HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG

Fakultät Informatik

Allensche Zeitlogik

Ausarbeitung im Fach Wissensbasierte Systeme
Prof. Dr. Ulrich Hedtstück
Wintersemester 2007/2008

von

Max Nagl
nagl@fh-konstanz.de

Andreas Hofmann
ahofmann@htwg-konstanz.de

Inhaltsverzeichnis

1. Einleitung

In der Regel werden Wissensdatenbanken statisch modelliert, ohne ein Konzept von Zeit. In vielen Fällen, etwa in der Sprachanalyse, Ereignissimulation oder Prozessmodellierung, spielen zeitlichen Beziehungen jedoch eine entscheidende Rolle. Wollen wir also Wissen über diese Ereignisse in einer Wissensdatenbank ablegen, so muss diese wohl oder übel auch zeitliches Wissen modellieren. Dies ist jedoch kein triviales Unterfangen, da zeitliche Beziehungen oft nicht in absoluten Zahlen ausdrückbar sind, sondern nur durch relative Informationen zueinander. Außerdem sind oft nicht alle Beziehungen explizit angegeben, sondern müssen implizit aus anderen Beziehungen hergeleitet werden.

Die Zeitlogik von James F. Allen [1983] ist ein Versuch, ein Modell zu entwickeln, dass diesen Eigenarten zeitlichen Wissens Rechnung trägt und dieses nicht nur modellieren kann, sondern auch die Möglichkeit bietet, computergestützt aus bekannten zeitlichen Relationen neues Wissen zu erschließen und die Konsistenz gegebener Zeitdaten zu überprüfen.

Allen ist Professor an der University of Rochester im Fachbereich Computer Science. Seine Hauptinteressengebiete liegen im Bereich der künstlichen Intelligenz, insbesondere der Wissensdarstellung und der Verarbeitung natürlicher Sprache. [University of Rochester, 2004]

2. Grundlagen

Dieses Kapitel beschäftigt sich mit den grundlegenden Problemen der Zeitmodellierung, stellt vorhandene Ansätze vor und bewertet diese.

2.1. Anforderungen an Zeitmodelle

Zunächst sollte erst einmal die Frage geklärt werden, was ein Zeitmodell überhaupt leisten muss. Allen [1983] identifiziert vier zentrale Anforderungen an Zeitmodelle und deren Darstellung von zeitlichem Wissen:

1. Die Darstellung sollte signifikante Ungenauigkeit erlauben. Zeitliches Wissen ist häufig relativ (zum Beispiel A ist vor B) und damit unabhängig von absoluten Daten.
2. Die Darstellung sollte unsichere Informationen berücksichtigen. Oft sind die genauen zeitlichen Beziehungen zwischen zwei Vorgängen nicht bekannt, es können jedoch schon einige Einschränkungen (Constraints) hinsichtlich der in Frage kommenden Möglichkeiten getroffen werden.
3. Die Darstellung sollte flexibel in Bezug auf die Granularität der Zeitdaten sein. Um zum Beispiel geschichtliche Ereignisse zu modellieren, sind grobgranulare Einheiten wie Tage, Jahre oder Jahrtausende nötig, während für die Modellierung von Ereignissen innerhalb eines Computersystems der Nanosekundenbereich interessant ist.
4. Das Modell sollte Persistenz unterstützen. Einmal etablierte Fakten sollten weiterhin als gültig vermutet werden, bis das Gegenteil nachgewiesen ist. Wenn jemand beispielsweise sein Auto am Montag auf einem Parkplatz abstellt, erwartet er, dass es am Dienstag immer noch dort steht. Dies muss nicht der Fall sein (das Auto könnte gestohlen oder abgeschleppt worden sein), wird jedoch erst einmal als Normalzustand angenommen.

Diese Liste ist natürlich weder vollständig, noch für alle Anwendungsgebiete obligatorisch. Sie trägt Allens Ausrichtung auf Künstliche Intelligenz, insbesondere Spracherkennung, Rechnung.

2.2. Bisherige Ansätze

Die bisherigen Ansätze zur Modellierung zeitlicher Zusammenhänge lassen sich grob in vier Kategorien einteilen [Allen, 1983]:

- Zustandsmodelle
- Zeitliniensysteme
- Vor/Nach-Ketten
- Formelle Modelle

Im Folgenden werden die verschiedenen Ansätze kurz umrissen und auf Erfüllung der im vorherigen Abschnitt aufgestellten Anforderungen überprüft.

2.2.1. Zustandsmodelle

Zustandsmodelle haben kein wirkliches Konzept von Zeit, ein Zustand stellt immer eine Beschreibung der Welt (also eine Datenbank mit Fakten) *in einem bestimmten Augenblick* dar. Durch das Eintreten von Ereignissen bzw. Aktionen kann ein Zustand in einen anderen Zustand überführt werden. Tritt zum Beispiel ein Ereignis ein, durch welches der Fakt A wahr und der Fakt B falsch wird, so ist der resultierende Zustand schlicht eine Kopie des Ursprungszustands, dem A hinzugefügt und aus dem B entfernt wurde. Durch Speicherung aller nacheinander folgenden Zustände erhält man eine Art grober zeitlicher Modellierung, was jedoch in der Regel wegen des hohen Speicherbedarfs nicht praktikabel ist, zumal diese Methode keine detaillierten Schlüsse über die zeitlichen Zusammenhänge zulässt. Die einzige der in Abschnitt 2.1 aufgestellten Anforderungen, die erfüllt wird, ist die Persistenz: Ein Fakt verbleibt im System, bis er explizit gelöscht wird.

2.2.2. Zeitliniensysteme

In Zeitliniensystemen ist jeder Fakt mit einem Zeitstempel indiziert. Ein Zeitstempel ist die Darstellung eines bestimmten Zeitpunktes. In der Regel sind Zeitstempel darauf ausgelegt, dass man durch möglichst einfache mathematische Operationen berechnen kann, in welcher Abfolge sie auftreten und wie lange sie auseinander liegen. Klassischerweise ist ein Zeitstempel eine Ganzzahl, so dass einfache Substraktion diese Informationen liefert. Dieser Ansatz bietet sich an, wenn *jedem* Ereignis im zu modellierenden System ein genauer, absoluter Zeitpunkt zugewiesen werden kann. Dies ist jedoch oft nicht der Fall, insofern bietet dieser Ansatz nicht genug Toleranz gegenüber Ungenauigkeit. Viele relative Beziehungen können nicht modelliert werden, zum Beispiel kann nicht festgehalten werden, dass A und B nicht gleichzeitig sind, ohne zu entscheiden, ob A vor oder nach B ist.

2.2.3. Vor/Nach-Ketten

Vor/Nach-Ketten sind eine abgewandelte Form von Zeitliniensystemen. Statt absoluter Zeitstempel, aus denen sich relative Vor/Nach-Beziehungen berechnen lassen, werden hier diese relativen Beziehungen direkt gespeichert. Für zwei Ereignisse A und B gibt es also eine Relation, die deren Beziehung zueinander beschreibt. Klassische Vor/Nach-Ketten kennen nur die Relationen „ist gleich“, „vor“ und „nach“. Damit unterliegen sie ähnlichen Einschränkungen wie Zeitliniensysteme, sprich nicht alle Beziehungen können abgebildet werden. Sie bieten jedoch schon eine Mechanik für zeitliches Schließen. So lässt sich zum Beispiel aus „A vor B“ und „B vor C“ durch Kettenbildung (daher der Name) schließen, dass „A vor C“ gilt. Die Allensche Zeitlogik baut auf diesem Modell auf und erweitert es signifikant.

