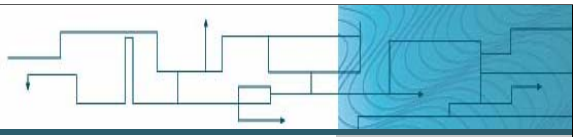
DAI-Labor
TU Berlin



Grundlagen der Künstlichen Intelligenz

Belief Revision

01.12.2005

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

AIOT
Agententechnologien in betrieblichen Anwendungen und der Telekommunikation

Gliederung

- ⇨ Einleitung
- ⇨ Default Logic
- ⇨ Truth Maintenance Systems (TMS)
- ⇨ Justification-based TMS
- ⇨ Assumption-based TMS
- ⇨ Zusammenfassung

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."

Theorie/Hypothese aufstellen und durch Beobachtungen stärken oder wieder verwerfen/anpassen/verbessern

Hypothesenbildung: Z.B Newton'sche Mechanik

Gliederung

- ⇨ Einleitung
- ⇨ Default Logic
- ⇨ Truth Maintenance Systems (TMS)
- ⇨ Justification-based TMS
- ⇨ Assumption-based TMS
- ⇨ Zusammenfassung

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 ($1 \leq i \leq n$) $\neg B_i$ nicht abgeleitet werden kann, dann leite C ab.**

A, Bi und C sind Formeln.

Semantik: „Wenn A bekannt ist und keine Evidenz für B besteht, dann kann C inferiert werden. Es gilt die Closed World Assumption.

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 $\nvdash$ 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

triggert C: X,M / A, denn weder ¬X noch ¬M sind aus {X,Y,Z} herleitbar und A folgt aus {A,B,C}

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 $(E_i, E) \}$

Zyklische (rekursive) Darstellung der Extension (es gibt auch anderer Definitionen), die schwierig algorithmisch umzusetzen ist.

Die Extension einer Default-Theorie beschreibt das aktuell gültige Wissen, bestehend aus dem sicheren Wissen W und allen gültigen Defauls aus D.

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.**

Außerdem: Nicht für alle Default-Theorien existieren Extensionen. Beispiel: eine Default-Theorie mit A: B / ¬B ∈ D besitzt keine Extension denn das Default kann weder getriggert noch nicht getriggert werden.

Aus Wikipedia: A normal default theory is guaranteed to have at least one extension. Furthermore, the extensions of a normal default theory are mutually inconsistent, i.e., inconsistent with each other.

Beispiel

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

  $E = W \cup \{ F(Tweety) \}$

- ⇨ $W = \{ V(T), P(T), \forall x\, P(x) \Rightarrow \neg F(x) \}$, $D = \{ V(x) : F(x) / F(x) \}$

  $E = W$

- ⇨ $W = \{ V(T), P(T) \}$, $D = \{ V(x): F(x) / F(x) , P(x): \neg F(x) / \neg F(x) \}$

  $E = W \cup \{ F(T) \}$ oder $E = W \cup \{ \neg F(T) \}$

- **„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

- ⇨ Einleitung
- ⇨ Default Logic
- ⇨ **Truth Maintenance Systems (TMS)**
- ⇨ Justification-based TMS
- ⇨ Assumption-based TMS
- ⇨ Zusammenfassung

Truth Maintenance — TMS

⇨ **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**

RETRACT entfernt ein Element aus der KB.

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, ¬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.**

Aufsteigende Nummerierung.

Die KB wird zu einem Stack, alle durch eine Perzeption inferierten Formeln tragen denselben Index.

sehr komplex!

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

Die Inferenzmaschine erzeugt neues Wissen, z.B. durch Theorembeweiser oder durch Interaktion mit einem Nutzer. Wissen wird als Annahmen und in Form von Rechtfertigungen dem TMS mitgeteilt. Das TMS verwaltet die Annahzmen und Rechtfertigungen und ist in der Lage, mit negierten Annahmen (Rücknahme einer Annahme) umzugehen.

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

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

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 ∈ KB drückt keinesfalls *Wissen* aus:**

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

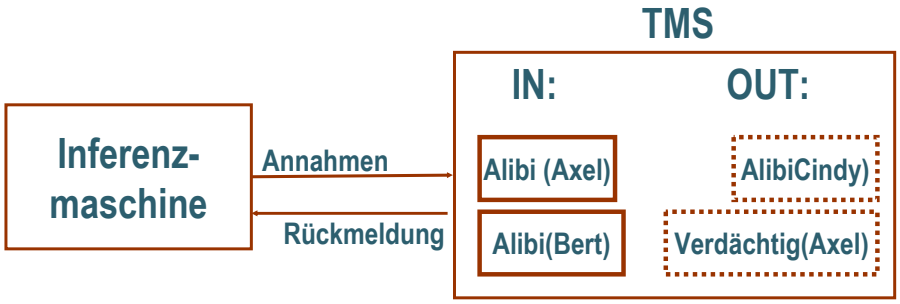
Knoten werden auch Node und in Spezialfällen (s.u.) Assumption genannt.
Eine Formel ist entweder IN oder OUT, beides gleichzeitig geht nicht.

