Universität Hannover
Institut für Allgemeine Nachrichtentechnik
Prof. Dr.-Ing. H.-P. Kuchenbecker
Prof. Dr.-Ing. K. Jobmann

Entwicklung und Implementierung von Algorithmen zur Regelbildung aus Ereignissequenzen in Kommunikationsnetzen

Diplomarbeit

Peter Tondl

29. August 2000

Erstprüfer	:	Prof. Dr.-Ing. K. Jobmann
Zweitprüfer	:	Prof. Dr.-Ing. H.-P. Kuchenbecker
Betreuer	:	Dipl.-Ing. Klaus-Dieter Tuchs

Inhaltsverzeichnis

Kapitel 1

Einleitung

Ein Kommunikationsnetzwerk kann als eine Ansammlung miteinander verbundener Elemente gesehen werden: Vermittlungsstellen, Übertragungs-, Sende- und Empfangseinrichtungen. Nahezu jede Komponente eines solchen Netzes enthält ihrerseits untergeordnete Subkomponenten. Die genaue Zahl der Komponenten ist abhängig von der Abstraktionsebene der Sichtweise auf das Netz.

Alle Elemente – sowohl Komponenten, Subkomponenten als auch Software-Module – sind in der Lage, *Alarmmeldungen* zu produzieren. Diese Alarmmeldungen stellen Informationen dar, die Klassen verschiedener, vom Normalfall abweichender Situationen beschreiben. Selbst in kleineren Netzen existiert eine große Zahl verschiedener Typen von Alarmmeldungen. Die Zahl täglich produzierter Meldungen schwankt dabei typischerweise zwischen 200 und 10.000 [HKM96].

In den Netzmanagement-Zentralen der Netzwerk-Betreiber werden die empfangenen Alarmmeldungen in Datenbanken gespeichert, unter Umständen gefiltert sowie dem zuständigen Operateur angezeigt. Dieser entscheidet daraufhin über die Art der einzuleitenden Reaktion.

Dabei können verschiedene Arten von Problemen auftreten: Zum einen bedeutet die Größe des Netzwerkes und die Mannigfaltigkeit der Alarmtypen eine hohe Anzahl möglicher auftretender Situationen. Erschwerend kommt hinzu, dass Alarmmeldungen oft in *Alarmfluten* auftreten, was bedeutet, dass sehr viele Meldungen innerhalb kurzer Zeit in den Zentralen eintreffen. Alarmfluten können entstehen, wenn wenige Fehlerursachen die Generierung von Alarmmeldungen in zahlreichen direkt oder indirekt betroffenen Netzelementen zur Folge haben. Neben der Belastung des Netzes durch die Vielzahl übermittelter Meldungen wird dem Operateur die Behandlung einzelner Alarmmeldungen durch Alarmfluten deutlich erschwert.

Zum anderen unterliegt die in Kommunikationsnetzwerken verwendete Hard- und Software ständigen Veränderungen. Ändert sich die Zusammensetzung der Netzele-

mente, so ändert sich auch die Charakteristik der die zeitliche Abfolge der Alarmmeldungen beschreibenden auftretenden *Alarmsequenzen*.

Zur Linderung dieser Probleme kommen Techniken wie *Alarmfilterung* und *Alarmkorrelation* zum Einsatz. Diese reduzieren die Zahl der dem Operateur angebotenen Alarmmeldungen und erhöhen die Abstraktionsebene der angezeigten Information. Alarmfilterung wird auf allen Hierachie-Ebenen eines Netzwerkes angewendet indem nur Teile der empfangenen Meldungen eines Elementes durch das übergeordnete Element weitergegeben werden. Alarmkorrelation bedeutet die Zusammenfassung und Rückführung von Einzelalarmen auf ihre Fehlerursache.

Zur Erfüllung ihrer Aufgaben benötigen Alarmkorrelatoren Regeln anhand derer sie Alarmmeldungen zu Alarmgruppen zusammenfassen können. Die vorliegende Diplomarbeit präsentiert eine mit Hilfe dafür entwickelter spezieller Algorithmen realisierte Lösung für das Problem der Regelgewinnung. Es wird gezeigt, wie aus einer gegebenen Datenbasis an *Alarmmustern* – also zusammengehörenden Alarmmeldungen – in sinnvoller und effizienter Weise Regeln gewonnen und bei Bedarf nachbearbeitet und geordnet werden können.

Die für das Verständnis der gewählten Lösung notwendigen theoretischen Betrachtungen, sowie die entwickelten Algorithmen werden in Kapitel 2 vorgestellt. Kapitel 3 verdeutlicht die Arbeitsweise der Algorithmen anhand eines einfachen praktischen Beispiels. In Kapitel 4 werden die aus der Untersuchung realer Alarmmeldungs-Datenbanken gewonnenen Erkenntnisse über das Verhalten der verwendeten Algorithmen und die bei der Anwendung auftretenden Probleme aufgezeigt. Kapitel 5 fasst die wesentlichen Aspekte dieser Arbeit zusammen und gibt einen Ausblick über mögliche weiterführende Projekte. In dem daran anschließenden Anhang A wird die in Form eines C++-Programms entwickelte Lösung im Detail vorgestellt. Hinweise zu den notwendigen Rahmenbedingungen werden in Anhang B gegeben. Die für eine Weiterentwicklung des Programms notwendige Übersicht über die Programmstruktur wird in Anhang C vermittelt, während Anhang D eine notwendige Korrektur einer dieser Diplomarbeit zugrunde liegenden Arbeit gibt.

Kapitel 2

Assoziative Regeln

Assoziative Regeln – im Kontext dieser Arbeit meist als *Regelmuster* bezeichnet – stellen eine einfache und nützliche Form von Wissen dar, das in effizienter Weise aus großen Datenbeständen gewonnen werden kann. Eine Sammlung von Regelmustern, die in einer gegebenen Relation enthalten ist, beinhaltet typischerweise große Mengen an Informationen über die zugrunde liegenden Daten. Das Problem bei Anwendung dieser *Data Mining*-Technik ist, dass die Zahl gefundener Muster im Allgemeinen zu groß sein wird, um die gebildete Regelsammlung direkt verwenden zu können. Ziel dieses Kapitels ist es somit, nach einigen grundlegenden Ausführungen in Abschnitt 2.1 und dem Aufzeigen von Möglichkeiten zur Regelmuster-Gewinnung aus Alarmmustern in Abschnitt 2.2, in Abschnitt 2.3 Verfahren vorzustellen, mit deren Hilfe die gewonnenen Bestände an Regelmustern von enthaltenen Redundanzen befreit werden können. Abschnitt 2.4 beschäftigt sich mit Möglichkeiten, den verbleibenden Bestand an Regelmustern in einer durch den Anwender beeinflussbaren Weise weiter zu unterteilen um die Übersichtlichkeit und damit den Nutzwert der Regelsammlung zu erhöhen. Im abschließenden Abschnitt 2.5 werden Ideen aufgezeigt, mit deren Hilfe die gefundenen Regelmuster auf einem höheren Abstraktionsniveau – und damit auf eine für den Anwender leichter erfassbare Weise – dargestellt werden können.

2.1 Das Warenkorbmodell

Assoziative Regeln dienen der Beschreibung von Datenbeständen und ihren Eigenschaften. Sie können als Vorhersage für das Auftreten von Attributwerten aufgefasst werden [Toi96, Fer98]. Eine solche Regel kann über ein „Warenkorbmodell“ definiert werden:

Definition 2.1 (Assoziative Regel)
Es existiere eine Grundmenge $R = \{P_1, \ldots, P_m\}$ von Produkten P_i. Mit p_i, p_j seien Teilmengen, mit P_i ein einzelnes Element der Grundmenge bezeichnet. Die *Regel* $p_i \Rightarrow P_i$ stellt dann eine *Assoziation* $p_i \cup \{P_i\}$ zwischen der Teilmenge p_i und dem Element P_i her. Die Regel $p_i \Rightarrow p_j$ stellt die entsprechende Assoziation $p_i \cup p_j$ zwischen den Teilmengen p_i und p_j her.

In einer Datenbank über das Kaufverhalten von Kunden einer Supermarktkette könnte beispielsweise die folgende assoziative Regel gefunden werden:

Beispiel 2.1 (Assoziative Regel)
`Bier` $\Rightarrow$ `Chips (0.84)`

Beispiel 2.1 bedeutet, dass 84% der Kunden, die Bier kauften, auch Chips in ihrem Warenkorb liegen hatten.

Definition 2.2 (Beispielmenge)
Sei $R = \{P_1, \ldots, P_m\}$ wiederum Grundmenge von Produkten P_i und gelte $p_i \subseteq R$. Dann wird eine *Beispielmenge* $r = \{p_1, \ldots, p_n\}$ durch eine Folge von n Teilmengen p_i der Grundmenge dargestellt.

Die Beispielmenge r entspricht der Menge an vorkommenden Warenkörben und wird durch eine Menge binärer Spaltenvektoren p_i der Länge m dargestellt. Dabei wird das Vorkommen eines Produkts P_i durch $p_i = 1$, das Fehlen durch $p_i = 0$ repräsentiert.

Definition 2.3 (Produktmenge)
Sei $R = \{P_1, \ldots, P_m\}$ wiederum Grundmenge von Produkten P_i und gelte $p_i \subseteq R$. Dann wird die *Produktmenge* p_i durch eine Instanz von p_i dargestellt.

Befinden sich in einem Warenkorb p_i beispielsweise die Produkte Butter, Brot und Milch, so entsprichte dem die zugehörige Produktmenge $p_i = \{Butter, Brot, Milch\}$.

Bei einer abstrakteren Darstellung der Produkte durch die allgemeinen Platzhalter A bis F lässt sich die Beispielmenge r mit den zugehörigen Produktmengen p_i wie folgt darstellen:

Spalten-ID	Spalte
p_1	$\{A, B, C, D, G\}$
p_2	$\{A, B, E, F\}$
p_3	$\{B, I, K\}$
p_4	$\{A, B, H\}$
p_5	$\{E, G, J\}$

Tabelle 2.1: Beispielmenge r über $R = \{A, \ldots, K\}$

Die für eine Verarbeitung in elektronischer Form günstigere zugehörige Darstellung mittels Spaltenvektoren in binärer Form wird durch die folgende Darstellung deutlich:

Spalten-ID	A	B	C	D	E	F	G	H	I	J	K
p_1	1	1	1	1	0	0	1	0	0	0	0
p_2	1	1	0	0	1	1	0	0	0	0	0
p_3	0	1	0	0	0	0	0	0	1	0	1
p_4	1	1	0	0	0	0	0	1	0	0	0
p_5	0	0	0	0	1	0	1	0	0	1	0

Tabelle 2.2: Binäre Beispielmenge r über $R = \{A, \ldots, K\}$

Die *relative Frequenz* $fr(p, r)$ einer Produktmenge $p \subseteq R$ über r wird definiert zu:

Definition 2.4 (Relative Frequenz)

$$fr(p, r) = \frac{|\{p \in r \mid p[i] = 1 \text{ für alle } P_i \in p\}|}{|r|}$$

was bedeutet, dass die relative Frequenz einer Produktmenge der Wahrscheinlichkeit entspricht, mit der die zugehörigen Produkte in einem zufällig gewählten Warenkorb vollständig vorkommen. Anders ausgedrückt entspricht die relative Frequenz dem Quotienten aus der Anzahl der Zeilen in r, die das zu p gehörende „Produktmuster“ aufweisen – in denen also *mindestens* die in p enthaltenen Produkte vorkommen – und der Gesamtzahl aller Zeilen in r. Die relative Frequenz wird im Folgenden nur noch als Frequenz bezeichnet.

Beispiel 2.2 (Frequenz einer Produktmenge)
Sei $p = \{A, B\}$ eine gegebene Produktmenge. Mit r aus Tabelle 2.1 gilt

dann:

$$fr(p,r) = \frac{|\{p_1, p_2, p_4\}|}{|r|} = \frac{3}{5} = 0,6$$

Das Ziel besteht nunmehr in dem Finden der Produkte, die genügend oft zusammen gekauft werden. Was als „genügend oft“ angesehen wird, muss durch die Festlegung eines *Frequenz-Schwellwertes* min_freq definiert werden. Eine Produktmenge p gilt dann als frequent, wenn $fr(p,r) \geq min\_freq$ erfüllt ist.

Eine Zusammenfassung aller frequenten Produktmengen über r unter Berücksichtigung des Frequenz-Schwellwertes kann wie folgt definiert werden:

Definition 2.5 (Set frequenter Produktmengen)

$$F(r, min\_fr) = \{p_i \subseteq R \mid fr(p_i, r) \geq min\_freq\}$$

Bei gegebenen nichtleeren und unzusammenhängenden Produktmengen $p_i, p_j \subseteq R$ bedeutet der Ausdruck $p_i \Rightarrow p_j$ gemäß Definition 2.1 eine assoziative Regel über r. Der einer Regel entgegengebrachte Grad an Vertrauen kann wie folgt definiert werden:

Definition 2.6 (Vertrauenswürdigkeit einer Regel)

$$conf(p_i \Rightarrow p_j, r) = \frac{fr(p_i \cup p_j, r)}{fr(p_i, r)}$$

Für die Vertrauenswürdigkeit einer Regel lässt sich ebenso wie für die Frequenz ein Schwellwert min_conf definieren. Mittels diesem kann entschieden werden, ob eine Regel stark genug ist, um als „interessant“ zu gelten. Sowohl der Frequenz- als auch der Vertrauens-Schwellwert werden für das Entfernen uninteressanter Regeln benutzt.

Beispiel 2.3 (Vertrauenswürdigkeit einer Regel)
Für die nichtleeren und unzusammenhängenden Produktmengen $p_i = \{A\}$ und $p_j = \{B\}$ sowie der aus diesen Mengen bildbaren Regel $p_i \Rightarrow p_j$ („wenn p_i dann auch p_j“) gilt wiederum mit r aus Tabelle 2.1:

$$conf(p_i \Rightarrow p_j, r) = \frac{|\{p_1, p_2, p_4\}|}{|\{p_1, p_2, p_4\}|} = \frac{3}{3} = 1$$

Assoziative Regeln besitzen im Hinblick auf die Expansion oder Kontraktion der linken Regelseite keine monotonen Eigenschaften. Wenn beispielsweise $p_i \Rightarrow p_j$ gilt, so gilt nicht notwendigerweise auch die expandierte Form $p_i \cup \{A\} \Rightarrow p_j$. Dies liegt daran, dass die Werte für die Frequenz und die Vertrauenswürdigkeit der Regel $p_i \cup \{A\} \Rightarrow p_j$ nicht zwangsläufig ausreichend sein müssen. Assoziative Regeln sind ebenfalls nicht monoton im Hinblick auf die Expansion der rechten Regelseite. Wenn $p_i \Rightarrow p_j$ zutrifft, so ist dies für die expandierte Form $p_i \Rightarrow p_j \cup \{A\}$ aus obigen Gründen ebenso nicht garantiert. Die einzige monotone Eigenschaft assoziativer Regeln besteht in Bezug auf die Kontraktion der rechten Regelseite: Trifft $p_i \Rightarrow p_j \cup \{A\}$ zu, so gilt auch die Kontraktion $p_i \Rightarrow p_j$. Alle in den späteren Abschnitten vorgestellten Verfahren zur Gewinnung und Nachbearbeitung von Regeln werden zur Erzielung hoher Effizienz von dieser Eigenschaft gebrauch machen.

Das Problem der Regelfindung kann nun wie folgt beschrieben werden: Gegeben sei ein Set R binärer Attribute, eine korrespondierende Relation r, ein Frequenz-Schwellwert min_freq sowie ein Vertrauens-Schwellwert min_conf. Finde alle assoziativen Regeln in r, die eine minimale Frequenz von min_freq und eine minimale Vertrauenswürdigkeit von min_conf aufweisen.

Die Entdeckung assoziativer Regeln kann in zwei Phasen unterteilt werden. In einem ersten Schritt werden die der Regelbildung zugrunde liegenden Datenmuster $p_i \subseteq R$ gesucht. Anschließend wird in der zweiten Phase für jedes frequente Datenmuster p_i geprüft, ob alle nichtleeren Teilmuster $p_{part} \subset p_i$ die Regel $p_i \backslash p_{part} \Rightarrow p_{part}$ unter der Bedingung ausreichender Vertrauenswürdigkeit erfüllen. Für den Fall einer *Ereignissequenz* – bei dieser lassen sich die vorkommenden Einzelereignisse E_i (Entsprechung der zuvor behandelten Produkte P_i) auf einem Zeitstrahl anordnen – wird dieser Vorgang in Kapitel 3 beispielhaft aufgezeigt.

2.2 Generierung von Regelmustern

In einer gegebenen Datenbasis können sehr viele assoziativer Regeln enthalten sein. Die weitaus größte Zahl dieser Regeln wird sich auf einem sehr niedrigen Abstraktionslevel bewegen und daher in ihrer Aussagekraft und in ihrem Nutzen beschränkt sein. Im Kontext dieser Arbeit wird daher von *Regelmustern* gesprochen, wenn sich die Auswahl der gewonnenen Regeln einzig auf die Parameter für minimale Frequenz min_freq und minimales Vertrauen min_conf stützt. Regelmuster stellen somit – obwohl bereits vollwertige Regeln – das Rohmaterial für die in einem späteren Prozess zu bildenden, auf einem höheren Abstraktionslevel stehenden, Regeln dar.

Ziel dieses Abschnittes ist es, die für den Prozess der Generierung von Regelmustern aus einem gegebenen Alarmmuster-Bestand entwickelten Algorithmen vorzustellen und ihre grundsätzliche Arbeitsweise zu erläutern.

Vor Beginn der Ausführungen zur Regelmuster-Gewinnung und -Nachbearbeitung eine Anmerkung zur in den folgenden Abschnitten verwendeten Nomenklatur: Ein Ausdruck der Form $X \Rightarrow Y$ bezeichnet eine Regel. X stellt somit die linke Regelseite, Y die rechte Seite der Regel dar. Da Regeln aus Alarmmustern gebildet werden (siehe folgenden Abschnitt) und diese beide Seiten der Regeln enthalten, ergibt sich als Bezeichnung für ein Alarmmuster der Ausdruck XY.

2.2.1 Alarmmuster

Regelmuster werden aus Alarmmustern konstruiert, die zuvor aus einem gegebenen Datenbestand extrahiert werden müssen. Die Aufgabe der Alarmmuster-Findung lässt sich dabei wie folgt umschreiben: Gegeben sei eine Klasse von Alarmmustern K, ein Zeitstrahl von Einzelalarmen s, eine Fensterbreite $window\_width$ sowie ein Frequenz-Schwellwert min_freq. Finde alle Alarmmuster $XY \subseteq K$, die eine genügende Häufigkeit $fr(XY, window\_width, s) \geq min\_freq$ auf s aufweisen. Für eine weitergehende Erläuterung dieser alles andere als trivialen Thematik sei auf [Toi96, Tdl00] verwiesen.

Alarmmuster lassen sich in vier Hauptklassen einteilen. Dazu gehören die parallelen sowie die seriellen Alarmmuster. Dürfen in einem Muster Ereignisse eines bestimmten Alarmtyps mehrfach vorkommen, wird von einem Alarmmuster mit Wiederholung gesprochen, andernfalls von einem Muster ohne Wiederholung.

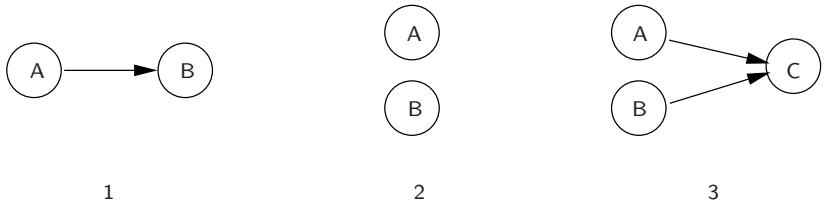

Abbildung 2.1: Alarmmuster

Beispiel 2.4 (Alarmmuster)
Sei XY ein Alarmmuster. A, B und C seien Einzelalarme. Dann wird XY (mit A und B als zugehörigen Einzelalarmen) als *serielles* Alarmmuster bezeichnet, wenn auf A immer B folgt (Abbildung 2.1, Fall 1). Ist die Reihenfolge hingegen beliebig, handelt es sich bei XY um ein *paralleles* Alarmmuster (Fall 2). Besteht XY aus allen drei Einzelalarmen und folgt C immer auf A und B (deren Reihenfolge wiederum beliebig ist), handelt es sich um ein nicht-serielles und nicht-paralleles Alarmmuster (Fall 3, im Kontext dieser Arbeit nicht weiter behandelt).

Ein paralleles Alarmmuster XY_{par} wird repräsentiert durch ein lexikographisch geordnetes Array von Einzelalarmen. Ein Array wird durch den Namen des Alarmmusters, die Einträge im Array über eine Notation mittels eckiger Klammern bezeichnet. Beispielsweise könnte sich ein Alarmmuster XY_{par} mit den zugehörigen Einzelalarmen A, B und C über das Array $XY_{par}[1] = A$, $XY_{par}[2] = B$, $XY_{par}[3] = B$, $XY_{par}[4] = C$ und $XY_{par}[5] = C$ darstellen.

Ein serielles Alarmmuster XY_{ser} wird ebenfalls durch ein Array von Einzelalarmen repräsentiert. Nunmehr beruhen die Alarme eines Arrays jedoch auf einer Ordnung. Beispielsweise wird das serielle Alarmmuster XY_{ser} mit den zugehörigen Einzelalarmen C, A, F und C – in dieser Reihenfolge – durch ein Array $XY_{ser}[1] = C$, $XY_{ser}[2] = A$, $XY_{ser}[3] = F$, $XY_{ser}[4] = C$ dargestellt.