2.2.4. Formale Modelle

Abgesehen von diesen Grundtypen existiert eine Vielzahl formaler Zeitmodelle aus verschiedensten wissenschaftlichen Bereichen, insbesondere der Philosophie und der Künstlichen Intelligenz. Eine detaillierte Beschreibung würde den Rahmen dieser Ausarbeitung sprengen, doch wie Allen [1983] kritisiert, handelt es sich stets um Modelle mit dem Zeitpunkt als atomare Einheit. Warum dies ein Problem darstellt, wird im nächsten Abschnitt erläutert.

2.3. Zeitpunkte und Zeitintervalle

Allen [1983] argumentiert, ein Ereignis könne gar nicht sinnvoll einem Zeitpunkt zugeordnet werden. Ein Zeitpunkt ist ein Augenblick von null Zeiteinheiten Dauer, während ein Ereignis immer eine bestimmte Dauer größer null hat. Jedes Ereignis lässt sich somit immer noch in Teilabschnitte zerlegen, welche zeitlich getrennt sind. Somit ist es unmöglich, dieses Ereignis einem einzigen Zeitpunkt zuzuordnen. Ein weiteres Problem mit Zeitpunkten der Dauer null ist das Eintreten undefinierter Zustände. Betrachtet man das Ereignis „Licht wird angeschaltet“, so gibt es zwei Zustände: den Zustand „Licht ist aus“ vor dem Ereignis und den Zustand „Licht ist an“ nach dem Ereignis. Zeitlich modelliert gibt es also zwei Zeitintervalle, einen während dem das Licht aus ist, und einen, während dem das Licht an ist. Lässt ein Modell diskrete Zeitpunkte zu, stellt sich die Frage, welcher Zustand an dem Zeitpunkt gilt, an dem diese beiden Zeitintervalle aufeinandertreffen. Dies ist abhängig davon, ob Zeitintervalle als offene oder geschlossene Intervalle definiert sind. Sind die Intervalle offen, ist zu diesem Zeitpunkt das Licht sowohl an als auch aus, was einen logischen Widerspruch darstellt. Ebenso im Falle von geschlossenen Intervallen – hier ist das Licht zum kritischen Zeitpunkt weder an noch aus. Diskrete Zeitpunkte in einem Zeitmodell widersprechen also unserem intuitiven Zeitverständnis und bringen viele komplexe Probleme mit sich. Dies war der Anlass für Allen, statt des Zeitpunktes das Zeitintervall zur atomaren Einheit seiner Zeitlogik zu machen.

3. Die Allensche Zeitlogik

Dieses Kapitel beschreibt die grundlegenden Elemente und die Anwendung der Allenschen Zeitlogik.

3.1. Das Grundelement

Wie im vorherigen Kapitel dargelegt ist die atomare Einheit in der Allenschen Zeitlogik das Zeitintervall, nicht der Zeitpunkt. Im folgenden werden zwar Zeitpunkte für die Definition von Zeitintervallen und der Relationen zwischen diesen verwendet, dies dient jedoch nur der einfacheren Darstellung. In der eigentlichen Logik spielen Zeitpunkte keine Rolle und auch die Definitionen wären ohne sie realisierbar, etwa rein grafisch oder deskriptiv.

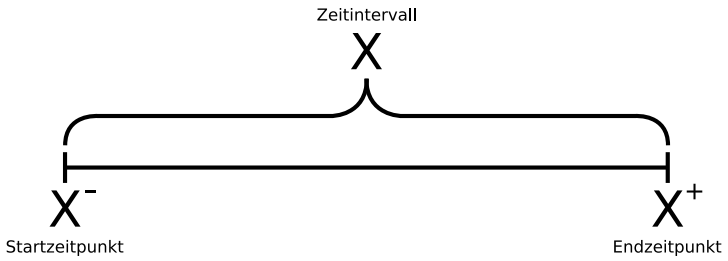

Abbildung 3.1.: Ein Zeitintervall X

Ein Zeitintervall X hat einen Startzeitpunkt und Endzeitpunkt. Der Startzeitpunkt wird meist mit X^-, der Endzeitpunkt mit X^+ bezeichnet.

Generell gilt $X^- < X^+$, also dass der Startzeitpunkt vor dem Endzeitpunkt liegt. Zeitintervalle mit $X^- = X^+$ (Zeitpunkte) existieren damit nicht.

3.2. Die Basisrelationen

Interessant wird es, wenn zwei oder mehr Intervalle existieren. Zwischen diesen bestehen zeitliche Beziehungen (Relationen). Die Menge aller Relationen, die zwischen jeweils zwei Intervallen bestehen können, wird von den Basisrelationen gebildet. In der Allenschen Zeitlogik gibt es sieben verschiedene Arten von Basisrelationen. Von diesen besitzen sechs

eine inverse Relation, so dass es insgesamt dreizehn Basisrelationen gibt. Diese werden in den folgenden Abschnitten beschrieben und sind in Tabelle 3.1 zusammengefasst.

Englisch	Deutsch	Symbol	Inverse
X before Y	X vor Y	$<$	$>$
X meets Y	X trifft Y	m	mi
X overlaps Y	X überlappt Y	o	oi
X during Y	X während Y	d	di
X starts Y	X startet Y	s	si
X finishes Y	X beendet Y	f	fi
X equals Y	X gleich Y	$=$	$=$

Tabelle 3.1.: Überblick über die dreizehn Basisrelationen

3.2.1. before

X before Y, abgekürzt $X < Y$, bedeutet, dass das Intervall X endet, bevor Intervall Y startet.

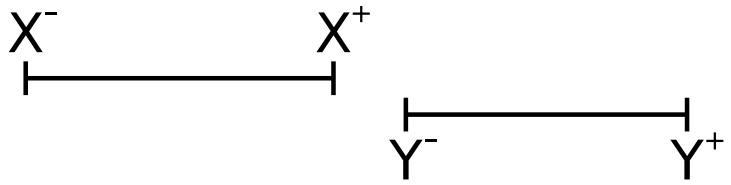

Abbildung 3.2.: X before Y

Die mathematische Bedingung lautet $X^- < X^+ < Y^- < Y^+$. Die inverse Relation („after“) wird mit $X > Y$ abgekürzt.

3.2.2. meets

X meets Y, abgekürzt X m Y, bedeutet, dass das Intervall X das Intervall Y trifft. Der Endzeitpunkt von X ist gleichzeitig der Startzeitpunkt von Y. Umgangssprachlich würde man hier ebenfalls von „X vor Y“ sprechen, in der Allenschen Zeitlogik wird jedoch streng zwischen „before“ und „meets“ unterschieden.

Die mathematische Bedingung lautet $X^- < X^+ = Y^- < Y^+$. Die inverse Relation wird mit X mi Y abgekürzt.

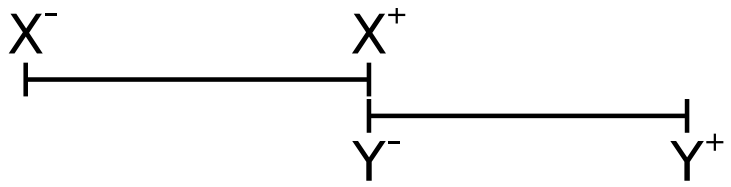

Abbildung 3.3.: X meets Y

3.2.3. overlaps

X overlaps Y, abgekürzt X o Y, bedeutet, dass das Intervall X vor dem Intervall Y startet und während diesem endet. Es gibt also einen Bereich, in dem sich beide Intervalle überlappen und für jedes Intervall je einen Bereich ohne Überlappung.

