Informatik-Seminar SS2010
Methoden der KI

Funktionsweise und Anwendung eines JTMS

Frederik Ladewig
(inf 8325)

Inhaltsverzeichnis

0. Vorwort....3
1. Einführung Truth Maintenance Systeme....3
2. JTMS Einführung....3
2.1. Darstellung eines JTMS....4
2.2. Modell....5
2.2.1 Fundiertes Modell....5
2.2.2. Abgeschlossenheit....5
2.2.3 Zulässiges Modell....6
3. JTMS Algorithmus....6
3.1. Benötigte Funktionen....6
3.1.1. J(n)....6
3.1.2. SJ(n)....6
3.1.3. Supp(n)....6
3.1.4 Ant(n)....6
3.1.5. Cons(n)....6
3.1.6. ACons(n)....6
3.1.7. Supp*(n)....6
3.1.8. Ant*(n)....7
3.1.9. Acons*(n)....7
3.3. Der-Algorithmus....7
3.4. Beispiel....9
3.5. Dependency Directed Backtracking – DDB....13
3.5.1 Begriffsdefinition....14
3.5.2. Der Algorithmus....14
3.6. DDB Beispiel....15
4. Schlusswort....16

0. Vorwort

Dieser Seminarvortrag basiert größtenteils auf dem Kapitel Justification-based Truth Maintenance Systeme aus dem Buch Methoden wissensbasierter Systeme von Christoph Beierle und Gabriele Kern-Isberner. Ich habe versucht in diesem Vortrag weniger abstrakte Definitionen zu geben, sondern stattdessen die Sachverhalte möglichst mit Worten und Beispielen zu beschreiben, da das Ziel dieses Vortrags war einen ersten Eindruck von Justification-based Truth Maintenance Systemen zu bekommen. Exakte Definitionen einiger Begriffe können, wenn gewünscht, in diesem Buch nachgelesen werden.

1. Einführung Truth Maintenance Systeme

Die typischen Regelbasierten Expertensystemen arbeiten in der Regel nur mit monotoner Logik, das heißt dass Wissen, welches einmal in eine Wissensbasis eingefügt wurde ist auch gültig und daran ändert sich auch nichts. Nun kann natürlich das Problem auftreten, dass eine neue Regel im Widerspruch zu den bisher existierenden Regeln steht. Wenn das der Fall ist bleibt nur noch die Möglichkeit die neue Regel einfach nicht einzufügen, oder die ganze Wissensbasis zu überarbeiten um diesen Widerspruch zu entfernen, was in einem einfachen regelbasierten System nicht automatisierbar ist.

Beispiel 1:

Nehmen wir an wir haben ein kleines regelbasiertes Expertensystem, bestehend aus den folgenden Zuständen von Objekten:

$s =$ Schalter ist geschlossen
$\neg s =$ Schalter ist offen
$k =$ Kabel ist in Ordnung
$\neg k =$ Kabel ist defekt
$l =$ Lampe ist an
$\neg l =$ Lampe ist aus

und der Regel $s \rightarrow l$

Wenn nun als neue Regel $s \wedge \neg k \rightarrow \neg l$ in das System eingefügt werden soll kann es zu dem Widerspruch kommen, dass sowohl l, als auch $\neg l$ *gilt.*

An diesem Punkt kommen Truth Maintenance Systeme ins Spiel. Ihre Aufgabe ist es jederzeit einen die Konsistenz des gesamten Systems zu erhalten, indem gegebenenfalls bestehende Aussagen zurück genommen werden, wenn diese im Widerspruch mit neuen Informationen stehen. Dabei werden allerdings niemals alte Informationen gelöscht. Stattdessen gibt es eine Menge von aktuell gültigen Informationen und der restlichen Menge von Informationen, die aktuell nicht gültig sind.

2. JTMS Einführung

Im Folgenden soll die Funktionsweise von Justification based Truth Maintenance Systemen vorgestellt werden. Ein JTMS besteht aus einer Menge von Begründungen (justification) und einer Menge von Aussagen (TODO) . Formal dargestellt: $T = (N, J)$.

2.1. Darstellung eines JTMS

Ein truth maintenance network T besteht aus einer Menge von Knoten (nodes) N und einer Menge von Begründungen (justifications) J.