TELL(Node(Verdächtig(Axel), OUT))).

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

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

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



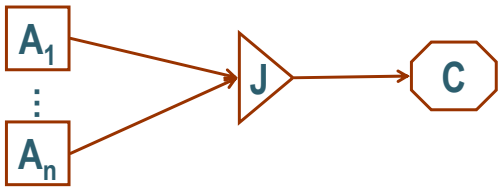
Das TMS verwaltet den Status aller definierten Knoten (IN oder OUT).

mit Node/2 wird ein Knoten auf IN oder OUT gesetzt. Falls der Knoten vorher nicht existierte, dann wird er erzeugt.

Justification

JTMS

- **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

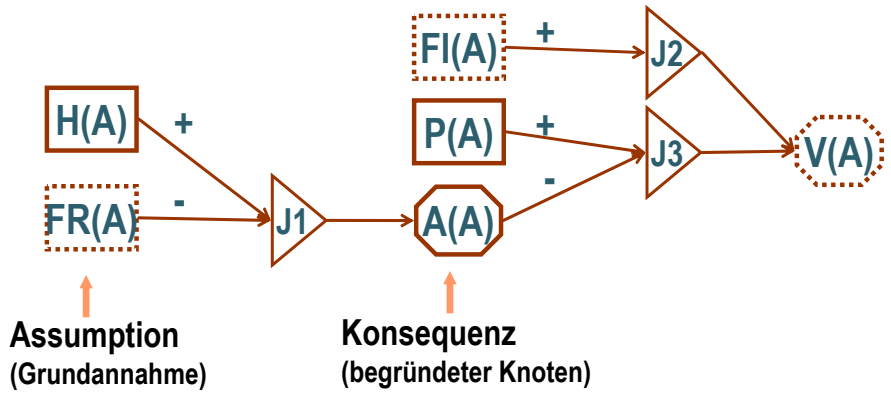


  - A1, ... An sind die Vorgänger, C ist die Konsequenz.
- **Justify([A1, ..., An], C).**

A1, ... An sind die unterstützenden Beliefs für die Konsequenz C. C wird geglaubt, wenn A1...An IN sind.

Abhängigkeitsnetze JTMS

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



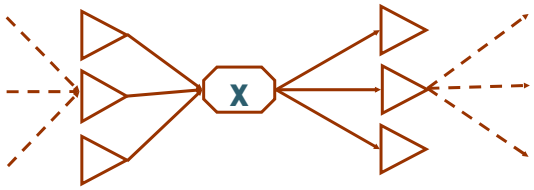
Unter der Annahme, dass Hotelreservierung vorliegt, die zunächst als echt anzunehmen ist, kann die links stehende Justifikation A(A) schlussfolgern. A(A) ist demnach IN, was das “Schalten” der rechten Justification verhindert, sodass V(A) OUT ist.

H = Hotelregistration; (A) = (Axel); FR = FakeReg; P = Profitiert; V = Verdächtig; A = Alibi; Fl = Flucht

Vereinbarungsgemäß werden (von dieser Stelle an für den Bereich JTMS) gepunktete Knoten als OUT und Knoten mit vollständigen Linien als IN gekennzeichnet.

Jeder Knoten im TMS besitzt … JTMS

- ⇨ **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

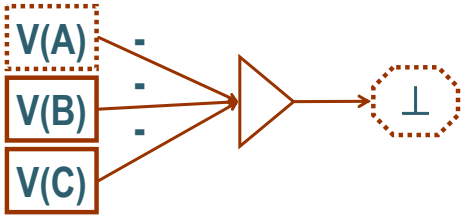


Unter der Annahme, dass Hotelreservierung vorliegt, die zunächst als echt anzunehmen ist, kann die links stehende Justifikation A(A) schlussfolgern. A(A) ist demnach IN, was das “Schalten” der rechten Justification verhindert, sodass V(A) OUT ist.

Ein Support genügt, damit ein Knoten IN ist.

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.**



Gelungeneres Beispiel: Außer A,B und C gibt es keine weiteren Verdächtigen:
Darzustellen als Contradiction([V(A), V(B), V(C), X])

Gliederung

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

Arbeitsweise eines TMS

- **Ein TMS arbeitet inkrementell**
  - Knoten und Justifications hinzufügen
  - Assumptions IN oder OUT setzen
  - Anfragen hinsichtlich IN / OUT stellen

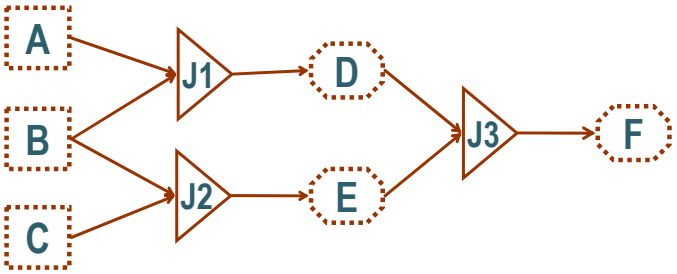
- **Bei einer Änderung ...**
  - ...wird das Label des betroffenen Knotens entsprechend gesetzt
  - … und werden dessen Konsequenzen durch das Abhängigkeitsnetz propagiert ...