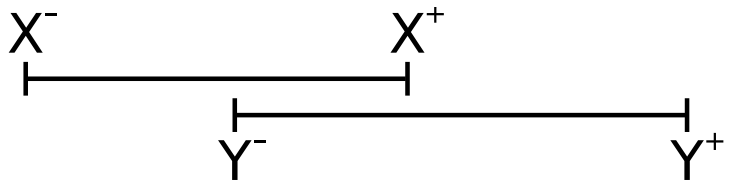

Abbildung 3.4.: X overlaps Y

Die mathematische Bedingung lautet $X^- < Y^- < X^+ < Y^+$. Die inverse Relation wird mit X oi Y abgekürzt.

3.2.4. starts

X starts Y, abgekürzt X s Y, bedeutet, dass das Intervall X gleichzeitig mit Intervall Y startet. Intervall X endet vor Intervall Y. Das Intervall X ist somit kürzer als Y.

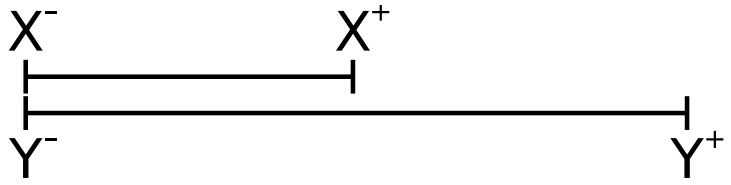

Abbildung 3.5.: X starts Y

Die mathematische Bedingung lautet $X^- = Y^- < X^+ < Y^+$. Die inverse Relation wird mit X si Y abgekürzt. Dabei ist Y kürzer als X.

3.2.5. finishes

X finishes Y, abgekürzt X f Y, bedeutet, dass das Intervall X gleichzeitig mit Intervall Y endet. Intervall X beginnt nach Intervall Y. Das Intervall X ist somit kürzer als Y.

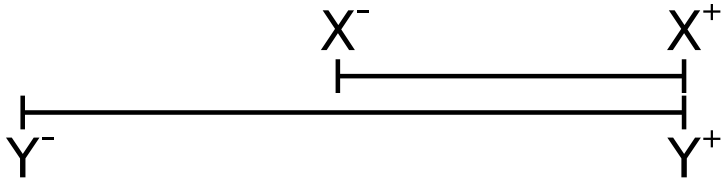

Abbildung 3.6.: X finishes Y

Die mathematische Bedingung lautet $Y^- < X^- < X^+ = Y^+$. Die inverse Relation wird mit X fi Y abgekürzt. Dabei ist Y kürzer als X.

3.2.6. during

X during Y, abgekürzt X d Y, bedeutet, dass das Intervall X während Intervall Y stattfindet. Intervall X startet nach dem Start von Intervall Y und endet vor dem Ende von Intervall Y. Das Intervall X ist somit kürzer als Y. Im allgemeinen Sprachgebrauch würden „starts“, „finishes“ und „during“ alle als „während“ bezeichnet werden. Auch Allen fasste ursprünglich alle diese Fälle in der Relation „during“ zusammen. Die Aufsplittung nahm er vor, da dies einen besseren Informationsgewinn durch Komposition (Siehe Abschnitt 3.4) ermöglicht. [Allen, 1983]

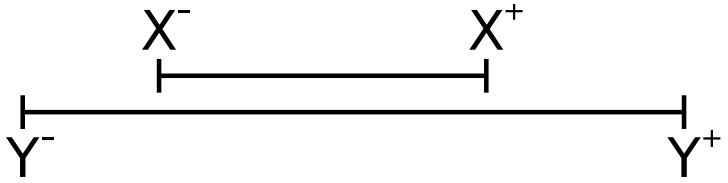

Abbildung 3.7.: X during Y

Die mathematische Bedingung lautet $Y^- < X^- < X^+ < Y^+$. Die inverse Relation wird mit X di Y abgekürzt. Dabei ist Y kürzer als X.

3.2.7. equals

X equals Y, abgekürzt $X = Y$, bedeutet, dass das Intervall X gleichzeitig mit Intervall Y startet und endet. Die Intervalle sind somit von gleicher Länge.

Die mathematische Bedingung lautet $X^- = Y^- < X^+ = Y^+$. Für die Identität gibt es keine inverse Relation, bzw. die inverse Relation ist wiederum die Identität.

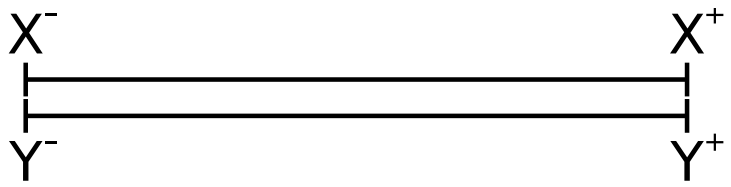

Abbildung 3.8.: X equals Y

3.3. Unvollständiges Wissen

Wenn eine einzige Basisrelation zwischen zwei Zeitintervallen besteht, bedeutet dies, dass wir alle Informationen über die Beziehung zwischen diesen beiden Intervallen haben. A d B besagt eindeutig, dass A nach B beginnt und vor B endet, alle anderen Basisrelationen sind ausgeschlossen. Wenn wir aber nur unvollständige Informationen haben, können wir keine so eindeutige Aussage treffen. Falls wir beispielsweise nur wissen, dass A weder ganz noch teilweise außerhalb von B liegt, kommen immer noch mehrere Basisrelationen in Frage. Diese Unsicherheit modellieren wir durch eine allgemeine Zeitrelation, welche durch eine Menge von Basisrelationen ausgedrückt wird. In unserem Beispiel wäre dies A {=,s,f,d} B, was bedeutet dass A entweder identisch mit B ist *oder* B startet *oder* B beendet *oder* während B ist. Wir wissen nicht, welche dieser Möglichkeiten tatsächlich wahr ist, nach unserem aktuellen Wissensstand sind alle möglich. Wir wissen jedoch, dass alle Basisrelationen, welche nicht in dieser Menge sind, definitiv *nicht* zutreffen. Wir haben also durch Einschränkungen (Constraints) unsere Unwissenheit über die Relation zwischen A und B verkleinert. Völlige Unwissenheit wäre eine Relation, welche aus der Menge aller Basisrelationen besteht. Diese spezielle Menge wird im Folgenden als „no info“ [Allen, 1983] bezeichnet.

3.4. Kompositionen

Unter der „Komposition von Relationen“ versteht man die Verkettung von zwei Relationen zwischen drei Zeitintervallen A, B, C nach folgendem Schema:

A r1 $B \wedge B$ r2 $C \Rightarrow A$ r3 C

Die Relationen r1 und r2 sind dabei bekannte Relationen, r3 ist die Relation, auf die wir schließen wollen. Gilt zum Beispiel $A > B$ und $B > C$, so muss auch $A > C$ gelten.

Nicht alle Kombinationen lassen auf eine eindeutige Relation r3 schließen. Wenn wir A m B und B d C gegeben haben, wissen wir sicher, dass der Endzeitpunkt von C nach B und damit auch nach A liegt. Wir wissen jedoch nicht genau, wann C beginnt, nur dass es vor B sein muss. C könnte also während A starten, vor A oder zum gleichen Zeitpunkt wie A. Entsprechend können wir also A {o,d,s} C schließen.