Knoten repräsentieren eine elementare Aussage wie zum Beispiel “Der Schalter ist geschlossen“. Dabei ist zu beachten, dass der Knoten nicht mit dieser Aussage identifiziert werden darf, weshalb auch zum Beispiel die Negation dieser Aussage, also “Der Schalter ist nicht geschlossen“ oder “Der Schalter ist offen“, nicht durch eine Negation des Knoten repräsentiert werden darf, sondern durch einen anderen, unabhängigen Knoten repräsentiert werden muss. Somit wäre es auch denkbar, dass in einem TMN zwei Knoten zugleich gültig sind, deren Aussagen jedoch widersprüchlich sind. Dieser Widerspruch ist für ein JTMS nicht zu erkennen und müsste vorher explizit modelliert werden. Dazu wird der Widerpruchsknoten $n_\perp$ verwendet.

Knoten können zwei verschiedene Status haben. Wenn es keinen Grund zu der Annahme gibt, dass ein Knoten gültig ist, so wird dieser als *out* bezeichnet. Gibt es hingegen eine gültige Begründung für einen Knoten, dann wird dieser Knoten selbst akzeptiert und als *in* bezeichnet.

In einem TMN werden Knoten graphisch durch Rechtecke dargestellt.

Begründungen sind n:1 Beziehungen zwischen Knoten. Eine Begründung (J) besteht immer aus zwei Mengen von Eingangsknoten und einem Ausgangsknoten. Die Mengen der Eingangsknoten werden *in*-Liste (I) bzw. *out*-Liste (O) genannt. Der Ausgangsknoten wird als Konsequenz (n) bezeichnet. Die Notation von Begründungen sieht wie folgt aus: $J = (I|O \rightarrow n)$.

Die Regel, dass das Licht brennt, wenn der Schalter geschlossen ist und es kein Grund zu der Annahme gibt, dass das Kabel defekt ist dann ist die Lampe an, wird durch die Begründung $J = (s|\neg k \rightarrow l)$ dargestellt.

Den Widerspruch, dass eine Lampe nicht gleichzeitig an und aus sein kann würde folgendermaßen definiert werden: $J = (l, \neg l|\emptyset \rightarrow n_\perp)$.

Eine Begründung ist genau dann gültig, wenn alle Knoten der *in*-Liste den Status *in* haben und alle Knoten der *out*-Liste den Status *out* haben. Ist die Begründung gültig dann wird der Konsequenz-Knoten akzeptiert und bekommt den Status *in*. Wenn die Begründung allerdings nicht gültig ist, so wird die Konsequenz von dieser Begründung nicht beeinflusst. Begründungen deren *in*- und *out*-Listen leer sind nennt man Prämissen. Diese sind natürlich immer gültig und damit wird auch jede Konsequenz einer Prämisse immer akzeptiert. Prämissen können in einem TMN dafür benutzt werden Fakten darzustellen.

In dem Beispiel aus der Einführung könnte mit der Begründung (Prämisse) J = (∅|∅ → s) den Fakt darstellen, dass der Schalter aktuell geschlossen ist.

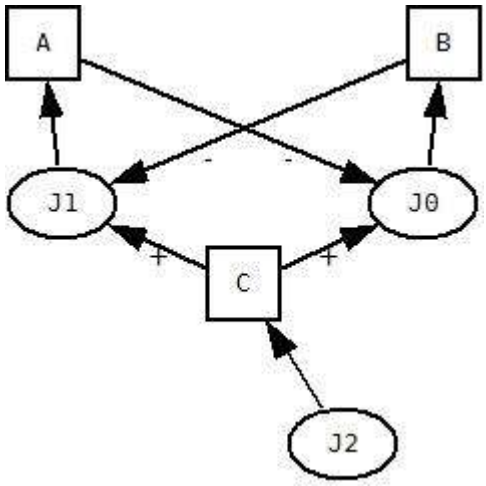
In einem TMN werden Begründungen als Kreise dargestellt und mit Pfeilen in Richtung Konsequenz verbunden. Die Pfeilen von den Knoten der *in*-Liste werden mit einem "+" markiert und die Pfeile der *out*-Liste mit einem "-".

Beispiel 2:

Gegeben sei folgendes TMN:

$T=(N,J)$
$N=\{A,B,C\}$
$J_0=(C|A\rightarrow B)$
$J_1=(C|B\rightarrow A)$
$J_2=(\emptyset|\emptyset\rightarrow C)$

In der graphischen Darstellung würde dies folgendermaßen aussehen:



2.2. Modell

Als Modell eines truth maintenance networks wird eine Menge von Knoten bezeichnet, die den Status *in* haben. Mögliche Modelle des TMN in Beispiel 2 wären: {}, {C}, {A}, {B}, {C, A}, {C, B}.