JTMS - Propagierung

Aufbau des Abhängigkeitsnetzes:

⇨ **Assumption([A,B,C], OUT).**

⇨ **Justification([A,B],D), Justification([B,C],E), Justification([D,E],F)**

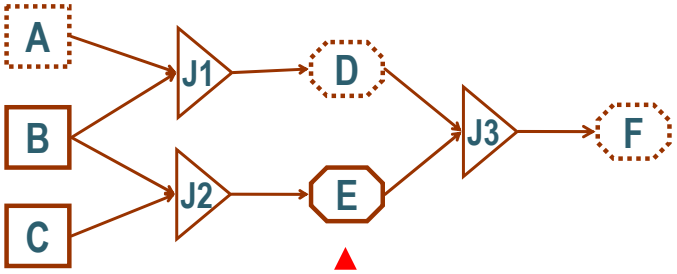


Die Knoten D, E und F werden durch die Justifications „automatisch“ ins Netz mit aufgenommen. Alternativ könnten sie auch über Node([D,E,F]) deklariert werden. Nodes können nicht IN oder OUT gesetzt werden, da dies ausschließlich implizit durch die Justifications geschieht.

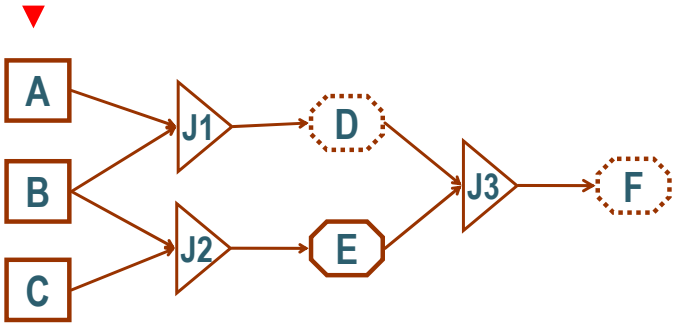
JTMS - Propagierung

⇨ **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

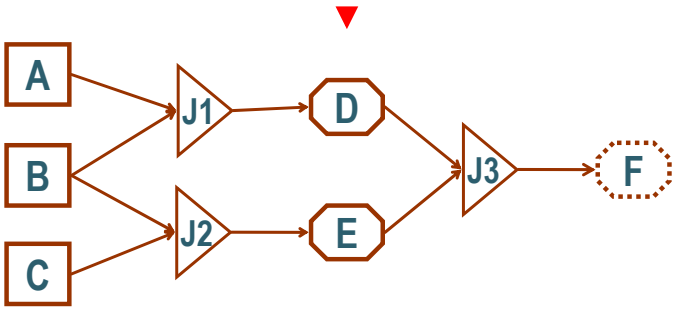


⇨ **Assume(A)**



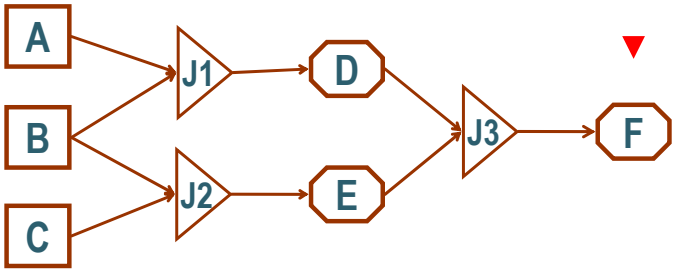
JTMS - Propagierung

- ⇨ 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

JTMS

⇨ **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.

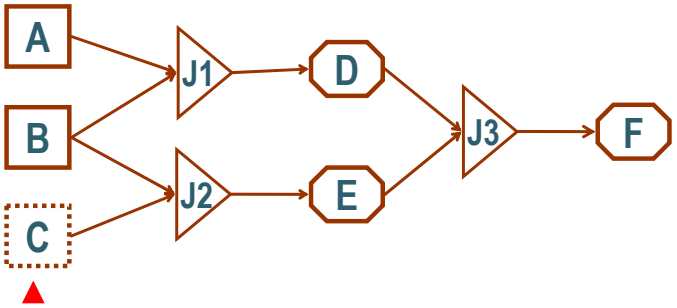
1.

2. … (durch A begründet sind)

2 und 3 natürlich nur dann, wenn A vorher IN war.

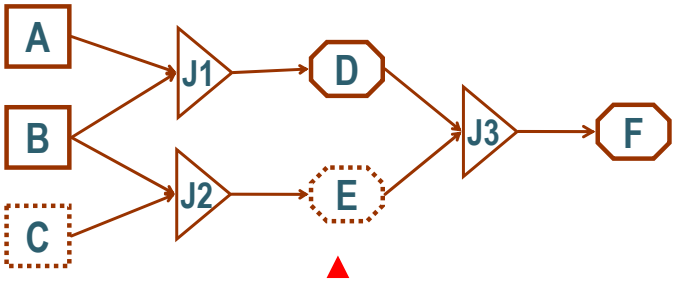
Beispiel OUT - Propagierung

⇨ **Assume(¬C).**



Beispiel OUT - Propagierung

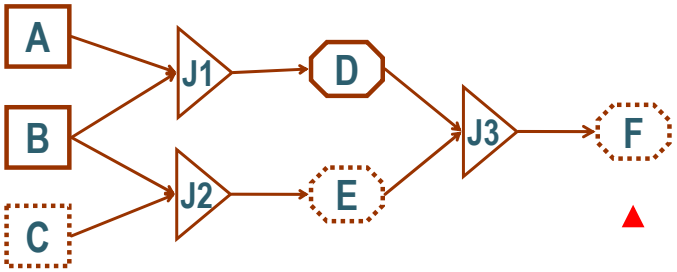
- ⇨ Assume(¬C).
- ⇨ E verliert (einzigen) Support.