Tabelle 3.2 zeigt die möglichen Kompositionen der Basisrelationen an, mit Ausnahme

von equals. Aus A r1 B und $B = C$ folgt trivialerweise stets A r1 C, entsprechend führt $A = B$ und B r2 C zu A r2 C.

A r1 B \ B r2 C	<	>	d	di	o	oi	m	mi	s	si	f	fi
"before" <	<	no info	< o m d s	<	<	< o m d s	<	< o m d s	<	<	< o m d s	<
"after" >	no info	>	> oi mi d f	>	> oi mi d f	>	> oi mi d f	>	> oi mi d f	>	>	>
"during" d	<	>	d	no info	< o m d s	> oi mi d f	<	>	d	> oi mi d f	d	< o m d s
"contains" di	< o m di fi	> oi di mi si	o oi dur con =	di	o di fi	oi di si	o di fi	oi di si	di fi o	di	di si oi	di
"overlaps" o	<	> oi di mi si	o d s	< o m di fi	< o m	o oi dur con =	<	oi di si	o	di fi o	d s o	< o m
"over-lapped-by" oi	< o m di fi	>	oi d f	> oi mi di si	o oi dur con =	> oi mi	o di fi	>	oi d f	oi > mi	oi	oi di si
"meets" m	<	> oi mi di si	o d s	<	<	o d s	<	f fi =	m	m	d s o	<
"met-by" mi	< o m di fi	>	oi d f	>	oi d f	>	s si =	>	d f oi	>	mi	mi
"starts" s	<	>	d	< o m di fi	< o m	oi d f	<	mi	s	s si =	d	< m o
"started by" si	< o m di fi	>	oi d f	di	o di fi	oi	o di fi	mi	s si =	si	oi	di
"finishes" f	<	>	d	> oi mi di si	o d s	> oi mi	m	>	d	> oi mi	f	f fi =
"finished-by" fi	<	> oi mi di si	o d s	di	o	oi di si	m	si oi di	o	di	f fi =	fi

Tabelle 3.2.: Kompositionstabelle der Basisrelationen [Allen, 1983]

3.5. Grafische Darstellung

Betrachten wir nun die Relationen zwischen mehr als drei Intervallen, so bilden diese ein temporales Constraint-Netz.

Betrachten wir beispielhaft ein Netz mit vier Zeitintervallen A, B, C und D. Folgende drei Relationen seien bekannt:

- A < B
- B < C
- D f C

Dieses Netz lässt sich auf zwei verschiedene Arten darstellen:

Als Graph (Abbildung 3.9) Die Knoten des Graphs sind die Intervalle, die Linien zwischen den Knoten symbolisieren die Relationen. Die Pfeilrichtung zeigt an, in welche Richtung die Relation gilt.

Als Matrix (Tabelle 3.3) Für jedes Intervall gibt es sowohl eine Spalte als auch eine Zeile. In jedes Feld wird die Relation zwischen dem Zeilenintervall und dem Spaltenintervall eingetragen. Die Felder der Hauptdiagonalen enthalten also immer „=“, an der Hauptachse gespiegelt kann immer die inverse Relation eingetragen werden.

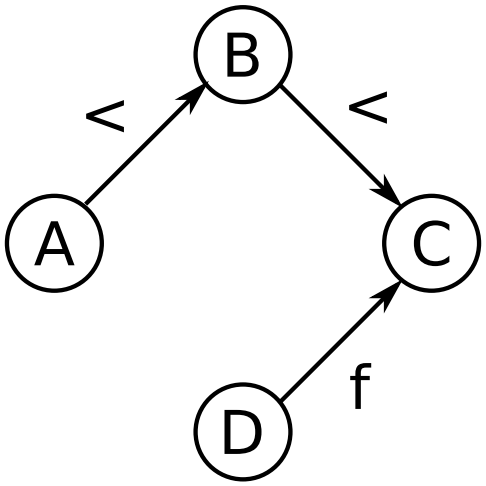

Abbildung 3.9.: Beispiel für ein einfaches Constraint-Netz

	A	B	C	D
A	=	<	no info	no info
B	>	=	<	no info
C	no info	>	=	fi
D	no info	no info	f	=

Tabelle 3.3.: Matrixdarstellung des Netzes aus Abbildung 3.9

4. Die Pfadkonsistenzmethode

In diesem Kapitel wird mit der Pfadkonsistenzmethode ein Verfahren zur Propagierung von Constraints in temporalen Netzen vorgestellt.

4.1. Ziel

Gegeben ist ein Constraint-Netz mit einer Anzahl von Intervallen und Relationen zwischen diesen. Wir interessieren uns nun für zwei Fragen:

1. Welche impliziten Relationen können wir aus den gegebenen Informationen schließen?
2. Ist das modellierte Zeitgefüge konsistent, also frei von Widersprüchen?

Das Ziel der Pfadkonsistenzmethode ist, bei einem gegebenen temporalen Constraint-Netz implizites Wissen über Relationen zwischen Zeitintervallen zu vervollständigen. Des Weiteren sollen mögliche Inkonsistenzen gefunden werden, um zeitliche Widersprüche zu identifizieren. Ein zeitlicher Widerspruch ist gegeben, wenn nicht alle gegebenen Constraints erfüllt werden können.

4.2. Das Verfahren

Die Pfadkonsistenzmethode durchläuft die Matrixdarstellung eines gegebenen Constraint-Netzes und ermittelt für jedes Feld mögliche Relationen aus Kompositionen anderer Felder der Matrix. Aus der ermittelten Menge der Relationen und der schon im Feld vorhandenen Menge wird die Schnittmenge gebildet und diese wird als neuer Wert in das Feld eingetragen. Auf diese Art werden inkonsistente Relationen aus der Tabelle entfernt – enthält ein Feld keine einzige Relation mehr, ist keine Beziehung zwischen diesen Intervallen möglich, was bedeutet, dass das gesamte Netz nicht pfadkonsistent ist. Meist ist ein einziger Durchlauf der Matrix für ein vollständiges Ergebnis nicht ausreichend, deshalb wird das Verfahren so lange wiederholt, bis sich keine Werte mehr ändern. Das Verfahren terminiert spätestens nach n Durchläufen. Hierbei ist n die Anzahl der Intervalle. Eine effiziente Implementierung hat eine Laufzeit von $O(n^3)$. Um das Verfahren zu veranschaulichen betrachten wir nun zwei Beispiele.

4.3. Beispiel 1

Als erstes betrachten wir das in Abbildung 3.9 dargestellte Constraint-Netz. Die Matrixform dieses Netzes vor der Durchführung der Pfadkonsistenzmethode ist in Tabelle 3.3 dargestellt.

Nun werden die Felder der Matrix durchlaufen. Die erste Änderung findet im Feld (A, C) statt. Bisher steht dort der Wert „no info“, also sind noch alle dreizehn Basisrelationen möglich. Nun werden alle möglichen Kompositionskandidaten für die Intervalle A und C untersucht. Fündig werden wir bei den Relationen zwischen A und B sowie zwischen B und C. Mit Hilfe der Kompositionstabelle (Tabelle 3.2) lässt sich von $A < B$ und $B < C$ auf $A < C$ schließen. Nun bilden wir aus der alten und der berechneten Menge die Schnittmenge:

$\{<, >, m, mi, o, oi, s, si, d, di, f, fi, =\} \cup \{<\} = \{<\}$

Das Ergebnis wird als neue Relation für A, B eingetragen. Die inverse Menge wird als neue Relationsmenge für B, A eingetragen. Das Ergebnis dieses ersten Schritts ist in Tabelle 4.1 dargestellt.

