DAI-Labor
TU Berlin

Grundlagen der Künstlichen Intelligenz

Belief Revision

01.12.2005

Dr.-Ing. Stefan Fricke
stefan.fricke@dai-labor.de

A|O|T
Agententechnologien in betrieblichen Anwendungen und der Telekommunikation

Gliederung

Probleme klassischer Logik

- ⇨ Klassische Logik ist monoton: $X \vdash H \Rightarrow X \cup Y \vdash H$
- ⇨ Menschliches Schließen ist nicht immer monoton

  Z.B. aus Nichtevidenz schlussfolgern und gegebenenfalls falsche Annahmen zurücknehmen
  - → „Solange ich nicht Gegenteiliges weiß, gilt die Annahme, dass alle Erwachsenen lesen können."

Gliederung

Default Logik

- ⇨ Eine Default-Theorie ist ein Paar (W, D)
  - → W (World) enthält FOL-Formeln, die wahr sind
  - → D ist eine Menge von rücknehmbaren Defaults
- ⇨ D besteht aus Inferenzregeln der Form A: $B_1 \ldots B_n$ / C
  - → A ist die Vorbedingung
  - → B_i sind Konsistenzannahmen
  - → C ist die (revidierbare) Konsequenz des Defaults
- ⇨ Semantik: Wenn A ableitbar ist und für alle i ($1 \le i \le n$) $\neg B_i$ nicht abgeleitet werden kann, dann leite C ab.

Beispiel

W = { Vogel(Tweety), Vogel(Hansi), Pinguin(Tweety),
$\forall x$ Pinguin(x) $\Rightarrow \neg$Fliegt(x) }

D = { d_1 }, wobei d_1 = Vogel(x): Fliegt(x) / Fliegt(x)

- ⇨ Aus Vogel(Hansi) kann mit d_1 Fliegt(Hansi) abgeleitet werden.
- ⇨ Mit d_1 kann nicht Fliegt(Tweety) abgeleitet werden, da W $\vdash$ Pinguin(Tweety).

Konsistenzkriterien

- Gegeben ein Paar von Formelmengen (T_1, T_2):

  (T_1, T_2) triggert A:B/C, gdw. T1 ⊢ A und nicht T_2 ⊢ ¬B
- Falls nicht T_2 ⊢ ¬ B, dann ist B konsistent mit T_2
- Beispiel:

  ({A,B,C}, {X,Y,Z}) triggert C: X, ¬B / A

  triggert nicht C: ¬X / B

  triggert nicht C: X / ¬B

Konsistenzkriterien

Eine Menge von Formeln E ist eine Extension einer Default-Theorie (W,D) genau dann, wenn:

$$E = \bigcup_{i=0}^{\infty} E_i \quad \text{mit:}$$

- $E_0 = W$
- $E_{i+1} = E_i \cup \{ C \mid A:B_1, ..., B_n/C \in D$ und $A:B_1, ..., B_n/C$ getriggert von $(E_i, E) \}$

Konsistenzkriterien

- Problem: Extensionen sind nicht immer eindeutig
- Beispiel: D = { A: B / ¬C, A: C / ¬B } hat 2 Extensionen

  W ∪ { ¬C } und

  W ∪ { ¬B } ,

  → denn nur jeweils eines der beiden Defaults kann getriggert werden.
- Alle Extensionen einer Default Theorie sind wechselseitig inkonsistent.

Beispiel

- W = { V(Tweety) }, D = { V(x) : F(x) / F(x) }

  E = W ∪ { F(Tweety) }
- W = { V(T), P(T), ∀x P(x) ⇒ ¬F(x) }, D = { V(x) : F(x) / F(x) }

  E = W
- W = { V(T), P(T) }, D = { V(x): F(x) / F(x) , P(x): ¬F(x) / ¬F(x) }

  E = W ∪ { F(T) } oder E = W ∪ { ¬F(T) }

Entscheidungsprozeduren

- „Skeptisches“ Default Reasoning
  - → Entscheidungsprozedur liefert zu Formel F genau dann ⊤, wenn F in allen Extensionen enthalten ist.
- „Leichtgläubiges“ Default Reasoning
  - → Entscheidungsprozedur liefert zu Formel F genau dann ⊤, wenn F in mindestens einer Extension enthalten ist.