E hat keinen alternativen Support (das wäre eine Justification, deren Antecetents alle IN sind).

Beispiel OUT - Propagierung

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

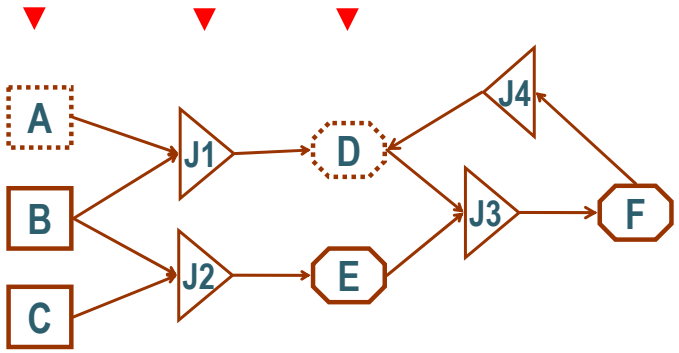


Re-Support

JTMS

⇨ Assume(¬A).

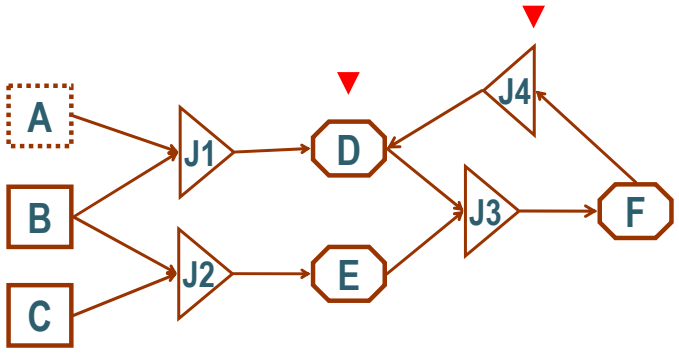
D verliert Support über J1.



D -> F, F -> D ist eine so genannte monotone Schleife, da alle Vorgänger IN sein müssen, damit die Justification schaltet.

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!



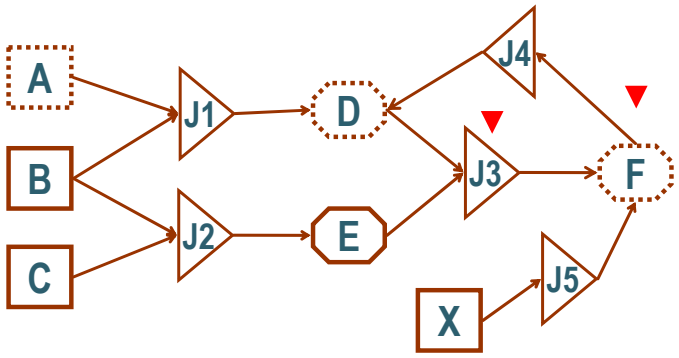
Derartige Probleme treten nur in Zyklen auf. Die Reihenfolge der „Propagierungsbearbeitung“ spielt eine Rolle: Wenn zunächst die OUT-Propagierung vollständig durchgeführt würde, könnte D nicht erneut über J4 begründet werden, da F nun OUT wäre.

Re-Support, 2. Beispiel

JTMS

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

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



Hier endet die Weiterpropagierung, da die Konsequenz von F==OUT (nämlich D) bereits OUT ist.

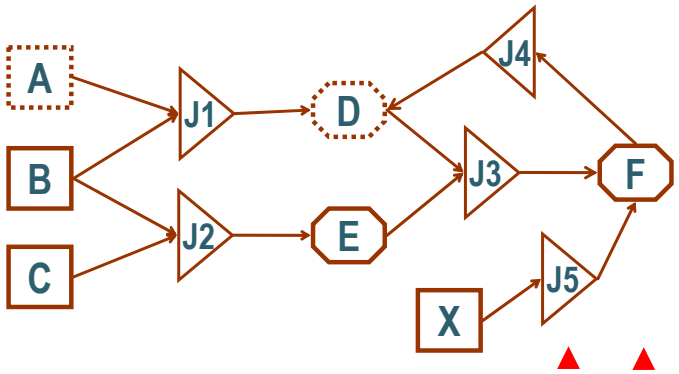
Monotone Zirkularitäten sind in vielen Anwendungsbereichen nützlich. Die Begründung von D ist wohlbegründet, da sie nicht zirkulär ist.