	A	B	C	D
A	=	<	**<**	no info
B	>	=	<	no info
C	**>**	>	=	fi
D	no info	no info	f	=

Tabelle 4.1.: Matrixdarstellung nach dem ersten Schritt der Pfadkonsistenzmethode

Nun läuft das Verfahren alle weiteren Felder ab und ergänzt die Felder um aus Kompositionen gewonnenes Wissen. Das Endergebnis ist in Tabelle 4.2 dargestellt.

	A	B	C	D
A	=	<	**<**	**<**
B	>	=	<	**<**
C	**>**	>	=	fi
D	**<**	**<**	f	=

Tabelle 4.2.: Matrixdarstellung nach Durchführung der Pfadkonsistenzmethode

In diesem Fall ist unser Netz nicht nur konsistent – es gibt keine leeren Felder in der Matrix – sondern auch eindeutig zu lösen. In jedem Feld steht eine Basisrelation, wir wissen also ganz genau, wie jedes Intervall mit allen anderen Intervallen zusammenhängt.

4.4. Beispiel 2

Nun betrachten wir ein etwas komplexeres Problem: das in Abbildung 4.1 dargestellte Constraint-Netz.

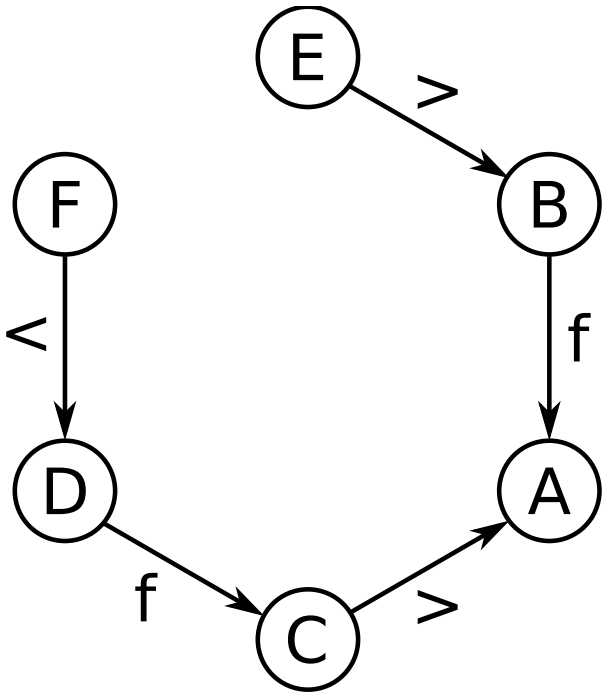

Abbildung 4.1.: Ein komplexeres Constraint-Netz [Heinsohn u. Socher-Ambrosius, 1999]

Die Matrix für dieses Beispiel ist etwas größer und es existiert noch viel unvollständiges Wissen, wie aus Tabelle 4.3 ersichtlich ist.

	A	B	C	D	E	F
A	=	fi	<	no info	no info	no info
B	f	=	no info	no info	>	no info
C	>	no info	=	fi	no info	no info
D	no info	no info	f	=	no info	>
E	no info	<	no info	no info	=	no info
F	no info	no info	no info	<	no info	=

Tabelle 4.3.: Beispiel 2 vor Durchführung der Pfadkonsistenzmethode

Bei dieser Matrix sind mehrere Durchgänge notwendig, bis keine Veränderungen mehr auftreten. Tabelle 4.4 zeigt das Endergebnis. Wie zu sehen ist, ist das Netz pfadkonsistent – kein Feld ist leer. Wir haben jedoch in einigen Feldern unvollständige Informationen – mehrere mögliche Relationen – oder sogar „no info“. Dies bedeutet, dass aus den

gegebenen Relationen keine vollständige Konstruktion aller zeitlichen Zusammenhänge möglich ist.

	A	B	C	D	E	F
A	=	fi	<	<	**>,di,oi,mi,si**	no info
B	f	=	<	<	>	no info
C	>	>	=	fi	>	**>,di,oi,mi,si**
D	>	>	f	=	>	>
E	**<,d,o,m,s**	<	<	<	=	no info
F	no info	no info	**<,d,o,m,s**	<	no info	=

Tabelle 4.4.: Beispiel 2 nach Durchführung der Pfadkonsistenzmethode

4.5. Unvollständigkeit

Die Pfadkonsistenzmethode ist nicht vollständig, es gibt also Netze, die von ihr als pfadkonsistent erkannt werden, aber trotzdem unerfüllbar sind. Dieses Problem ist laut Allen u. Hayes [1990] in der Praxis nicht relevant, da es nur in extrem wenigen Fällen auftritt. Man kann ein solches Szenario konstruieren, in der Praxis kommen solche Bedingungen in der Regel jedoch nicht vor.

Wir betrachen dazu nun ein Beispiel von Heinsohn u. Socher-Ambrosius [1999], ein Constraint-Netz mit vier Intervallen A, B, C und D. Folgende Relationen sind gegeben:

1. A beendet C oder C beendet A
2. A ist während B oder B ist während A
3. B ist während C oder C ist während B
4. D überlappt B
5. D startet mit A oder A startet mit D
6. D startet mit C oder C startet mit D

Nach der Pfadkonsistenzmethode ist dieses Netz pfadkonsistent. Bei genauerer Betrachtung stellt sich aber heraus, dass es nicht erfüllbar ist. Aus Regel 1 ergibt sich, dass A und C auf jeden Fall den gleichen Endzeitpunkt und verschiedene Startzeitpunkte haben müssen.

Betrachten wir nun die Beziehung zwischen A und D sowie zwischen D und C. Hierbei gibt es vier verschiedene Fälle:

Es gilt D s A und D s C Hierbei haben A und C gleiche Startzeitpunkte. Dies widerspricht allerdings Regel 1.

Es gilt D m A und D m C Dies würde ebenfalls bedeuten, dass A und C die gleichen Startzeitpunkte haben. Dies widerspricht erneut Regel 1.

Es gilt D s A und D m C In diesem Fall sind die Startzeitpunkte von A und D sowie die Endzeitpunkte von A und C jeweils identisch. Der Startzeitpunkt von C ist gleich mit dem Endzeitpunkt von D und liegt innerhalb von A. Aus den Regeln 2 und 3 ergibt sich, dass B entweder innerhalb von C ist oder A innerhalb von B. Beide Möglichkeiten widersprechen allerdings Regel 4.

Es gilt D m A und D s C Hier sind die Startzeitpunkte von D und C sowie die Endzeitpunkte von A und C jeweils identisch. Der Startzeitpunkt von A ist gleich mit dem Endzeitpunkt von D und liegt innerhalb von C. Auf Grund von Regel 4 liegt der Startzeitpunkt innerhalb von C, aber noch vor dem Startzeitpunkt von A. Nach Regel 2 muss der Endzeitpunkt von B nun nach A sein, dies widerspricht allerdings Regel 3, nach der der Endzeitpunkt von B noch vor dem Ende von A sein muss.