2.2.2 Generierung paralleler Regelmuster

Algorithmus 2.1 wurde zur Generierung von Regelmustern aus parallelen Alarmmustern entwickelt. Die dem Algorithmus zugrunde liegende Idee ist dabei folgende: In einem ersten Schritt werden aus dem aktuellen Alarmmuster alle Regelmuster generiert, die auf der rechten Regelseite lediglich ein Ereignis enthalten. Dabei wird während des i-ten Schrittes das jeweils i-te Element des Alarmmusters der rechten Regelseite zugeordnet, während die übrigen Elemente der linken Regelseite entsprechen. Um Redundanzen zu vermeiden, wird jedem so gebildeten Regelmuster eine Zahl $skip = i - 1$ beigefügt, die besagt, wie viele Schritte bei der daran anschließenden – auf diesem Regelmuster basierenden – Ableitung weiterer Regelmuster übersprungen werden dürfen. Beispiel 2.5 erläutert diese Vorgehensweise. Zur Vermeidung weiterer Redundanzen, die bei Alarmmustern mit Wiederholungen entstehen können, wird zusätzlich das aktuelle Element des Alarm- oder Regelmusters mit dem vorhergehenden Element verglichen und bei Übereinstimmung kein neues Muster generiert (dieses wäre völlig identisch mit dem zuvor gebildeten Muster).

Beispiel 2.5 (Generierung paralleler Regelmuster)
Im ersten Schritt werden alle Regelmuster aus einem Alarmmuster gebildet. In allen folgenden Schritten werden die Regelmuster aus ihren übergeordneten Regelmustern abgeleitet. Die Darstellung in runden Klammern bedeutet, dass dieses Muster redundant ist und daher nicht gebildet zu werden braucht. Erkennbar ist, dass entsprechend ihren *skip*-Zahlen, kein Nachfolger des ersten Regelmusters, der erste Nachfolger des zweiten Regelmusters sowie beide Nachfolger des dritten Regelmusters redundant sind.

Schritt 1: zugrunde liegendes Alarmmuster {1, 2, 3}

skip	{1, 2, 3}
0	2 3 → 1
1	1 3 → 2
2	1 2 → 3

Schritt 2: zugrunde liegende Regelmuster
{2 3 → 1}, {1 3 → 2}, {1 2 → 3}

skip	2 3 → 1	*skip*	1 3 → 2	*skip*	1 2→ 3
0	3 → 1 2	(0)	(3 → 1 2)	(0)	(2 → 1 3)
1	2 → 1 3	1	1 → 2 3	(1)	(1 → 2 3)

Funktionsnachweis Algorithmus 2.1
Der Algorithmus funktioniert, da alle ausgegebenen parallelen Regelmuster $rule = X \Rightarrow Y$ in der zugrunde liegenden Datenbasis enthalten sind. Es gilt $fr(rule) \geq min\_freq$, da $fr(XY) \geq min\_freq$ (Zeile 2) und $conf(rule) \geq min\_conf$ (Zeilen 12, 26) gefordert ist. Alle Regelmuster $rule$, die in der zugrunde liegenden Datenbasis enthalten sind, werden andererseits vom Algorithmus erkannt und ausgegeben, da $fr(rule) \geq min\_freq$ sowie $fr(XY) \geq min\_freq$ gilt und XY in $F(r, min\_freq)$ sein muss (Zeile 2). Das möglicherweise interessante Regelmuster $rule$ wird überprüft und ausgegeben, wenn gilt $conf(rule) \geq min\_conf$ (Zeilen 34, 35).

Die Vertrauenswürdigkeit von Regelmustern fällt monoton durch das Wechseln von Elementen der linken Seite auf die rechte Seite der Regel (siehe Ende Abschnitt 2.1). Diese Eigenschaft kann ausgenutzt werden, um die Anzahl potenziell interessierender Muster zu reduzieren, die in Algorithmus 2.1 auf Vertrauenswürdigkeit getestet werden. Die Idee ist dabei folgende: Für ein Set $XY \in F(r)$ wird das Regelmuster $X \Rightarrow Y$ nur dann getestet, wenn alle Regelmuster $X \cup Y \backslash Z \Rightarrow Z$ mit $Z \subset Y$ einen ausreichend hohen Grad an Vertrauenswürdigkeit besitzen.

In Algorithmus 2.1 wird dieser Umstand durch die Nichtaufnahme von Regelmustern mit zu geringem Vertrauenswert in das Set gültiger Muster berücksichtigt. Bedingt durch die schrittweise Herangehensweise an die Musterbildung (Beispiel 2.5) können aus ungültigen Mustern keine weiteren Regelmuster abgeleitet werden.

Eingabe: R, r über R, min_freq, min_conf
Ausgabe: Alle parallelen Regelmuster als Funktion von min_freq, min_conf und r sowie deren Frequenz und Vertrauenswürdigkeit.

```
1:  // Finde alle frequenten parallelen Alarmmuster
2:  Berechne F(r, min_freq) := {XY ⊆ R | fr(XY, r) ≥ min_freq};
3:  for alle XY ∈ F(r, min_freq) do
4:    k := 0;
5:    S := ∅;
6:    last_member := ∅;
7:    current_size := |XY| − 1;
8:    for i := 1 to i = |XY| do
9:      if last_member ≠ XY[i] then
10:       rule[left_side] := XY \ XY[i];
11:       rule[right_side] := XY[i];
12:       if conf(rule) ≥ min_conf then
13:         rule[skip] := i − 1;
14:         rule[size] := current_size;
15:         rule[conf] := conf(rule);
16:         k := k + 1;
17:         S[k] := rule;
18:     last_member := XY[i];
19:   while current_size > 1 do
20:     for alle S[k] mit S[k][size] = current_size do
21:       last_member := ∅;
22:       for i := 1 + S[k][skip] to current_size do
23:         if last_member ≠ S[k][left_side][i] then
24:           rule[left_side] := S[k][left_side] \ S[k][left_side][i]
25:           rule[right_side] := S[k][right_side] ∪ S[k][left_side][i]
26:           if conf(rule) ≥ min_conf then
27:             rule[skip] := i − 1;
28:             rule[size] := current_size − 1;
29:             rule[conf] := conf(rule);
30:             k := k + 1;
31:             S[k] := rule;
32:         last_member := S[k][left_side][i];
33:     current_size := current_size − 1;
34:   for i := 1 to k do
35:     Ausgabe des Regelmusters S[k], fr(XY) und S[k][conf];
```

Algorithmus 2.1: Generierung paralleler Regelmuster

Definition 2.7 (Zahl bildbarer paralleler Regelmuster)
Sei L die Länge des zugrunde liegenden parallelen Alarmmusters XY_{par}, so gilt für die Zahl der aus XY_{par} bildbaren Regelmuster G (für Alarmmuster ohne Wiederholungen):

$$G(L) = \sum_{n=1}^{L-1} \binom{L}{n}$$

Definition 2.7 kann durch das klassische Urnenmodell erklärt werden, bei dem die Reihenfolge der gezogenen Kugeln keine Rolle spielt (da parallele Alarmmuster zugrunde liegen) und die Forderung besteht, dass mindestens eine Kugel gezogen wird (nichtleere Teilmenge der linken Regelseite) sowie mindestens eine Kugel in der Urne verbleibt (nichtleere Teilmenge der rechten Regelseite). Die Kombinatorik gibt für diesen Fall die Berechnungsgrundlage $G = \binom{L}{n}$ an, mit den $L-1$ Möglichkeiten eine unterschiedliche Zahl von Kugeln zu ziehen ergibt sich somit $G(L)$ in angegebener Weise.

2.2.3 Generierung serieller Regelmuster

Algorithmus 2.2 wurde zur Erzielung hoher Effizienz bei der Generierung von Regelmustern aus seriellen Alarmmustern entwickelt. Um die speziellen Eigenschaften dieser Alarmmusterart nutzen zu können, besteht eine wesentliche Forderung in der Bildung nichtleerer *rechter Teilmengen*, also Mengen, die von der rechten Seite des aktuellen Alarmmusters beginnend gebildet werden. Beispiel 2.6 erläutert dieses Verfahren.

Beispiel 2.6 (Bildung rechter Teilmengen)
Sei $\{A, B, C, D\}$ ein serielles Alarmmuster XY_{ser}. Dann sind die aus XY_{ser} bildbaren nichtleeren rechten Teilmengen $Y \subset XY_{ser}$: $\{D\}$, $\{C, D\}$ sowie $\{B, C, D\}$ und zwar in genau dieser Reihenfolge.

Funktionsnachweis Algorithmus 2.2
Der Algorithmus funktioniert, da alle ausgegebenen seriellen Regelmuster $rule = X \Rightarrow Y$ in der zugrunde liegenden Datenbasis enthalten sind. Es gilt $fr(rule) \geq min\_freq$, da $fr(XY) \geq min\_freq$ (Zeile 2) und $conf(rule) \geq min\_conf$ (Zeile 10) gefordert ist. Alle Regelmuster $rule$, die in der zugrunde liegenden Datenbasis enthalten sind, werden andererseits vom Algorithmus erkannt und ausgegeben, da $fr(rule) \geq min\_freq$

Eingabe: R, r über R, min_freq, min_conf
Ausgabe: Alle seriellen Regelmuster als Funktion von min_freq, min_conf und r sowie deren Frequenz und Vertrauenswürdigkeit.

```
1:  // Finde alle frequenten seriellen Alarmmuster
2:  Berechne F(r, min_freq) := {XY ⊆ R | fr(XY, r) ≥ min_freq};
3:  for alle XY ∈ F(r, min_freq) do
4:    for i := 1 to i = |XY| − 1 do
5:      rule := ∅;
6:      for x := 1 to |XY| − i do
7:        rule[left_side][x] := XY[x];
8:      for x := |XY| − i + 1 to |XY| do
9:        rule[right_side][x + i − |XY|] := XY[x];
10:     if conf(rule) ≥ min_conf then
11:       Ausgabe des Regelmusters rule, fr(XY) und conf(rule);
12:     else
13:       setze fort mit Zeile 3
```

Algorithmus 2.2: Generierung serieller Regelmuster

sowie $fr(XY) \geq min\_freq$ gilt und XY in $F(r, min\_freq)$ sein muss (Zeile 2). Das möglicherweise interessante Regelmuster $rule$ wird überprüft und ausgegeben, wenn gilt $conf(rule) \geq min\_conf$ (Zeilen 10, 11).

In Bezug auf die Vertrauenswürdigkeit von Regelmustern beim Wechsel von Elementen der linken Seite auf die rechte Seite der Regel gilt das in Abschnitt 2.2.2 gesagte. Zusammen mit der Forderung nach Bildung rechter Teilmengen (Zeilen 4 bis 9) wird diese Idee von Zeile 13 umgesetzt.

Definition 2.8 (Zahl bildbarer serieller Regelmuster)
Sei L die Länge des zugrunde liegenden seriellen Alarmmusters XY_{ser}, so gilt für die Zahl der aus XY_{ser} bildbaren Regelmuster G:

$$G(L) = L - 1$$

Definition 2.8 erklärt sich in einfacher Weise dadurch, dass bei der Bildung serieller Regelmuster das zugrunde liegende Alarmmuster geteilt wird, ohne die Reihenfolge der Einzelereignisse zu verändern. In Verbindung mit der Forderung, dass auf beiden Seiten der Regel keine leeren Teilmengen vorkommen dürfen, ergeben sich somit genau $L - 1$ verschiedene Möglichkeiten dies zu tun.

2.3 Nachbearbeitung von Regelmustern

Methoden wie Alarmmuster- und Regelmuster-Findung kommen zum Einsatz, um die Menge zu verarbeitender Daten zu reduzieren und ihren Nutzwert durch Anhebung des Abstraktionsniveaus zu erhöhen. Paradoxer Weise können diese Methoden selbst eine große Menge an Daten erzeugen, so dass ein neues Problem entsteht: Da in einer gegebenen Datenbasis leicht tausende oder gar Millionen von Alarm- und damit auch von Regelmustern enthalten sein können, wird bereits durch die hohe Zahl eine geeignete Präsentation dieser Muster für den Anwender unmöglich gemacht. Im Folgenden sollen Methoden aufgezeigt werden, die es ermöglichen, durch Zusammenfassung und Gruppierung von Regelmustern den erzeugten Datenbestand überschaubar – und damit nutzbar – zu machen.

2.3.1 Bildung repräsentativer linker Regelsets

Im folgenden Abschnitt werden die Möglichkeiten der Redundanz-Reduktion zunächst aus dem Blickwinkel der Vorhersage weiterer Ereignisse durch bereits eingetretene Ereignisse untersucht. Diese Betrachtungsweise äußert sich in der Frage: Was folgt auf X, also: $X \Rightarrow ?$

Definition 2.9 (Linkes Regelset)
Sei R ein Set von Regelmustern mit identischer linker Regelseite X, so dass gilt:

$$R = \{X \Rightarrow Y_i \mid i = 1, \ldots, n\}$$

Das heißt, jedes Regelmuster in R enthält unter identischen Voraussetzungen eine Vorhersage über das weitere Auftreten von Ereignissen. Eine grundsätzliche Annahme besteht nun darin, dass innerhalb eines linken Regelsets eine große Redundanz bezüglich dieser Voraussagen existiert. Eine Möglichkeit diese Redundanz zu beseitigen besteht in der Bildung *repräsentativer linker Regelsets* (abkürzend wird dieser Vorgang auch als *L-Methode* bezeichnet).

Definition 2.10 (Repräsentatives linkes Regelset)
Sei R ein Set von Regelmustern, dann ist $S \subseteq R$ ein repräsentatives linkes Regelset wenn gilt:

$$\bigcup_{(X \Rightarrow Y) \in R} XY = \bigcup_{(X \Rightarrow Y) \in S} XY$$

Definition 2.10 bedeuted, dass die Regelmuster in S die zugrunde liegenden Alarmmuster XY in gleicher und vollständiger Weise beschreiben, wie dies die Regelmuster in R tun.

Definition 2.11 (Voraussagekraft von Regelmustern)
Seien A und B zwei Regelmuster mit identischer linker Regelseite X, die die geforderten Werte für min_freq und min_conf erfüllen. Dann besitzt Regelmuster A die höhere Voraussagekraft gegenüber Regelmuster B, wenn für die rechten Regelseiten Y_A und Y_B der Muster gilt:

$$Y_B \subset Y_A$$

Die zu lösende Aufgabe besteht nun darin, aus einer gegebenen Menge M von Regelmustern diejenigen Muster zu entfernen, deren Voraussagekraft von mindestens einem weiteren Regelmuster mit identischer linker Regelseite übertroffen wird. Beispiel 2.7 veranschaulicht diese Idee. Die zugrunde liegende Vorgehensweise ist dabei folgende: Eine gegebene Menge von Regelmustern wird in Regelsets nach Definition 2.9 unterteilt. Aus diesen Regelsets werden anschließend alle redundanten Muster entfernt. Die verbleibenden Muster – sie entsprechen den repräsentativen linken Regelsets – werden ausgegeben. Algorithmus 2.3 implementiert diesen Ablauf.

Beispiel 2.7 (Voraussagekraft von Regelmustern)
Seien $A \Rightarrow B$ und $A \Rightarrow (B, C)$ zwei Regelmuster, die die geforderten Werte für min_freq und min_conf erfüllen. Regelmuster $A \Rightarrow (B, C)$ besitzt gegenüber Regelmuster $A \Rightarrow B$ die höhere Voraussagekraft, da B echte Teilmenge von (B, C) ist.

Funktionsnachweis Algorithmus 2.3
Der Algorithmus funktioniert, da alle durch M erfassten Alarmmuster auch durch S erfasst werden. Die Regelmuster in M werden zunächst nach ihrer linken Regelseite geordnet (Zeilen 5, 7) in Regelsets einsortiert (Zeilen 9 bis 12). Dabei wird sichergestellt, dass nur Regelmuster aus M in R vorhanden sind (Zeile 2) und kein Regelmuster aus M unberücksichtigt bleibt (Zeile 4). In S können wiederum nur Muster aus R gelangen (Zeile 14). Alle Regelmuster, für die keine Muster mit höherer Voraussagekraft existieren, werden in S aufgenommen (Zeilen 17, 18). Gleichzeitig bleiben alle Muster unberücksichtigt, für die in S bereits ein

Eingabe: Menge M von Regelmustern
Ausgabe: alle aus M bildbaren repräsentativen linken Regelsets S

1: // Generieren aller aus M bildbaren linken Regelsets R
2: $R := \emptyset$;
3: $k := 0$;
4: **while** $M \neq \emptyset$ **do**
5: $rule := (X \Rightarrow Y)$ mit $rule$ beliebiges Regelmuster aus M;
6: $k := k + 1$;
7: $R[k] := R[k] \cup rule$;
8: $M := M \setminus rule$;
9: **for** alle $(X \Rightarrow Y)$ aus M mit $|X| = |X_{rule}|$ **do**
10: **if** $X = X_{rule}$ **then**
11: $R[k] := R[k] \cup (X \Rightarrow Y)$;
12: $M := M \setminus (X \Rightarrow Y)$;
13: // Bilden repräsentativer linker Regelsets S
14: $S := \emptyset$;
15: **for** alle $R[k]$ **do**
16: **while** $R[k] \neq \emptyset$ **do**
17: $rule := (X \Rightarrow Y)$, so dass gilt $|Y| = |Y|_{max}$;
18: $S[k] := S[k] \cup rule$;
19: $R[k] := R[k] \setminus rule$;
20: **for** alle $(X \Rightarrow Y_j) \in R[k]$ mit $|Y_j| < |Y_{rule}|$ **do**
21: **if** $Y_j \subset Y_{rule}$ **then**
22: $R[k] := R[k] \setminus (X \Rightarrow Y_j)$;
23: **for** alle $S[k]$ **do**
24: Ausgabe aller in $S[k]$ befindlichen Regelmuster

Algorithmus 2.3: Bildung repräsentativer linker Regelsets

Muster höherer Voraussagekraft existiert (Zeilen 21, 22). Beim Test auf Aufnahmefähigkeit in S wird sichergestellt, dass alle Regelmuster aus R berücksichtigt werden (Zeilen 15, 16). Die Abarbeitung wird beendet mit der vollständigen Ausgabe der Ergebnisse (Zeilen 23, 24).

2.3.2 Bildung repräsentativer rechter Regelsets

Eine andere Herangehensweise mit der Möglichkeit zur Redundanz-Reduktion besteht in der Rückführung von Ereignissen auf ihre Ursachen. Formuliert werden kann dies in der Frage: Was muss passieren damit Y eintritt, also: $? \Rightarrow Y$. Dieser Ansatz wird im Folgenden näher beleuchtet.

Definition 2.12 (Rechtes Regelset)
Sei R ein Set von Regelmustern mit identischer rechter Regelseite Y, so dass gilt:

$$R = \{X_i \Rightarrow Y \mid i = 1, \ldots, n\}$$

Das heißt, jedes Regelmuster in R enthält unter verschiedenen Voraussetzungen eine identische Aussage über das weitere Auftreten von Ereignissen. Eine grundsätzliche Annahme besteht – wie in ähnlicher Weise bereits im Abschnitt 2.3.1 erläutert – darin, dass innerhalb eines rechten Regelsets eine große Redundanz bezüglich der verwendeten Voraussetzungen zur Vorhersage eines Ereignisses oder einer Ereignissequenz existiert. Eine Möglichkeit diese Redundanz zu beseitigen besteht in der Bildung *repräsentativer rechter Regelsets* (abkürzend wird dieser Vorgang auch als *R-Methode* bezeichnet).

Definition 2.13 (Repräsentatives rechtes Regelset)
Sei R ein Set von Regelmustern, dann ist $Z \subseteq R$ ein repräsentatives rechtes Regelset wenn gilt:

$$\bigcup_{(X \Rightarrow Y) \in R} XY = \bigcup_{(X \Rightarrow Y) \in Z} XY$$

Definition 2.13 bedeuted, dass die Regelmuster in Z die zugrunde liegenden Alarmmuster XY in gleicher und vollständiger Weise beschreiben, wie dies die Regelmuster in R tun.

Definition 2.14 (Detektionskraft von Regelmustern)
Seien A und B zwei Regelmuster mit identischer rechter Regelseite Y, die die geforderten Werte für min_freq und min_conf erfüllen. Dann besitzt Regelmuster A die höhere Detektionskraft gegenüber Regelmuster B, wenn für die linken Regelseiten X_A und X_B der Muster gilt:

$$X_A \subset X_B$$

Der Unterschied in der Qualität der Regelmuster liegt demnach darin, dass Muster A in der Lage ist, mit geringeren quantitativen Voraussetzungen die gleichen qualitativen Aussagen wie Muster B zu treffen. Die zu lösende Aufgabe kann nunmehr wie folgt formuliert werden: Bei einer gegebenene Menge M von Regelmustern sollen alle Muster aus M entfernt werden, deren Detektionskraft von mindestens einem weiteren Regelmuster mit identischer rechter Regelseite übertroffen wird. Beispiel 2.8 veranschaulicht diese Idee. Die Menge M wird nach Definition 2.12 in Regelsets unterteilt. Aus diesen Regelsets werden anschließend alle redundanten Muster entfernt. Die verbleibenden Muster werden als Ergebnis ausgegeben, da sie den repräsentativen rechten Regelsets entsprechen. Algorithmus 2.4 implementiert diesen Ablauf.

Beispiel 2.8 (Detektionskraft von Regelmustern)
Seien $A \Rightarrow C$ und $(A, B) \Rightarrow C$ zwei Regelmuster, die die geforderten Werte für min_freq und min_conf erfüllen. Regelmuster $A \Rightarrow C$ besitzt gegenüber Regelmuster $(A, B) \Rightarrow C$ die höhere Detektionskraft, da A echte Teilmenge von (A, B) ist.