Gliederung

Truth Maintenance

- Wenn Wissen und Schlussfolgerungen nicht monoton sind, kommen Belief Revision-Mechanismen zur Anwendung.
- Beispiel: Wissensbasierter Agent
  - Wenn $P \in KB$ und TELL(KB, $\neg P$), dann RETRACT(KB, P)
  - Was geschieht aber mit den Formeln, die von P inferiert wurden?
- Truth Maintenance Systeme geben eine Antwort auf diese Frage

Einfacher Ansatz zur Belief Revision

- Nummerierung der Formeln P_i: TELL(KB, P_1), TELL(KB, Q_2) ...
- Indexierung aller von P_i inferierten Formeln in der KB mit i
  - Die KB wird zu einem Stack.
- Bei einem TELL(KB, $\neg P$) wird der Stack einschließlich P_i leergeräumt
  - und damit auch alle Inferenzen aus dem Tell(KB, P) entfernt.
- Anschließend alle Tell(KB, P_j) (j > i) erneut durchgeführt.

Interaktion mit einem TMS

- Aufgabe eines TMS: Beliefs speichern und effizient verwalten
  - Wechselnde Annahmen erzwingen Update aktueller Beliefs
  - Inkonsistenzen erkennen und behandeln
  - Default-Reasoning unterstützen
  - Erklärungen generieren

Beispiel

- Axel, Bert und Cindy sind eines Verbrechens verdächtig
  - Axel hat ein Alibi: Registrierung in einem Hotel in Aachen
  - Bert Alibi: Schwiegersohn bezeugt seinen Besuch in Berlin
  - Cindy Alibi: Behauptung, ein Konzert in Celle gehört zu haben
- Aus diesem Sachverhalt schließen wir:
  - Axel, Bert oder Cindy haben das Verbrechen begangen
  - Axel ist unschuldig, Bert ist unschuldig
- Später belegt Cindy ihr Alibi
  - Eine Fernsehkamera hat sie aufgenommen

...

Gliederung

Justification-based Truth Maintenance System

- Jeder Belief ist ein TMS-Knoten.
- Das Label eines Knoten drückt aus, ob die mit dem Knoten assoziierte Proposition aktuell geglaubt wird oder nicht.
  - IN = aktuell geglaubt
  - OUT = zurzeit nicht geglaubt
- Eine Formel $P \in KB$ drückt keinesfalls *Wissen* aus:

	P: IN	P: OUT
¬P: IN	Widerspruch	¬P wahr
¬P: OUT	P wahr	P unbekannt

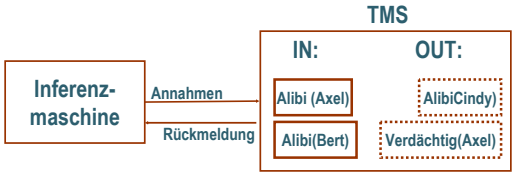
Knoten

TELL(Node(Verdächtig(Axel), OUT))).

TELL(Node(Alibi(Cindy, OUT))).

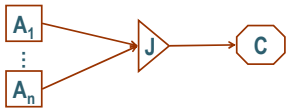
TELL(Node(Alibi(Axel, IN))).

TELL(Node(Alibi(Bert, IN))).



Justification

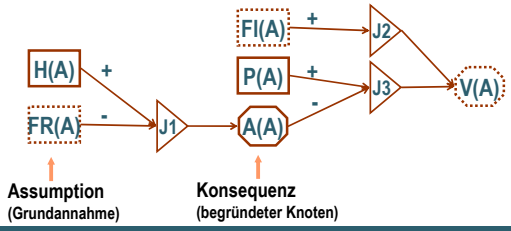
- **Justifications drücken Beziehungen zwischen Beliefs aus.**
- **Justifications sind Hornklauseln: $A_1 \wedge ... \wedge A_n \Rightarrow C$**
  - A_1, ... A_n sind Literale, C eine (positive) Proposition



  - A1, ... An sind die Vorgänger, C ist die Konsequenz.
- **Justify([A1, ..., An], C).**

