

Truth Maintenance System

John Doyle

Vortrag: Stefan Zimmermann

Projekt: Robustes Verarbeiten von
Instruktionen

Certainty? In this world, nothing is certain
but death and taxes.

(Benjamin Franklin, 1706-1790)

Overview

- Introduction
- Theory
- Basic of TMS
- Justifications
- TMS process
- Dependency-direct backtracking
- Outlook and discussion

Introduction

Technical terms and shortcuts:

- TMS - Wahrheitserhaltungssystem
- belief - Überzeugung
- node - Knoten
- justification - Rechtfertigung
- argument - Argument
- support-list (SL) - Unterstützungsliste
- conditional proof (CP) - bedingt bewiesen

Introduction

Truth maintenance is an area of AI concerned with revising sets of beliefs and maintaining the truth in the system when new information contradicts existing information. Truth maintenance systems (TMSs) work with inference engines that act as problem solvers within large search spaces. The inference engine explores alternatives, makes choices, and examines the consequences of the choices. If a contradiction is detected during this process, the TMS eliminates it by revising the knowledge base. Together, the TMS and inference engine can solve problems where algorithmic solutions don't exist, and thus offer an efficient way to deal with search spaces that are large due to combinatorial explosions of alternatives....

IEEE Educational Activities Department Piscataway, NJ, USA

Robustes Verarbeiten von
Instruktionen

5

Theory

- **Kontraposition:**
 - Aus $A \rightarrow B$ folgert man: $\neg B \rightarrow \neg A$
- **Transitivität:**
 - Aus $A \rightarrow B$ und $B \rightarrow C$ folgert man: $A \rightarrow C$
- **Monotonie:**
 - Aus $A \rightarrow B$ folgert man: $A \cap B \rightarrow C$

Robustes Verarbeiten von
Instruktionen

7

Introduction

- Das TMS ist ein problem solver Subsystem, das Funktionen bereit stellt um...
 - Annahmen zu machen
 - Abhängigkeiten zwischen Aussagen zu verwalten
 - Änderungen in seinen beliefs zu vollziehen, wenn es zu Kontradiktionen kommt
 - Konsistenz (soweit möglich) zu sichern

Robustes Verarbeiten von
Instruktionen

6

Theory

- Kontraposition und Transitivität beruhen auf der Monotonie Eigenschaft.
- **Justifications** sind wahr und das ändert sich auch nicht. Somit kann durch das Schlussfolgern nur die Wissensbasis erweitert, aber nicht revidiert werden.



Alle revidierbaren Inferenzoperation müssen nichtmonoton sein.

Robustes Verarbeiten von
Instruktionen

8

Theory

Definition der Zustände „in“ und „out“:

- p hat mindestens einen akzeptablen Grund und ist deswegen ein Mitglied der derzeitigen Menge an justifications („in“)
- P hat derzeit keinen akzeptablen Grund (entweder keine Gründe, oder nur unakzeptable) und ist deswegen kein Mitglied der derzeitigen Menge der justifications („out“).
- Zustände sind aber nicht symmetrisch.

Justifications

Beispiel:

Linie	Aussage	justification	Abhängigkeit
1.	$A \rightarrow B$	Prämisse	{1}
2.	$B \rightarrow C$	Prämisse	{2}
3.	A	Hypothese	{3}
4.	B	MP 1,3	{1,3}
5.	C	MP 2,4	{1,2,3}
6.	$A \rightarrow C$	Entladung 3,5	{1,2}

Basis of TMS

Fundamentale Aktionen des TMS:

- Es können neue nodes erschaffen werden. Die können mit Beschreibungen versehen werden.
- Neue justifications zu nodes hinzufügen oder zurücknehmen. Dieses sind meist Argumentationsschritte, die oft für die Anwendung von Regeln oder Prozeduren des Problem-Lösungsprogramms stehen.
- Die TMS kann nodes als einen Widerspruch markieren um Inkonsistenzen einer Menge von justifications darzustellen, die in die Argumente für den node greifen

Justifications

Beispiel:

N-1	A → B	(SL () ())	Prämisse
N-2	B → C	(SL () ())	Prämisse
N-3	A	(SL () ())	Prämisse
N-4	B	(SL (N-1 N-3) ())	MP
N-5	C	(SL (N-2 N-4) ())	MP
N-6	A → C	(CP N-5 (N-3) ())	Entladung

Justifications

- **Justifications** haben 2 Teilaufgaben:
 - nach außen gerichtete wichtig für den **problem solver**
 - nach innen gerichtete für das TMS
- Das TMS betrachtet oder benutzt dabei nie die äußerliche Darstellungsform, zeichnet sie lediglich für den **problem solver** auf.

Justifications

- Natürliche Argumente nutzen eine Fülle von Darstellungstypen, somit muss das TMS diese erstmal in eine bestimmte gewöhnliche Form bringen.
- TMS gebraucht nur 2 (interne) Formen für **justifications**, die **support-list (SL)** und **conditional proof (CP)** heißen.

Support-list justifications (SL)

- Support-list justifications haben die Form:
(SL<inlist> <outlist>)
- Diese ist nur dann gültig, wenn alle Knoten in der inlist „in“ sind und alle Knoten und der outlist „out“ sind.
- Somit ist der node, den er rechtfertigt, immer „in“.