Alle vier möglichen Fälle führen also zu Widersprüchen. Das Netz ist somit inkonsistent, obwohl die Pfadkonsistenzmethode dieses Netz als konsistent ausgewiesen hat.

4.5.1. Lösungsansätze

Es gibt bisher keine zufriedenstellende Lösung des Unvollständigkeitsproblems, nur zwei teilweise Lösungsansätze:

Anzahl der Basisrelationen verringern Durch Verringerung der Basisrelationen auf eine Teilmenge der ursprünglichen Basisrelationen erhält man ein vollständiges Verfahren. Hierdurch wird allerdings die Anzahl der darstellbaren Szenarien stark eingeschränkt. Das beste vollständige Verfahren [Nebel, 1997] modelliert lediglich 10% des Umfangs der Allenschen Zeitlogik.

Kombination mit vollständiger Suche Durch die Kombination mit einer vollständigen Suche kann das Verfahren ebenfalls vervollständigt werden. Das Problem hierbei ist allerdings, dass die vollständige Suche NP-vollständig ist. [Heinsohn u. Socher-Ambrosius, 1999]

Diese Lösungsansätze sind damit nur in Ausnahmesituationen brauchbar, was jedoch wie oben erklärt in der Praxis meist nicht relevant ist.

5. Fazit

Die Allensche Zeitlogik liefert ein solides Grundgerüst zur Modellierung von zeitlichen Zusammenhängen. Die Wahl des Zeitintervalls anstelle des Zeitpunktes als grundlegendes Element löst viele Probleme früherer Modelle. Das Allensche Modell ist gut geeignet für die oft ungenauen und relativen Angaben zeitlicher Bezüge und kann auch unsichere und unvollständige Informationen abbilden. Mit der Pfadkonsistenzmethode steht eine einfache und automatisierbare Methode zum zeitlichen Schließen und zur Überprüfung auf Inkonsistenzen zur Verfügung, welche für alle praktisch relevanten Fälle verlässliche Ergebnisse liefert.

Alles in Allem stellt die Allensche Zeitlogik eine für den Bereich der computergestützten Wissensverarbeitung sinnvolle Weiterentwicklung dar. Sie ist auf Grund ihres an die natürliche Sprache angelehnten Relationenmodells besonders geeignet zur Extraktion von zeitlichem Wissen aus gesprochenen oder geschriebenen Sätzen. Dies ist nicht weiter verwunderlich, da Sprachverarbeitung eines der Fachgebiete von John F. Allen ist.

A. Implementierung in Prolog

Im Rahmen dieser Veranstaltung haben wir selbst die Pfadkonsistenzmethode in Prolog implementiert. Dieser Anhang enthält den gesamten Quelltext dieser Implementierung sowie einige Anwendungsbeispiele.