Wenn die Wissensrepräsentation eine Vorberechnung der Schleifen erlaubt, (z.B. ist es notwendig, dass für Schleifen verwendete Regeln keine Variablen enthalten) dann kann das System selbständig Zirkularitäten verwalten. Das Immediate-Check TMS [Puppe 87] erledigt dies durch Markierungen Zirkel-verdächtiger Schlussfolgerungen bereits beim Aufbau des TMS.

Re-Support

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

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



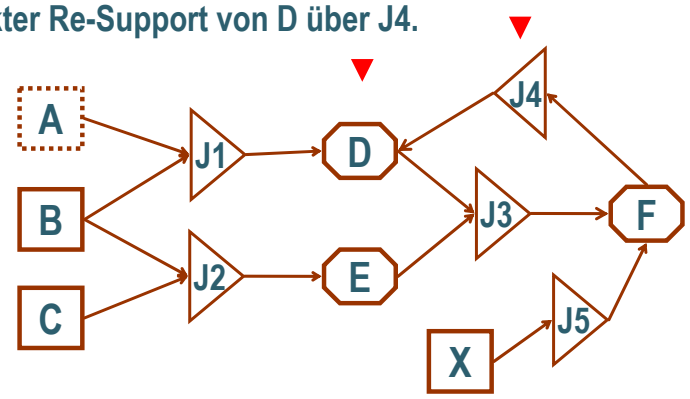
Re-Support

JTMS

⇨ 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.



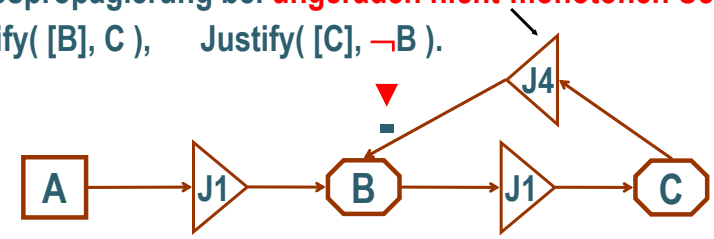
Dadurch, dass D wieder IN wird, müsste die Propagierung über J3 weitergehen. Sie endet, weil die Konsequenz von J3 (F) bereits IN ist.

D wird in diesem Fall auch nicht zyklisch supported, wie man am „Weg“ X -> F -> D erkennt.

JTMS-Probleme mit negativen Schlussfolgerungen

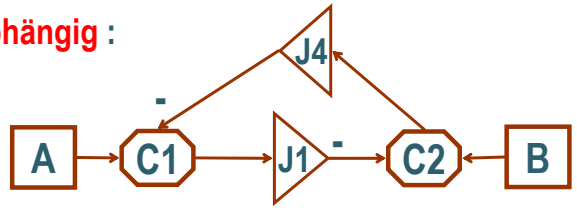
⇨ Endlospropagierung bei ungeraden nicht-monotonen Schleifen

Justify([B], C), Justify([C], ¬B).



⇨ Beliefs sind reihenfolgeabhängig :

Assume([A,B], IN) ≠
Assume([B,A], IN)



Monotone Schleifen haben wir oben bereits kennen gelernt. Nichtmonotonie bezieht sich hier auf das „Entfernen“ eines Beliefs, d.h. das OUT-Setzen als Konsequenz einer Justification.

Ungerade bezieht sich auf die Anzahl der Negationen (OUT) in der Schleife.

Nach assume(S1, IN) führt die JTMS-Propagierung in eine Endlosschleife. Im Wechsel werden S1 und S2 etabliert und wieder zurückgezogen.

Gliederung

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

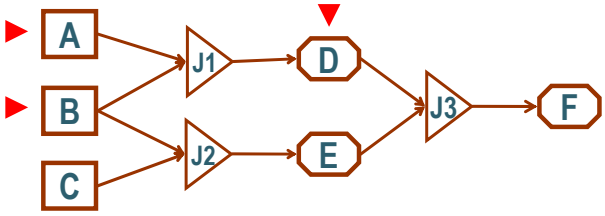
Vom JTMS zum Assumption-based TMS (ATMS)

- ⇨ 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.

ATMS macht Reasoning in multiplen Kontexten.

Idee

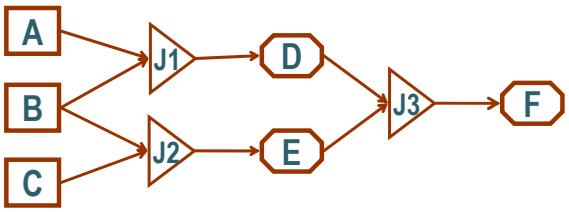
⇨ **Ob ein Knoten geglaubt wird, hängt von einer Menge von Assumptions ab:**



→ 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.**

- **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.

Man spricht dabei auch vom charakteristischen Environment.

E2 ist Spezialfall von E1, denn alles was aus E1=A folgt ist natürlich auch aus A und B und C.

Um Charakteristisches Environment zu berechnen sind Mengenoperationen notwendig.