Um besser mit dem Begriff des Modells arbeiten zu können werden zusätzlich einige Erweiterungen definiert.

2.2.1 Fundiertes Modell

Eine Knotenmenge heißt fundiert bezüglich eines TMNs, wenn es eine vollständige Ordnung der Elemente in M gibt, so dass für jedes $n_j \in M$ gilt: Es gibt eine in M gültige Begründung $(I|O\rightarrow n_j)\in J$ von n derart, dass $I\subseteq\{n_1,\ldots,n_{j-1}\}$. Eine solche Begründung wird stützende Begründung genannt.

2.2.2. Abgeschlossenheit

Eine Knotenmenge heißt abgeschlossen bezüglich eines TMNs, wenn jede Konsequenz einer in dem Modell M gültigen Begründung in M enthalten ist. Damit ist also jeder *in*-Knoten in einem abgeschlossenen Modell enthalten aber kein *out*-Knoten. Das Modell M={B} wäre zum Beispiel kein abgeschlossenes Modell, da auch C *in* ist.

2.2.3 Zulässiges Modell

Als zulässiges Modell wird ein Modell bezeichnet, das fundiert und abgeschlossen ist. Im folgenden werden wir nur noch mit zulässigen Modellen arbeiten.

Das TMN in Beispiel 2 hätte also genau zwei zulässige Modelle: {C, A} und {C, B}. Alle anderen fallen raus, da diese nicht abgeschlossen sind.

3. JTMS Algorithmus

Die eigentliche Aufgabe eines JTMS-Verfahrens ist das Etablieren eines neuen zulässigen Systemzustandes, wenn neue Informationen dem System hinzugefügt wurden, durch die der bisherige Status unhaltbar geworden ist. Bei einer neu hinzugefügten Begründung versucht der JTMS Algorithmus also aus einem gegebenen zulässigen Modell ein neues zulässiges Modell zu erstellen, welches die neue Begründung berücksichtigt.
Neue Knoten können problemlos in ein bestehendes System eingefügt werden, da sie zunächst von keiner Begründung betroffen sind und damit in jedem Fall OUT sind. Spannend wird es erst beim Hinzufügen neuer Begründungen.

3.1. Benötigte Funktionen

Zum besseren Verständnis wird jeder Begriff auch an dem Beispiel 2 und das Modell M={C, A} erklärt.

3.1.1. J(n)

Menge der Begründungen, die n als Konsequenz haben. J(A) wäre also { J_1 }.

3.1.2. SJ(n)

Die Menge der stützenden Begründungen von n. SJ(C) = { J_2 }.

3.1.3. Supp(n)

Als Supp(n) wird die Menge aller Eingangsknoten einer stützenden Begründung von n bezeichnet. Supp(B) = {C, A}.

3.1.4 Ant(n)

Die stützenden Knoten Supp(n) eines Knotens mit dem Status *in* werden auch Antezedenzen genannt. Ant(A) = {C,B}, Ant(B) ist nicht definiert.

3.1.5. Cons(n)

Cons(n) umfasst alle Knoten, die n in ihrer Begründung erwähnen. Cons(C)={A, B}.

3.1.6. ACons(n)

Affected consequences sind alle Knoten, die n in ihrer stützenden Begründung erwähnen. Acons(C)={A,B}.

3.1.7. Supp*(n)

Die Vorfahren eines Knoten n ist der transitive Abschluss der stützenden Knoten von n. Supp(B) = {C, A}.

3.1.8. Ant*(n)

Die Fundamente sind der transitive Abschluss der Antezedenzen von n. Ant(A) = {C,B}.

3.1.9. Acons*(n)

Die Auswirkungen sind der transitive Abschluss der betroffenen Konsequenzen von n. Acons(C)={A,B}.

3.3. Der-Algorithmus

Eingabe: Ein TMN T=(N,J), ein zulässiges Modell M bzgl. Z und eine neue Begründung $J_0 = (I_0 | O_0 \rightarrow n_0)$

Ausgabe: Ein zulässiges Modell M´ bzgl. des TMN $T' = (N, J \cup \{J_0\})$

Anmerkung: Es gibt bestimmte TMN für die der Algoritmus nicht terminiert oder kein zulässiges Modell liefert. Dies passiert bei TMN, die sogenannte odd looops enthalten, in denen eine Aussage genau dann akzeptiert wird, wenn eben diese Aussage nicht akzeptiert wird. Solch eine odd loop entsteht zum Beispiel durch die Begründung $J = (\emptyset | n \rightarrow n)$ .Diese odd loops müssen vorher ausgeschlossen werden.

