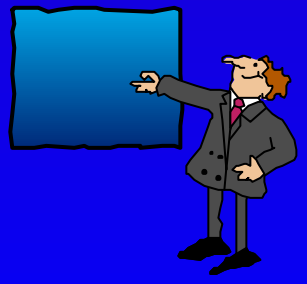
A formal Framework for positive and negative detection schemes

Vortrag von Michael Schmitt

Seminar Künstliche Immunsysteme



Agenda

- Worum geht es?
- Biologischer Hintergrund
- Technische Anwendung
  - Begriffsklärung, bisherige Systeme, Probleme
- Wie funktioniert "anomaly detection"?
  - Ähnlichkeitsmaße
  - Erzeugungswahrscheinlichkeit
  - Erzeugungs-/Erkennungsmethoden/-schemata
  - Probleme und mögliche Lösungen
  - Effizienzsteigerung
- Fazit
- Offene Probleme

Worum geht es?

- Anomalie-Erkennung in technischen Systemen
- Abweichungen vom Norm-Verhalten eines Systems
  - Welche Vorgänge sind erlaubt?
  - Wie können Eindringlinge erkannt werden?
- Verschiedene Erkennungsansätze
  - Effizienzuntersuchung

Biologischer Hintergrund

- Ziel: Vermeidung von Auto-Immunität durch Körperabwehr
- Vorbild Thymusdrüse
  - Isoliertes Heranwachsen von T-Zellen
  - T-Zellen werden nur mit körpereigenen Zellen konfrontiert
  - Bei Immunreaktionen wird die Zelle zerstört
  - Nur die überlebenden ausgewachsene Zellen werden in den Körper entlassen
- Im Körper reagieren die T-Zellen dann nur auf körperfremde Stoffe (nonself)

Technische Anwendung

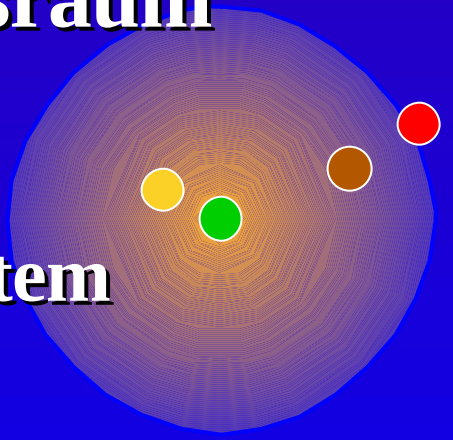
- Erkennung von Eindringlingen in ein technisches System
- Antiviren-Software
- Netzwerkangriffe (TCP, UDP, ICMP)
- Unerlaubte Benutzer bzw. -aktionen in Computersystemen

Grundlegende Begriffsklärung

- *U:* Universum aller möglichen Zeichenketten mit Länge *l*
- *s:* Eine Zeichenkette aus U (mit Länge l)
- *RS:* Menge aller s in "self"
- *S*: Auswahl von Beispielen aus RS
- *U-RS:* Menge aller Elemente im "non-self"
- *r:* Grad der Näherung an S
- *Löcher:* Nicht erkennbare Zeichenketten in self bzw. non-self
- *xMd:* x wird von d erkannt (match)
- *O.B.d.A. Betrachtung von Bit-Strings*

Bisherige Systeme

- Erkennung von Sonderfällen durch Wahrscheinlichkeiten
  - Verdacht ~ 1 / Häufigkeit in der Vergangenheit
- Erkennung durch Abstand im Ereignisraum
  - L-Norm
  - Hamming-Abstand/Manhattan-Abstand
  - Verdacht ~ Kleinster Abstand zu bekanntem Ereignis
- Kombination verschiedener Ansätze
- Grundlage ist die Korrelation von Sicherheitsverletzungen und Anomalien



Bekannte Probleme

- Kleine Beispielmenge S
  - Wie werden Objekte aufgrund gegebener Beispiele eingeordnet?
- Ähnlichkeiten müssen sinnvoll definiert sein
  - Ansonsten bedeutungslose Klassifikationen
- Partial Matching kann zu Löchern in der Erkennung führen

Wie funktioniert anomaly detection?

