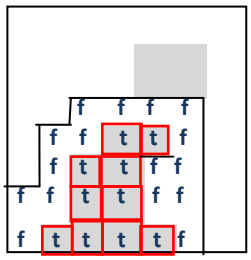
alle Nachbarpixel, die denselben Wert (0 oder 1) wie das Saatpixel haben, sind ebenfalls Teil der Region und werden der Region hinzugefügt.



f = false, das Pixel wird nicht mehr näher betrachtet, da es nicht zur Region hinzugefügt worden ist.
t = zur Region zugehörig.
markierte Nachbarpixel → **N**



Vergleiche neu markierte N mit den Regionspixeln und nimm die Nachbarpixel mit denselben Werten zur Region hinzu



usw. bis zum Endergebnis!

Integral Image und Predictive Code

B-Matix

1er einfüllen, so wie es die Angabe vorschreibt

I-Matrix

Es wird immer ein Viereck von der Ecke links oben gezogen bis inklusive des Pixel, das man ausrechnen will. Alles, was drinnen liegt, wird addiert und in das Pixel im rechten unteren Pixel des aufgezogenen Vierecks geschrieben. Dadurch wird die I-Matrix nach unten rechts hin von den Werten her ansteigend. Wenn der Rand nicht anders vorgegeben ist, am besten einen 0-Abschluss wählen (geht am schnellsten).

Bsp:

B-Matrix

Integral Image soll mit einem linearen Prädiktor mit den Koeffizienten „überzogen“ werden (Der Prädiktor wird wie eine Maske einer Faltung angewendet). Die drei Pixel, die nicht *B* sind, multipliziert man mit denen, die in der B-Matrix darunter liegen und addiert dann die Ergebnisse. Was raus kommt, schreibt man in das Pixel, das an der Stelle des *B* liegt.

Danach werden mittels Lauflängencodierung die Spalten aufgeschrieben.

Achtung: Es wird die Differenz der Werte der B-Matrix zur I-Matrix verlangt!
Achtung: Einfacher ist es, wenn in der B-Matrix gleich die Pixel markiert werden, die zur I-Matrix anders sind. Es geht dann schneller, den Lauflängencode zu schreiben.
Achtung: Hinweis darauf aufschreiben, dass wir erst ein Flag (eigentlich !) gesetzt wird und Sinn macht, wenn es mehr als 3 Zeichen in Folge sind.

Array Grammatik

Bei der Zuweisung einer Grammatik zu einem Hilbert-Scan empfiehlt es sich, rechts neben jede Grammatikzeile das Zeichen zu zeichnen, das sich daraus ergibt.

Definition

G = (N, T, #, P, {(v0, S)})

N … Nichtterminale/Metasymbole
T … Terminale (1er)
… leeres Pixel, Null, halt noch leeres Blatt
P … Produktionsmenge (Regeln)
v0 … Startvektor
S … Startvektor

Beispiel: **Rechtwinkeliges Dreieck, mit einem rechten Winkel in der linken unteren Ecke!**

G = ({S, 0}, {1}, #, P, S)

P: **p1:** S # → 1 0 **p2:** S → 1 **p3:** 0 1 / # → # **p4:** 0 / 1 # → 1 0

Ableitung:

			S		S		**S**		S		**S**	
p1	**S**	p1	1 0	p3	1 **0**	p4	1 1	p1	1 **0**	p4	1 1	p2
S →	1 0	→	1 **0**	→	1 **1**	→	1 1 0	→	1 **1**	→	1 1 0	→

1		1		1		1
1 1		1 1		1 1		1 1
1 1 0	p3	1 1 **0**	p4	1 1 1	p3	1 1 1
1 1 **0**	→	1 1 **1**	→	1 1 1 **0**	→	1 1 1 1

Beispiel: **Aufgestelltes Quadrat:**

G = ({S, 0}, {1}, #, P, S)

P: **p1:** S → 0 / 1 1 1 / 1 **p2:** 0 → 0 / 1 / 1 / 1 **p3:** # 1 # / # 1 # → 1 1 1 / 1 1 1 1 1

p4: # / 1 # → 1 1 1 / 1 **p5:** 1 / 1 # → 1 1 1 / 1 **p6:** 0 → 1

Ableitung:

S p1→ **0** / 1 1 1 / 1

p2→ 0 / 1 / 1 / 1 1 1 / 1

p3→ 0 / 1 / 1 / 1 1 1 / 1 1 1 1 1 / 1 1 1 / 1

p2→ 0 / 1 / 1 / 1 1 1 / 1 1 1 1 1 / 1 1 1 / 1

p3→ 0 / 1 / 1 / 1 1 1 / 1 1 1 1 1 / 1 1 1 / 1 1 1 1 1 / 1 1 1 / 1

p4→ 0 / 1 / 1 / 1 1 1 / 1 1 1 1 1 / 1 1 1 1 1 / 1 1 1 1 1 / 1 1 1 / 1

p5→ 0 / 1 / 1 / 1 1 1 / 1 1 1 1 1 / 1 1 1 1 1 1 1 / 1 1 1 1 1 / 1 1 1 / 1

p2→ 0 / 1 / 1 / 1 1 1 / 1 1 1 1 1 / 1 1 1 1 1 1 1 / 1 1 1 1 1 / 1 1 1 / 1

p3→ 0 / 1 / 1 / 1 1 1 / 1 1 1 1 1 / 1 1 1 / 1 1 1 1 1 / 1 1 1 / 1

p4→ 0 / 1 / 1 / 1 1 1 / 1 1 1 1 1 / 1 1 1 1 1 / 1 1 1 1 1 1 1 / 1 1 1 1 1 / 1 1 1 / 1

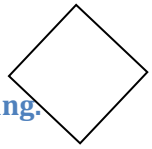
p4→ 0 / 1 / 1 / 1 1 1 / 1 1 1 1 1 / 1 1 1 1 1 / 1 1 1 1 1 1 1 / 1 1 1 1 1 / 1 1 1 / 1

p5→ 0 / 1 / 1 / 1 1 1 / 1 1 1 1 1 / 1 1 1 1 1 / 1 1 1 1 1 1 1 / 1 1 1 1 1 / 1 1 1 / 1

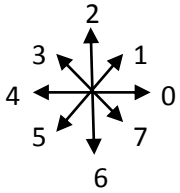
p5→ 0 / 1 / 1 / 1 1 1 / 1 1 1 1 1 / 1 1 1 1 1 / 1 1 1 1 1 1 1 / 1 1 1 1 1 / 1 1 1 / 1

p5→ 0 / 1 / 1 / 1 1 1 / 1 1 1 1 1 / 1 1 1 1 1 / 1 1 1 1 1 1 1 / 1 1 1 1 1 / 1 1 1 / 1

p6→ 1 / 1 1 1 / 1 1 1 1 1 / 1 1 1 1 1 1 1 / 1 1 1 1 1 / 1 1 1 / 1



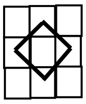
Beispiel: **Freeman Chain Code zum letzten Beispiel, also Umfang.**

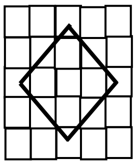


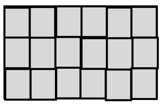
G = ({A, B, C, D, E, R, S, T}, {1, 3, 5, 7}, #, P, S)

P:	**p1:** S → TRE	**p2:** R → RR	**p3:** TE → #	**p4:** R → ABCD	**p5:** BA → AB
	P6: CA → AC	**p7:** DA → AD	**p8:** CB → BC	**p9:** DB → BD	**p10:** DC → CD
	P11: AE → E1	**p12:** BE → E3	**p13:** CE → E5	**p14:** DE → E7	

Ableitung:

	p1		p4		p14		p13		p12	
S	→	TRE	→	TABCDE	→	TABCE7	→	TABE57	→	
	p11		p3							
TAE357	→	TE1357	→	#1357	=>					

	p1		p2		p4		p7		p9	
S	→	TRE	→	TRRE	→	TABC**DA**BCDE	→	TABCA**DB**CDE	→	TABCAB**DC**DE
	p10		p6		p8		p5		p14	
	→	TAB**CA**BCDDE	→	TABA**CB**CDDE	→	TA**BA**BCCDDE	→	TAABBCCD**DE**	→	
			p14		p13/12/11/3					
		TAABBCC**DE**7	→	**TAABBCCE**77	→	#11335577	=>			

Beispiel: **Rechteck mit Seitenverhältnis** 

G = ({S, 0}, {1}, #, P, S)

P:	**p1:** S → 0 0	**p2:** 0 → 0 1 1 1	**p3:** 1 1 # 1 → 1 1	**p4:** 1 # 1 1 1 → 1	**p5:** 0 → 1

	p1		p2		p2		p3	
S	→	0 0	→	0 1 0 1 1 1 1 1	→	0 1 0 1 1 1 1 1 1 1 1 1 1 1	→	0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
	p4							
	→							

	p2		p3		p3	
0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1	→	0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	→	0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	→	0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
	p4					
	→					

	p4		p5	
0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	→	0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	→	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1