Funktionsnachweis Algorithmus 2.4
Der Algorithmus funktioniert, da alle durch M erfassten Alarmmuster auch durch Z erfasst werden. Die Regelmuster in M werden zunächst nach ihrer rechten Regelseite geordnet (Zeilen 5, 7) in Regelsets einsortiert (Zeilen 9 bis 12). Dabei wird sichergestellt, dass nur Regelmuster aus M in R vorhanden sind (Zeile 2) und kein Regelmuster aus M unberücksichtigt bleibt (Zeile 4). In Z können wiederum nur Muster aus R gelangen (Zeile 14). Alle Regelmuster, für die keine Muster mit höherer Detektionskraft existieren, werden in Z aufgenommen (Zeilen 17, 18). Gleichzeitig bleiben alle Muster unberücksichtigt, für die in Z bereits ein Muster höherer Detektionskraft existiert (Zeilen 21, 22). Beim Test auf Aufnahmefähigkeit in Z wird sichergestellt, dass alle Regelmuster aus R berücksichtigt werden (Zeilen 15, 16). Die Abarbeitung wird beendet mit der vollständigen Ausgabe der Ergebnisse (Zeilen 23, 24).

Eingabe: Menge M von Regelmustern
Ausgabe: alle aus M bildbaren repräsentativen rechten Regelsets Z

1: // Generieren aller aus M bildbaren rechten Regelsets R
2: $R := \emptyset$;
3: $k := 0$;
4: **while** $M \neq \emptyset$ **do**
5: $rule := (X \Rightarrow Y)$ mit $rule$ beliebiges Regelmuster aus M;
6: $k := k + 1$;
7: $R[k] := R[k] \cup rule$;
8: $M := M \setminus rule$;
9: **for** alle $(X \Rightarrow Y)$ aus M mit $|Y| = |Y_{rule}|$ **do**
10: **if** $Y = Y_{rule}$ **then**
11: $R[k] := R[k] \cup (X \Rightarrow Y)$;
12: $M := M \setminus (X \Rightarrow Y)$;
13: // Bilden repräsentativer rechter Regelsets Z
14: $Z := \emptyset$;
15: **for** alle $R[k]$ **do**
16: **while** $R[k] \neq \emptyset$ **do**
17: $rule := (X \Rightarrow Y)$, so dass gilt $|X| = |X|_{min}$;
18: $Z[k] := Z[k] \cup rule$;
19: $R[k] := R[k] \setminus rule$;
20: **for** alle $(X_j \Rightarrow Y) \in R[k]$ mit $|X_j| > |X_{rule}|$ **do**
21: **if** $X_{rule} \subset X_j$ **then**
22: $R[k] := R[k] \setminus (X_j \Rightarrow Y)$;
23: **for** alle $Z[k]$ **do**
24: Ausgabe aller in $Z[k]$ befindlichen Regelmuster

Algorithmus 2.4: Bildung repräsentativer rechter Regelsets

2.3.3 Kombinationen von Reduktions-Methoden

Nach Bildung der repräsentativen linken und rechten Regelsets bleibt die Frage, ob eine Kombination beider Methoden sinnvoll ist. Im Besonderen ist zu klären, in welcher Reihenfolge die verschiedenen Ansätze für die Reduktion von Regelmustern auf die Sets angewendet werden sollen. Da es sich hierbei um anwendungsbezogene Probleme handelt, werden diese Fragen nur bei Kenntnis aller Randbedingungen des jeweiligen Anwendungsfalls zufriedenstellend beantworten werden können. Eine vielversprechende Vorgehensweise besteht beispielsweise darin, in einem ersten Schritt aus dem gegebenen Bestand an Regelmustern repräsentative rechte Regelsets zu bilden. In einem zweiten Schritt werden aus diesen, bereits einer Reduktion unterworfenen Mustern, repräsentative linke Regelsets gebildet. Der Grund diese *rechts-links Methode* (*RL-Methode*) der ebenfalls möglichen *links-rechts Methode* vorzuziehen besteht darin, dass bei dieser Variante die jeweiligen Seiten der Regelmuster erst auf Allgemeinheit und anschließend auf Spezialisiertheit untersucht werden. Durch dieses Vorgehen wird ein möglichst hoher Grad an Reduktion erreicht, da Muster, die zur Aussonderung anderer Muster im zweiten Bearbeitungsschritt notwendig sind, nicht vorzeitig aus dem Datenbestand entfernt werden. Ein weiterer Vorteil gegenüber der weiter unten vorgestellten Vergleichsmethode ist die Tatsache, dass die Reduktion garantiert verlustfrei bleibt. Es werden also alle Alarmmuster, die durch den ursprünglichen Bestand an Regelmustern beschrieben wurden, auch durch den aus der Anwendung der rechts-links Methode entstandenem, reduzierten Bestand an Regelmustern beschrieben (gleiches gilt für die links-rechts Methode). Der Grund liegt in der Eigenschaft repräsentativer Regelsets, Redundanzen ohne Verlust an „Alarmmuster-Abdeckung“ reduzieren zu können, die auch bei wiederholter Anwendung erhalten bleibt.

Ein ebenfalls möglicher Ansatz zur weiteren Reduktion von Redundanzen in Beständen von Regelmustern besteht im Vergleich der zuvor gebildeten repräsentativen linken und rechten Regelsets. Bei dieser *Vergleichsmethode* werden nur die Muster in den endgültigen Bestand übernommen, die in beiden Sets vorkommen. Vorteil dieser Methode ist die Einfachheit der Realisierung, Nachteil ist die entstehende Unsicherheit bezüglich der Abdeckung der Alarmmuster, da eine verlustfreie Reduktion auf diese Weise nicht mehr garantiert werden kann.

2.4 Klassifizierungen von Regelmustern

Trotz der Anwendung der in den vorhergehenden Abschnitten beschriebenen Möglichkeiten der Zusammenfassung von Regelmustern der Form $X \Rightarrow Y$ zu repräsentativen Regelsets, kann der verbleibende Bestand an Regelmustern weiterhin sehr umfangreich sein.

Regelmuster können basierend auf ihrer vermuteten Interessantheit geordnet werden. Eine offensichtliche Möglichkeit der Messung, ob ein Muster interessant ist oder nicht, besteht in der Auswertung der Parameter min_conf und min_freq. Denkbar sind weiterhin komplexere Arten von Messungen um beispielsweise die statistische Signifikanz von Regelmustern zu bestimmen. Derartige Messungen sind sinnvoll zur Ermittlung der interessantesten Muster, sie helfen aber nicht bei der Präsentation von großen Regelsammlungen [TKR95].

Ziel der folgenden beiden Abschnitte ist es, durch Ordnen und Gruppieren der Regelmuster einen höheren Grad an Übersicht und Verständlichkeit zu erreichen. In Abschnitt 2.4.3 wird anschließend die Möglichkeit untersucht, mit Hilfe von Templates die für den Anwender interessanten Regelmuster von den weniger interessanten Mustern eines Bestandes zu trennen.

2.4.1 Bildung von Regelgruppen durch Bereichsbildung

Eine Methode der Strukturierung von Regelsets ist die Bereichsbildung. Bei der Bereichsbildung werden diejenigen Regelmuster in Gruppen zusammengefasst, die gleiche Aussagen über die rechte oder linke Regelseite machen. Dies entspricht der schon bekannten Unterteilung in rechte oder linke Regelsets. Erscheint diese Strukturierung als zu fein – nähert sich also die Zahl bildbarer Regelsets der Zahl vorhandener Regelmuster zu stark an – muss ein gewisser Unterschied zwischen den *signifikanten Regelseiten* zugelassen werden. Die signifikante Regelseite entspricht dabei der zur Bildung der Regelsets benutzten Seite der Regel. Der Unterschied – oder die Entfernung – zwischen zwei Regelmustern kann auf verschiedene Weisen definiert werden. Eine Möglichkeit besteht bespielsweise in der Anzahl der Spalten, in denen sich die Muster unterscheiden.

Definition 2.15 (Entfernung linker Regelmuster)
Seien $X_1 \Rightarrow U$ und $X_2 \Rightarrow V$ zwei Regelmuster mit ähnlicher linker Regelseite. Dann beträgt die Entfernung d zwischen diesen Mustern:

$$d\big(X_1 \Rightarrow U, X_2 \Rightarrow V\big) = ||X_1|| + ||X_2|| - 2||X_1X_2||$$

Definition 2.15 besagt, dass die Entfernung der Regelmuster der Summe der „belegten“ Positionen der signifikanten Regelseiten der Einzelmuster entspricht, von der die Zahl der mit gleichen Werten belegten Positionen in doppelter Gewichtung abgezogen wird. Beispiel 2.9 erläutert diesen Gedanken.

Beispiel 2.9 (Entfernung linker Regelmuster)
Seien $(1,2) \Rightarrow 4$ und $(1,3) \Rightarrow 5$ zwei Regelmuster. Dann ergibt sich die gesuchte Entfernung zwischen den Mustern zu:

$$d\big((1,2) \Rightarrow 4, (1,3) \Rightarrow 5\big) = 2 + 2 - 2 \cdot 1 = 2$$

Eine anschaulichere Darstellung ist mit Hilfe folgender Tabelle möglich, bei der Unterschiede zwischen den Mustern an der Stelle n mit $d_n = 1$ gekennzeichnet werden, sowie Gemeinsamkeiten mit $d_n = 0$.

$(1,2) \Rightarrow 4$	1	2	
$(1,3) \Rightarrow 5$	1		3
d_n	0	1	1
n	1	2	3

Damit ergibt sich die gesuchte Entfernung zwischen den Mustern wiederum zu:

$$d\big((1,2) \Rightarrow 4, (1,3) \Rightarrow 5\big) = \sum_{n=1}^{3} d_n = 2$$

In Analogie zur Definition 2.15 gilt für Muster mit signifikanter rechter Regelseite:

Definition 2.16 (Entfernung rechter Regelmuster)
Seien $U \Rightarrow Y_1$ und $V \Rightarrow Y_2$ zwei Regelmuster mit ähnlicher rechter Regelseite. Dann beträgt die Entfernung d zwischen diesen Mustern:

$$d(U \Rightarrow Y_1, V \Rightarrow Y_2) = ||Y_1|| + ||Y_2|| - 2||Y_1 Y_2||$$

Die praktische Umsetzung der aufgeführten Methoden zur Bereichsbildung lässt sich durch Übernahme und Modifikation des ersten Teils der bereits für die Bildung von Regelsets benutzten Algorithmen 2.3 und 2.4 erreichen.

Funktionsnachweis Algorithmus 2.5
Der Algorithmus funktioniert, da alle in M vorhandenen Regelmuster in genau einem der gebildeten Bereichen $G[k]$ vertreten sind. Für die zu bildenden Regelgruppen wird zunächst ein Referenzmuster gewählt (Zeile 4), gegen das anschließend alle in M vorkommenden Muster (Zeile 8) getestet werden (Zeile 9). Verläuft der Test positiv, ist also die Entfernung zwischen den Mustern nicht zu groß, wird das untersuchte Regelmuster der Gruppe zugefügt (Zeile 10) und aus M entfernt (Zeile 11). Alle Muster in M werden genau einem Bereich zugewiesen (Zeilen 6, 7 und 10, 11).

Eingabe: Menge M von Regelmustern, Entfernungswert $max\_distance$
Ausgabe: alle aus M bildbaren Gruppen $G[k]$ von Regelmustern

1: $G := \emptyset$;
2: $k := 0$;
3: **while** $M \neq \emptyset$ **do**
4: $rule := (X \Rightarrow Y)$ mit $rule$ beliebiges Regelmuster aus M;
5: $k := k + 1$;
6: $G[k] := G[k] \cup rule$;
7: $M := M \setminus rule$;
8: **for** alle $(X \Rightarrow Y)$ aus M **do**
9: **if** $distance(rule, X \Rightarrow Y) \leq max\_distance$ **then**
10: $G[k] := G[k] \cup (X \Rightarrow Y)$;
11: $M := M \setminus (X \Rightarrow Y)$;
12: **for** alle $G[k]$ **do**
13: Ausgabe aller in $G[k]$ befindlichen Regelmuster

Algorithmus 2.5: Gruppierung von Regelmustern ohne Vorgaben

Dabei wird sichergestellt, dass nur Muster aus M in $G[k]$ vorhanden sind (Zeile 1) und kein Muster unberücksichtigt bleibt (Zeile 3). Die Abarbeitung wird beendet mit der vollständigen Ausgabe der Ergebnisse (Zeilen 12, 13).

Algorithmus 2.5 ist gleichermaßen zur Bildung von Gruppen basierend auf der linken wie rechten Regelseite als signifikanter Seite geeignet, da die Unterschiede in der Bereichsbildung einzig in der Berechnung von $distance(rule, X \Rightarrow Y)$ liegen.

Ein offensichtlicher Nachteil dieses Verfahrens ist es, dass die gebildeten Gruppen nicht mehr in jedem Fall eindeutig sind, sobald die zulässige Entfernung der Gruppenmitglieder vom Referenzmuster ≥ 1 ist. Beispiel 2.10 verdeutlicht dieses Verhalten.

Beispiel 2.10 (Gruppierung von Regelmustern ohne Vorgaben) Seien $(1, 1, 1) \Rightarrow Y_1$, $(1, 1, 2) \Rightarrow Y_2$ und $(1, 2, 2) \Rightarrow Y_3$ drei Regelmuster. Sei $max\_distance = 1$. Dann sind die aus diesen Mustern bildbaren Regelgruppen oder Bereiche:

Referenzmuster	$(1,1,1) \Rightarrow Y_1$	$(1,1,2) \Rightarrow Y_2$	$(1,2,2) \Rightarrow Y_3$
Gruppe 1	$(1,1,1) \Rightarrow Y_1$ $(1,1,2) \Rightarrow Y_2$	$(1,1,1) \Rightarrow Y_1$ $(1,1,2) \Rightarrow Y_2$ $(1,2,2) \Rightarrow Y_3$	$(1,1,2) \Rightarrow Y_2$ $(1,2,2) \Rightarrow Y_3$
Gruppe 2	$(1,2,2) \Rightarrow Y_3$	—	$(1,1,1) \Rightarrow Y_1$

Je nach Aussehen des Referenzmusters – welches bei Anwendung des obigen Algorithmus zufällig gewählt wird – besitzen die resultierenden Gruppen eine unterschiedliche Zusammensetzung. Es muss somit für jeden einzelnen Anwendungsfall geprüft werden, ob das beschriebene Verfahren sinnvoll eingesetzt werden kann oder nicht.

Ein weiterer möglicher Ansatz für die Gruppierung von Regelmustern besteht in der Vorgabe von Referenzmustern durch den Anwender. Aus dem vorliegenden Bestand an Mustern werden dann nur diejenigen Regelmuster herausgesucht und zu Gruppen gebildet, die zu den angegebenen Referenzmustern passen. Die einzige notwendige Änderung von Algorithmus 2.5 betrifft Zeile 4:

Eingabe: Menge M von Regelmustern, Menge $V \subset M$ von Referenzmustern, Entfernungswert $max\_distance$
Ausgabe: alle mittels V bildbaren Gruppen $G[k]$ von Regelmustern

// Übernehme Algorithmus 2.5 und ersetze Zeile 4 durch die Zeilen 4a, 4b:
$rule := (X \Rightarrow Y)$ mit $rule$ beliebiges Regelmuster aus V;
$V := V \setminus rule$;

Algorithmus 2.6: Gruppierung von Regelmustern durch Vorgaben

Funktionsnachweis Algorithmus 2.6
Der Algorithmus funktioniert, da das für Algorithmus 2.5 Gesagte in gleicher Weise gilt (bei Ersatz von Zeile 4 durch die Zeilen 4a, 4b). Insbesondere ist sichergestellt, dass die Menge der Referenzmuster Teilmenge der ursprünglich getesteten Muster ist (Forderung von $V \subset M$).

Der Nachteil dieses Verfahrens besteht in der Vorgabe der Referenzmuster selbst und dem damit notwendigen Eingreifen des Anwenders. Zugleich ergibt sich die Schwierigkeit bei großen Datenbeständen – und nur bei diesen erscheint eine Gruppierung vorteilhaft – sinnvolle Referenzmuster zu finden. Ein weitaus praktikablerer Weg bei der anwenderbezogenen Unterteilung von Regelmustern in Gruppen besteht in der Anwendung von Templates. Diese Idee wird in Abschnitt 2.4.3 erläutert.

2.4.2 Bildung von Regelgruppen durch Zuordnung

Ein weiterer Ansatz zur Bildung von Regelgruppen besteht in der Zuordnung von Regelmustern zu einem – die Eigenschaften der Gruppe bestimmenden – Regelmuster (*Gruppenmuster*). Im Unterschied zu der Vorgehensweise des vorhergehenden Abschnittes ist dabei die Zuordnung eines Musters zu mehreren Gruppen möglich. Ziel ist es, Gruppen von Regelmustern zu bilden, die in einer vom Anwender zu definierenden Weise Ähnlichkeiten untereinander aufweisen. Diese Ähnlichkeiten können beispielsweise darin bestehen, dass sich alle Muster einer Gruppe aus einer bestimmten Menge oder Teilmenge von Alarmmeldungen zusammensetzen. Die jeweils zur Anwendung kommende Menge von Alarmen kann von dem zuvor erwähnten Gruppenmuster definiert werden. Algorithmus 2.7 implementiert diese Idee.

Eingabe: Menge M von Regelmustern mit den Eigenschaften $compl = x$ und $groups = 0$
Ausgabe: alle mittels M bildbaren Zuordnungen $G[k]$ von Regelmustern

1: $G := \emptyset$;
2: $k := 0$;
3: **while** $M \neq \emptyset$ **do**
4: $rule := (X \Rightarrow Y)$, so dass gilt $|rule| = max$, $rule_{compl} = max$ in M;
5: $M := M \setminus rule$;
6: **if** $rule_{groups} = 0$ **then**
7: $k := k + 1$;
8: $G[k] = G[k] \cup rule$;
9: **for** alle $(X \Rightarrow Y)$ aus M **do**
10: **if** $(X \Rightarrow Y) \subseteq rule$ **then**
11: $(X \Rightarrow Y)_{groups} = (X \Rightarrow Y)_{groups} + 1$;
12: $G[k] := G[k] \cup (X \Rightarrow Y)$;
13: **if** $|X \Rightarrow Y| = |rule|$ **then**
14: $M := M \setminus (X \Rightarrow Y)$;
15: **for** alle $G[k]$ **do**
16: Ausgabe aller in $G[k]$ befindlichen Regelmuster

Algorithmus 2.7: Gruppierung von Regelmustern durch Zuordnung

Die *Komplexität* eines Regelmusters entspricht der Anzahl verschiedener Alarmmeldungen, aus denen das Muster besteht. In dem vorliegenden Algorithmus wird die Komplexität und die Länge der Regelmuster benutzt, um die Gruppenmuster festzulegen. Um eine neue Gruppe „aufmachen“ zu dürfen, muss ein Regelmuster in dem zugrunde liegenden Regelmuster-Bestand bei maximaler Länge die höchste Komplexität aufweisen. Weiterhin darf ein derartiges Muster noch keiner Gruppe zugeordnet

worden sein. Gibt es mehrere Muster die diesen Kriterien entsprechen, wird das erste mögliche Muster zum Gruppenmuster. Trotz dieses Vorgehens sind die gebildeten Gruppen eindeutig, da nur die Alarmmeldungen, nicht aber ihre Anordnung oder Anzahl innerhalb eines Musters als Auswahlkriterium für die Gruppenmitglieder dienen. Die Zählvariable *groups* kennzeichnet die Zahl der Gruppen, in denen das betreffende Regelmuster vorkommt. Gruppenmuster sind an $groups = 0$ erkennbar.

Beispiel 2.11 (Komplexität und Länge eines Regelmusters)
Es sei folgendes Regelmuster gegeben:

```
2244 2244 6800 8035 -> 8035
```

Dann beträgt die *Komplexität* des Regelmusters $compl = 3$, da genau 3 verschiedene Alarmmeldungen in dem Muster vorkommen. Die *Länge* des Musters ist $l = 5$, da das Muster aus 5 Alarmmeldungen besteht.

Würde das in Beispiel 2.11 vorgestellte Regelmuster Gruppenmuster werden, so würde die dadurch entstehende Auswahlmenge A an Alarmmeldungen durch $A = \{2244, 6800, 8035\}$ definiert werden. Regelmuster, die dieser Gruppe zugeordnet werden sollen, müssen also aus Alarmmeldungen bestehen, die Teilmenge von A sind.

Funktionsnachweis Algorithmus 2.7
Der Algorithmus funktioniert, da alle Regelmuster aus M einer Gruppe zugeordnet werden (Zeilen 8, 12) und keine Muster unberücksichtigt bleiben (Zeile 3). Gleichzeitig ist sichergestellt, dass nur Regelmuster aus M in den gebildeten Gruppen $G[k]$ vorkommen (Zeile 1). Neue Gruppen können nur von den Regelmustern gebildet werden, die den Kriterien von maximaler Komplexität und Länge genügen (Zeile 4), sowie selbst noch kein Mitglied einer anderen Gruppe sind (Zeile 6). Alle Regelmuster, die die geforderte Ähnlichkeit zum Gruppenmuster aufweisen (Zeile 10) werden der entprechenden Gruppe zugeordnet (Zeile 12). Der Überprüfung werden wiederum alle Muster des Datenbestandes unterzogen (Zeile 9). Mehrfachzuordnungen werden ermöglicht, da nur diejenigen Regelmuster aus M entfernt werden, die gleiche Länge zum Gruppenmuster aufweisen (Zeilen 13, 14). Die Abarbeitung wird beendet mit der vollständigen Ausgabe der Ergebnisse (Zeilen 15, 16).