Support-list justifications (SL)

- **SL-justifications** mit nichtleerer **inlist** und leerer **outlist** werden als normale Schlussfolgerungen bezeichnet.
- Eine **justification** ohne **inlist** und **outlist** wird als Prämisse bezeichnet und ist immer gültig.

conditional proof (CP)

- Conditional proof-justifications haben die Form:
(CP<consequent><inhypotheses><outhypotheses>)
- Die Menge der Hypothesen wird hier aufgeteilt, weil jeder node von nodes, die „in“ sind *und* von nodes die „out“ sind, abgeleitet werden können.

conditional proof (CP)

- Die CP-justification ist gültig, wenn der consequent node „in“ ist, indem entweder...
 - Jeder node der inhypotheses ist „in“
 - Jeder node der outhypotheses ist „out“
- Einfache Handhabung für die validity einer CP-justifications, wenn consequent und inhypothese „in“ sind und outhypotheses „out“.

conditional proof (CP)

- Liegen andere support statuses für diesen node vor, muss der support status des hypothesis nodes und seiner Auswirkungen auf eine hypothetische Situation gewechselt werden, damit die validity des conditional proof bewertet werden kann.
- Dieses benötigt TM Bearbeitung, eventuell Gültigkeitsprüfung von anderen CP-justifications, sodass alles sehr komplex werden kann.

conditional proof (CP)

- Da hier kein bekannter Algorithmus gefunden wurde, wird CP genauso berechnet wie SL.
- Bei der Entstehung hat die neue SL-justification die gleichen dependencies und ihr kann einfach validity nachgewiesen werden.
- Jedesmal, wenn eine CP justification valid ist, berechnet es eine äquivalente SL justification und rechnet fortan mit dieser weiter und zieht dadurch „Parallelen“.

TMS process

- Bei dem Prozess werden nötige Änderungen in der derzeitigen Menge der beliefs gemacht, wenn der User neue justification sets zu einem node hinzufügt, oder entfernt.
- Ablauf:
 - Es muss nur wenig getan werden, wenn die justification invalid ist, oder der node schon „in“ ist.

TMS process

- Wenn die justification valid ist, aber der node invalid, muß der node und seine Auswirkungen geupdated werden.
- Dazu macht die TMs eine Liste , die alle nodes und ihr Auswirkungen enthalten. Um dies zu verdeutlichen werden alle diese markiert, damit man sieht, das hier nicht genug well-founded support erfolgt ist.

TMS process

- Dann werden alle justifications dieser Nodes betrachtet, ob einer von diesen vielleicht valid ist, nur aufgrund von unmarkierten nodes (diese müssten dann gut begründeten support haben).
- Wenn es solche nodes gibt, werden diese „in“ (oder „out“, wenn alle ihre justifications invalid sind, allein aufgrund von gut begründetem support).

TMS process

- Dann wird geschaut, ob den markierten Konsequenzen well-founded support gegeben werden kann.
- Manchmal kommt es vor, das schon nach Betrachtung aller markierten nodes, diesen allen well-founded support status gegeben werden konnte.
- Aufgrund von Zyklen können immernoch nodes markiert bleiben. Das TMS initialisiert dann ein constraint relaxations Prozess, der dann support status für die übrig gebliebenen nodes festlegt.

TMS process

- Ganz am Ende prüft das TMS auf Widersprüche und CP-justifications, führt dependency direct backtracking und CP-justification Verarbeitung durch, wenn nötig.
- Letzendlich werden dem Benutzerprogramm die Veränderungen in den support-statuses von den beteiligten Knoten mitgeteilt.

Dependency-direct backtracking

- Immer wenn das TMS einen Kontradiktion node „in“ macht, wird dependency direct backtracking aufgerufen.
- Dies dient dazu, mindestens eine Annahme zu finden und zu entfernen, um den Kontradiktions node wieder „out“ zu machen.

Dependency-direct backtracking

- Die einzelnen Schritte im Detail:
 - Durchlaufe die foundations des Kontradictions nodes C, um die Menge $S=\{A_1, \dots, A_n\}$ zu finden, sodass eine Annahme A enthalten ist, die nur in C's foundations ist und es keine weitere Annahme B gibt, die auch in der foundations von C ist sodass A in der foundation von B ist.
 - Wenn noch nie backtracking auf C angewandt wurde, das S als Menge der maximalen Annahmen ausgewählt hatte, kreierte einen node NG, sonst wähle den alten node.

Dependency-direct backtracking

- Hierbei präsentiert C die falsche Aussage und
- NG präsentiert

$$\text{St}(A_1) \cap \dots \cap \text{St}(A_n) \rightarrow \text{false}$$

Oder

$$\neg (\text{St}(A_1) \cap \dots \cap \text{St}(A_n))$$

Justify NG mit

$$(\text{CP } C \text{ } S())$$

Dependency-direct backtracking

- Jetzt wird ein A_i ausgewählt, der den „Täter“ darstellt. Dazu gibt es D_1, \dots, D_k , welche die out nodes der outlist von A_i 's supporting justification sind. Dann wird D_j aus der Menge ausgewählt und mit
 $(SL (NG A_1 \dots A_{i-1} A_{i+1} \dots A_n)(D_1 \dots D_{j-1} D_{j+1} \dots D_k))$ justified wird.
- Wenn die TMS weitere Argumente findet, sodass C weiterhin „in“ bleibt, wiederhole das backtracking.

Outlook and discussion



Outlook and discussion

- Finden sich Parallelen zu unserem GA?
- Wie kann man die Funktionen in den GA aus dem Projekt einbringen?