- Normverhalten basiert auf Modell
- Modell wird aus Beispielen (samples) erschlossen ("Konzept-Lernen")
  - Problem führt oft zu Übergeneralisierung
  - In der Praxis häufig nicht lösbar
- Positive oder negative Erkennung möglich
- Bisherige Formen beschränkt auf kleine, genau bekannte und gleichbleibende Probleme
- In anderen Fällen Herausforderung, evtl. auch für negative Erkennung

Ähnlichkeitsmaße

- Kleinster Hamming-Abstand von sample(t)
  - Keine Unterscheidung von Elementen in sample(t)
- 1 / (# Instanzen von p_k in sample(t))
  - 1 / 0 wird auf große Konstante C festgelegt
  - Keine Unterscheidung von Elementen nicht aus sample(t)
- Hybride Methode
- Erzeugungsregel *ähnlich zu sample(t)* bzw. *unähnlich zu sample(t)*
  - Effektivere Erkennung

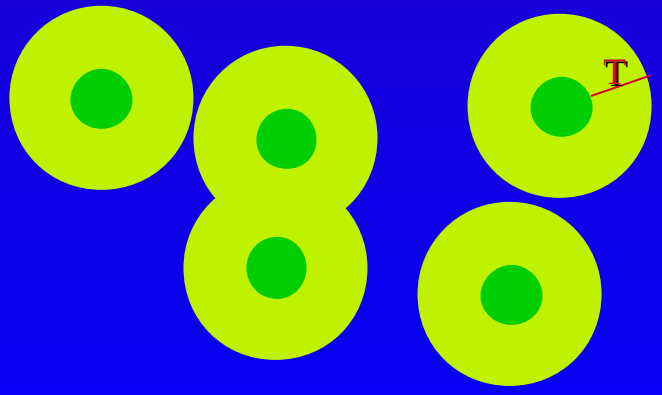
Erzeugungswahrscheinlichkeit
durch Prozeß G

- **Erzeugungsregel Q(S)**
  - **Erzeugungswahrscheinlichkeit allg.:** $p = 1/Q(S)$
  - **Elemente in Q(sample(*t*)) wahrscheinlicher durch G erzeugt als Elemente außerhalb**
  - **Multiplikation mit Faktor f**
    - $p'_1 = f / |Q(sample(t))|, \quad x(t) \in Q(sample(t))$
    - $p'_2 = 1 / |U - Q(sample(t))|, \quad x(t) \notin Q(sample(t))$
- **Allgemeine Regel**
  - $Pr\{x(t) \mid y(t) = 0, sample(t), t\} =$
    p_1, wenn $x(t) \in Q(sample(t))$
    p_2, wenn $x(t) \notin Q(sample(t))$
  - $0 \leq p_2 < p_1 \leq 1, \; p_1 + p_2 = 1$

Erzeugungs-/Erkennungsmethoden

Hamming-Abstand

- Kategoriemaß mit Grenzwert T
  - Hier ist $T = l - r$
- $Hd_T(S)$ ist Menge von Strings in einem Abstand von maximal T
  - xMd « $Hd(x, d) \pounds l - r$
  - x wird von d erkannt, wenn es sich in höchstens $l - r$ Positionen von d unterscheidet

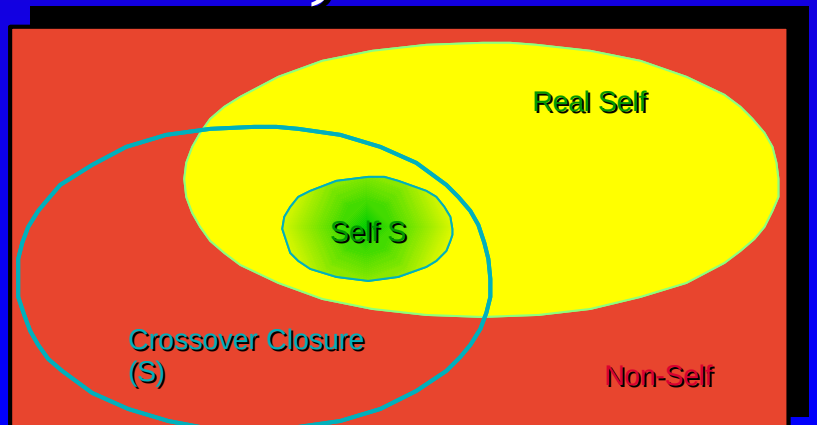


Erzeugungs-/Erkennungsmethoden
n-Gramme

- Trace eines gültigen Programmablaufs im Code speichern
- Bei erneuter Ausführung in jedem Programmschritt vergleichen
  - Anomalie, wenn Programmzustand nicht im Trace
- Speicherung als DAG möglich
- Gültiges Programm ist $CC^*_n(S) = \cup_{l \hat{\imath} L} CC^l_n(S)$