Der Vorteil des aufgezeigten Vorgehens gegenüber dem in Abschnitt 2.4.1 vorgestellten Verfahren ist die Eindeutigkeit der gebildeten Gruppen, die auch ohne Eingreifen des Anwenders erreicht wird. Nachteilig ist die entstehende Redundanz, da Regelmuster mehr als einmal im Regelmuster-Bestand vorhanden sein können.

2.4.3 Templates

Interessante und uninteressante Klassen von Regelmustern können mit Hilfe von *Templates* spezifiziert werden. Die Idee von Templates wurde in [KMR94] vorgestellt. Templates beschreiben ein Set von Regeln durch die Festlegung, welche Attribute die Einzelereignisse der linken Regelseite und welche Attribute die Einzelereignisse der rechten Regelseite aufweisen müssen. Ein Einzelereignis bezeichnet dabei beispielsweise eine Alarmmeldung, wenn das zugehörige Regelmuster aus einem Alarmmuster gebildet wurde.

Beispiel 2.12 (Attribute von Regelmustern)
Alarmmeldungen in Mobilfunknetzen lassen sich abhängig von ihrer Bedeutung verschiedenen Kategorien zuordnen. Diese Kategorien können als Attribute für die Bildung von Templates benutzt werden. Für Regelmuster, die aus Alarmmeldungen bestehen, könnten sich beispielsweise folgende Attribute ergeben:

Notice, Disturbance, Diagnosis, Failure, BTS-Alarms, MSC-Alarms

Erscheint diese Unterteilung als zu unhandlich, kann durch den Anwender eine geeignete Hierarchie von Attributen mittels Einteilung in Klassen geschaffen werden:

Beispiel 2.13 (Klassifizierung von Regelmuster-Attributen)
Eine mögliche Einteilung der Attribute aus Beispiel 2.12 wäre folgendermaßen: Jedes Attribut gehört zur Klasse ‚*all_alarms*‘. Weniger wichtige Meldungen werden in der Klasse ‚*less_important*‘, wichtige Meldungen in der Klasse ‚*important*‘ zusammengefasst. Die Klassifizierung der Attribute könnte demzufolge so aussehen:

$\{$*Notice, Disturbance, Diagnosis*$\} \subseteq$ *less_important* $\subset$ *all_alarms*

$\{$*Failure, BTS-Alarms, MSC-Alarms*$\} \subseteq$ *important* $\subset$ *all_alarms*

Ein Template lässt sich jetzt als ein Ausdruck folgender Form darstellen:

Definition 2.17 (Template)

$$T_1, \ldots, T_k \Rightarrow T_{k+1}, \ldots, T_n$$

Jedes T_i entspricht einem Ausdruck der Form C^+ oder C^*, wobei C der Name einer Klasse ist, die im einfachsten Fall aus nur einer Alarmmeldung besteht. Die Ausdrücke C^+ und C^* korrespondieren mit *einer oder mehr* und *null oder mehr* Instanzen der Klasse C. Eine Regel $X \Rightarrow Y$ genügt einem Template, wenn sie als eine Instanz des Templates angesehen werden kann.

Beispiel 2.14 (Instanz eines Templates)
Es sei folgender Zusammenhang zwischen Alarmmeldungen und ihren Attributen sowie den zugehörigen Alarmnummern bekannt:

Nummer	Attribut	Alarmtext
2567	Failure	Failure in Sending System Information to BTS Site
7706	BTS-Alarms	BTS O&M Link Failure

Angenommen ein Operator eines Betreibers für Mobilfunknetze ist interessiert an der Alarmmeldung 2567. Weiterhin interessieren ihn Zusammenhänge zwischen diesem Alarm und anderen wichtigen Alarmen. Der Operator sucht also beispielsweise nach Regelmustern, die die Alarmmeldung 2567 auf der rechten Seite der Regel besitzen und auf der linken Seite Meldungen der Klasse *important* besitzen. Somit lautet das zugehörige Template:

$$important^+, all\_alarms^* \Rightarrow 2567^+$$

und ein Regelmuster der Form

$$7706 \Rightarrow 2567$$

würde unter Berücksichtigung der in Beispiel 2.13 dargelegten Klassifizierung diesem Template genügen.

Mittels Templates ist ein Anwender in der Lage anzugeben, was interessant ist und was nicht. Um als interessant zu gelten, muss ein Regelmuster auf ein *einschließendes Template* passen. Passt ein Muster auf ein *ausschließendes Template*, wird es als uninteressant eingestuft. Um einem Anwender präsentiert werden zu können, muss ein Regelmuster auf mindestens ein einschließendes Template passen, während es gleichzeitig von keinem ausschließenden Template erfasst werden darf.

Beispiel 2.15 (Ein- und ausschließende Templates 1)
Bei Benutzung des *einschließenden* Templates

$$important^+, all\_alarms^* \Rightarrow 2567^+$$

wird ein Anwender eine gewisse Anzahl von Regelmustern erhalten. Einige dieser Muster werden vielleicht als uninteressant eingestuft werden. Durch Hinzunahme des *ausschließenden* Templates

$$less\_important^+, all\_alarms^* \Rightarrow all\_alarms^+$$

können alle die Muster herausgefiltert werden, deren linke Regelseite mindestens einen weniger wichtigen Alarm enthält.

Eine weitere interessante Frage könnte lauten: Welche Alarmmeldungen folgen, wenn Alarm 2567 auf der linken Seite der Regel erscheint?

Beispiel 2.16 (Ein- und ausschließende Templates 2)
Bei Benutzung des *einschließenden* Templates

$$2567^+, all\_alarms^* \Rightarrow all\_alarms^+$$

und der Hinzunahme der *ausschließenden* Templates

$$less\_important^+, all\_alarms^* \Rightarrow all\_alarms^+$$

und

$$all\_alarms^+ \Rightarrow less\_important^+, all\_alarms^*$$

werden alle die Muster erfasst, deren linke Regelseiten die Alarmmeldung 2567 enthalten und in denen nur wichtige Alarmmeldungen vorkommen.

Eine weit weniger restrektive Handhabung kann mit dem Template

$$all\_alarms^+ \Rightarrow less\_important^+$$

als zweiter ausschließender Bedingung bewirkt werden, bei dem nur die Muster ausgesondert werden, die ausnahmslos weniger wichtige Meldungen auf der rechten Seite der Regel beinhalten.

Templates eröffnen bei Anwendung auf Regelmuster einen Weg, Hintergrundwissen zu nutzen. Ausschließende Templates können beispielsweise zur Ausfilterung von Mustern benutzt werden, die bereits bekannte Eigenschaften repräsentieren.

Templates stellen ein sehr mächtiges Mittel zur Nachbearbeitung von Regelmuster-Beständen dar. Ihr ‚Gebrauchswert' hängt dabei ganz wesentlich von der Art und Weise der Klassifizierung der Regelmuster ab. Je feiner und geschickter diese gewählt werden kann, um so größer sind die Möglichkeiten einer differenzierten und problemorientierten Bearbeitung der vorhandenen Datenbestände. In der sinnvollen Einteilung der Regelmuster in Klassen liegt somit die Hauptschwierigkeit bei der Verwendung von Templates. Der einfachere Teil besteht in der konkreten Anwendung von Templates auf Regelmuster. Die zugrunde liegende Idee ist dabei folgende: Aus dem Bestand zu prüfender Regelmuster wird ein Muster ausgewählt. Dieses Muster wird zunächst dahingehend getestet, ob es zu einem ausschließenden Template passt. Die Menge der ausschließenden Templates wird dabei als T_{out} bezeichnet. Im einfachsten Fall ist dazu genau ein Vergleich notwendig, im schlechtesten Fall müssen alle $|T_{out}|$ ausschließenden Templates geprüft werden. Wird ein passendes Template gefunden, kann das Regelmuster sofort verworfen werden, da es damit als uninteressant eingestuft wird. In einem zweiten Schritt wird das Muster auf mögliche Zuordnungen zu einschließenden Templates getestet. Die Menge der einschließenden Templates wird dabei als T_{in} bezeichnet. Im einfachsten Fall ist wiederum nur ein Vergleich notwendig, da das Muster nur zu einem Template passen muss. Im schlechtesten Fall müssen alle $|T_{in}|$ einschließenden Templates geprüft werden. Wird ein entsprechendes Template gefunden, wird das Regelmuster in den Bestand interessanter Muster übernommen. Algorithmus 2.8 implementiert dieses Vorgehen.

Funktionsnachweis Algorithmus 2.8

Der Algorithmus funktioniert, da alle potenziell interessanten Regelmuster bei der Überprüfung berücksichtigt werden (Zeilen 2, 3 und 4). Gleichzeitig wird sichergestellt, dass nur Regelmuster aus M in R vorkommen können (Zeile 1). Genügt das zu testende Regelmuster mindestens einem ausschließenden Template, wird es verworfen (Zeilen 6, 7). Dabei wird sichergestellt, dass bei Bedarf alle ausschließenden Templates bei der Überprüfung mit einbezogen werden (Zeile 5). Wird das Regelmuster nicht verworfen und genügt es mindestens einem einschließenden Template (Zeile 10), wird es in den Bestand interessierender Muster aufgenommen (Zeile 14) und ausgegeben (Zeilen 15, 16). Dabei wird sichergestellt, dass bei Bedarf alle einschließenden Templates bei der Überprüfung mit einbezogen werden (Zeile 9).

Eingabe: Menge M von Regelmustern, Menge T_{in} einschließender Templates, Menge T_{out} ausschließender Templates
Ausgabe: alle Regelmuster die T_{in} genügen und nicht von T_{out} erfasst werden

```
1:  R := ∅;
2:  while M ≠ ∅ do
3:     rule := (X ⇒ Y) mit rule beliebiges Regelmuster aus M;
4:     M := M \ rule;
5:     for i := 1 to |T_out| do
6:        if rule ∈ T[i]_out then
7:           setze fort mit Zeile 2
8:     found := false;
9:     for i := 1 to |T_in| do
10:       if rule ∈ T[i]_in then
11:          found := true;
12:          setze fort mit Zeile 13
13:    if found = true then
14:       R := R ∪ rule;
15: for alle Regelmuster in R do
16:    gib aktuelles Regelmuster aus
```

Algorithmus 2.8: Anwendung von Templates auf Regelmuster

2.5 Visualisierung von Regelmustern

Die interessantesten Regelmuster sind oft diejenigen Muster, die unerwartete Informationen liefern. Um derartige Muster zu finden, müssen Probleme, die bei der Verwaltung großer Regelbestände auftreten können, gelöst werden. Templates sind ein möglicher Weg, um die Suche in diesen Beständen zu optimieren. Als konsequente Weiterführung dieser Strategie sollen nunmehr Ansätze aufgezeigt werden, mit deren Hilfe umfangreiche Bestände an Regelmustern besser verwaltet werden können. Die wesentlichen Ideen des diesem Anliegen gewidmeten Abschnittes wurden ebenfalls in [KMR94] vorgestellt.

2.5.1 Anzeige einzelner Regelmuster

Eine Aufgabe bei der Visualisierung von Regelmustern besteht in der Bereitstellung von Verfahren, die es einem Anwender erlauben, aus einem gegebenen Datenbestand die für ihn interessantesten Muster so schnell wie möglich herauszufinden. Eine einfache Methode ist die Repräsentation der gefundenen Zusammenhänge in Textform. Ein Nachteil dieses Ansatzes ist die rasant verlorengehende Übersicht bei länger werdenden Regellisten. Wird der Umstand berücksichtigt, dass selbst bei hohen Frequenz- und Vertrauenswerten die Zahl der aus einer Datenbasis gewinnbaren Regelmuster sehr hoch sein kann, erscheint die fehlende Praktikabilität dieses Verfahrens noch deutlicher.

Beispiel 2.17 (Textuelle Repräsentation)

```
Loss of Frame Alignment
⇒ Loss of CRC Multiframe Alignment
(0.0004, 0.85)

Loss of Frame Alignment, Loss of CRC Multiframe Alignment
⇒ Bit Error Rate (BER) > 10E-3
(0.000295, 0.7265)
```

Für einzelne Regelmuster bietet es sich an, nur die Werte für Vertrauen und Frequenz zu visualisieren. Die vollständige Visualisierung eines Musters ist – beispielsweise gemeinsam mit anderen Mustern – mittels eines Graphen möglich. Diese Vorgehensweise wird in Abschnitt 2.5.2 behandelt. Abbildung 2.2 verdeutlicht die Repräsentation von Regelmustern mit Hilfe von Balkendiagrammen.

Die linken und rechten Balken repräsentieren die Werte für Vertrauen und Frequenz, während die mittleren Balken den aus Vertrauen und Frequenz gebildeten Wert für die *Gemeinsamkeit* darstellen (siehe Definition 2.18). Die Abbildung am rechten

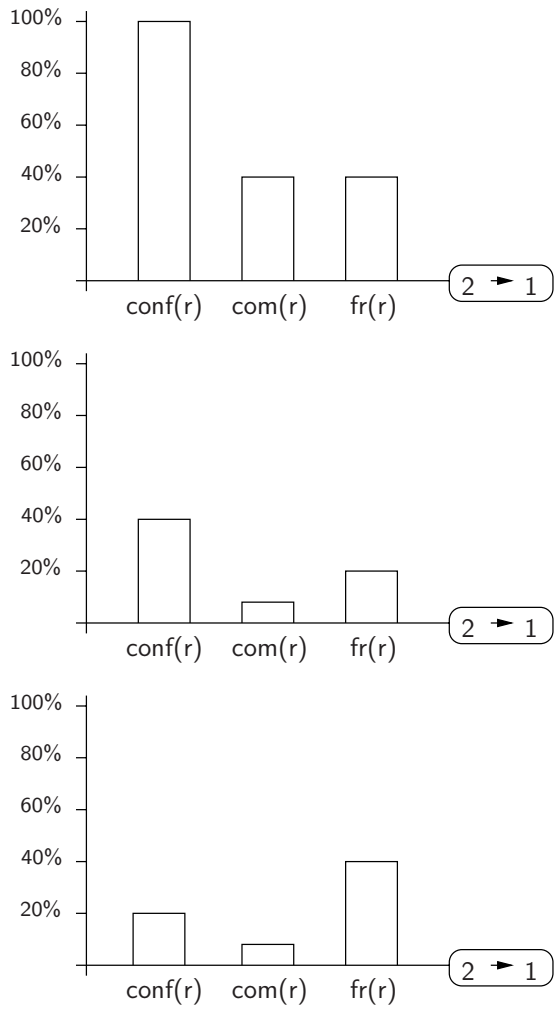

Abbildung 2.2: Repräsentation mittels Balkendiagrammen

Rand des Diagramms gibt Aufschluss über den Aufbau der Regel, in diesem Beispiel besitzen alle dargestellten Regelmuster zwei Elemente auf der linken Regelseite und ein Element auf der rechten Seite der Regel. Es gilt $\mu = 1$.

Definition 2.18 (Gemeinsamkeit)
Sei r ein Regelmuster, $fr(r)$ der Wert für die Frequenz und $conf(r)$ der Wert für die Vertrauenswürdigkeit dieses Musters. $\mu > 0$ sei ein vom Anwender vorgegebener Wert. Dann lässt sich die *Gemeinsamkeit* $com(r)$ eines Regelmusters wie folgt definieren:

$$com(r) = \mu \cdot fr(r) \cdot conf(r)$$

Durch den künstlich geschaffenen Wert für die Gemeinsamkeit eines Regelmusters können Muster mit geringem Frequenz- *und* geringem Vertrauenswert leichter er-

fasst werden. Es handelt sich somit um eine zusätzliche Hilfe bei der Suche nach interessanten Regelmustern durch den Anwender.

Die Kombination der textuellen und diagrammbasierten Repräsentation erhöht die Effizienz beim manuellen Durchsuchen der Regelmuster. Die Anzeige der Diagramme vermittelt selbst beim schnellen Durchlaufen des Bestandes immer eine ungefähre Vorstellung von den Eigenschaften des jeweiligen Musters. Durch diesen Umstand ist es möglich, potenziell interessante Regelmuster alleine durch Betrachten der Diagramme zu finden, da Abweichungen der Werte von den übrigen Mustern leicht erkannt werden können. In einem daran anschließenden Schritt können diese Muster dann mittels der textuellen Repräsentation näher untersucht werden.

2.5.2 Gemeinsame Anzeige mehrerer Regelmuster

Die Visualisierung eines ganzen Regelsets ist – im Vergleich zu einem einzelnen Regelmuster – weitaus schwieriger. Das Problem besteht in dem Entwurf eines Graphen, dessen Knoten die in den Regelmustern vorkommenden Elemente repräsentieren, wobei jedes dieser Elemente unterschiedliche Nachfolger besitzt sowie ebenfalls unterschiedliche Vertrauens- und Frequenzwerte aufweisen kann. Eine praktische Umsetzung des Graphenansatzes kommt daher nur für einfache Konstellationen von Regelmustern in Betracht.

Die dem hier vorgestellten Graphenmodell zugrunde liegende Idee ist folgende: Die Elemente eines Regelmusters werden als Knoten, die Beziehungen zwischen diesen Elementen als Pfeile dargestellt (Abbildung 2.3). Durch die Dicke der verwendeten Linien wird die Vertrauenswürdigkeit einer Regel angezeigt, durch eine der Linie zugeordnete Zahl die Frequenz (die umgekehrte Variante ist ebenso möglich). Mittels verschiedener Farben können zusätzliche Informationen in das Modell eingebracht werden.

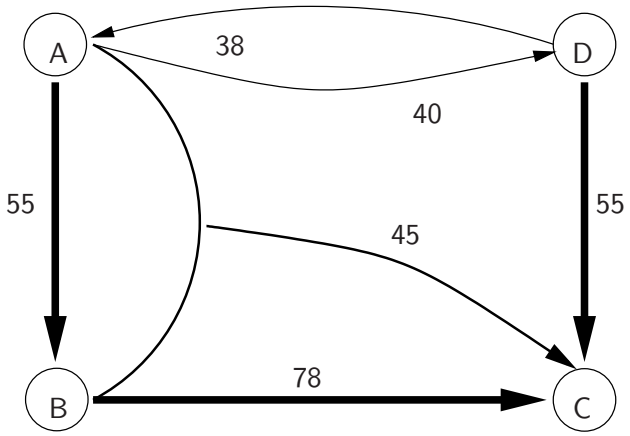

Abbildung 2.3: Repräsentation mittels Graphen

Das Hauptproblem bei der Entwicklung eines derartigen Graphen ist die geschickte Platzierung der Knoten. Selbst bei einer geringen Anzahl an Elementen (> 4) und darzustellenden Regelmustern wird der Graph im Allgemeinen bereits an Übersichtlichkeit verlieren. Um die Komplexität der Darstellung zu verringern, müssen Reduktionsverfahren auf den Regelmuster-Bestand angewendet werden. Das mit Abstand bedeutenste Verfahren sind die bereits in Abschnitt 2.4.3 vorgestellten Templates. Dabei entspricht das Löschen eines Knotens dem Hinzufügen zweier ausschließender Templates: Ein Template beinhaltet das zugehörige Attribut auf der linken, das zweite Template auf der rechten Regelseite. Das besondere Hervorheben eines Knotens entspricht demgegenüber der Anwendung zweier einschließender Templates.

Ein einfacher, aber wirkungsvoller Weg, die Komplexität des darzustellenden Graphen zu reduzieren, ist eine Beschränkung der Elementzahl der Regelmuster. Beispielsweise könnte der Anwender festlegen, dass nur Muster zur Anzeige kommen, die maximal drei Elemente auf der linken und zwei Elemente auf der rechten Seite der Regel aufweisen.

Die Zusammenfassung von Knoten ist ein dritter Weg zur Komplexitäts-Verringerung. Dem Anwender wird dabei die Möglichkeit eingeräumt, gewisse Knoten – beispielsweise solche, die mit sehr hohen Vertrauenswerten untereinander verbunden sind – durch einen neuen gemeinsamen Knoten zu repräsentieren.

Ein letzter Ansatz zur Reduktion der darzustellenden Komplexität besteht in der Anwendung einer Graphentechnik, die mehrere Instanzen eines Knotens zulässt. Ein möglicher Anwendungsfall besteht in der Nutzung separater Knoten für die Anzeige der linken und rechten Regelseiten. Die konsequente Umsetzung dieses Weges wäre die Darstellung der Knoten der linken Regelseite auf der linken Seite des Anzeigemediums, sowie die Darstellung der Knoten der rechten Regelseite auf der rechten Seite des Anzeigemediums. Ein Nachteil dieser Technik besteht in dem Verlust des „Netzwerkeffekts“ bei der Darstellung des Graphen.

Kapitel 3

Praktisches Beispiel

Nach den theoretischen Ausführungen der vorhergehenden Abschnitte soll nunmehr die Gewinnung von Regeln aus Alarmmustern anhand einer gegebenen Ereignissequenz beispielhaft aufgezeigt werden. Abbildung 3.1 stellt die verwendete Sequenz dar. Es handelt sich dabei um das zeitlich aufeinanderfolgende dreimalige Auftreten der Einzelalarme 1, 2 und 3. Obwohl eine solch kurze Sequenz keinerlei praktische Bedeutung hat und den im Folgenden betriebenen Aufwand für sich allein gesehen nicht rechtfertigen würde, ist sie doch sehr gut für die Erläuterung der zum Einsatz kommenden Ideen geeignet, da der Blick auf das Wesentliche nicht durch ein Überangebot an Information behindert wird. Für die Gewinnung von Alarmmustern aus derartigen Sequenzen sei auf [Toi96, Tdl00] verwiesen.