⇨ **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

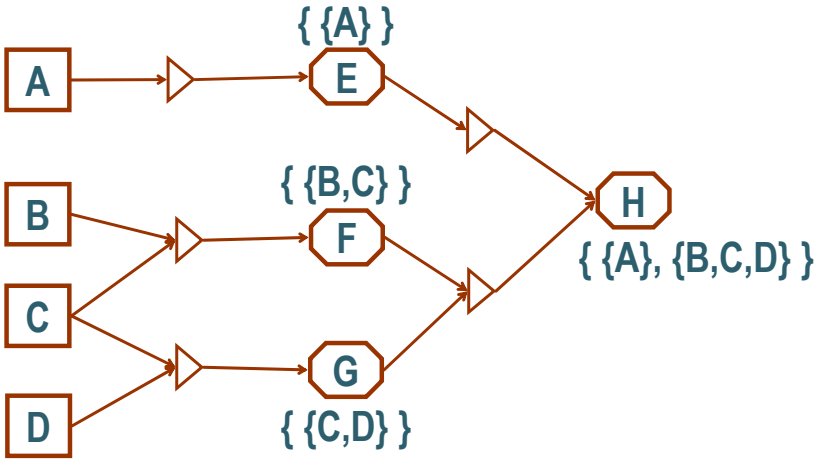
- **Ein Label L ist vollständig, wenn für jedes konsistente Environment E mit $E \notin L$ gilt: $E,T \models X \Rightarrow \exists E_i \in L : E_i \subset E$.**
  - "Zu jedem nicht minimalen E existiert ein minimales E_i in L"
  - T (Theorie) enthält alles Wissen, inklusive Regeln (Justifications)
- **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.**

Formale Definition der Vollst.: Ein Label L ist vollständig, wenn für jedes konsistente Environment E mit E nicht Element von L gilt: $E,T \models X \Rightarrow \exists E_i \in L : E_i$ ist echte Teilmente von E.

L ist minimal, wenn für jedes Ei in L, für das E eine echte Teilmenge von Ei ist gilt, dass X nicht aus E und T folgerbar ist. M.a.W., L besteht nur aus minimalen Environments, die zusammen Vollständigkeit der Folgerbarkeit garantieren.

Label: Beispiel

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

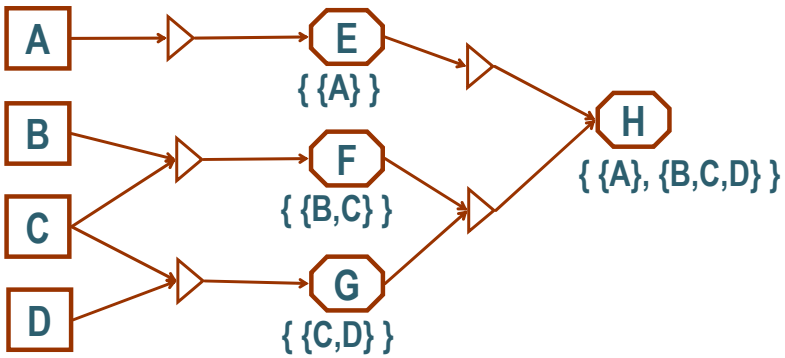


IN oder OUT ist vom Label ablesbar

ATMS

⇨ **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 }



Beispiel zeigt die Bedeutung der Minimalität: Durch einfache Teilmengenbestimmung ist erkennbar, ob ein Knoten aus einer gegebenen Menge von Assumptions folgerbar ist oder nicht.

Gliederung

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

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)**

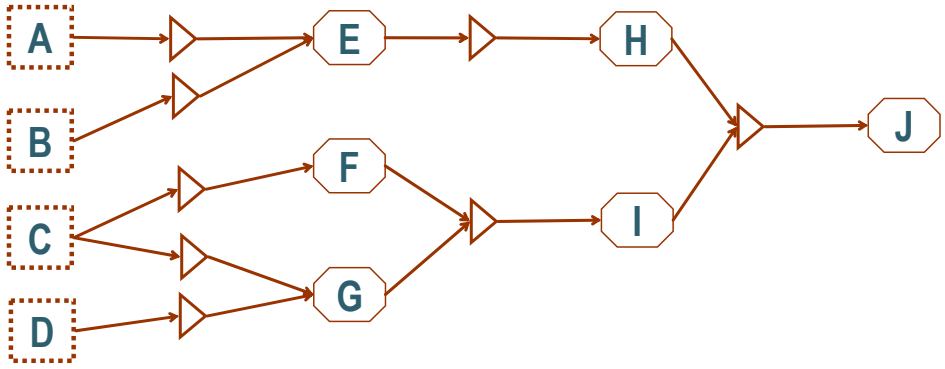
{ {A}, {B} } ∧ { {B}, {C} } ⇒
{ {A,B}, {A,C}, {B,B}, {B,C} } ⇒ { {B}, {A,C} }

Summierung und Multiplikation bei Disjunktionen bzw. Konjunktionen bewahren die Minimalität und Vollständigkeit. VerODERte Environments werden kombiniert, verUNDete Environments in allen ihren Teilmengen kombiniert.

Disjunktion ist, wenn ein Knoten mehrere Justifications hat (deren Labels sind dann entsprechend zu kombinieren).