Der Algorithmus:

1. // Hinzufügen von J_0 , aktualisieren von Cons(n)

   $J := J + J_0$

   $J(n_0) := J(n_0) + J_0$

   for $n \in I_o \cup O_0$ **do**

   $Cons(n) := Cons(n) + n_0$

   if $Label(n_0) = in$ **then** HALT;

   if J_0 ungültig in M **then**

   $Supp(n_0) := Supp(n_0) + n'$; HALT;

   //n' ist ein *out*-Knoten aus I_0 oder ein *in*-Knoten aus O_0

2. // Überprüfe $ACons(n_0)$

   if $ACons(n_0) = \emptyset$ **then**

   $Label(n_0) := in;$

   $Supp(n_0) := I_o \cup O_0$;

   $SJ(n_0) := J_0$;

   $M := M + n_0$;

   HALT;

3. // Es gilt: Label(n_0) = *out*

// J_0 ist gültig in M

// *ACons(* n_0 *)* ist nicht leer

L := ACons(* n_0 *)* + n_0

Markiere jeden Knoten in L mit *unknown*

4. **for** $n \in L$ **do** (4A)

4A. **if** *Label(n) = unknown* **then**

if es gibt eine fundiert gültige Begründungen $J' = (I' | O' \rightarrow n) \in J(n)$ **then**

SJ(n) := J';

Supp(n) := $I' \cup O'$ *;*

Label(n) := in

Wende (4A) auf alle Konsequenzen von n mit Status *unknown* an.

else if es gibt nur fundiert ungültige Begründungen in *J(n)* **then**

Label(n) := out;

Bestimme *Supp(n)*

Wende (4A) auf alle Konsequenzen von n mit Status *unknown* an.

else

Status bleibt *unknown* bis Abschnitt 5.

5. **for** $n \in L$ **do** (5A)

5A. **if** *Label(n) = unknown* **then**

if es gibt eine nicht-fundiert gültige Begründungen $J' = (I' | O' \rightarrow n) \in J(n)$ **then**

if Acons(n) ist nicht leer **then**

for $n' \in ACons(n) + n$ **do**

Label(n') := unknown;

Wende (5A) auf n' an;

else

SJ(n) := J';

Supp(n) := $I' \cup O'$;

Label(n) := in

for $n' \in O'$ do

if *Label(n') = unknown* **then**

Label(n') := out;

for $n' \in Cons(n), Label(n') = unknown$ **do** (5A)

else // Alle Begründungen von n sind nicht-fundiert gültig

Label(n) := out;

for $J = (I' | O' \rightarrow n) \in J(n)$ **do**

Wähle $n' \in I'$ mit *Label(n') = unknown*

Label(n') := out;

Bestimme *Supp(n)*

for $n' \in Cons(n), Label(n') = unknown$ **do** (5A)

//Nach dem Durchlaufen des Algorithmus muss noch überprüft werden ob ein Widerspruch vorliegt.

$M' := \{ n \in N | Label(n) = in \}$

for $n \in N$ **do**

if n ist Widerspruchsknoten AND n ist *in* **then** DDB()

3.4. Beispiel

Als Beispiel wollen wir einen Ablauf in einer Bibliothek abbilden und zwar die Bestimmung ob ein bestimmtes Buch ausleihbar ist oder nicht.

Gegeben ist ein TMN T = (N, J) mit

$K = \{V, P, \neg P, N, H, G, F, A, \neg A, n_{\perp}\}$ und $J = \{J_1, J_2, J_3, J_4, J_5, J_6, J_7, J_8, J_9,\}$

Interpretation der Knoten:

V: „als verfügbar gekennzeichnet“

P: „am angegebenen Platz vorhanden“

¬P: „nicht am angegebenen Platz vorhanden“

N: „Nachschlagewerk“

H: „im Handapparat einer Vorlesung“

G: „gestohlen“

F: „falsch einsortiert“

A: „ausleihbar“

¬A : „nicht ausleihbar“

$n_\perp$: Widerspruchsknoten

Definition der Begründungen:

$J_1 = (\emptyset | \emptyset \rightarrow V)$

$J_2 = (V | F, G \rightarrow P)$

$J_3 = (F | \emptyset \rightarrow \neg P)$

$J_4 = (G | \emptyset \rightarrow \neg P)$

$J_5 = (P | N, H \rightarrow A)$

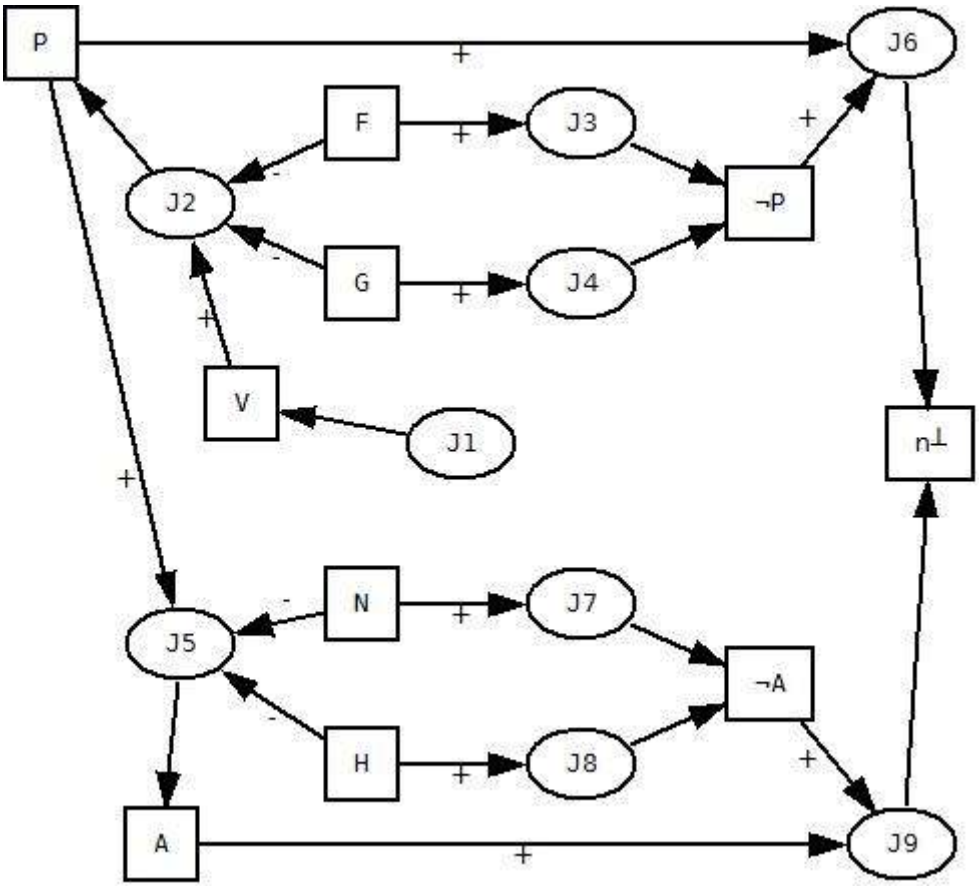
$J_6 = (P, \neg P | \emptyset \rightarrow n_\perp)$

$J_7 = (N | \emptyset \rightarrow \neg A)$

$J_8 = (H | \emptyset \rightarrow \neg A)$

$J_9 = (A, \neg A | \emptyset \rightarrow n_\perp)$

Grafische Darstellung



Mögliche Modelle

Jedes zulässige Modell enthält alle Knoten, die durch Prämissen begründet werden, also ist V wegen J_1 in jedem Modell enthalten. Wenn V *in* ist, dann ist auch J_2 gültig und auch P *in,* woraus wiederum J_5 als gültig erkannt werden kann und damit auch A *in* ist.

Also haben wir das Modell M = {V, P, A} gefunden. Es ist abgeschlossen, weil alle anderen Knoten *out* sind und es ist fundiert durch die Fundierungsordnung V<P<A.

Somit ist M ein zulässiges Modell und es ist auch das einzige.