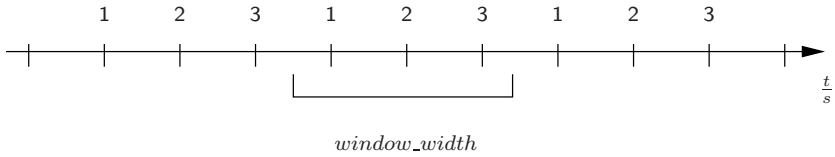

Abbildung 3.1: Beispielhafte Sequenz mit 9 aufgetretenen Alarmen

Begonnen wird im Abschnitt 3.1 zunächst mit der Erläuterung der Gewinnung von Regelmustern aus parallelen Alarmmustern. Abschnitt 3.2 beschäftigt sich mit der Gewinnung von Regelmustern aus seriellen Alarmmustern. Notwendig wurde diese Unterteilung, da die Generierung der Regelmuster auf unterschiedlichen Prinzipien basiert und die gewonnenen Ergebnisse unterschiedlicher Interpretation bedürfen.

Allen im Folgenden aufgeführten Beispielen liegen gemeinsame Werte für Frequenz, Vertrauen (je 0%) und Fensterbreite (3 Sekunden) zugrunde. Der Wert der Fensterbreite wurde bewusst so gewählt, dass in Kombination mit der Beschaffenheit der Ereignissequenz keine Alarmmuster mit Wiederholungen auftreten können. Der Grund ist in dem Verlust an Übersichtlichkeit zu suchen, der nicht durch einen Gewinn an Information ausgeglichen werden kann.

3.1 Regelmuster aus parallelen Alarmmustern

Tabelle 3.1 enthält alle aus obiger Ereignissequenz gewinnbaren parallelen Alarmmuster. Wird das Beobachtungsfenster über die Sequenz bewegt, so zeigt sich, dass jeder Einzelalarm in genau 9 Fenstern vertreten ist. Gemäß Definition 3.1 beträgt die Gesamtzahl aller Fenster 11.

Definition 3.1 (Zahl der Fenster einer Ereignissequenz)
Sei $window\_width$ die Breite des Fensters, T_s der erste Zeitpunkt innerhalb, T^s der erste Zeitpunkt nach dem Ende der Ereignissequenz. Dann beträgt die Zahl W der Fenster einer Ereignissequenz:

$$W(window\_width, T_s, T^s) = T^s - T_s + window\_width - 1$$

Ein Frequenzwert von 81,8182 % bedeutet also, dass das betreffende Alarmmuster in ca. 82 % aller Fenster – und somit in 9 von 11 Fenstern – vorkam.

Alarmmuster	Frequenz in %
{1}	81,8182
{2}	81,8182
{3}	81,8182
{1, 2}	72,7273
{1, 3}	63,6364
{2, 3}	72,7273
{1, 2, 3}	63,6364

Tabelle 3.1: Parallele Alarmmuster

An den Alarmmustern {1}, {1,2} und {1,2,3} ist erkennbar, dass die Frequenz paralleler Alarmmuster durch Hinzunahme von Ereignissen monoton fällt. Die Muster {1,3} und {1,2,3} zeigen, dass die Monotonie aber nicht im strengen Sinne gilt. Es handelt sich hierbei um eine fundamentale Eigenschaft paralleler Alarmmuster – und wie später zu sehen sein wird von Alarmmustern generell – wie sie in ähnlicher Form auch bei Regelmustern durch Wechseln von Elementen der linken Regelseite auf die rechte Seite der Regel beobachtet und ausgenutzt werden kann.

Tabelle 3.2 enthält die aus den obigen Alarmmustern generierbaren Regelmuster. Dabei nehmen die ersten drei Einträge der Form $0 \Rightarrow Y$ eine Sonderstellung ein, da 0 kein gültiges Ereignis ist. Sie bringen die Wahrscheinlichkeiten zum Ausdruck,

Regelmuster	Frequenz in %	Vertrauen in %
$0 \Rightarrow 1$	81,8	100,0
$0 \Rightarrow 2$	81,8	100,0
$0 \Rightarrow 3$	81,8	100,0
$2 \Rightarrow 1$	72,7	88,9
$1 \Rightarrow 2$	72,7	88,9
$3 \Rightarrow 1$	63,6	77,8
$1 \Rightarrow 3$	63,6	77,8
$3 \Rightarrow 2$	72,7	88,9
$2 \Rightarrow 3$	72,7	88,9
$3 \Rightarrow (1,2)$	63,6	77,8
$2 \Rightarrow (1,3)$	63,6	77,8
$1 \Rightarrow (2,3)$	63,6	77,8
$(2,3) \Rightarrow 1$	63,6	87,5
$(1,3) \Rightarrow 2$	63,6	100,0
$(1,2) \Rightarrow 3$	63,6	87,5

Tabelle 3.2: Parallele Regelmuster

mit der die jeweils zugehörigen Einzelereignisse in einem beliebigen Fenster vorkommen werden. Das Vertrauen in ein solches „Regelmuster“ beträgt per Definition 100%. Von weit größerem Interesse sind die daran anschließenden Muster. Auffallendes Merkmal der folgenden beiden Gruppen von Regelmustern ist ihre Symmetrie bezüglich der Parameter Frequenz und Vertrauen bei Vertauschung von Elementen der rechten und linken Regelseite. Dies ist eine Folge sowohl aus der Eigenschaft paralleler Alarmmuster, keine Ordnung bezüglich ihrer Einzelereignisse zu besitzen, als auch aus der Regelmäßigkeit der untersuchten Datenbasis bezüglich der Reihenfolge des Auftretens der Einzelereignisse. Es handelt sich somit nicht um eine Eigenschaft paralleler Regelmuster, die in vorteilhafter Weise genutzt werden kann. Übereinstimmung zwischen Regelmustern, die ihren Ursprung im selben Alarmmuster haben besteht lediglich in Bezug auf die Frequenz, da diese immer identisch zur Frequenz des Alarmmusters ist. Deutlich wird dies auch an der letzten Gruppe von Mustern, in der das Regelmuster $(1,3) \Rightarrow 2$ einen anderen Vertrauenswert besitzt, als die übrigen Muster der Gruppe.

Die Monotonie in Bezug auf den Vertrauenswert beim Wechsel von Elementen der linken Regelseite auf die rechte Seite der Regel lässt sich an den letzten beiden Mustergruppen beobachten, bei der die Werte für die Gruppe mit zwei Elementen auf der linken und einem Element auf der rechten Seite der Regel durchgehend höher sind, als die Werte der Gruppe mit einem Element auf der linken und zwei Elementen auf der rechten Seite der Regel.

3.2 Regelmuster aus seriellen Alarmmustern

Tabelle 3.3 enthält alle aus der betrachteten Ereignissequenz gewinnbaren seriellen Alarmmuster. Bezüglich der Einzelalarme und der Gesamtzahl der Fenster gilt das in Abschnitt 3.1 gesagte. Ebenso gibt es keine Unterschiede bei den Alarmmustern der Länge 1, da diese Muster noch nicht die für parallele und serielle Alarmmuster typischen Eigenschaften aufweisen können. Für die übrigen Alarmmuster gilt für die Summe der Frequenzwerte: $fr(XY) = fr(X \rightarrow Y) + fr(Y \rightarrow X)$ in erwarteter Weise.

Alarmmuster	Frequenz in %
$\{1\}$	81,8182
$\{2\}$	81,8182
$\{3\}$	81,8182
$\{1 \rightarrow 2\}$	54,5455
$\{1 \rightarrow 3\}$	27,2727
$\{2 \rightarrow 1\}$	18,1818
$\{2 \rightarrow 3\}$	54,5455
$\{3 \rightarrow 1\}$	36,3636
$\{3 \rightarrow 2\}$	18,1818
$\{1 \rightarrow 2 \rightarrow 3\}$	27,2727
$\{2 \rightarrow 3 \rightarrow 1\}$	18,1818
$\{3 \rightarrow 1 \rightarrow 2\}$	18,1818

Tabelle 3.3: Serielle Alarmmuster

Auch bei seriellen Alarmmustern ist der Frequenzwert bei Hinzunahme weiterer Ereignisse eine monoton fallende Funktion. Es gilt ohne Einschränkung das in Abschnitt 3.1 gesagte, auch wenn sich in dem zugrunde liegenden Alarmmusterbestand kein Beispiel befindet, mit dessen Hilfe die strenge Monotonie ausgeschlossen werden kann.

Tabelle 3.4 enthält die aus den obigen Alarmmustern generierbaren seriellen Regelmuster. Dabei nehmen die ersten drei Einträge wiederum die unter Abschnitt 3.1 beschriebene Sonderstellung ein. Sehr gut beobachtet werden kann der „Randeffekt“ der verwendeten Ereignissequenz an Hand der folgenden Gruppe von Regelmustern: Während bei der gegebenen Fensterbreite von 3 Sekunden auf eine 1 immer ein weiteres Ereignis folgt – die Summe der Vertrauenswerte beträgt 100% – ist die Wahrscheinlichkeit, dass auf eine 2 ein Ereignis folgt nur noch 89%. Bei einer 3 sinkt diese weiter auf 67%.

Bezüglich der Frequenz nehmen die Regelmuster wiederum den Wert „ihres“ Alarm-

Regelmuster	Frequenz in %	Vertrauen in %
$0 \Rightarrow 1$	81,8	100,0
$0 \Rightarrow 2$	81,8	100,0
$0 \Rightarrow 3$	81,8	100,0
$1 \Rightarrow 2$	54,5	66,7
$1 \Rightarrow 3$	27,3	33,3
$2 \Rightarrow 1$	18,2	22,2
$2 \Rightarrow 3$	54,5	66,7
$3 \Rightarrow 1$	36,4	44,4
$3 \Rightarrow 2$	18,2	22,2
$1 \Rightarrow (2 \rightarrow 3)$	27,3	33,3
$2 \Rightarrow (3 \rightarrow 1)$	18,2	22,2
$3 \Rightarrow (1 \rightarrow 2)$	18,2	22,2
$(1 \rightarrow 2) \Rightarrow 3$	27,3	50,0
$(2 \rightarrow 3) \Rightarrow 1$	18,2	33,3
$(3 \rightarrow 1) \Rightarrow 2$	18,2	50,0

Tabelle 3.4: Serielle Regelmuster

musters an. So sind beispielsweise die Werte für das Alarmmuster $\{1 \rightarrow 2 \rightarrow 3\}$ sowie für die Regelmuster $1 \Rightarrow (2 \rightarrow 3)$ und $(1 \rightarrow 2) \Rightarrow 3$ in erwarteter Weise identisch. Ebenso ist das bereits angesprochene Verhalten bezüglich der Monotonie von Regelmustern beim Wechsel von Elementen der linken Regelseite auf die rechte Seite der Regel zu beobachten. Beispielsweise sinkt der Vertrauenswert der zusammengehörenden Regelmuster $(1 \rightarrow 2) \Rightarrow 3$ und $1 \Rightarrow (2 \rightarrow 3)$ von 50% auf 33,3%.

3.3 Reduktion von Regelmustern

Nach der erfolgreichen Generierung der Regelmuster, gilt es in einem weiteren Schritt den Nutzwert dieser Muster durch Reduktion noch vorhandener Redundanzen zu erhöhen.

Tabelle 3.5 veranschaulicht die Bildung eines repräsentativen linken Regelsets für Regelmuster aus parallelen Alarmmustern. Zu erkennen ist, dass, im Vergleich mit den „unbehandelten“ Mustern, alle Regelmuster entfernt wurden, für die jeweils mindestens ein weiteres Muster existierte, dessen rechte Regelseite – unter der Bedingung identischer linker Regelseiten – die rechte Seite des entfernten Musters als echte Teilmenge enthielt. Die entfernten Muster werden somit, gemäß Definition 2.11, durch Muster höherer Voraussagekraft repräsentiert.

Regelmuster	Frequenz in %	Vertrauen in %
$0 \Rightarrow 1$	81,8	100,0
$0 \Rightarrow 2$	81,8	100,0
$0 \Rightarrow 3$	81,8	100,0
$3 \Rightarrow (1,2)$	63,6	77,8
$2 \Rightarrow (1,3)$	63,6	77,8
$1 \Rightarrow (2,3)$	63,6	77,8
$(2,3) \Rightarrow 1$	63,6	87,5
$(1,3) \Rightarrow 2$	63,6	100,0
$(1,2) \Rightarrow 3$	63,6	87,5

Tabelle 3.5: Repräsentatives linkes Regelset (parallel)

Regelmuster	Frequenz in %	Vertrauen in %
$0 \Rightarrow 1$	81,8	100,0
$0 \Rightarrow 2$	81,8	100,0
$0 \Rightarrow 3$	81,8	100,0
$2 \Rightarrow 1$	72,7	88,9
$1 \Rightarrow 2$	72,7	88,9
$3 \Rightarrow 1$	63,6	77,8
$1 \Rightarrow 3$	63,6	77,8
$3 \Rightarrow 2$	72,7	88,9
$2 \Rightarrow 3$	72,7	88,9
$3 \Rightarrow (1,2)$	63,6	77,8
$2 \Rightarrow (1,3)$	63,6	77,8
$1 \Rightarrow (2,3)$	63,6	77,8

Tabelle 3.6: Repräsentatives rechtes Regelset (parallel)

Bei der Bildung des repräsentativen rechten Regelsets wurden alle Muster aus dem ursprünglichen Bestand entfernt, für die, bei nunmehr identischer rechter Regelseite, mindestens ein weiteres Muster existierte, dessen linke Regelseite echte Teilmenge der linken Seite des entfernten Musters war. Die entfernten Muster werden somit, gemäß Definition 2.14, durch Muster höherer Detektionskraft repräsentiert.

Regelmuster	Frequenz in %	Vertrauen in %
$0 \Rightarrow 1$	81,8	100,0
$0 \Rightarrow 2$	81,8	100,0
$0 \Rightarrow 3$	81,8	100,0
$1 \Rightarrow (2 \rightarrow 3)$	27,3	33,3
$2 \Rightarrow (3 \rightarrow 1)$	18,2	22,2
$3 \Rightarrow (1 \rightarrow 2)$	18,2	22,2
$(1 \rightarrow 2) \Rightarrow 3$	27,3	50,0
$(2 \rightarrow 3) \Rightarrow 1$	18,2	33,3
$(3 \rightarrow 1) \Rightarrow 2$	18,2	50,0

Tabelle 3.7: Repräsentatives linkes Regelset (seriell)

Die Tabellen 3.7 und 3.8 vervollständigen die Darstellung für die auf seriellen Alarmmustern basierenden Regelmuster. Auch für diese Muster gilt das zuvor Gesagte in entsprechender Weise.

Regelmuster	Frequenz in %	Vertrauen in %
$0 \Rightarrow 1$	81,8	100,0
$0 \Rightarrow 2$	81,8	100,0
$0 \Rightarrow 3$	81,8	100,0
$1 \Rightarrow 2$	54,5	66,7
$1 \Rightarrow 3$	27,3	33,3
$2 \Rightarrow 1$	18,2	22,2
$2 \Rightarrow 3$	54,5	66,7
$3 \Rightarrow 1$	36,4	44,4
$3 \Rightarrow 2$	18,2	22,2
$1 \Rightarrow (2 \rightarrow 3)$	27,3	33,3
$2 \Rightarrow (3 \rightarrow 1)$	18,2	22,2
$3 \Rightarrow (1 \rightarrow 2)$	18,2	22,2

Tabelle 3.8: Repräsentatives rechtes Regelset (seriell)

Eine Anwendung der links-rechts Methode auf die gebildeten repräsentativen Regelsets bleibt ohne Wirkung, da die rechten Regelseiten nach der Bildung der repräsen-

tativen linken Regelsets schon zu speziell geworden sind um noch rechte Regelsets mit mehr als einem zugehörigen Muster bilden zu können (bei Nichtberücksichtigung der für eine Reduktion nicht verwendbaren $0 \Rightarrow Y$-Muster). Durch Anwendung der rechts-links Methode ist hingegen eine weitere Reduktion möglich, da die nunmehr zur Bildung der Regelsets herangezogenen linken Regelseiten ein sehr allgemeines Aussehen besitzen und folglich alle erzeugten Sets mehr als ein zugehöriges Muster aufweisen. Diese Beobachtung stützt somit die im Abschnitt 2.3.3 gemachten Ausführungen.

Regelmuster	Frequenz in %	Vertrauen in %
$0 \Rightarrow 1$	81,8	100,0
$0 \Rightarrow 2$	81,8	100,0
$0 \Rightarrow 3$	81,8	100,0
$3 \Rightarrow (1, 2)$	63,6	77,8
$2 \Rightarrow (1, 3)$	63,6	77,8
$1 \Rightarrow (2, 3)$	63,6	77,8

Tabelle 3.9: Anwendung der rechts-links Methode (parallel)

Regelmuster	Frequenz in %	Vertrauen in %
$0 \Rightarrow 1$	81,8	100,0
$0 \Rightarrow 2$	81,8	100,0
$0 \Rightarrow 3$	81,8	100,0
$1 \Rightarrow (2 \rightarrow 3)$	27,3	33,3
$2 \Rightarrow (3 \rightarrow 1)$	18,2	22,2
$3 \Rightarrow (1 \rightarrow 2)$	18,2	22,2

Tabelle 3.10: Anwendung der rechts-links Methode (seriell)

Zusammenfassend kann festgestellt werden, dass – abgesehen von den künstlich gebildeten und daher nicht weiter berücksichtigten Regelmustern der Form $0 \Rightarrow Y$ – aus den gefundenen 7 parallelen Alarmmuster des zugrunde liegenden Datenbestandes 12 Regelmuster gebildet werden konnten. Mittels Bildung repräsentativer linker Regelsets konnte die Zahl der Muster auf 6, mittels repräsentativer rechter Regelsets auf 9 reduziert werden. Die Anwendung der rechts-links Methode erlaubte schließlich eine Reduktion auf 3 Muster und somit auf 25 % des ursprünglichen Bestandes. Auf Grund der Einfachheit der verwendeten Ereignissequenz ist erkennbar, dass eine weitere verlustfreie Reduktion nicht mehr möglich ist. Die konsequente Anwendung der vorgestellten Reduktions-Methoden hat demzufolge bei dem hier verwendeten einfachen Beispiel auf die minimal mögliche Zahl von Regelmustern geführt. Für die

seriellen Alarm- und Regelmuster gilt in gleicher Weise: Aus 9 Alarmmustern wurden 12 Regelmuster gebildet, die Reduktion führte auf 6 Muster für das repräsentative linke Regelset, auf 9 Muster für das repräsentative rechte Regelset sowie auf wiederum 3 Muster bei Anwendung der rechts-links Methode. Die links-rechts Methode blieb ebenfalls ohne Effekt.

Kapitel 4

Ergebnisse

Im folgenden Kapitel sollen Ergebnisse vorgestellt werden, die bei der Untersuchung von Alarmdatenbanken eines Mobilfunk-Betreibers gewonnen wurden. Der Beobachtungszeitraum dieser Datenbanken überstreicht dabei eine Periode von 33 Wochen. Die den Datenbanken zugrunde liegende minimale Zeitunterteilung beträgt eine Sekunde. Zur Verdeutlichung der Erkenntnisse werden die Werte einer Datenbank stellvertretend für alle untersuchten Datenbanken aufgezeigt.

4.1 Vorbemerkungen

Um die Ausführungen dieses Kapitels im Detail verstehen zu können, ist die Kenntnis der in [Tdl00], Kapitel 5 vorgestellten Zusammenhänge notwendig. Da es sich bei der vorliegenden Arbeit um eine auf [Tdl00] aufbauende Arbeit handelt, wird auf eine Wiederholung der dort dargestellten Ausführungen an dieser Stelle verzichtet.

Eine weitere Vorbemerkung ist zu den in den folgenden Abschnitten gezeigten Diagrammen notwendig. Obwohl beinahe alle Diagramme in „Kurvenform“ dargestellt werden, handelt es sich bei den zugrunde liegenden Daten grundsätzlich um diskrete Werte. Die kontinuierliche Darstellungsform wurde aus Gründen der Zweckmäßigkeit gewählt, da die wesentlichen Aussagen der Darstellungen so deutlicher zum Ausdruck kommen.

4.2 Eigenschaften der betrachteten Daten

Der vorliegende Datenbestand wurde mittels dreier Zeitfenster, deren Längen 90, 180 und 360 Sekunden betrugen, analysiert. Dabei wurden jeweils Alarmmuster mit den relativen Frequenz-Schwellwerten von x, 10 und 20 Prozent des ermittel-

ten Durchschnittswertes gesucht. Die Durchschnittswerte selbst können Tabelle 4.1 entnommen werden. Für den Frequenz-Schwellwert x gilt die Zuordnung $x_{90s} = 7\%$, $x_{180s} = 8\%$ und $x_{360s} = 8,5\%$ – wobei der Wert für x so gewählt wurde, dass gerade alle in dem jeweiligen Datenbestand vorkommenden Einzelalarme bei der Alarmmuster-Findung berücksichtigt wurden.

Bei dem verwendeten Cluster handelt es sich um die Sammlung der Alarmmeldungen der Basisstation 12-70-0061 und ihrer zugehörigen Tochter-Elemente. Abbildung 4.1 verdeutlicht den Aufbau des Clusters.