Erzeugungs-/Erkennungsmethoden
Crossover Closure (CC)

- **Verallgemeinerung einer Beispielmenge S**
  - **Erzeugung aller möglichen Kombinationen der Merkmale der Elemente von S**
  - ***CC(S)*** = {*u* Î *U*| " **Merkmale** *w*. *s* Î *S* .*u*[*w*] = *s*[*w*]}
- **S ist *abgeschlossen*, wenn CC(S) = S**



Fahrzeuge

Räder	Farbe	Geschw.
4	rot	100
2	schwarz	200

Nach CC...

Räder	Farbe	Geschw.
4	rot	100
4	rot	200
4	schwarz	100
4	schwarz	200
2	rot	100
2	rot	200
2	schwarz	100
2	schwarz	200

Erkennungsmethoden

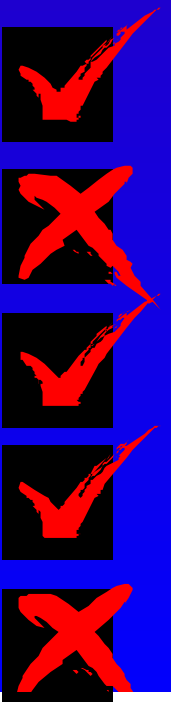
r-consecutive bit detection

- rcb-Detektor *d* ist String der Länge *l*
- *d* erkennt einen anderen String, wenn dieser mindestens *r* gleiche Zeichen (Bits) hat
  - Einschränkung auf benachbarte Bits
  - *dMx* « $ Fenster *w*. *d*[*w*] = *x*[*w*]

Detektor (*l* = 5, *r* = 3)

1	1	0	0	1

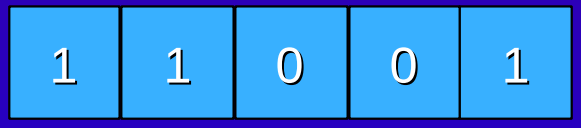
1	1	0	1	1
0	0	0	1	1
0	1	0	0	1
0	0	0	0	1
1	0	0	1	1



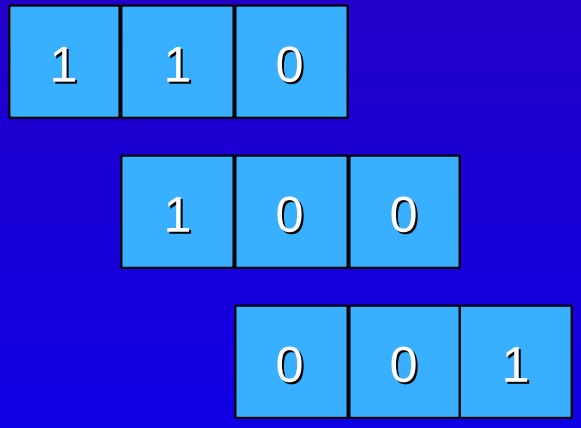
Erkennungsmethoden
r-chunk detection

- **Vereinfachung von rcb**
  - **Beschränkung auf Teil-Strings**
  - **Genaue Übereinstimmung**
- **Detektoren werden für Fenster *w* definiert**
  - ***dMx « x[w] = d***
- **Beispiel**
  - **String-Länge *l* = 5**
  - ***r* = 3**

Zu erkennender String



r-chunks



4 Erkennungsschemata
Positive vs. negative Erkennung

- **Positive Erkennung**
  - ✓ Erkennen von Elementen des "self"
  - ✓ Self i.A. kleiner als Non-Self
  - ✓ Genauer als negative Erkennung
  - ✗ Höherer Berechnungsaufwand
- **Negative Erkennung**
  - ✓ Erkennen von "non-self"-Elementen
  - ✗ Non-Self meist größer
  - ✓ Mittels genetischem Algorithmus
  - ✓ Verteilte Erkennung möglich
  - ✓ In abgeschlossener Umgebung gleicher Informationsgehalt wie bei positiver Erkennung