Aktuelle Belegung für das Modell M

Knoten	Status	J(n)	SJ(n)	Supp(n)	Supp*(n)
V	*in*	J_1	J_1	-	-
P	*in*	J_2	J_2	V, F, G	V, F, G
¬P	*out*	J_3, J_4	*	F, G	F, G
F	*out*	-	*	-	-
G	*out*	-	*	-	-
H	*out*	-	*	-	-
N	*out*	-	*	-	-
A	*in*	J_5	J_5	P, N, H	P, N, H, F, G, V
¬A	*out*	J_7, J_8	*	N, H	N, H
$n_\perp$	*out*	J_6, J_9	*	¬P, ¬A	¬P, ¬A, F, G, N, H

Knoten	Ant(n)	Ant*(n)	Cons(n)	ACons(n)	ACons*(n)
V	-	-	P	P	P, A
P	V, F, G	V, F, G	A, $n_\perp$	A	A
¬P	*	*	$n_\perp$	$n_\perp$	$n_\perp$
F	*	*	P, ¬P	P, ¬P	P, ¬P, A $n_\perp$
G	*	*	P, ¬P	P, ¬P	P, ¬P, A $n_\perp$
H	*	*	A, ¬A	A, ¬A	A, ¬A, $n_\perp$
N	*	*	A, ¬A	A, ¬A	A, ¬A, $n_\perp$
A	P, N, H	P, N, H, V, F, G	$n_\perp$	-	-
¬A	*	*	$n_\perp$	$n_\perp$	$n_\perp$
$n_\perp$	*	*	-	-	-

(*: Wert nicht definiert; -: Menge ist leer)

Durchführung des Algorithmus

Wir starten den Algorithmus mit M={V, P, A} und wählen als neue Begründung $J_{10}=(\emptyset|\emptyset \rightarrow N)$. Somit ist für den Algorithmus $n_0=N$ und $J_0=J_{10}$. Damit legen wir also fest, dass das Buch, um das es grade geht, ab sofort ein Nachschlagewerk ist.

1. Begründung in das System einfügen

   N ist in dem aktuellen Modell *out*, aber die Begründung J_{10} ist gültig in M

2. ACons(N) = {A, ¬A} und damit nicht leer

3. L := {N, A, ¬A, $n_\perp$ }

   Alle Knoten in L werden nun mit *unknown* markiert

4. Wir beginnen mit n=N (Schleife#1)

4A. J_{10} ist fundiert gültig, also SJ(N)= J_{10} , Supp(N)= $\emptyset$ und Label(N)=*in*

(4A) auf {A, ¬A} anwenden, beginnen mit A (Schleife#2)

Da N nun den Status *in* hat ist J_5 nun ungültig

Es gibt also nur fundiert ungültige Begründungen für A

Label(A)=*out*, Supp(A)={P, N, H}, Cons(A)={ $n_\perp$ }

(4A) auf $n_\perp$ anwenden (Schleife#3)

$n_\perp$ hat mit J_6 und J_9 nur fundiert ungültige Begründungen, also

Label($n_\perp$)=*out,* Supp($n_\perp$), Cons($n_\perp$)= $\emptyset$

Es gibt keine Konsequenzen, also zurück zu Schleife#2

//Bisher wurde A und $n_\perp$ der Status *out* zugewiesen ,

//N den Status *in* und ¬A ist weiter *unknown*

Nächster Schleifendurchlauf: n=¬A

Es gibt jetzt mit J_7 eine fundiert gültige Begründung, also

Label(¬A)=*in*, Supp(¬A)=N, Cons(¬A)= $n_\perp$

Es gibt keine Konsequenzen mit status unknown mehr, also zu Schleife#1

Nun hat keiner der Knoten mehr den Status *unknown*, also wird weder 4A noch 5A in den restlichen Schleifendurchläufen etwas ändern.

6. Der Widerspruchsknoten ist *out*, also wird auch hier nichts mehr geändert.

Das neue Modell M' beinhaltet also die Knoten {V, P, N, ¬A}. Das Buch ist also verfügbar, an seinem Platz, es ist ein Nachschlagewerk und nicht ausleihbar.

3.5. Dependency Directed Backtracking – DDB

Wenn ein Widerspruchsknoten in ist muss, versuchen wir mit dem DDB-Algoritmus diesen Widerspruch aufzulösen.

3.5.1 Begriffsdefinition

MaxAnn($n_\perp$):

Die Menge der maximale Annahmen von $n_\perp$ sind eine Teilmenge der Fundamente von $n_\perp$. Eine Annahme n_A ist genau dann in *MaxAnn(* $n_\perp$ *)*, wenn es keine andere Annahme n_B in *Ant*(* $n_\perp$ *)* gibt derart, $n_A \in Ant*(n_B)$ ist.