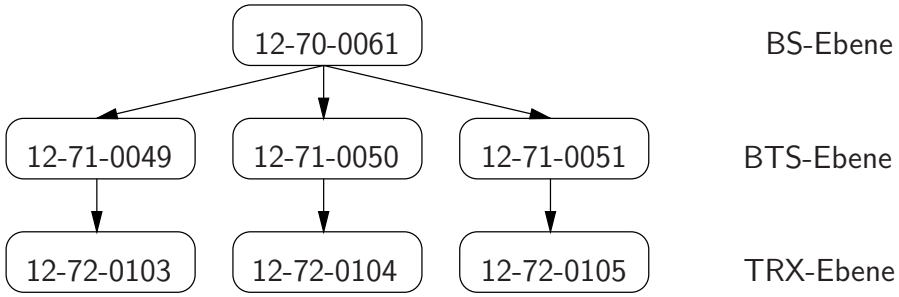

Abbildung 4.1: Aufbau des Clusters 12-70-0061

Der Basisstation sind drei Base Transceiver Stations (BTS) zugeordnet, die ihrerseits je eine TRX steuern.

Anzahl Alarmmeldungen	:	586
Anzahl Einzelalarme	:	42
Frequenz-Bezugswert 90s	:	$5,74776 \cdot 10^{-5}$
Frequenz-Bezugswert 180s	:	$1,10015 \cdot 10^{-4}$
Frequenz-Bezugswert 360s	:	$2,07278 \cdot 10^{-4}$
Tag der ersten Meldung	:	30.12.1997
Tag der letzten Meldung	:	17.08.1998
Gesamtzeitraum	:	231 Tage

Tabelle 4.1: Eigenschaften des Clusters 12-70-0061

In dem erfassten Beobachtungszeitraum von 231 Tagen traten 586 Alarmmeldungen auf. Die Gesamtzahl unterschiedlicher Alarme betrug dabei 42. Einige dieser Alarme kamen jedoch nicht häufig genug vor, um bei der Analyse berücksichtigt zu werden. Die Zahl berücksichtigter Alarme lag je nach Konfiguration zwischen 26 und 42.

4.3 Qualität der Kandidaten-Generierung

Die Tabellen 4.2 und 4.3 geben einen Überblick über das Verhalten der zur Implementierung der Alarmmuster-Findung verwendeten Algorithmen in Bezug auf die

Qualität der Kandidaten-Generierung. Dargestellt ist bei einer Fensterbreite von 360 Sekunden und einem relativen Frequenz-Schwellwert von 8,5% die Zahl generierter („cand“) und frequenter Alarmmuster („freq“) sowie das Verhältnis beider Zahlen (Trefferquote in Prozent – „%“).

parallel	mit Wiederholungen			ohne Wiederholungen		
l	cand	freq	%	cand	freq	%
1	42	42	100,0	42	42	100,0
2	903	219	24,3	861	205	23,8
3	846	585	69,1	614	468	76,2
4	1247	1116	89,5	700	685	97,9
5	1727	1661	96,2	729	721	98,9
6	1989	1958	98,4	574	574	100,0
7	1843	1827	99,1	355	355	100,0
8	1339	1335	99,7	167	167	100,0
9	753	753	100,0	55	55	100,0
10	320	320	100,0	11	11	100,0
11	99	99	100,0	1	1	100,0
12	20	20	100,0	–	–	–
13	2	2	100,0	–	–	–

Tabelle 4.2: Qualität der Kandidaten-Generierung (parallel, 360s, 8,5%)

Im ersten Schritt – er entspricht der Suche nach Alarmmustern der Länge 1 – müssen alle 42 Einzelalarme auf Frequenz getestet werden. Der Frequenz-Schwellwert von 8,5 Prozent wurde so gewählt, dass dieser Test von allen Einzelalarmen bestanden werden konnte. Dementsprechend beträgt die Trefferquote 100 Prozent.

Je länger die Alarmmuster werden, um so mehr kombinatorisch nutzbare Informationen stehen zur Verfügung, die in vorteilhafter Weise ausgenutzt werden können. Ab einer Länge von vier Einzelalarmen werden bereits Trefferquoten nahe 100 Prozent erreicht (parallel, ohne Wiederholungen).

Eine mögliche Verbesserung der verwendeten Algorithmen lässt sich aus der sehr guten Trefferquote für größere Alarmmuster-Längen ableiten: Überschreitet die Trefferquote für Alarmmuster der Länge l einen bestimmten Grenzwert, beispielsweise 80 Prozent, werden alle folgenden Kandidaten der Längen $l+1$, $l+2$, …, max_l in einem Schritt generiert und anschließend *gemeinsam* anhand der Datenbasis auf Frequenz geprüft. Diese Vorgehensweise verringert die Zahl der notwendigen Durchläufe durch die Datenbasis was sich besonders bei größeren Alarmmuster-Längen positiv auswirken wird, da bei diesen einerseits nur noch wenige Kandidaten zu testen sind, andererseits die Trefferquote aber bereits 100 Prozent erreicht hat. Zur Verdeut-

seriell	mit Wiederholungen			ohne Wiederholungen		
l	cand	freq	%	cand	freq	%
1	42	42	100,0	42	42	100,0
2	1764	246	13,9	1722	232	13,5
3	1749	736	42,1	1334	564	42,3
4	2155	1219	56,6	1007	696	69,1
5	1341	1124	83,8	446	430	96,4
6	592	579	97,8	121	121	100,0
7	175	175	100,0	11	11	100,0
8	28	28	100,0	–	–	–
9	2	2	100,0	–	–	–

Tabelle 4.3: Qualität der Kandidaten-Generierung (seriell, 360s, 8,5%)

lichung der Verhältnisse sind in Abbildung 4.2 die erzielten Trefferquoten für die Kandidaten-Generierung noch einmal gesondert dargestellt.

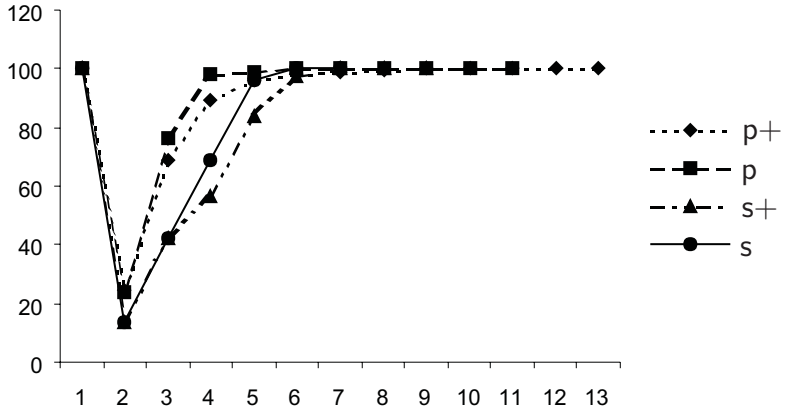

Abbildung 4.2: Qualität der Kandidaten-Generierung

4.4 Quantität der Regelmuster-Generierung

Die Tabellen 4.4 und 4.5 geben einen Überblick über das quantitative Verhalten der bei der Implementierung verwendeten Algorithmen in Bezug auf den Prozess der Regelmuster-Generierung. Im Unterschied zur Kandidatenbildung des vorhergehenden Abschnittes kann bei der Generierung von Regelmustern nicht von einem qualitativ bewertbaren Verhalten gesprochen werden, da es sich um einen an das jeweilige Alarmmuster starr gekoppelten Vorgang handelt.

parallel		mit Wiederholungen			ohne Wiederholungen		
w	f	AM	RM	RM (95%)	AM	RM	RM (95%)
90s	7%	1123	21438	16209	577	2999	981
180s	8%	1925	29269	18119	1105	9933	3421
360s	8,5%	9937	433146	195523	3284	99533	36506

Tabelle 4.4: Regelmuster-Generierung (parallel, $conf = 95\%$)

Dargestellt ist jeweils die Anzahl gefundener frequenter Alarmmuster („AM“), die Anzahl der daraus bildbaren Regelmuster („RM“) sowie die Anzahl Regelmuster mit einer Vertrauenswürdigkeit von mindestens 95 Prozent („RM (95%)“). Für die einzelnen Fensterbreiten wurden diejenigen Frequenz-Schwellwerte gewählt, bei denen alle 42 Einzelalarme gerade als frequent eingestuft wurden.

seriell		mit Wiederholungen			ohne Wiederholungen		
w	f	AM	RM	RM (95%)	AM	RM	RM (95%)
90s	7%	524	573	91	427	468	83
180s	8%	989	1027	80	694	726	74
360s	8,5%	4151	4439	335	2096	2274	225

Tabelle 4.5: Regelmuster-Generierung (seriell, $conf = 95\%$)

Abbildung 4.3 verdeutlicht die in den Tabellen dargestellten Verhältnisse für den besonders extremen Fall paralleler Alarmmuster mit Wiederholungen.

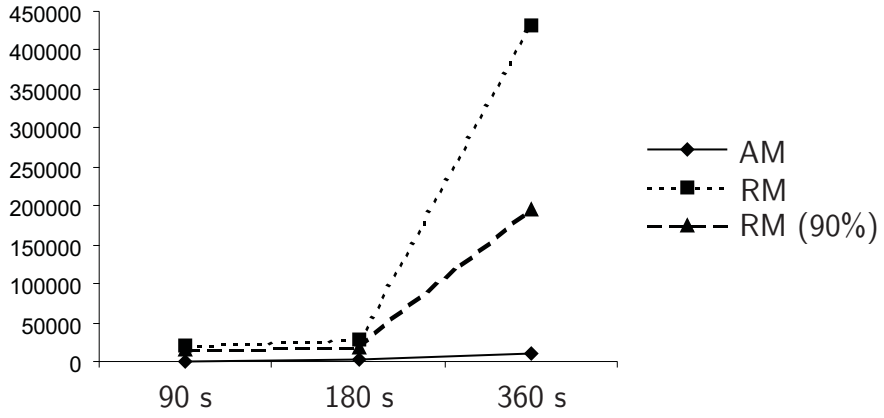

Abbildung 4.3: Quantität der Regelmuster-Generierung

Erkennbar ist, dass bei steigender Fenstergröße (Faktor 4) die Zahl gefundener Alarmmuster sehr stark ansteigt. Bei parallelen Mustern mit Wiederholung um den Faktor 8,9, bei Mustern ohne Wiederholung um den Faktor 5,7. Die Zahl generierter Regelmuster steigt noch schneller: Bei parallelen Mustern mit Wiederholung um den Faktor 20,2, bei Mustern ohne Wiederholung sogar um den Faktor 33,2. Bei seriellen Mustern sieht die Situation nicht ganz so dramatisch aus: Bei Mustern mit Wiederholungen steigt die Zahl gefundener Alarmmuster und generierter Regelmuster jeweils um etwas weniger als das 8-fache, bei Mustern ohne Wiederholungen um etwas weniger als das 5-fache.

4.5 Quantität der Redundanz-Reduktion

Die im Abschnitt 4.4 aufgeführten Zahlen verdeutlichen die Notwendigkeit der Anwendung von Verfahren, die es erlauben, mit derart wachsenden Datenmengen umzugehen. In erster Linie ist dabei die Beseitigung von Redundanzen in den erzeugten Regelmuster-Beständen zu sehen. Auch dieser Abschnitt kann sich nur mit der Quantität der zur Redundanz-Reduktion eingesetzten Algorithmen auseinander setzen, da über die Qualität der Reduktion nur im konkreten Anwendungsfall geurteilt werden kann.

parallel		mit Wiederholungen				ohne Wiederholungen			
w	f	RM	R	L	RL	RM	R	L	RL
90s	7%	16209	2148	855	187	981	414	379	146
		100,0%	13,3%	5,25%	1,15%	100,0%	42,2%	38,6%	14,9%
180s	8%	18119	2245	1630	261	3421	783	931	198
		100,0%	12,4%	9,0%	1,4%	100,0%	22,9%	27,2%	5,8%
360s	8,5%	195523	10421	10562	538	36506	2691	3247	300
		100,0%	5,33%	5,4%	0,28%	100,0%	7,37%	8,89%	0,82%

Tabelle 4.6: Redundanz-Reduktion (parallel, $conf = 95\%$)

Dargestellt ist wiederum die Anzahl generierter Regelmuster mit einem Vertrauenswert von mindestens 95 Prozent („RM“) sowie die daraus gewonnenen repräsentativen rechten Regelsets (R-Methode – „R“), die repräsentativen linken Regelsets (L-Methode – „L“) und die Ergebnisse der Anwendung der rechts-links-Methode (RL-Methode – „RL“). Weiterhin sind für jede Fenstergröße die relativen Größen der reduzierten Regelmuster-Bestände im Verhältnis zum nichtreduzierten Bestand dargestellt.

An den aufgeführten Werten in den Tabellen 4.6 und 4.7 ist deutlich erkennbar,

seriell		mit Wiederholungen				ohne Wiederholungen			
w	f	RM	R	L	RL	RM	R	L	RL
90s	7%	91	79	79	70	83	71	71	62
		100,0%	86,8%	86,8%	76,9%	100,0%	85,5%	85,5%	74,7%
180s	8%	80	71	77	69	74	65	71	63
		100,0%	88,8%	96,2%	86,3%	100,0%	87,8%	95,9%	85,1%
360s	8,5%	335	106	318	96	225	94	221	84
		100,0%	31,6%	94,9%	28,7%	100,0%	41,8%	94,2%	37,3%

Tabelle 4.7: Redundanz-Reduktion (seriell, $conf = 95\%$)

dass die Effizienz der Redundanz-Reduktion mit steigender Größe des Regelmuster-Bestandes ebenfalls sehr schnell steigt. Besonders deutlich wird dies am Beispiel der parallelen Regelmuster mit Wiederholungen und 360 Sekunden Fensterbreite: Hier wird bei einer Ausgangslage von circa 200.000 Regelmustern eine Reduktion auf nur noch 2,8 Promille des ursprünglichen Datenbestandes erreicht. Im Gegensatz dazu stehen die 86,3 Prozent der seriellen Regelmuster mit Wiederholungen und 180 Sekunden Fensterbreite mit einer Ausgangslage von 80 Regelmustern.

Dieser besonders auffällige Sachverhalt ist in Abbildung 4.4 noch einmal im direkten Vergleich gesondert dargestellt.

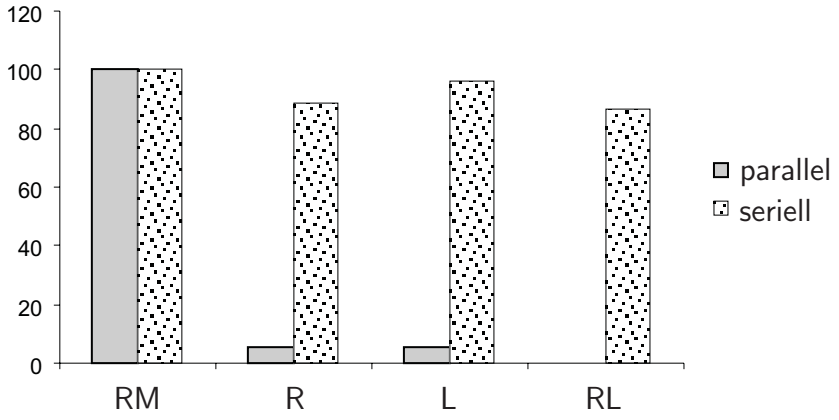

Abbildung 4.4: Quantität der Redundanz-Reduktion

Zusammenfassend kann festgestellt werden, dass dem rasanten Wachstum der erzeugten Regelmuster-Bestände mit Hilfe der in Abschnitt 2.3 vorgestellten Reduktionsmethoden wirksam begegnet werden kann. Weiterhin lässt sich die Vermutung formulieren, dass bei dem zugrunde liegenden Datenbestand eine Fenstergröße von 360 Sekunden bereits als zu groß angesehen werden muss, da im Vergleich zu ei-

ner Fenstergröße von 90 Sekunden zum überwiegenden Teil nur noch Redundanzen erzeugt werden.

4.6 Einfluss der Fenstergröße

$f = 10\%$	mit Wiederholungen		ohne Wiederholungen	
w	parallel	seriell	parallel	seriell
90s	424	326	331	281
180s	926	671	623	491
360s	5632	2720	2159	1433

Tabelle 4.8: Einfluss der Fenstergröße

Der Einfluss der Fenstergröße auf die Anzahl gefundener frequenter Alarmmuster wurde anhand des Zahlenmaterials der vorhergehenden Abschnitte bereits sichtbar. Daher soll an dieser Stelle nur noch kurz darauf eingegangen werden. Deutlich wird anhand der Tabelle 4.8, dass der Alarmmuster-Bestand im Vergleich zur Fenstergröße (Faktor 4) stärker wächst: Bei Alarmmustern mit Wiederholungen ist ein Anwachsen um den Faktor 13,3 bei parallelen, um den Faktor 8,4 bei seriellen Mustern zu beobachten. Bei Alarmmustern ohne Wiederholungen ist der Wachstumsvorgang etwas weniger heftig: Bei parallelen Mustern um den Faktor 6,5, bei seriellen Mustern um den Faktor 5,1.

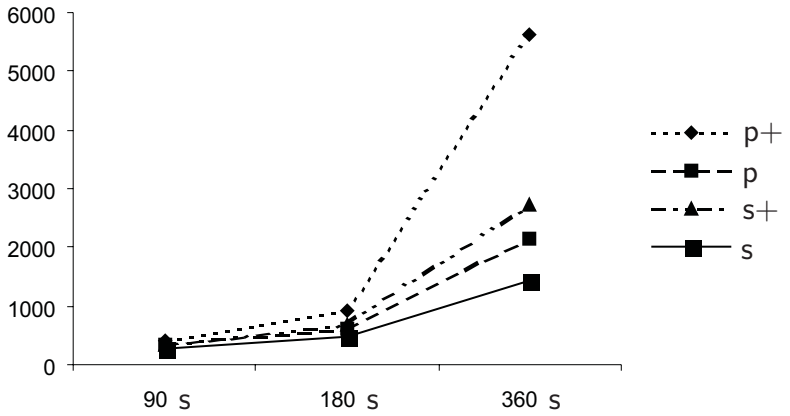

Abbildung 4.5: Einfluss der Fenstergröße

Berücksichtigt werden muss diese „Nichtlinearität“ bei der Anwendung von Suchvorgängen nach frequenten Alarmmustern über große Datenbestände. Bei diesen

wird das Arbeiten mit großen Fenstern sehr schnell in Bereiche führen, die von der verwendeten Hardware nur noch schwer beherrscht werden können. So war es dem Autor dieser Arbeit mit den entsprechenden Einstellungen problemlos möglich, aus einem Alarmmeldungs-Bestand von circa 100 kB ein Alarmmuster-Bestand von 3 MB (parallele Muster mit Wiederholungen) erzeugen zu lassen aus dem wiederum ein Regelmuster-Bestand von über 330 MB generiert wurde.

4.7 Einfluss des Frequenz-Schwellwertes

$w = 360$s	mit Wiederholungen		ohne Wiederholungen	
f	parallel	seriell	parallel	seriell
8,5%	9937	4151	3284	2096
10%	5632	2720	2159	1433
20%	791	640	456	405

Tabelle 4.9: Einfluss des Frequenz-Schwellwertes

Ganz ähnlich wie der Einfluss der Fenstergröße verhält sich der Einfluss des Frequenz-Schwellwertes auf die Anzahl gefundener frequenter Alarmmuster. Auch hier existiert eine „Grenze“, ab der die Anforderungen hardware- und zeitmäßig nicht mehr zufriedenstellend erfüllt werden können. Der Eingangs des Kapitels aufgeführte virtuelle Frequenz-Schwellwert x ist ein Beispiel dafür: Die konkreten Werte ließen für ihre jeweiligen Zeitfenster noch ein vernünftiges Arbeiten zu und erfassten dennoch alle im Alarmmeldungs-Bestand vorkommenden Einzelalarme.

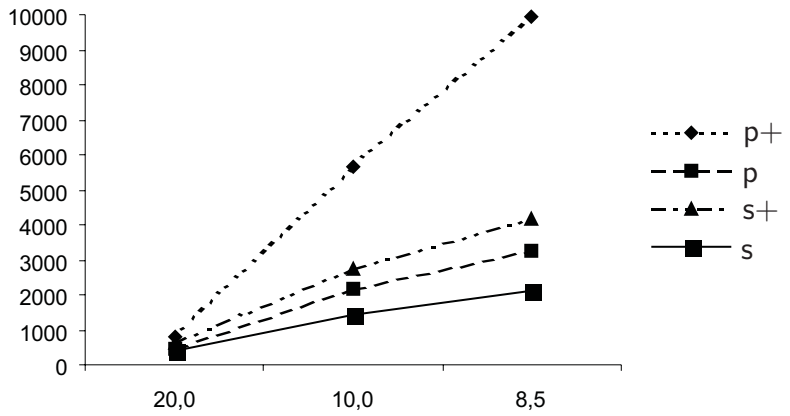

Abbildung 4.6: Einfluss der Frequenz

Kapitel 5

Zusammenfassung

Die Aufgabenstellung der vorliegenden Diplomarbeit umfasste die Schritte von der Erarbeitung der theoretischen Grundlagen von Regelgewinnungs- und Regelbearbeitungs-Prozessen über die Entwicklung der zugehörigen Algorithmen bis zur Implementierung eines kompletten, das Problem der Regelfindung lösenden Programms in der objektorientierten Programmiersprache C++.

Aufbauend auf dem in einer vorhergehenden Arbeit entwickelten Programm ISIS (ISIS steht für „Intelligent Search of Interesting Patterns in Sequences“) wurden die bestehenden Bereiche Datenvorverarbeitung und Alarmmuster-Findung um Fähigkeiten zur Regelmuster-Generierung, zur Reduktion vorhandener Redundanzen in den generierten Regelmuster-Beständen, zur Gruppierung mutmaßlich zusammengehörender Regelmuster und zur Filterung der Regelmuster-Bestände durch anwenderspezifische Vorgaben erweitert. Durch die Implementierung dieser Funktionen wurde der Ausbau von ISIS zu einem vollständigen Data Mining-Tool vollzogen.