4 Erkennungsschemata
Disjunktive vs. konjunktive Erkennung

- **Disjunktive Erkennung**
  - **Ein Match pro Paket ist ausreichend**
- **Konjunktive Erkennung**
  - **Paket muß an allen Stellen matchen**
- **Vier Kombinationen aus positiver, negativer und disjunktiver bzw. konjunktiver Erkennung möglich**

4 Erkennungsschemata
ND, PD, PC, PD

- $Scheme_{ND}(¡) = \{x \hat{I} U \mid "w.Ø\$d \hat{I} ¡.dMx\}$

- $Scheme_{PD}(¡) = \{x \hat{I} U \mid \$w.\$d \hat{I} ¡.dMx\}$

- $Scheme_{PC}(¡) = \{x \hat{I} U \mid "w.\$d \hat{I} ¡.dMx\}$

- $Scheme_{NC}(¡) = \{x \hat{I} U \mid \$w.Ø\$d \hat{I} ¡.dMx\}$

Vergleich der Methoden
Ähnlichkeiten

- **r-chunks matching umfaßt rcb-matching**
  - ***l* = 4**
  - ***r* = 2**

- **Scheme_{PC} und Scheme_{ND} mit r-chunks erkennen genau dieselben Sprachklassen wie Crossover Closure.**

Vergleich der Methoden
Ähnlichkeiten

- $Scheme_{ND}(¡) = (Scheme_{PD}(¡))'$
  $= Scheme_{PC}(¡') = (Scheme_{NC}(¡'))'$
  $\Leftrightarrow (Scheme_{ND}(¡'))' = Scheme_{PD}(¡')$
  $= (Scheme_{PC}(¡))' = Scheme_{NC}(¡)$

Vergleich der Methoden
Unterschiede/Probleme

- rcb
  - Scheme_{PC} und Scheme_{ND} erkennen nicht alle unter CC abgeschlossenen Sprachen
    - Scheme_{PD} erkennt S ® Scheme_{ND} erkennt S'
- r-chunks
  - Scheme_{PC} und Scheme_{ND} erkennen nicht alle unter CC abgeschlossenen Sprachen, aber einige, die nicht abgeschlossen sind
    - Scheme_{ND} erkennt S ® Scheme_{PD} erkennt S'
- Beispiel: $l = 3$, $r = 2$, $S = \{000\}$
  - $CC(S) = S$ ist abgeschlossen
    - $S' = \{001, 010, 011, 100, 101, 110, 111\}$
    - $CC(S') = U$

Effizienzsteigerung
Vorüberlegungen

- Je größer self, umso umfangreicher die Anzahl der Detektoren
  - Wie kann man die Detektoranzahl reduzieren?
  - Recall: Gleicher Informationsgehalt bei positiver und negativer Erkennung!
- An welchem Punkt ist negative Erkennung effizienter?
  - Bestimmung der Detektoranzahl in Abhängigkeit von |S|

Effizienzsteigerung
Bestimmung der Detektoranzahl

- Anzahl der Strings in U mit einem bestimmten Pattern bei Fenstergröße r
  - 2^{l-r}
- Ein solcher String wird mit Wahrscheinlichkeit $2^{l-r}/2^{l} = 1/2^{r}$ gewählt
- Wahrscheinlichkeit für bestimmtes Pattern bei zufälligem self set und kleinem r
  - $1 - (1 - 1/2^{r})^{|S|}$
- Anzahl unterschiedlicher Pattern in S bei Fenstergröße r
  - $E_r \approx 2^r - 2^r (1 - 1/2^r)^{|S|}$

Effizienzsteigerung
Bestimmung der Detektoranzahl

- Geschätzte Anzahl der Detektoren bei PD-Erkennung
  - $E_{pos} = tE_r$
- Geschätzte Anzahl bei NC-Erkennung
  - $E_{neg} = t\,(2^r - E_r)$
- Günstigervergleich
  - Recall: $E_r \approx 2^r - 2^r(1 - 1/2^r)^{|S|}$
  - $E_{neg} = E_{pos} = t\,(2^r(1- 1/2^r)^{|S|}) = t\,(2^r - 2^r(1 - 1/2^r)^{|S|})$
- Lösung für |S| (im binären Fall)
  - $|S| = -\ln(2) / \ln(1 - 1/2^r) \approx (0{,}693) * 2^r$