Als Konjunktionen sind die Antecedents einer Justification zu behandeln, zur Berechnung des Labels für die Konsequenz.

Label-Propagierung

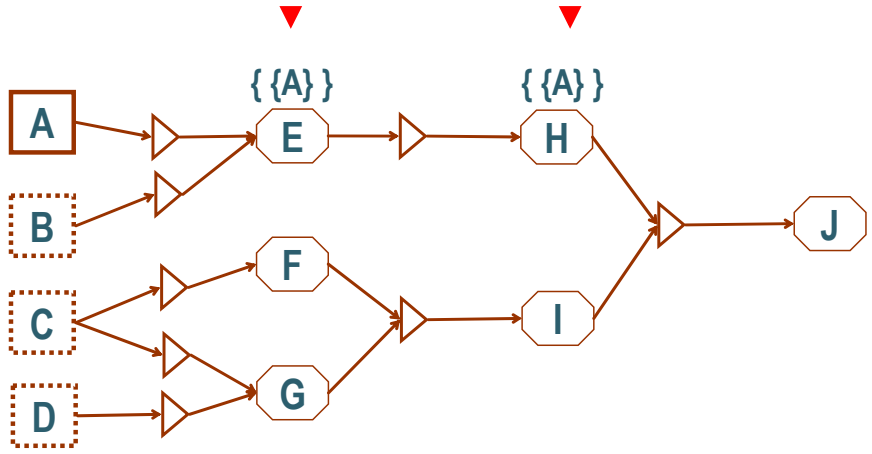


Gestrichelte Asumjptions sind noch nicht bekannt / propagiert. (Nicht etwa OUT wie in den JTMS-Folien!)

Label-Propagierung

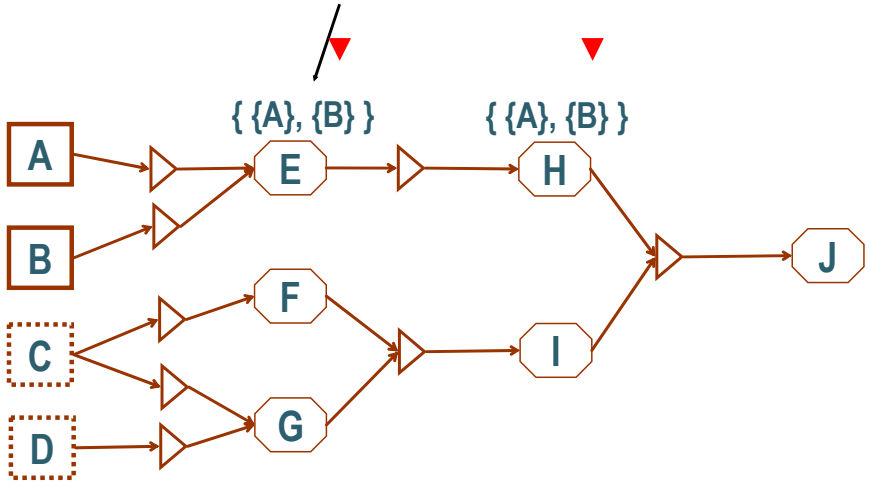
ATMS

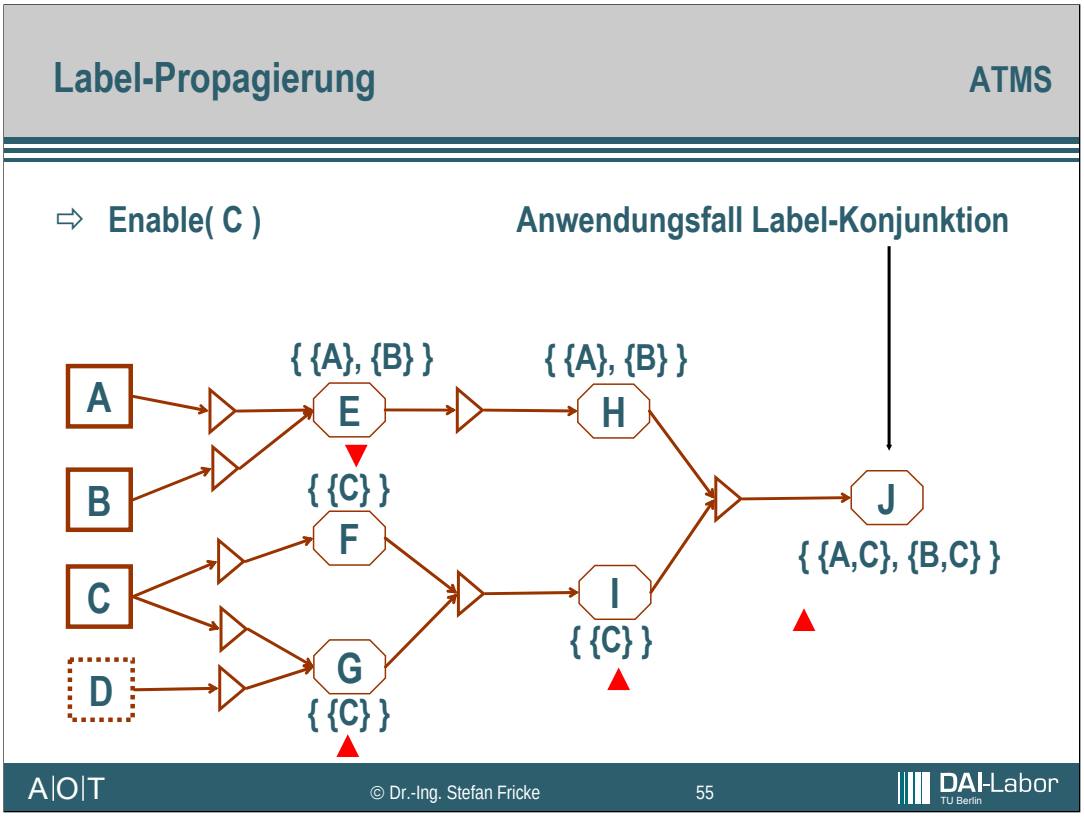
⇨ Enable(A)