Bei der Bearbeitung der gestellten Aufgabe kristallisierte sich als größtes Problem das enorme Anwachsen des zu betrachtenden Datenbestandes bei der Erzeugung von Regelmustern aus Alarmmustern heraus. Dieses Verhalten erschwert in erheblichem Maße die Erreichung des eigentlichen Ziels eines jeden Data Mining-Prozesses, der Gewinnung von Wissen aus den ursprünglichen Daten. Ein Schwerpunkt der vorliegenden Arbeit bestand daher darin, den erzeugten Datenbestand an Regelmustern von vorhandenen Redundanzen zu befreien und somit eine sinnvolle Weiterverarbeitung zu ermöglichen. Primäres Ziel bei der Realisierung des Reduktionsvorganges war der Erhalt der im erzeugten Datenbestand vorhandenen Information, was bedeutete, dass der durch den ursprünglichen Bestand an Regelmustern beschriebene Alarmmuster-Bestand auch durch den reduzierten Bestand an Regelmustern in gleicher und vollständiger Weise beschrieben werden musste.

Nachteil dieser – in Bezug auf die vorhandene Information – verlustfreien Reduktionsmethode ist, dass in Abhängigkeit der für den gesamten Vorgang der Alarmmus-

ter-Findung und Regelmuster-Generierung gewählten Parameter der verbleibende Bestand an Regelmustern weiterhin zu groß sein kann um einen schnellen und einfachen Erkenntnisgewinn zu ermöglichen. Aus diesem Grund wurden Methoden entwickelt, die es einem Anwender ermöglichen, aus den vorliegenden Regelmustern mit Hilfe einfach zu formulierender Beschreibungen gezielt die für ihn wichtigen Muster herauszufiltern. Abschließend wurde die Möglichkeit geschaffen, den gesamten Bestand an Regelmustern hinsichtlich von Ähnlichkeiten der Muster untereinander zu ordnen.

Ausblick

Trotz des Einsatzes aller vorgestellten Maßnahmen zur Nachbearbeitung und Ordnung von Regelmustern kann nicht garantiert werden, unter allen Umständen den verbleibenden Bestand an Mustern auf eine gewünschte Größe reduzieren zu können. Zur weiteren Bearbeitung dieses Problems bieten sich vor allem zwei Wege an: Zum einen die Entwicklung von Methoden, die den verbleibenden Bestand an Regelmustern durch gezieltes Hinnehmen von Informationsverlusten weiter reduzieren können.

Zum anderen die Realisierung der in Abschnitt 2.5 vorgestellten Methoden zur Visualisierung von Regelmustern. Diese bieten vor allem Anwendern eine direkte Möglichkeit, aus einem gegebenen Regelmuster-Bestand interessante Muster manuell und intuitiv herauszufinden.

Denkbar sind weiterhin darüber hinaus gehende Ansätze, die in geeigneter Weise Zusammenhänge zwischen den gefundenen Regelmustern und der zugrunde liegenden Architektur des realen Kommunikationsnetzwerkes sichtbar machen.

Anhang A

Programmbedienung

In diesem Kapitel soll der Umgang mit dem im Rahmen dieser Arbeit weiter entwickelten Programm ISIS kurz erläutert werden. Dabei wird im folgenden Abschnitt zunächst auf die Bedienung von ISIS selbst eingegangen. Im Abschnitt A.2 wird anschließend die Möglichkeit aufgezeigt, ISIS mittels einer eigens dafür geschaffenen grafischen Benutzeroberfläche (Frontend) zu steuern. Abschnitt A.3 erläutert die Installation der vorgestellten Programme. Der abschließende Abschnitt A.4 gibt einen kurzen Überblick über bisherige Programmversionen.

A.1 Bedienung über Kommandozeile

Bei dem C++-Programm ISIS (Intelligent Search of Interesting Patterns in Sequences) handelt es sich um ein unter dem Betriebssystem Linux lauffähiges Werkzeug, das auf Ebene der Kommandozeile gesteuert wird. Die Bedienung von ISIS beschränkt sich somit auf die Übergabe von Parametern beim Programmaufruf. Während des Programmablaufs ist keine Interaktion mit dem Benutzer vorgesehen. Die Teile eines vollständigen Durchlaufs können einzeln durchgeführt werden, soweit dies logisch sinnvoll ist. Beispielsweise kann in einem ersten Arbeitsschritt die Alarmmuster-Findung durchgeführt werden, während die Regelmuster-Generierung erst später vorgenommen wird (da die Regelmuster-Generierung auf der Alarmmuster-Findung aufbaut, ist die umgekehrte Reihenfolge nicht sinnvoll). Ein typischer Aufruf sieht beispielsweise so aus:

Beispiel A.1 (Programmaufruf ISIS 1)

```
isis -A alarm.dbn -C config.dbn
     -L /home/mein_verzeichnis/log
     -f 100 -c 80 -w 120 -ap pPsS -rp pPsS
```

Großbuchstaben bezeichnen generell „fest stehende“ Dateien (Alarm-Datenbanken, Konfigurations-Datenbanken, ...) und Pfadangaben (Log = Verzeichnis, in das die Ergebnis-Dateien geschrieben werden sollen), während einstelligen Kleinbuchstaben „variable“ Werte oder auch keine Werte folgen. Zweistellige Kleinbuchstaben bezeichnen die Modus-Parameter. Im Einzelnen haben die Übergabeparameter folgende Bedeutungen:

`-A`	:	(Pfad und) Dateiname der Alarm-Datenbank
`-C`	:	(Pfad und) Dateiname der Konfigurations-Datenbank
`-c`	:	Minimale Vertrauenswürdigkeit in Prozent
`-e`	:	Erweiterte Form der Alarmnummer verwenden
`-f`	:	Minimale Frequenz in Prozent
`-h`	:	(oder `--help`) Hilfeausgabe
`-L`	:	Pfadangabe für die Ergebnis-Dateien (Verzeichnis muss existieren)
`-T`	:	(Pfad und) Dateiname der Template-Datenbank
`-t`	:	(Pfad und) Name der anzuwendenden Template-Datei
`-v`	:	Versionsangaben
`-w`	:	Angabe der Fensterbreite in Sekunden

Tabelle A.1: ISIS: Allgemeine Parameter

`-ap`	:	Modus für die Art der Suche nach Alarmmustern
`-rp`	:	Modus für die Art der Generierung von Regelmustern
`-pa`	:	Redundanz-Reduktion durch RL-Methode
`-pr`	:	Redundanz-Reduktion durch R-Methode
`-pl`	:	Redundanz-Reduktion durch L-Methode
`-ga`	:	Gruppierung von RL-behandelten Regelmustern
`-gr`	:	Gruppierung von R-behandelten Regelmustern
`-gl`	:	Gruppierung von L-behandelten Regelmustern
`-gu`	:	Gruppierung von unbehandelten Regelmustern
`-ta`	:	Template-Anwendung auf RL-behandelte Regelmuster
`-tr`	:	Template-Anwendung auf R-behandelte Regelmuster
`-tl`	:	Template-Anwendung auf L-behandelte Regelmuster
`-tu`	:	Template-Anwendung auf unbehandelte Regelmuster

Tabelle A.2: ISIS: Modus-Parameter

Die Reihenfolge der Parameter ist beliebig. Als einzige Bedingung gilt, dass die den Modus-Parametern nachgestellten Bezeichner für die zugrunde liegende Alarmmusterart zusammen geschrieben werden, also beispielsweise `-ap pPsS` und nicht `-ap p P s S`.

Beispiel A.1 lässt sich nunmehr wie folgt interpretieren: Gesucht werden soll nach allen vier Alarmmusterarten, wobei die zugehörigen Parameter $min\_freq = 100\%$ und

`P`	:	parallele Alarmmuster mit Wiederholungen
`p`	:	parallele Alarmmuster ohne Wiederholungen
`S`	:	serielle Alarmmuster mit Wiederholungen
`s`	:	serielle Alarmmuster ohne Wiederholungen

Tabelle A.3: ISIS: Arten von Alarmmustern

$window\_width = 120s$ betragen. Die Alarm-Datenbank ist in der Datei *alarm.dbn*, die Konfigurations-Datenbank ist in der Datei *config.dbn* des aktuellen Verzeichnisses enthalten. Die Ergebnisse werden in das Verzeichnis */home/mein_verzeichnis/log* geschrieben. Anschließend werden die erzeugten Dateien zur Generierung von Regelmustern aus allen vier Alarmmusterarten bei einem zugrunde liegenden Wert von $min\_conf = 80\%$ benutzt. Die Ergebnisse dieses Vorgangs werden wiederum nach */home/mein_verzeichnis/log* geschrieben.

Ein daran anschließender Aufruf könnte beispielsweise so aussehen:

Beispiel A.2 (Programmaufruf ISIS 2)

```
isis -L /home/mein_verzeichnis/log
     -f 100 -c 80 -w 120 -pa pPsS
```

Nunmehr sollen die erzeugten Regelmuster einer Redundanz-Reduktion nach RL-Methode unterworfen werden. Die dafür nötigen Regelmuster-Dateien sind wiederum im Verzeichnis */home/mein_verzeichnis/log* zu finden. Die Reduktion soll auf alle vier möglichen Regelmusterarten (der Begriff „Regelmusterart“ steht synonym für „generiertes Regelmuster aus Alarmmusterart“) angewendet werden. Mit Hilfe der Werte für min_freq, min_conf und $window\_widht$ werden die entsprechenden Namen der Regelmuster-Dateien generiert. Die Ergebnisse werden wie schon zuvor nach */home/mein_verzeichnis/log* geschrieben.

A.2 Bedienung über grafische Oberfläche

Um den Benutzer von ISIS von den Unannehmlichkeiten der Kommandozeile zu entbinden, wurde für den populären KDE-Desktop des Betriebssystems Linux eine grafische Benutzeroberfläche – entsprechend der Namenskonvention von KDE *KISIS* genannt – entworfen und realisiert. Der KDE-Desktop ist mittlerweile für die meisten kommerziellen und nichtkommerziellen Unix-Arten verfügbar, so dass einer Portierung beider Programme – ISIS und KISIS – auf diese Systeme nichts im Wege steht.

Die Bedienung von KISIS ist sowohl über Maus, als auch über Tastatur möglich, da alle Funktionen über Menü und „Shortcut“-Leiste erreichbar sind. Im linken oberen Fenster der Benutzeroberfläche können die eingestellten Parameter jederzeit überprüft werden, während die Ausgaben von ISIS im rechten oberen Fenster beobachtet werden können. Status- und Fehlermeldungen werden in den unteren, mittels Karteikartenreitern getrennten, Fenstern angezeigt. Die Pfad- und Parametereinstellungen erfolgen über entsprechende Dialogfenster.

A.3 Installation

Die Übersetzung und Installation beider Programmpakete erfolgt in der bei GNU-Programmen üblichen Weise mittels:

```
root@at73:~/temp > ./configure
root@at73:~/temp > make
root@at73:~/temp > make install
```

Dabei wird davon ausgegangen, dass die jeweiligen gepackten Programmpakete nach „`~/temp`“ entpackt wurden. Die Befehle können auch zusammengefasst werden:

```
root@at73:~/temp > ./configure && make && make install
```

Zur Vermeidung möglicher Schwierigkeiten sollten diese Schritte durch einen Benutzer mit „root“-Rechten erfolgen.

A.4 Programm-Historie

Im Folgenden soll ein kurzer Überblick über die bisherigen Programmversionen und ihren jeweiligen Fähigkeiten gegeben werden. Die Versionsnummer von KISIS muss für eine problemlose Zusammenarbeit mit der von ISIS übereinstimmen (beispielsweise ISIS 1.4 und KISIS 1.4.x), da es sonst – durch eine Vielzahl von Parameteränderungen während der Weiterentwicklung von ISIS bedingt – zu „Missverständnissen“ zwischen den beiden Programmen kommen kann.

Version 1.0	:	Alarmmuster-Findung
Version 1.1	:	Regelmuster-Generierung
Version 1.2	:	Redundanz-Reduktion
Version 1.3	:	Template-Anwendung
Version 1.4	:	Gruppierung

Tabelle A.4: ISIS: Programm-Historie

Anhang B

Programmdateien

Zum Betrieb von ISIS sind neben dem eigentlichen Programm einige zusätzliche Dateien notwendig. Dazu gehören vor allem Dateien mit den zu untersuchenden Alarmmeldungen sowie eine die Zusammensetzung und den Aufbau des zugehörigen Netzes wiederspiegelnde Konfigurationsdatei. Weiterhin werden die Ergebnisse eines Programmdurchlaufs in einer Vielzahl weiterer Dateien gespeichert. Der Aufbau dieser Dateien und ihre jeweiligen Aufgaben sollen im Folgenden näher beleuchtet werden.

B.1 Grundsätzliches

Für alle im Weiteren aufgeführten Dateien gelten folgende Regeln:

- Es ist nur eine Definition pro Zeile zulässig (eine Alarmmeldung, ein Konfigurationseintrag, ein Template usw.).
- Die Dateien dürfen keine Leerzeilen enthalten.
- Eine Zeile darf keine Leerzeichen enthalten (Ausnahme: *alarm.dbn*).
- Kommentare werden mit einem „`#`“am Zeilenanfang gekennzeichnet.
- Separator ist das Semikolon „`;`“.

Alle benötigten Dateien müssen in Textform vorliegen. Alle erzeugten Dateien werden ebenfalls in dieser Form gespeichert. Es ist also jederzeit möglich, die Ergebnisse eines Programmdurchlaufs mit Hilfe eines einfachen Editors zu betrachten. Dieses Verfahren bietet die Möglichkeit, Unzulänglichkeiten der gewählten Parameter frühzeitig erkennen und gegebenenfalls korrigierend eingreifen zu können.

B.2 Zusammensetzung der Dateinamen

Für die von ISIS selbstständig generierten Dateinamen gelten folgende Regeln:

Beispiel B.1 (Dateinamen)

```
s+100-60s.apn
p+100-60s-90.rpn
s-80-120s-20.pln
p-80-120s-20.prn
```

Die ersten beiden Zeichen definieren die Art der zugrunde liegenden Alarmmuster. Dabei steht „`p`“ für parallele und „`s`“ für serielle Alarmmuster. Das folgende Plus „`+`“ steht für Alarmmuster mit Wiederholungen, das Minus „`-`“ für Alarmmuster ohne Wiederholungen (siehe Abschnitt 2.2.1). Die darauf folgende Zahl kennzeichnet den Wert für min_freq in Prozent. Alle auf min_freq folgenden Minuszeichen dienen nur noch der Übersichtlichkeit. Eine Zahl gefolgt von einem s definiert den zugrunde liegenden Wert für $window\_width$ in Sekunden. Bei Regelmusterdateien folgt auf $window\_width$ noch die Kennzeichnung für min_conf – wiederum in Prozent.

ap	:	Alarmmuster (alarm patterns)
rp	:	Regelmuster (rule patterns)
		Regelmuster, reduziert nach:
pa	:	RL-Methode (pruning all – RL-Muster)
pr	:	R-Methode (pruning right – R-Muster)
pl	:	L-Methode (pruning left – L-Muster)
ga	:	gruppierte RL-Muster (grouping all)
gr	:	gruppierte R-Muster (grouping right)
gl	:	gruppierte L-Muster (grouping left)
gu	:	gruppierte Regelmuster (grouping unpruned)
ta	:	templatebehandelte RL-Muster (template all)
tr	:	templatebehandelte R-Muster (template right)
tl	:	templatebehandelte L-Muster (template left)
tu	:	templatebehandelte Regelmuster (template unpruned)
db	:	Datenbank-Datei (data base)

Tabelle B.1: Dateiendung: Bedeutung der Stellen 1 und 2

Die Dateiendungen – sie sind immer dreistellig – sind nach folgenden Festlegungen aufgebaut: Die Stellen eins und zwei kennzeichnen den Inhalt der Datei, Stelle drei enthält ein Kürzel für die zugrunde liegende Netzhardware.

n	:	Netzhardware der Firma Nokia

Tabelle B.2: Dateiendung: Bedeutung der Stelle 3

In Beispiel B.1 steht demnach der erste Eintrag für eine Datei mit Alarmmuster, die aus Alarmmeldungen gewonnen wurden, welche von Netzelementen der Firma Nokia generiert wurden. Es handelt sich dabei um serielle Alarmmuster mit Wiederholungen. Die zugehörigen Werte sind: $min\_freq = 100\%$ und $window\_width = 60s$.

B.3 Alarmmeldungen – *alarm.dbn*

ISIS wurde mit dem Ziel entwickelt, Korrelationen von Alarmmeldungen in GSM-Mobilfunknetzen zu entdecken. Dazu wird eine Alarm-Datenbank in Textform benutzt, deren Einträge folgende Form besitzen:

Beispiel B.2 (Alarmmeldung)

```
12-60-0001;12-70-0093;12-71-0102;12-72-0176;
  24;7166811;4;7006;TRX/FU:NO DATA FROM DSPx TO FUCx;
    1998-02-27 03:16:30;0;1998-02-27 03:16:30;0;
      autoack;1998-02-27 03:16:30;system;
```

Eine solche Alarmmeldung entspricht im Wesentlichen der Art von Meldung, die ein Netzelement der Firma Nokia erzeugt. Prinzipiell ist aber jede in Textform darstellbare Art von Alarmen nach einer Anpassung der Einlesefunktion verarbeitbar.

Den einzelnen Elementen der Meldung werden dabei folgende Bedeutungen zugeordnet:

`12-60-0001`	:	root_object
`12-70-0093`	:	child_object_1
`12-71-0102`	:	child_object_2
`12-72-0176`	:	child_object_3
`24`	:	managed_object_class
`7166811`	:	consec_number
`4`	:	severity_number
`7006`	:	alarm_number
`TRX/FU...`	:	alarm_text
`1998-02-27`	:	event_time
`03:16:30`	:	event_time
`0`	:	alarm_status

```
1998-02-27   :  acknowledge_time
03:16:30     :  acknowledge_time
0            :  acknowledge_status
autoack      :  acknowledged_by
1998-02-27   :  cancel_time
03:16:30     :  cancel_time
system       :  cancelled_by
```

Für die weitere Verarbeitung der Alarmmeldungen sind jedoch nur wenige Elemente von Bedeutung. Dazu gehört ‚child_object_3', das als Auslöser des Alarms betrachtet wird, sowie ‚alarm_number' als eindeutigen Bezeichner für die Art des Alarms. Durch ‚event_time' wird die Auftretenszeit des Alarms festgehalten.

Beispiel B.3 (Erweiterte Alarmnummer)
Sei `7706` eine Alarmnummer, die von einem Netzelement mit der Bezeichnung `12-72-0176` mit einer Alarmmeldung erzeugt wurde. Dann lautet die zugehörige *erweiterte Alarmnummer*: `770620176`.

Es besteht die Möglichkeit, child_object_3 und alarm_number zu verschmelzen um eine „feinere" Auflösung des Alarmmeldungs-Bestandes zu erreichen. Diese *erweiterte Alarmnummer* besitzt die Form $xxxxYzzzz$. Dabei entspricht $xxxx$ dem Ursprungsalarm, Y der Hierarchieebene (6 = BSC, 7 = BS, 1 = BTS, 2 = TRX) und $zzzz$ der Elementnummer von child_object_3 (Beispiel B.3).

B.4 Netzabbild – *config.dbn*

Die für den Programmablauf benötigte Konfigurations-Datenbank spiegelt den logischen Aufbau der Netzelemente wider. Ein Eintrag dieser Datenbank ordnet einem Mutter-Element die zugehörigen Tochter-Elemente zu. Die Datenbank selbst liegt wie die Alarm-Datenbank in Textform vor:

Beispiel B.4 (Konfigurations-Datenbank)

```
12-60-0001(12-70-0048;12-70-0049;12-70-0051;12-70-0052;...)
12-70-0048(12-71-0022;12-71-0023;12-71-0024)
12-71-0022(12-72-0070;12-72-0133)
12-71-0023(12-72-0071;12-72-0134)
12-71-0024(12-72-0072;12-72-0135)
```

```
12-72-0070()
12-72-0133()
...
```

Der Bezeichner eines Netzelementes setzt sich aus seinem Gültigkeitsbereich oder Sektor (12 steht beispielsweise für Hannover), seinem Typ (60 = BSC, 70 = BS, 71 = BTS, 72 = TRX), sowie einer Nummer für die Unterscheidung der Netzelemente einer Hierarchiestufe untereinander zusammen. Die Nomenklatur der Netzelement-Bezeichner entspricht ebenfalls der der Firma Nokia. Wie bei der Alarm-Datenbank ist eine Anpassung an andersartig aufgebaute Bezeichner durch Modifikationen an der zuständigen Einlesefunktion möglich.

B.5 Templates – *template.dbn*

Für die Anwendung von Templates auf den Regelmusterbestand sind zwei weitere Dateien nötig. Die Datei *template.dbn* enthält dabei Angaben darüber, welche Templates überhaupt existieren und welche Alarmmeldungen ihnen zugeordnet werden. Bei *template.dbn* handelt es sich somit um eine in Textform vorliegende Template-Datenbank:

Beispiel B.5 (Template-Datenbank)

```
important(6082;6085;8000:8999)
less_important(0000:2999;3099;3150:3999)
all_alarms(important;less_important)
```

Ein Ausdruck der Form `something(...)` stellt dabei ein einzelnes Template dar, wobei pro Zeile nur ein derartiges Template vorkommen darf. Innerhalb der Klammern wird definiert, welche Alarmmeldungen diesem Template zuzuordnen sind. Dabei ist `6082` ein Beispiel für eine einzelne Alarmmeldung, `8000:8999` schließt alle Meldungen von 8000 bis 8999 ein. Zur Vereinfachung einer Template-Definition können innerhalb der Klammern auch weitere Templates erscheinen. Diese stehen dann stellvertretend für die durch sie umfassten Alarmmeldungen. Eine einzelne Alarmmeldung besitzt für sich alleine bereits Template-Charakter und muss daher nicht explizit als solches definiert werden (Beipsiel B.6).