Effizienzsteigerung
Weniger Detektoren

- Ausnutzen von Redundanz bei Detektoren
- Algorithmus bei negativer Erkennung
  - Detektor für 1. Fenster *t* erstellen, mit *t* Ï S
  - Für jedes nächste Fenster und jedes Paar $w_1 = v_i \ldots v_{r+i-2}a$, $w_2 = v_i \ldots v_{r+i-2}\bar{a}$
    - w1 und w2 Î S ® kein Detektor möglich
    - w_1 und w_2 Ï S ® Detektor nicht nötig, da schon in *t-1* abgedeckt
    - w_1 Å w_2 Î S ® komplementären Detektor erstellen, da er in keinem *x* Î *S* vorkommt
- $E_{minN} = 2^r - E_r + (l - r)(E_r - 2(E_r - E_{r-1}))$
- $E_{minP} = E_r + (l - r)(E_r - 2(E_r - E_{r-1}))$

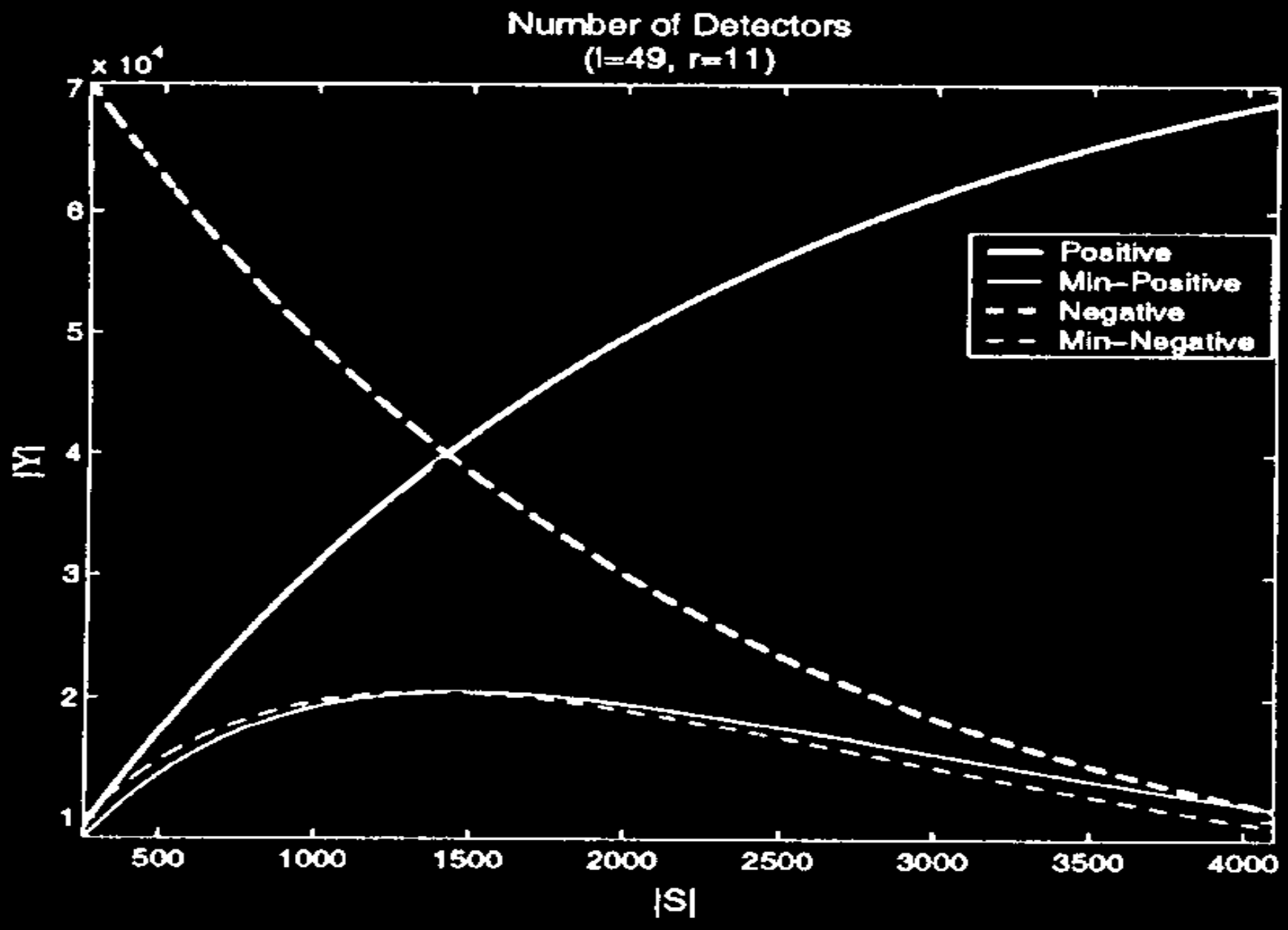


Fig. 2. Number of positive and negative detectors as a function of the size of the self set, assuming the self set was generated randomly. The plot shows both complete and reduced detector sets.