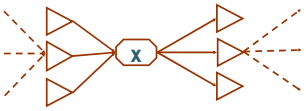
Abhängigkeitsnetze

- TELL(Justify([Profitiert(Axel), ¬Alibi(Axel)], Verdächtig(Axel))
  TELL(Justify([Hotelregistration(Axel), ¬FakeReg], Alibi(Axel))
  TELL(Justify([Flucht(Axel)], Verdächtig(Axel))



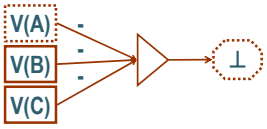
Jeder Knoten im TMS besitzt …

- **Justifications, die ihn als Konsequenz haben,**
- **Konsequenzen, d.h. Justifications, die ihn als Vorgänger haben,**
  - Knoten ohne Vorgänger werden Assumptions genannt
- **Support: Jede Justification, aufgrund derer er IN ist.**
  - Assumptions benötigen keinen Support



Kontradiktion

- **Negatives Wissen wird durch eine Justification eines speziellen Contradiction-Nodes (⊥) ausgedrückt**
  - Dieser Knoten darf nie IN werden
- **Beispiel: Mindestens eine der Personen Axel, Bert und Cindy ist verdächtig.**



Gliederung

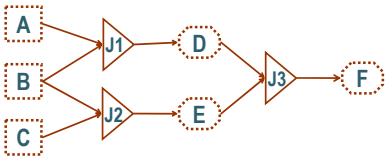
Arbeitsweise eines TMS

- **Ein TMS arbeitet inkrementell**
  - Knoten und Justifications hinzufügen
  - Assumptions IN oder OUT setzen
  - Anfragen hinsichtlich IN / OUT stellen
- **Bei einer Änderung ...**
  - ...wird das Label des betroffenen Knotens entsprechend gesetzt
  - ... und werden dessen Konsequenzen durch das Abhängigkeitsnetz propagiert ...

JTMS - Propagierung

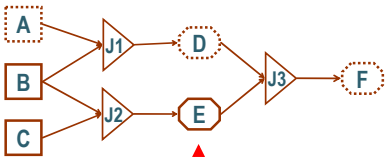
Aufbau des Abhängigkeitsnetzes:

- **Assumption([A,B,C], OUT).**
- **Justification([A,B],D), Justification([B,C],E), Justification([D,E],F)**



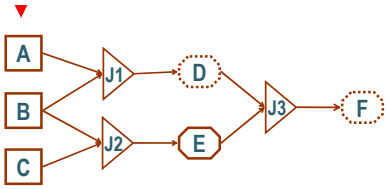
JTMS - Propagierung

- **Assume([B,C]) führt zur Propagierung von B=IN und C=IN:**
  - Justification J2 hat E zur Konsequenz
  - Die Propagierung endet, weil J3 nicht „aktiv“, da D=OUT



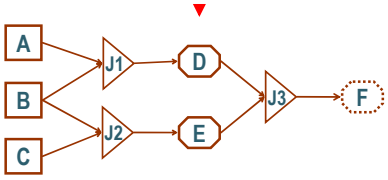
JTMS - Propagierung

- **Assume(A)**



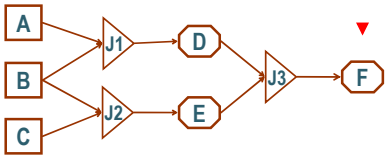
JTMS - Propagierung

- **Assume(A)**
- **D wird geglaubt über J1.**



JTMS - Propagierung

- **Assume(A)**
- **D wird geglaubt über J1.**
- **F wird geglaubt über J3, die Propagierung endet.**



Rücknahme von Informationen

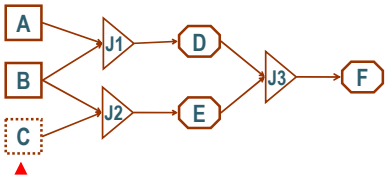
- Konsequenzen und Kontradiktionen sind nicht löschbar.
- Justifications werden nicht revidiert.
  - Dadurch können zirkuläre Begründungen entstehen (s.u.)
- Assumptions können zurückgenommen werden:

Assume(¬A)

1. setze A auf OUT.
2. Setze alle Knoten OUT, die durch A IN sind.
3. Suche alternativen Support für diese OUT-Knoten.