A.1. Quelltext

```
% Berechnet alle moeglichen Contraints mittels Pfadkonsistenzmethode

berechne(X) :-  ( relationen(X,E), write('Lade Ereignisse\n')),
( extend(E); write('Erweitere Ereignisse\n')),
( ( member(_,E), pfad(E)); write('Berechner Pfad\n')),
ausgabe(E), !.

% Gibt alle Constraints aus.

ausgabe(E) :- member(A,E),
member(B,E),
((relation(A,X,B),
write(A),
write(' '),
relationsname(X,N),
write(N),
write(' '),
write(B),
write('\n'));
(not(relation(A,X,B)),
write(A),
write(' '),
write('hat keine Beziehung zu'),
write(' '),
write(B),
write(' <- FEHLER'),
write('\n'))),
fail.

% Die eigentliche Pfadkonsistenzmethode

pfad(E) :- member(A,E),
member(B,E),
member(C,E),
A \= B,
```

```
B \= C,
A \= C,
notkomposition(A,B,C,X),
relation(A,X,C),
retractall(relation(A,X,C)),
fail.

% Erweiter leere Ereignisse um "no info" und invertiert Constraints.

extend(E) :- member(A,E),
member(B,E),
((A \= B,
not(relation(A,_,B)),
((
relation(B,X2,A),
inverse(X2,X2I),
assert(relation(A,X2I,B))
);(
not(relation(B,X2,A)),
relation(X1),
assert(relation(A,X1,B))
)));(
A == B,
assert(relation(A,i,B))
)),
fail.

% Ermittelt das inverse Element

inverse(v,vi).
inverse(vi,v).
inverse(m,mi).
inverse(mi,v).
inverse(o,oi).
inverse(oi,o).
inverse(s,si).
inverse(si,s).
inverse(d,di).
inverse(di,d).
inverse(f,fi).
inverse(fi,f).
inverse(i,i).

% Auflistung aller Relationen

relation(v).
relation(vi).
relation(m).
relation(mi).
relation(o).
relation(oi).
relation(s).
```

```
relation(si).
relation(d).
relation(di).
relation(f).
relation(fi).
relation(i).

% Deutsche Namen zu den Relationen

relationsname(v, 'ist vor').
relationsname(vi, 'ist nach').
relationsname(m, 'trifft').
relationsname(mi, 'wird getroffen von').
relationsname(o, 'ueberlappt').
relationsname(oi, 'wir ueberlappt von').
relationsname(s, 'startet').
relationsname(si, 'wird gestartet von').
relationsname(d, 'ist innerhalb von').
relationsname(di, 'enthaelt').
relationsname(f, 'beendet').
relationsname(fi, 'wird beendet von').
relationsname(i, 'ist gleichzeitig mit').

% Auflistung aller Relationen die zwischen
% A und C über B NICHT möglich sind.

notkomposition(A,B,C,R) :-relation(R),
not(kompositionenonce(A,B,C,R)).

% Auflistung aller Relationen die zwischen
% A und C über B möglich sind.

kompositionenonce(A,B,C,X) :- relation(X),
kompositionen(A,B,C,X).

% Auflistung aller Relationen die zwischen
% A und C über B möglich sind (kann Duplikate enthalten).

kompositionen(A,B,C,X) :- relation(A,R1,B),
relation(B,R2,C),
komposition(R1,R2,X),
!.

% Kompositionstabelle

komposition(X  ,i  ,X  ).
komposition(i  ,X  ,X  ) :- X \= i.

komposition(v ,v ,X ) :- member(X, [v]).
komposition(vi,v ,X ) :- relation(X).
komposition(d ,v ,X ) :- member(X, [v]).
komposition(di,v ,X ) :- member(X, [v,o,m,di,fi]).
```

```
komposition(o ,v ,X ) :- member(X, [v]).
komposition(oi,v ,X ) :- member(X, [v,o,m,di,fi]).
komposition(m ,v ,X ) :- member(X, [v]).
komposition(mi,v ,X ) :- member(X, [v,o,m,di,fi]).
komposition(s ,v ,X ) :- member(X, [v]).
komposition(si,v ,X ) :- member(X, [v,o,m,di,fi]).
komposition(f ,v ,X ) :- member(X, [v]).
komposition(fi,v ,X ) :- member(X, [v]).

komposition(v ,vi,X ) :- relation(X).
komposition(vi,vi,X ) :- member(X, [vi]).
komposition(d ,vi,X ) :- member(X, [vi]).
komposition(di,vi,X ) :- member(X, [vi,oi,mi,di,si]).
komposition(o ,vi,X ) :- member(X, [vi,oi,mi,di,si]).
komposition(oi,vi,X ) :- member(X, [vi]).
komposition(m ,vi,X ) :- member(X, [vi,oi,mi,di,si]).
komposition(mi,vi,X ) :- member(X, [vi]).
komposition(s ,vi,X ) :- member(X, [vi]).
komposition(si,vi,X ) :- member(X, [vi]).
komposition(f ,vi,X ) :- member(X, [vi]).
komposition(fi,vi,X ) :- member(X, [vi,oi,mi,di,si]).

komposition(v ,d ,X ) :- member(X, [v,o,m,d,s]).
komposition(vi,d ,X ) :- member(X, [vi,oi,mi,d,f]).
komposition(d ,d ,X ) :- member(X, [d]).
komposition(di,d ,X ) :- member(X, [o,oi,i,d,di,f,fi,s,si]).
komposition(o ,d ,X ) :- member(X, [o,d,s]).
komposition(oi,d ,X ) :- member(X, [oi,d,f]).
komposition(m ,d ,X ) :- member(X, [o,d,s]).
komposition(mi,d ,X ) :- member(X, [oi,d,f]).
komposition(s ,d ,X ) :- member(X, [d]).
komposition(si,d ,X ) :- member(X, [oi,d,f]).
komposition(f ,d ,X ) :- member(X, [d]).
komposition(fi,d ,X ) :- member(X, [o,d,s]).

komposition(v ,di,X ) :- member(X, [v]).
komposition(vi,di,X ) :- member(X, [vi]).
komposition(d ,di,X ) :- relation(X).
komposition(di,di,X ) :- member(X, [di]).
komposition(o ,di,X ) :- member(X, [v,o,m,di,fi]).
komposition(oi,di,X ) :- member(X, [vi,oi,mi,di,si]).
komposition(m ,di,X ) :- member(X, [v]).
komposition(mi,di,X ) :- member(X, [vi]).
komposition(s ,di,X ) :- member(X, [v,o,m,di,fi]).
komposition(si,di,X ) :- member(X, [di]).
komposition(f ,di,X ) :- member(X, [vi,oi,mi,di,si]).
komposition(fi,di,X ) :- member(X, [di]).

komposition(v ,o ,X ) :- member(X, [v]).
komposition(vi,o ,X ) :- member(X, [vi,oi,mi,d,f]).
komposition(d ,o ,X ) :- member(X, [v,o,m,d,s]).
komposition(di,o ,X ) :- member(X, [o,di,fi]).
```

```
komposition(o ,o ,X ) :- member(X, [v,o,m]).
komposition(oi,o ,X ) :- member(X, [o,oi,i,d,di,f,fi,s,si]).
komposition(m ,o ,X ) :- member(X, [v]).
komposition(mi,o ,X ) :- member(X, [oi,d,f]).
komposition(s ,o ,X ) :- member(X, [v,o,m]).
komposition(si,o ,X ) :- member(X, [o,di,fi]).
komposition(f ,o ,X ) :- member(X, [o,d,s]).
komposition(fi,o ,X ) :- member(X, [o]).

komposition(v ,oi,X ) :- member(X, [v,o,m,d,s]).
komposition(vi,oi,X ) :- member(X, [vi]).
komposition(d ,oi,X ) :- member(X, [vi,oi,mi,d,f]).
komposition(di,oi,X ) :- member(X, [oi,di,si]).
komposition(o ,oi,X ) :- member(X, [o,oi,i,d,di,f,fi,s,si]).
komposition(oi,oi,X ) :- member(X, [vi,oi,mi]).
komposition(m ,oi,X ) :- member(X, [o,d,s]).
komposition(mi,oi,X ) :- member(X, [vi]).
komposition(s ,oi,X ) :- member(X, [oi,f,d]).
komposition(si,oi,X ) :- member(X, [oi]).
komposition(f ,oi,X ) :- member(X, [vi,oi,mi]).
komposition(fi,oi,X ) :- member(X, [oi,di,si]).

komposition(v ,m ,X ) :- member(X, [v]).
komposition(vi,m ,X ) :- member(X, [vi,oi,mi,d,f]).
komposition(d ,m ,X ) :- member(X, [v]).
komposition(di,m ,X ) :- member(X, [o,di,fi]).
komposition(o ,m ,X ) :- member(X, [v]).
komposition(oi,m ,X ) :- member(X, [o,di,fi]).
komposition(m ,m ,X ) :- member(X, [v]).
komposition(mi,m ,X ) :- member(X, [s,si,i]).
komposition(s ,m ,X ) :- member(X, [v]).
komposition(si,m ,X ) :- member(X, [o,di,fi]).
komposition(f ,m ,X ) :- member(X, [m]).
komposition(fi,m ,X ) :- member(X, [m]).

komposition(v ,mi,X ) :- member(X, [v,o,m,d,s]).
komposition(vi,mi,X ) :- member(X, [vi]).
komposition(d ,mi,X ) :- member(X, [vi]).
komposition(di,mi,X ) :- member(X, [oi,di,si]).
komposition(o ,mi,X ) :- member(X, [oi,di,si]).
komposition(oi,mi,X ) :- member(X, [vi]).
komposition(m ,mi,X ) :- member(X, [f,fi,i]).
komposition(mi,mi,X ) :- member(X, [vi]).
komposition(s ,mi,X ) :- member(X, [mi]).
komposition(si,mi,X ) :- member(X, [mi]).
komposition(f ,mi,X ) :- member(X, [vi]).
komposition(fi,mi,X ) :- member(X, [oi,di,si]).

komposition(v ,s ,X ) :- member(X, [v]).
komposition(vi,s ,X ) :- member(X, [vi,oi,mi,d,f]).
komposition(d ,s ,X ) :- member(X, [d]).
komposition(di,s ,X ) :- member(X, [o,di,fi]).
```

```
komposition(o ,s ,X ) :- member(X, [o]).
komposition(oi,s ,X ) :- member(X, [oi,d,f]).
komposition(m ,s ,X ) :- member(X, [m]).
komposition(mi,s ,X ) :- member(X, [d,f,oi]).
komposition(s ,s ,X ) :- member(X, [s]).
komposition(si,s ,X ) :- member(X, [s,si,i]).
komposition(f ,s ,X ) :- member(X, [d]).
komposition(fi,s ,X ) :- member(X, [o]).

komposition(v ,si,X ) :- member(X, [v]).
komposition(vi,si,X ) :- member(X, [vi]).
komposition(d ,si,X ) :- member(X, [vi,oi,mi,d,f]).
komposition(di,si,X ) :- member(X, [di]).
komposition(o ,si,X ) :- member(X, [di,fi,o]).
komposition(oi,si,X ) :- member(X, [oi,vi,mi]).
komposition(m ,si,X ) :- member(X, [m]).
komposition(mi,si,X ) :- member(X, [v1]).
komposition(s ,si,X ) :- member(X, [s,si,i]).
komposition(si,si,X ) :- member(X, [si]).
komposition(f ,si,X ) :- member(X, [vi,oi,mi]).
komposition(fi,si,X ) :- member(X, [di]).

komposition(v ,f ,X ) :- member(X, [v, o, m, d, s]).
komposition(vi,f ,X ) :- member(X, [vi]).
komposition(d ,f ,X ) :- member(X, [d]).
komposition(di,f ,X ) :- member(X, [di,si,oi]).
komposition(o ,f ,X ) :- member(X, [d,s,o]).
komposition(oi,f ,X ) :- member(X, [oi]).
komposition(m ,f ,X ) :- member(X, [d,s,o]).
komposition(mi,f ,X ) :- member(X, [mi]).
komposition(s ,f ,X ) :- member(X, [d]).
komposition(si,f ,X ) :- member(X, [oi]).
komposition(f ,f ,X ) :- member(X, [f]).
komposition(fi,f ,X ) :- member(X, [f, fi, i]).

komposition(v ,fi,X ) :- member(X, [v]).
komposition(vi,fi,X ) :- member(X, [vi]).
komposition(d ,fi,X ) :- member(X, [v,o,m,d,s]).
komposition(di,fi,X ) :- member(X, [di]).
komposition(o ,fi,X ) :- member(X, [v,o,m]).
komposition(oi,fi,X ) :- member(X, [oi,di,si]).
komposition(m ,fi,X ) :- member(X, [v]).
komposition(mi,fi,X ) :- member(X, [mi]).
komposition(s ,fi,X ) :- member(X, [v,m,o]).
komposition(si,fi,X ) :- member(X, [di]).
komposition(f ,fi,X ) :- member(X, [f,fi,i]).
komposition(fi,fi,X ) :- member(X, [fi]).
```