Löcher

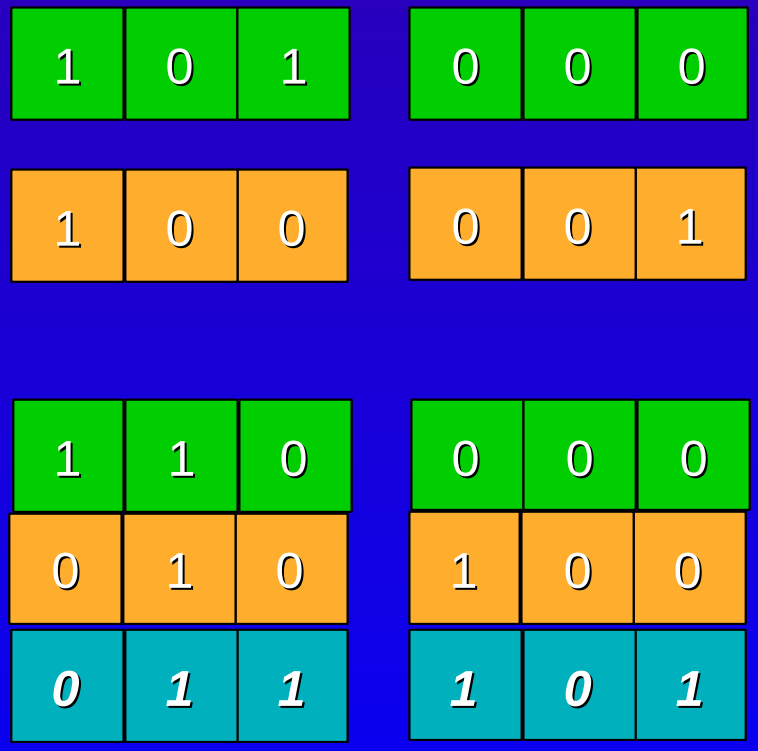
Problemstellung

- Trennung von self und non-self nicht immer möglich
- Löcher sind Sprachbereiche, die aufgrund des Erkennungsschemas nicht kollisionsfrei erkannt werden können
- Wie läßt sich die Anzahl der Löcher verringern?

Löcher

Reduktion der Anzahl

- Permutationen p (permutation maps)
- Beispiel mit rcb
  - S = {101, 000}
  - l = 3, r = 2
  - h_1 = 100, h_2 = 001 Ï S
  - Detektoren 00*, 10*, *00, *01
- Nach Permutation
  - p(s1) = 110, p(s2) = 000
  - p(h1) = 010, p(h2) = 100
  - ✓ Gültige Detektoren 011, 101



Löcher

Permutation kann nicht alles

- Nicht erkennbare Strings unter jeder Permutation bei gegebenem S und rcb

Self	010 011 100 101	001 011 100 110	100 101 010 011	100 110 001 011	001 101 010 110	010 110 001 101
h	110	101	110	101	011	011
detector templates	11* *10	10* *01	11* *10	10* *01	01* *11	01* *11

Fazit

- Große Anzahl an Detektoren möglich in umfangreichen Systemen
- Negative Erkennung ist manchmal besser
- Intelligente Wahl des Erkennungsschemas
  - Unterschiede bei der Erkennung von Self
  - Unterschiedliche Erkennung von Non-Self Elementen
  - Reduktion der Löcher
- Effizienzsteigerung durch reduzierte Detektoranzahl möglich

Noch offene Fragen

- Wie wird sich negative Erkennung in der Praxis behaupten?
- Wie kann die Ähnlichkeit von relationalen Datenbanken und $Scheme_{ND}$/$Scheme_{PC}$ zur Datenspeicherung verwendet werden?
- Aktualisierung dynamischer Samples?
- Zwischenlösung zwischen disjunktiver und konjunktiver Erkennung?
- Erstellung, Speicherung und Zugriff von/auf Detektoren effizienter gestalten?
- Zusammenhang zwischen GA und rcb?

Alles klar?!?