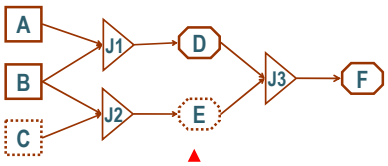
Beispiel OUT - Propagierung

- Assume(¬C).



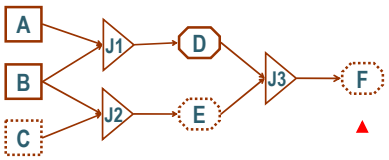
Beispiel OUT - Propagierung

- Assume(¬C).
- E verliert (einzigen) Support.



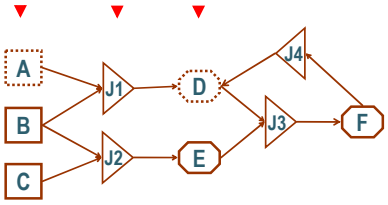
Beispiel OUT - Propagierung

- Assume(¬C).
- E verliert Support.
- F verliert Support.



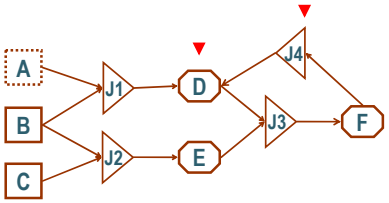
Re-Support

- Assume(¬A). D verliert Support über J1.



Re-Support: Problematik durch Zyklen

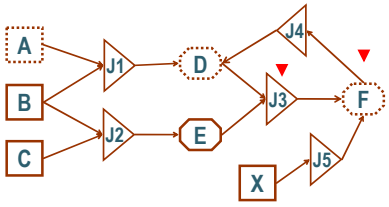
- Assume(¬A). D verliert Support über J1.
- Alternativer Support von D über J4
- Support von D über J4 ist unfundiert!



Re-Support, 2. Beispiel

⇨ Assume(¬A). D verliert Support über J1.

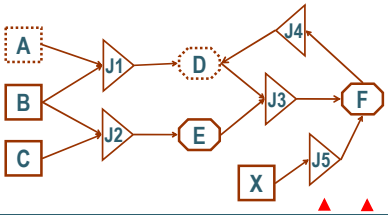
⇨ Weiterpropagierung statt Re-Support: F verliert Support über J3.



Re-Support

⇨ Assume(¬A). D verliert Support über J1.

⇨ F verliert Support über J3. Alternativer Support von F über J5.

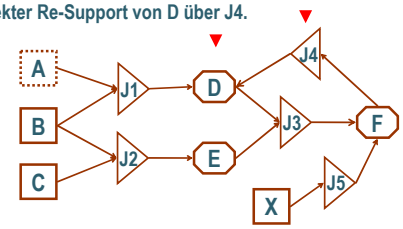


Re-Support

⇨ Assume(¬A). D verliert Support über J1.

⇨ F verliert Support über J3. Alternativer Support von F über J5.

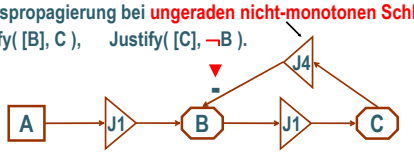
⇨ Korrekter Re-Support von D über J4.



JTMS-Probleme mit negativen Schlussfolgerungen

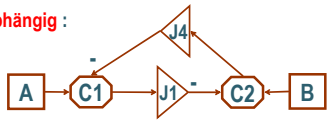
⇨ Endlospropagierung bei ungeraden nicht-monotonen Schleifen

Justify([B], C), Justify([C], ¬B).



⇨ Beliefs sind reihenfolgeabhängig :

Assume([A,B], IN) ≠
Assume([B,A], IN)



Gliederung

- ⇨ Einleitung
- ⇨ Default Logic
- ⇨ Truth Maintenance Systems (TMS)
- ⇨ Justification-based TMS
- ⇨ Assumption-based TMS
  - → Aufbau eines ATMS
  - → Propagierung im ATMS
- ⇨ Zusammenfassung

Vom JTMS zum Assumption-based TMS (ATMS)

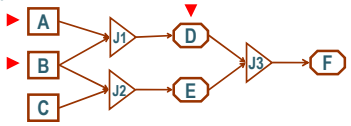
⇨ Ein JTMS berechnet die Auswirkungen der Änderung einer Assumption durch Propagation.