In einer zweiten Datei, deren Name vom Anwender beliebig gewählt werden kann, werden die auf den Regelmusterbestand anzuwendenden Templates definiert. Der Aufbei dieser Datei ist dabei folgender:

Beispiel B.6 (Template-Datei)

```
in(all_alarms+)(important+;all_alarms*)
ex(all_alarms+)(6082+;all_alarms*)
```

Ein Ausdruck der Form `in(...)(...)` bezeichnet ein einschließendes Template, ein Ausdruck der Form `ex(...)(...)` ein ausschließendes Template. Wiederum ist pro Zeile nur eine Definition möglich. Es können beliebig viele ein- und ausschließende Templates aufgeführt werden.

Ein dem Templatenamen nachgestelltes Plus („`+`“) bedeutet, dass die damit verbundene Bedingung mindestens einmal erfüllt werden muss, ein nachgestellter Stern („`*`“) bedeutet, dass die verbundene Bedingung erfüllt werden darf (siehe Abschnitt 2.4.3 für weitergehende Erläuterungen). Jedem Templatenamen muss eines der beiden Zeichen nachgestellt werden, da nur so Fehldefinitionen durch Vergessen eines Zeichens vermieden werden können.

Durch die jeweils erste Klammerung einer Zeile werden die auf die linke Seite der Regel anzuwendenden Templates beschrieben, durch die zweite Klammerung die auf die rechte Seite anzuwendenden Templates. Pro Seite muss mindestens ein Template das nachgestellte Plus enthalten, da nur so garantiert wird, dass die linken und rechten Regelseiten keine leeren Mengen enthalten. So werden durch Beispiel B.6 alle Regelmuster gefunden, die mindestens eine Alarmmeldung auf der linken Seite der Regel enthalten und mindestens eine „wichtige“ Alarmmeldung auf der rechten Seite – solange darunter nicht mindestens einmal die Meldung 6082 vorkommt. Das einschließende Template `in(all_alarms+)(important+;all_alarms*)` erlaubt beispielsweise unwichtige Meldungen auf der rechten Regelseite, während ein Template der Form `in(all_alarms+)(important+)` nur Regeln zulassen würde, die *ausschließlich* wichtige Alarmmeldungen auf der rechten Seite aufweisen. Templates lassen somit immer nur Regeln zu, die exakt zu ihnen passen.

B.6 Alarmmuster-Dateien – *.apn*

In die *.apn*-Dateien werden die gefundenen Alarmmuster geschrieben.

Beispiel B.7 (Alarmmuster-Datei)

```
12-70-0054;2.92146e-05;2567;7706;7711;
12-70-0054;3.18705e-05;7705;7706;7711;
12-70-0054;2.91626e-05;2567;7705;7706;7711;
```

Jede Zeile enthält ein Alarmmuster. Die einzelnen Teile eines Musters werden mittels Semikolon separiert. Der erste Eintrag bezeichnet das Cluster, dem das Muster zuzuordnen ist (siehe [Tdl00], Abschnitt 5.3 für weiterführende Erläuterungen). Die zweite Zahl entspricht der Frequenz des Musters. Alle daran anschließenden Zahlen stellen Alarmmeldungen dar und entsprechen somit dem gefundenen Alarmmuster.

B.7 Regelmuster-Dateien – *.rpn*

In die *.rpn*-Dateien werden die generierten Regelmuster geschrieben.

Beispiel B.8 (Regelmuster-Datei)

```
12-70-0054;5.0045e-05;0.993794;2567;7706;->;7705;
12-70-0054;3.18705e-05;0.998368;7706;7711;->;7705;
12-70-0054;2.91626e-05;0.99822;2567;7706;7711;->;7705;
```

Jede Zeile enthält ein Regelmuster. Die einzelnen Teile eines Musters werden mittels Semikolon separiert. Der erste Eintrag bezeichnet wiederum das Cluster, dem das Regelmuster zuzuordnen ist. Die zweite Zahl entspricht der Frequenz, die dritte Zahl dem Vertrauenswert des Musters. Alle daran anschließenden Zahlen stellen Alarmmeldungen dar und entsprechen somit dem eigentlichen Regelmuster. Die Trennung von linker und rechter Regelseite wird durch das Pfeilzeichen „`->`“ symbolisiert.

B.8 Regelsets und RL-Methode – *.pXn*

In die *.pXn*-Dateien werden die redundanz-reduzierten Regelmuster geschrieben. Die Unterscheidung zwischen den einzelnen Reduktionsarten wird über den Platzhalter *X* in der Dateiendung realisiert (Tabelle B.2). Der Aufbau der Dateien ist identisch mit den *.rpn*-Dateien.

B.9 Gruppierte Regelmuster – *.gXn*

In die *.gXn*-Dateien werden die gruppierten Regelmuster geschrieben. Die Unterscheidung zwischen den zugrunde liegenden Regelmustern wird über den Platzhalter *X* in der Dateiendung realisiert (Tabelle B.2). Der Aufbau der Dateien ist nahezu identisch mit den *.rpn*-Dateien. Der Unterschied besteht in der mitaufgeführten *group*-Variablen am Ende eines Regelmusters (siehe Abschnitt 2.4.2).

B.10 Templatebehandelte Regelmuster – *.tXn*

In die *.tXn*-Dateien werden die templatebehandelten Regelmuster geschrieben. Die Unterscheidung zwischen den zugrunde liegenden Regelmustern wird über den Platzhalter *X* in der Dateiendung realisiert (Tabelle B.2). Der Aufbau der Dateien ist wiederum identisch mit den *.rpn*-Dateien.

Anhang C

Programmklassen

In diesem Kapitel werden die bei der Implementierung des Programms ISIS entworfenen C++-Klassen kurz vorgestellt. Dabei werden keine Erklärungen zum realisierten Programmcode geliefert, da dafür die in den Quelltexten reichlich vorhandenen Kommentare besser geeignet sind. Ziel ist es vielmehr, dem Entwickler künftiger Erweiterungen zur leichteren Einarbeitung eine Übersicht über die vorhandene Programmstruktur zu verschaffen.

Obwohl die folgenden Ausführungen keine Code-Elemente enthalten und das vorliegende Kapitel nicht dem Zweck dient, die bei der Umsetzung von ISIS gewählten Lösungen im Detail zu erläutern, so werden doch für das Verständnis des Textes grundsätzliche Kenntnisse der objektorientierten Programmierung in C++ vorausgesetzt.

C.1 Wurzelklasse – *GenericObject*

erbt von	:	—
vererbt an	:	*ParmeterAnalysis, AlarmElement, NetElement, RuleElement, TimeElement, GenericCluster, AlarmPatterns, RulePatterns, RulePruning, RuleClassification*

Die Klasse *GenericObject* stellt den Ursprung der verwendeten Klassenhierarchie dar. In ihr sind Funktionen implementiert, die für alle anderen Klassen ebenfalls nützlich sind. Dazu gehört zur Zeit vor allem die Möglichkeit, C- oder C++-Strings in die entsprechenden Integer- oder Float-Werte zu wandeln. Desweiteren werden in der Header-Datei dieser Klasse alle global notwendigen Bibliotheks-Dateien wie *iostream*, *string*, *vector*, *list* usw. eingebunden.

C.2 Übergabeparameter – *ParameterAnalysis*

erbt von : *GenericObject*
vererbt an : —

ISIS ist ein Programm dessen Parameter – beim gegenwärtigen Entwicklungsstand – ausschließlich auf der Ebene der Kommandozeile übergeben werden. Die Auswertung dieser Parameter und Umsetzung in die entsprechenden Datenstrukturen wird von der Klasse *ParameterAnalysis* übernommen. Weiterhin werden in dieser Klasse alle benötigten Dateinamen für die Speicherung der gewonnenen Ergebnisse generiert. Die Prüfung der Existenz der für einen Programmdurchlauf notwendigen Datenbank-Dateien gehört ebenso zu den Aufgaben von *ParameterAnalysis* wie die Bereitstellung einer Hilfefunktion.

C.3 Interne Datenstruktur – *NetElement*

erbt von : *GenericObject*
vererbt an : —

Die Datenstruktur *NetElement* bildet den Aufbau eines Netzelementes ab. Dazu gehört der Sektor oder Gültigkeitsbereich, der Typ sowie die Unterscheidungsnummer des Elementes (siehe Abschnitt B.4). Die Klasse unterstützt die Operatoren für ‚kleiner als', ‚größer als' und Identität. *NetElement* wird von *AlarmElement* benutzt.

C.4 Interne Datenstruktur – *TimeElement*

erbt von : *GenericObject*
vererbt an : —

TimeElement bildet die verschiedenen, in einer Alarmmeldung vorkommenden Zeitenangaben nach Datum und Uhrzeit getrennt ab. Die Klasse unterstützt die Operatoren für ‚kleiner als', ‚größer als' und Identität, so dass die Anordnung der zugehörigen Meldungen auf einem Zeitstrahl möglich wird. *TimeElement* wird wie *NetElement* von *AlarmElement* benutzt.

C.5 Interne Datenstruktur – *AlarmElement*

erbt von : *GenericObject*
vererbt an : —

Bei der Klasse *AlarmElement* handelt es sich um eine Datenstruktur, die eine komplette Alarmmeldung, wie sie in Abschnitt B.3 näher erläutert wird, aufnehmen kann. Dabei werden die einzelnen Teile der Meldung aus der vorliegenden, durch die Speicherung in einer Textdatei bedingten, Zeichenkettenform in ihre jeweiligen „natürlichen“ Formen gewandelt (ganze Zahlen nach Integer, Text nach String, Datum- und Zeitangaben nach *TimeElement* ...).

C.6 Interne Datenstruktur – *RuleElement*

erbt von : *GenericObject*
vererbt an : —

Eine Regel ist eine Struktur auf einem hohen Abstraktionsniveau. Diesem Umstand wird durch die Klasse *RuleElement* Rechnung getragen. Die Klasse realisiert die häufig gebrauchten Operationen für Vergleiche von Regeln, Test auf Teilmenge einer Regelseite sowie Sortieren. Weiterhin ermöglicht *RuleElement* einen schnellen Zugriff auf die Elementanzahlen der beiden Regelseiten sowie auf die in Algorithmus 2.1 benötigte Skip-Zahl.

C.7 Interne Datenstruktur – *TemplateElement*

erbt von : *GenericObject*
vererbt an : —

Durch die Klasse *TemplateElement* wird die Abbildung der in Abschnitt 2.4.3 vorgestellten einschließenden und ausschließenden Templates in die entsprechende Datenstruktur realisiert. Die Klasse implementiert darüber hinaus keine weiteren Funktionen.

C.8 Datenvorverarbeitung – *GenericCluster*

erbt von : *GenericObject*
vererbt an : *NokiaCluster*

Um zu verhindern, dass Zusammenhänge in einem Datenbestand gesucht werden, die aus verschiedenen Gründen nicht existieren können, muss vor der Suche nach Alarmmustern eine geeignete Datenvorverarbeitung durchgeführt werden. Diese Aufgabe wird von den in der Klassenhierarchie von ISIS unterhalb der Klasse *GenericCluster* angesiedelten Klassen übernommen. Bei *GenericCluster* selbst handelt es sich beim gegenwärtigen Entwicklungsstand um eine reine Stummelklasse, in die zu einem späteren Zeitpunkt gemeinsame Funktionen der unter ihr angeordneten Klassen implementiert werden sollen.

C.9 Nokia-Netzelemente – *NokiaCluster*

erbt von : *GenericCluster*
vererbt an : —

Die Klasse *NokiaCluster* stellt die Implementierung der Datenvorverarbeitung für Netzelemente und Netzstrukturen der Firma Nokia dar ([Tdl00], Abschnitt 5.3). Der Benutzung der Klasse erfolgt in einfacher Weise durch die einzige nach außen sichtbare Funktion *nokia_cluster()* nach Instanzierung der Klasse.

C.10 Alarmmuster-Findung – *AlarmPatterns*

erbt von : *GenericObject*
vererbt an : —

Die Suche nach Alarmmustern auf einer Ereignissequenz wird durch die Klasse *AlarmPatterns* realisiert. Die Klasse unterstützt dabei die Suche nach parallelen und seriellen Mustern sowohl mit als auch ohne Wiederholungen (Abschnitt 2.2.1). Nach Instanzierung der Klasse wird die Suche in einfacher Weise durch Aufruf der einzigen nach außen sichtbaren Funktion *alarm_patterns()* gestartet. *AlarmPatterns* stellt die bedeutendste Klasse innerhalb von ISIS dar, da die von ihr gefundenen Alarmmuster Grundlage für alle weiteren Bearbeitungsschritte sind.

C.11 Regelmuster-Generierung – *RulePatterns*

erbt von : *GenericObject*
vererbt an : —

Die auf die Alarmmuster-Findung aufbauende Regelmuster-Generierung (Vorstellung in Abschnitt 2.2) wird durch die Klasse *RulePatterns* implementiert. Regelmuster können aus allen vier zuvor erwähnten Alarmmusterarten generiert werden. Nach Instanzierung der Klasse ist dies durch Aufruf der einzigen nach außen sichtbaren Funktion *rule_patterns()* möglich.

C.12 Regelmuster-Reduktion – *RulePruning*

erbt von : *GenericObject*
vererbt an : —

Die Redundanz-Reduktion von Regelmustern (Abschnitt 2.3) übernimmt die Klasse *RulePruning*. Sie ermöglicht die Reduktion nach repräsentativen linken Regelsets (L-Methode), nach repräsentativen rechten Regelsets (R-Methode), sowie durch Kombination beider Verfahren mittels rechts-links Methode (RL-Methode). Der Reduktions-Vorgang wird gestartet durch Aufruf der einzigen nach außen sichtbaren Funktion *rule_pruning()* der instanzierten Klasse.

C.13 Regelmuster-Klassifizierung – *RuleClassification*

erbt von : *GenericObject*
vererbt an : *RuleGrouping, RuleTemplates*

Die Klasse *RuleClassification* bietet gemeinsam nutzbare Funktionen und Variablen für die mit der Klassifizierung von Regelmustern betrauten Klassen an (Abschnitt 2.4). Dazu gehört eine Funktion für das Einlesen der Regelmuster-Dateien sowie eine Funktion für das Speichern der Ergebnis-Dateien.

C.14 Gruppierung – *RuleGrouping*

erbt von : *RuleClassification*
vererbt an : —

Das Gruppieren von Regelmustern mittels Zuordnung (Abschnitt 2.4.2) wird durch die Klasse *RuleGrouping* realisiert. Die Benutzung dieser Klasse besteht in dem Aufruf der einzigen nach außen sichtbaren Funktion *rule_grouping()* nach Instanzierung der Klasse.

C.15 Templates – *RuleTemplates*

erbt von : *RuleClassification*
vererbt an : —

Die Auswahl bestimmter Regelmuster nach dem Template-Verfahren (Abschnitt 2.4.3) wird duch die Klasse *RuleTemplates* realisiert. Die Benutzung dieser Klasse besteht in dem bereits gewohnten Aufruf der wiederum einzigen nach außen sichtbaren Funktion *rule_templates()* nach Instanzierung der Klasse.

Anhang D

Ergänzungen

Bei der Weiterführung der in [Tdl00] begonnenen Arbeit hat sich herausgestellt, dass der dort in Kapitel 5 aufgeführte und auf [Toi96] zurückgehende Algorithmus 5.4 unvollständig ist. Zumindest war es dem Autor dieser Arbeit nicht möglich, mit dem originalen Algorithmus den zugehörigen Programmteil in ISIS so zu implementieren, dass er fehlerfreie Ergebnisse liefern konnte.

Als Konsequenz wurde der Algorithmus erweitert, so dass nunmehr die erzeugten Ergebnisse mit den erwarteten Ergebnissen übereinstimmen. Die vorliegende Ausführung des Algorithmus ermöglicht somit eine fehlerfreie Implementierung des Prozesses der Erkennung frequenter serieller Alarmmuster.

Die Änderungen gegenüber Algorithmus 5.4 aus [Tdl00] betreffen die dortigen Zeilen 30, 32 und 34. Hinzugekommen sind in Algorithmus D.1 die Zeilen 32 und 36.

C_l bezeichnet wiederum das Set zu prüfender Kandidaten, F_l ist das Set frequenter Alarmmuster. Geprüft wird anhand der Ereignissequenz s, wobei durch T_s der erste zur Sequenz gehörende, mit T^s der erste *nicht* mehr zur Sequenz gehörende Zeitpunkt dargestellt wird. Durch $window\_width$ und min_freq werden Fensterbreite und minimale Frequenz der Alarmmuster in bekannter Weise definiert.

Eingabe: $C_l, s = (T_s, T^s, s), window\_width, min\_freq$
Ausgabe: $F_l = f(window\_width, min\_freq)$

// Fortsetzung auf Seite 75

1: **for** jedes a in C **do**
2: **for** $i := 1$ **to** $|a|$ **do**
3: $a\_initialized[i] := 0$;
4: $waits\_A(a[i]) := \emptyset$;
5: **for** jedes $a \in C$ **do**
6: $waits\_A(a[1]) := waits\_A(a[1]) \cup \{(a, 1)\}$;
7: $a\_freq\_count := 0$;
8: **for** $t := T_s - window\_width$ **to** $T_s - 1$ **do**
9: $begins\_at(t) := \emptyset$;
10: **for** $start := T_s - window\_width + 1$ **to** T^s **do**
11: $begins\_at(start + window\_width - 1) := \emptyset$;
12: $transitions := \emptyset$;
13: **for** alle Alarme (A, t) in s, so dass gilt $t = start + window\_width - 1$ **do**
14: **for** alle $(a, j) \in waits\_A(A)$ **do**
15: **if** $j = |a|$ **and** $a\_initialized[j] = 0$ **then**
16: $a\_in\_window := start$;
17: **if** $j = 1$ **then**
18: $transitions := transitions \cup \{(a, 1, start + window\_width - 1)\}$;
19: **else**
20: $transitions := transitions \cup \{(a, j, a\_initialized[j - 1])\}$;
21: $begins\_at(a\_initialized[j - 1] :=$
$begins\_at(a\_initialized[j - 1]) \setminus \{(a, j - 1)\}$;
22: $a\_initialized[j - 1] := 0$;
23: $waits\_A(A) := waits\_A(A) \setminus \{(a, j)\}$;
24: **for** alle $(a, j, t) \in transitions$ **do**
25: $a\_initialized[j] := t$;
26: $begins\_at(t) := begins\_at(t) \cup \{(a, j)\}$;
27: **if** $j < |a|$ **then**
28: $waits\_A(a[j + 1]) := waits\_A(a[j + 1]) \cup \{(a, j + 1)\}$;
29: **for** alle $(a, l) \in begins\_at(start - 1)$ **do**
30: **if** $l = |a|$ **and** $a\_initialized[l] = start - 1$ **then**
31: $a\_freq\_count := a\_freq\_count - a\_in\_window + start$;
32: $a\_initialized[l] := 0$;
33: **else if** $l \neq |a|$ **then**
34: $waits\_A(a[l + 1]) := waits\_A(a[l + 1]) \setminus \{(a, l + 1)\}$;
35: $a\_initialized[l] := 0$;
36: $begins\_at(start - 1) := \emptyset$;
37: **for** alle Alarmgruppen a in C **do**
38: **if** $a\_freq\_count/(T^s - T_s + window\_width - 1) \geq min\_freq$ **then**
39: gib a aus

Algorithmus D.1: Erkennen frequenter serieller Alarmgruppen

Abkürzungsverzeichnis

BS	Base Station
BTS	Base Transceiver Station
GNU	GNU Not Unix
GSM	Global System for Mobile Communication
KDE	K Desktop Environment
MSC	Mobile Station Controller
TRX	Transmitter Receiver Exchange

Abbildungsverzeichnis

Algorithmenverzeichnis

Tabellenverzeichnis

Index

Literaturverzeichnis

[KMR94] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, A. I. Verkamo
Finding Interesting Rules from Large Sets of Discovered Association Rules
University of Helsinki
1994

[TKR95] H. Toivonen, M. Klemettinen, P. Ronkainen, K. Hätönen, H. Mannila
Pruning and Grouping Discovered Association Rules
University of Helsinki
1995

[HKM96] K. Hätönen, M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen
Knowledge Discovery from Telecommunication Network Alarm Databases
University of Helsinki
1996

[HK96a] K. Hätönen, M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen
Rule Discovery in Alarm Databases
University of Helsinki
1996

[Toi96] Hannu Toivonen
Discovery of Frequent Patterns in Large Data Collections
University of Helsinki
ISBN 951-45-7531-8
1996

[Toi96a] Hannu Toivonen
Sampling Large Databases for Association Rules
University of Helsinki
1996

[DLM98] Gautam Das, King-Ip Lin, Heikki Mannila, Gopal Renganathan, Padhraic Smyth
Rule discovery from time series
University of Memphis and University of Helsinki and University of California at Irvine
1998

[Fer98] Reginald Ferber
Data Mining und Information Retrieval
Technische Universität Darmstadt
1998

[RMS98] Sridhar Ramaswamy, Sameer Mahajan, Avi Silberschatz
On the Discovery of Interesting Patterns in Association Rules
Bell Labs and Informix Corp.
1998

[Tdl00] Peter Tondl
Erkennung wiederkehrender Fehlerszenarien in Kommunikationsnetzen durch Data Mining-Algorithmen
Universität Hannover
2000