$J_\perp$: Menge der stützenden Begründungen der Knoten in *MaxAnn(* $n_\perp$ *)*.

3.5.2. Der Algorithmus

1. Ist $MaxAnn(n_\perp) = \emptyset$ so liegt ein unlösbarer Widerspruch vor.

2. Wähle eine der maximalen Annahmen $n_A \in MaxAnn(n_\perp) = \emptyset$
   Wähle einen (*out*-) Knoten n' in der *out*-Liste der stützenden Begründungen von n_A .

3. Füge eine neue Begründung der Form $J = (I_\perp | O_\perp \rightarrow n')$ hinzu, wobei
   $I_\perp$ = Vereinigung der IN-Listen von $J_\perp$ und
   $O_\perp$ = (Vereinigung der OUT-Listen von $J_\perp$)-{n'}.

4. JTMS Algorithmus Schritte 1-5

5. Ist $n_\perp$ auch im neuen Modell IN so wende DDB auf das neue Modell an.

6. Ist der Status von $n_\perp$ schließlich OUT so ist das DDB-Verfahren
   Erfolgreich abgeschlossen worden.

3.6. DDB Beispiel

Nehmen wir wieder das TMN aus Beispiel 3 bevor wir den JTMS Algorithmus darauf angewendet haben. Nun wollen wir die Begründung $J = (A | \neg A \rightarrow n_{\perp})$ dem TMN hinzufügen.

Anmerkung: (Diese Begründung ist nicht sehr sinnvoll, wenn man die Aussagen hinter den Knoten beachtet, aber für dieses Beispiel ließ sich kein sinnvolle Möglichkeit finden um den DDB Algorithmus vorführen zu können.)

Es wird mit dem JTMS Algorithmus gestartet. Dieser endet in Abschnitt 2 ab, da der Widerspruchsknoten keine betroffenen Konsequenzen hat. Ergebnis ist das Modell $M'=\{V, P, A, n_{\perp}\}$. Am Ende wird allerdings entdeckt, dass der Widerspruchsknoten den Status *in* hat, weshalb der DDB-Algorithmus ausgeführt werden muss.

1. $Ant^*(n_{\perp}) = \{V, P, G, F, \neg P, A, \neg A, N, H\}$

   V, P, G, F, N, H fallen raus, wegen $Ant^*(A) = \{V, P, G, F, N, H\}$

   ¬P und ¬A fallen raus, da sie beide den Status *out* haben.

   Übrig bleibt A, also $MaxAnn(n_{\perp}) = \{A\}$

2. Wir wählen die einzig mögliche maximale Annahme A.

   Aus der out-Liste von A wählen wir H (Wählt man N verhält sich der Algorithmus genauso, wie bei H, nur dass die Rollen von N und H vertauscht sind).

3. Nun erstellen wir die neue gültige Begründung $J = (P | N \rightarrow H)$.

4. Der JTMS-Algorithmus wird jetzt mit der neuen Begründung ausgeführt.

5. Als Ergebnis liefert der JTMS-Algorithmus das Modell $M''=\{V, P, H, \neg A\}$

6. Somit ist $n_{\perp}$ *out* und der DDB-Algorithmus erfolgreich beendet.

4. Schlusswort

Das JTMS wurde in den Siebziger Jahren als erstes TMS entwickelt. Seit dem wurden verschiedene neue Truth Maintenance Systeme entwickelt, die für meisten realen Anwendungsgebieten besser geeignet sind als das JTMS, weshalb es schwer war Beispiele für die praktische Anwendung von Justification-based Truth Maintenance Systemen zu finden. Eine Anwendung war das Verringern der Anzahl von falschen Alarmen, bei der Überwachung von Netzen.

Weitere TMS sind zum Beispiel das ATMS, welches auch in der KI-Vorlesung von Herrn Dr. Iwanowski vorgestellt wird und das Logic-based Truth Maintenance System.

Quellen:

Methoden wissensbasierter Systeme
Christoph Beierle, Gabriele Kern-Isberner
2008

Anwendung eines JTMS zur Verbesserung der NetzSicherheit von 2008:
http://www.hicss.hawaii.edu/hicss_41/decisionbp/INCTE01.pdf

Präsentation eines Logic-based Truth Maintenance Systems:
http://www.cs.northwestern.edu/~forbus/c44/Lectures/ltms.pdf