⇨ Ein ATMS berechnet die Auswirkungen der Änderung einer Assumption bereits beim Aufbau des Abhängigkeitsnetzes.

⇨ Schlüssel ist hierbei der Kontext eines Knoten.

Idee

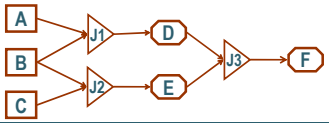
- Ob ein Knoten geglaubt wird, hängt von einer Menge von Assumptions ab:



  - D=IN hängt von A und B, nicht aber von C ab.
- Idee: Labeling der Knoten mit Mengen von Assumptions, sodass ohne Propagierung Änderungen in der Assumption-Menge direkt abgelesen werden können.

Begriffe

- Ein Environment (E) ist eine Menge von Assumptions:
  - Inkonsistent genau dann, wenn $\perp$ inferiert werden kann.
  - Konsistent genau dann, wenn nicht inkonsistent.
  - z.B: {}, {A}, {A, B}, {B, C}.
- Kontext: Konsistentes Environment $\cup$ alle inferierbaren Knoten
  - "alles, was aus einer Menge von Annahmen folgt"
  - z.B. {A, B} $\cup$ {D}
    oder {A,C} $\cup$ Ø
    oder {A,B,C,D,E,F}



Environment

- Gesucht ist das konsistente und minimale Environment eines Kontexts.
- Beispiel:

  E1 = {A}

  E2 = {A,B,C}

  E3 = {B,D}

  E1 und E3 sind minimal, E2 ist Spezialfall von E1.

Begriffe

- Das Label eines Knoten x ist die Menge konsistenter Environments $\{E_1, ..., E_m\}$ in denen x gilt.
- Spezialfälle:

  L = { } ist das leere Label
  - Es existiert kein konsistentes Environment für den Knoten x
  - D.h. entweder sind alle E inkonsistent oder x „hängt in der Luft"

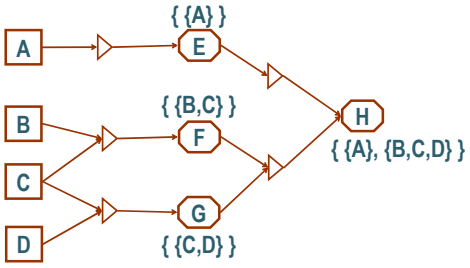
  L = { { } } ist ein leeres Environment
  - Der Knoten ist gültig in jedem konsistenten Environment

Eigenschaften von Labels

- Ein Label L ist vollständig, wenn für jedes konsistente Environment E mit $E \notin L$ gilt: $E,T \models X \Rightarrow \exists E_i \in L : E_i \subset E$.
  - "Zu jedem nicht minimalen E existiert ein minimales E_i in L"
  - T (Theorie) enthält alles Wissen, inklusive Regeln (Justifications)
- Ein Label L ist minimal, wenn $\forall E_i \in L \mid E \subset E_i \Rightarrow \neg(E,T \models X)$
- Ziel des ATMS: Berechnung der vollständigen, minimalen Label für alle Knoten.

Label: Beispiel

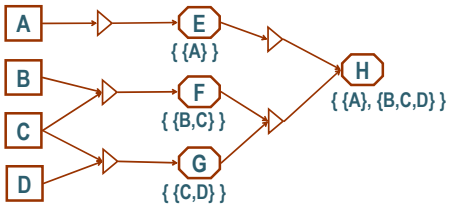
- Label = Menge der Environments, in denen ein Knoten geglaubt wird.



IN oder OUT ist vom Label ablesbar

- Jeder Knoten mit nicht leerem Label kann IN sein.

  z.B.: H ist IN im Kontext {B,C,D,X}, H ist OUT im Kontext {C,D }



Gliederung

Wie werden Label propagiert?