A.2. Anwendungsbeispiele

A.2.1. Einstieg

Das Prolog-Programm ist so ausgelegt, dass man für jedes Constraint-Netz eine Regel anlegt, welche wiederum die Wissensbasis dynamisch um Fakten mit allen bekannten Relationen dieses Netzes erweitert. Dabei vergibt man pro Regel eine Ganzzahl als Index, welche dann als Parameter für das berechne-Prädikat verwendet wird.

Betrachten wir als erstes Beispiel nun das Constraint-Netz aus Abbildung 3.9. Die Regel, die dieses Netz modelliert, sieht so aus:

```
relationen(1,[a,b,c,d]) :- retractall(relation(_,_,_)),
 assert(relation(a,v,b)),
 assert(relation(b,v,c)),
 assert(relation(d,f,c)).
```

Der erste Parameter ist der Index, in diesem Fall 1. Der zweite Parameter ist eine Liste mit den beteiligten Intervallen - hier gibt es also vier Intervalle mit den Bezeichnungen A, B, C und D. Im Regelrumpf werden dann zunächst alle - eventuell von vorherigen Durchläufen noch vorhandenen - dynamischen Relationsfakten entfernt und dann die von diesem Netz bekannten Relationen der dynamischen Wissensbasis hinzugefügt. In diesem Beispiel die Relationen $A < B$, $B < C$ und D f C (Da „<“ und „>“ ungültige Bezeichnernamen sind, verwenden wir „v“ und „vi“). Nun können die unbekannten Relationen berechnet werden:

```
?- berechne(1).
Lade Ereignisse
Erweitere Ereignisse
Berechner Pfad
a ist gleichzeitig mit a
a ist vor b
a ist vor c
a ist vor d
b ist nach a
b ist gleichzeitig mit b
b ist vor c
b ist vor d
c ist nach a
c ist nach b
c ist gleichzeitig mit c
c wird beendet von d
d ist nach a
d ist nach b
d beendet c
d ist gleichzeitig mit d
```

Bei diesem Netz ist das Ergebnis konsistent und es gibt für jedes Intervallpaar eine eindeutige Relation. Die nächsten zwei Beispiele stellen Szenarien dar, in denen dies nicht der Fall ist.

A.2.2. Inkonsistenz

Wir definieren nun ein Constraint-Netz, welches offensichtlich widersprüchlich ist:

```
relationen(4,[a,b,c]) :- retractall(relation(_,_,_)),
 assert(relation(a,v,b)),
 assert(relation(b,v,c)),
 assert(relation(a,vi,c)).
```

Als Ergebnis erhalten wir:

```
?- berechne(4).
Lade Ereignisse
Erweitere Ereignisse
Berechner Pfad
a ist gleichzeitig mit a
a hat keine Beziehung zu b <- FEHLER
a hat keine Beziehung zu c <- FEHLER
b hat keine Beziehung zu a <- FEHLER
b ist gleichzeitig mit b
b hat keine Beziehung zu c <- FEHLER
c hat keine Beziehung zu a <- FEHLER
c hat keine Beziehung zu b <- FEHLER
c ist gleichzeitig mit c
```

Durch die widersprüchlichen Daten hat der Algorithmus alle Basisrelationen zwischen den verschiedenen Intervallen verworfen. Das Netz ist damit inkonsistent.

A.2.3. Unvollständiges Wissen

Als nächstes modellieren wir ein etwas komplexeres Netz:

```
relationen(5,[a,b,c,d]) :- retractall(relation(_,_,_)),
 assert(relation(a,d,b)),
 assert(relation(a,di,b)),
 assert(relation(a,f,c)),
 assert(relation(a,fi,c)),
 assert(relation(c,d,b)),
 assert(relation(c,di,b)),
 assert(relation(d,s,a)),
 assert(relation(d,m,a)),
 assert(relation(d,s,c)),
 assert(relation(d,m,c)),
 assert(relation(d,o,b)).
```

Die Ausgabe der Berechnung lautet wie folgt:

```
?- berechne(5).
Lade Ereignisse
Erweitere Ereignisse
Berechner Pfad
a ist gleichzeitig mit a
a ist innerhalb von b
a enthaelt b
a beendet c
a wird beendet von c
a wird gestartet von d
a wird getroffen von d
b enthaelt a
b ist innerhalb von a
b ist gleichzeitig mit b
b enthaelt c
b ist innerhalb von c
b wir ueberlappt von d
c wird beendet von a
c beendet a
c ist innerhalb von b
c enthaelt b
c ist gleichzeitig mit c
c wird gestartet von d
c wird getroffen von d
d startet a
d trifft a
d ueberlappt b
d startet c
d trifft c
d ist gleichzeitig mit d
```

Hier werden für einige Intervallpaare mehrere Zeilen mit verschiedenen Intervallpaaren ausgegeben. Dies bedeutet, dass für diese Intervalle mehrere Möglichkeiten in Frage kommen und wir auf Grund des gegebenen Wissens nicht mit Sicherheit darauf schließen können, welche eindeutig zutrifft.

Literaturverzeichnis

[Allen 1983] Allen, James F.: Maintaining knowledge about temporal intervals. In: *Communications of the ACM* 26 (1983), Nr. 11, S. 832–843

[Allen 1991] Allen, James F.: Time and Time Again: The Many Ways to Represent Time. In: *International Journal of Intelligent Systems* 6 (1991), Nr. 4, S. 341–355

[Allen u. Ferguson 2005] Allen, James F. ; Ferguson, George: Actions and Events in Interval Temporal Logic. In: *Journal of Logic and Computation* 4 (2005), Nr. 5, S. 531–579

[Allen u. Hayes 1990] Allen, James F. ; Hayes, Patrick J.: Moments and points in an interval-based temporal logic. In: *Comput. Intell.* 5 (1990), Nr. 4, S. 225–238. – ISSN 0824–7935

[Heinsohn u. Socher-Ambrosius 1999] Heinsohn, Jochen ; Socher-Ambrosius, Rolf: *Wissensverarbeitung: Eine Einführung*. Spektrum, Akademischer Verlag, 1999

[Nebel 1997] Nebel, Bernhard: Solving hard qualitative temporal reasoning problems. In: *Constraints* 1 (1997), Nr. 3, S. 175–190

[University of Rochester 2004] University of Rochester: *James F. Allen's Home Page*. Version: Januar 2004. `http://www.cs.rochester.edu/~james/`, Abruf: 2. Februar 2008