- Disjunktionen werden „aufsummiert“ (Kombination)

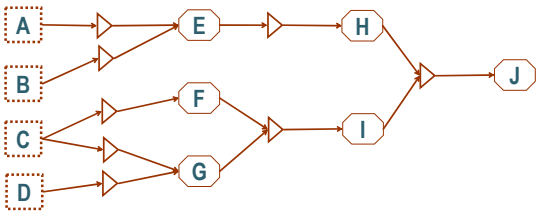
  { {A}, {B} } ∨ { {B}, {C} } ⇒ { {A}, {B}, {C} }

  { {A}, {B,C} } ∨ { {B}, {C} }
  ⇒ { {A}, {B,C}, {B}, {C} } ⇒ { {A}, {B}, {C} }

- Konjunktionen werden „multipliziert“ (Submengenkombination)

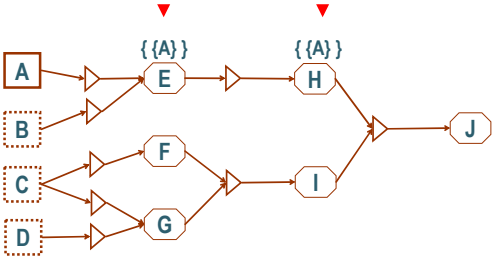
  { {A}, {B} } ∧ { {B}, {C} } ⇒
  { {A,B}, {A,C}, {B,B}, {B,C} } ⇒ { {B}, {A,C} }

Label-Propagierung



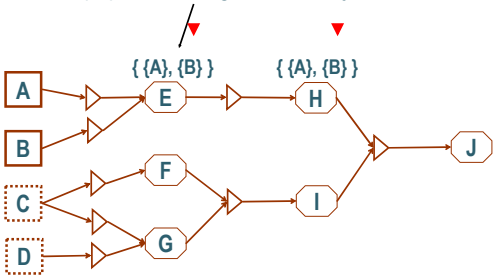
Label-Propagierung

- Enable(A)



Label-Propagierung

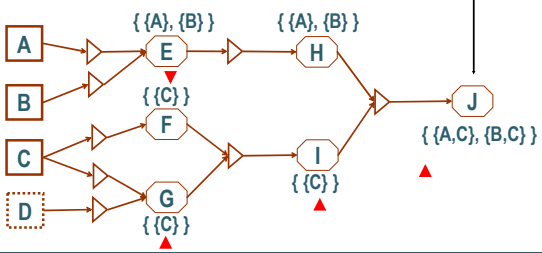
- Enable(B) Anwendungsfall Label-Disjunktion



Label-Propagierung

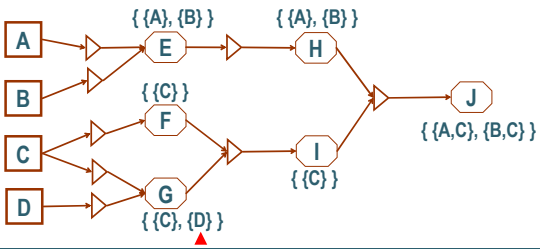
⇨ Enable(C)

Anwendungsfall Label-Konjunktion



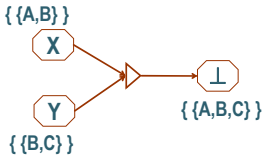
Label-Propagierung

⇨ Enable(D)



Nogoods

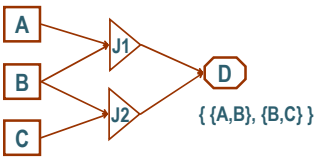
⇨ Inkonsistente Environments werden als Nogoods repräsentiert.



⇨ Die Semantik ergibt sich aus dem Kontext, aber…

Propagierung von Nogoods

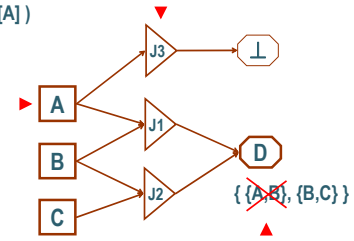
⇨ Beispiel: Ausgangssituation



Nogoods blockieren die Label-Propagierung

⇨ Nogood(E): E und alle E‘ mit E ⊂ E‘ werden aus sämtlichen Knotenlabeln entfernt.

Z.B.: Nogood([A])



Gliederung

JTMS und ATMS

- **JTMS: nur ein Kontext**
  - Knotenlabel ist entweder IN oder OUT
- **ATMS: multiple Kontexts**
  - Knotenlabel enthält valide Environments
- **JTMS: Suchprozeduren stellen einen konsistenten Zustand her**
  ATMS: Lösung wird aus dem Environment abgelesen
- **Die Propagierungsalgorithmen sind NP-vollständig**

JTMS und ATMS im Vergleich

	JTMS	ATMS
Art der Rechtfertigung	Direkte Begründungen	Basisannahmen
Behandlung von Zirkularitäten	Aufwändig	Einfach
Behandlung von Kontradiktionen	Einfach	Aufwändig
Effizienz	abhängig vom Vernetzungsgrad	abhängig von der Menge der Assumptions

DAI-Labor
TU Berlin

Grundlagen der Künstlichen Intelligenz

Belief Revision

01.12.2005

Dr.-Ing. Stefan Fricke
stefan.fricke@dai-labor.de

A|O|T
Agententechnologien in betrieblichen Anwendungen und der Telekommunikation

Referenzen

- JTMS: Doyle, J.: A truth maintenance system. Articial Intelligence, 12(3). S.231-272, 1979.
- ATMS: J. De Kleer: An assumption based truth maintenance system. Artificial Intelligence 28, S. 127-162, 1986.
- Stuart C. Shapiro: Belief Revision and Truth Maintenance Systems: An Overview and a Proposal. CSE Technical Report 98-10, 1998.

Anhang

Abgeschlossenheit und Konsistenz

Default Logik

- **Eine Formelmenge S ist bzgl. (W, D) abgeschlossen, gdw.**
  - $W \subseteq S$
  - Th(S) = S (S ist vollständige Folgerungsmenge aus W)
  - Falls $A : B_1, ..., B_n / C \in D$, $A \in S$, $\neg B_i \notin S$ $(1 \leq i \leq n)$, dann $C \in S$ (abgeschlossen bzgl. D)
- **S ist konsistent, wenn S keine Formeln enthält, die sich nicht aus W und den Konsequenzen anwendbarer Defaults in D herleiten lassen.**

Beispiel

- W = ∅ D = { true:b/a }
- S_1 = Th({a}) ist abgeschlossen und konsistent
- S_2 = Th({b}) ist abgeschlossen, aber nicht konsistent
- ¬b kann nicht aus W und den Konsequenzen aus D hergeleitet werden

ATMS-Propagierungsalgorithmus

- nach (De Kleer, 1986)
- Propagate:
- Update: Neue Environments zu jeder Konsequenz propagieren
- Weave: Gegeben neue Environments, kreiere ein neues Label für einen Knoten
- Nogood

ATMS: Propagate

Propagate([X<=X1,...,Xn],a,l)

L=Weave(a,l,{X1,...,Xn})

Update(L,X)

- Propagate is used to trigger the ATMS, typically with the inclusion of a justification:

  Propagate([X <= X1,...,Xn], Φ, { })

ATMS: Weave

Weave(a,l,X)

if X is empty, return l

if X = [H|T] (i.e. a list, H=head and T=tail)

if H=a, return Weave(Φ,l,T)

if H≠a, let l'=set of all environments formed by computing the union of an enviroment of l and an environment of a label of H

remove from l' duplicate environments, inconsistent environments and environments that are subsets of other environments

return Weave(a,l',T)

ATMS: Update

Update(L,X)

if X=false, call Nogood(E) for each E in L and halt (return { })

if X≠false,

Delete every E in L such that
there is an E' in the label of X and E' is a subset of E

Delete every E' in the label of X such that
there is an E in L and E is a subset of E'

Make the label of X as the union of the remaining environments

for every justification J in which X appears as an antecedent call Propagate(J,X,L)

ATMS: Nogood

Nogood(E)

tag E as inconsistent

delete E and any E' such that E is a subset of E'
from all labels of all nodes