Gestrichelte Asumjptions sind noch nicht bekannt / propagiert. Assumptions mit vollen Linien sind „enabled“, d.h. dem ATMS bekannt gemacht. Das bedeutet nicht, dass sie auch IN sind! Denn das ATMS verwaltet ja sämtliche IN/OUT-Permutationen aller bekannter (enabled‘ter) Assumptions.

⇨ Enable(B) Anwendungsfall Label-Disjunktion



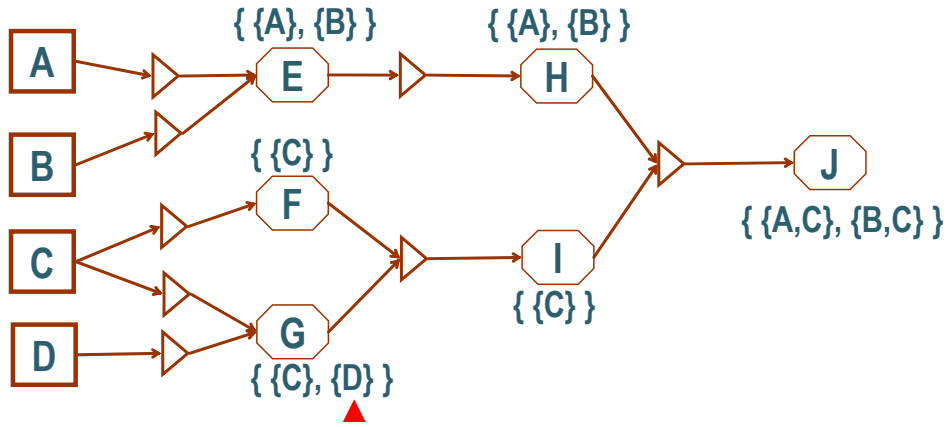


Das Label von J berechnet sich durch die VerUNDung der Label von I und H.

Label-Propagierung

ATMS

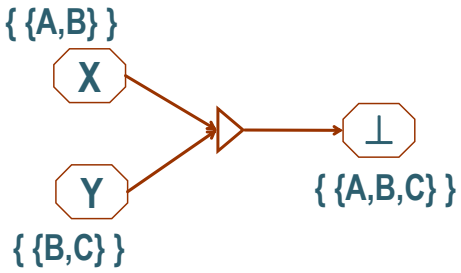
⇨ Enable(D)



Die Label-Konjunktion für I erbringt keine Änderung aufgrund der Minimalitätsanforderung. C ist auf jeden Fall ein Environment, und D taucht nicht alleine auf, sodass {C} auch das einzige, minimale Env. für den Knoten I ist.

Nogoods

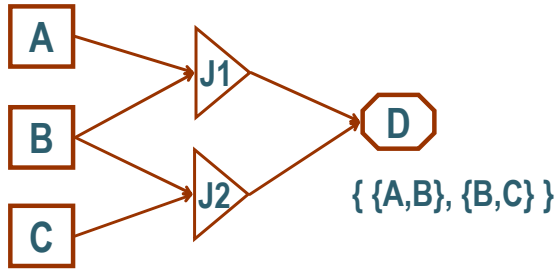
⇨ Inkonsistente Environments werden als Nogoods repräsentiert.



⇨ Die Semantik ergibt sich aus dem Kontext, aber…

Propagierung von Nogoods

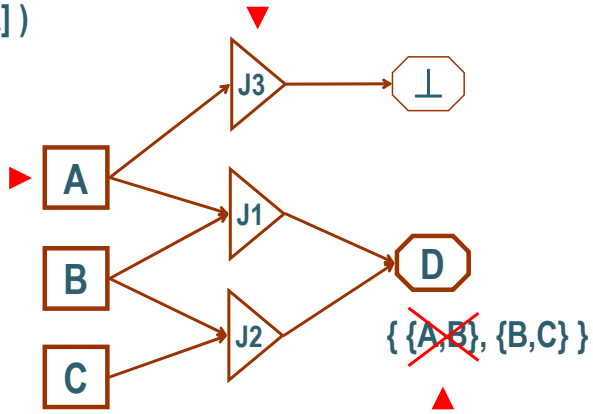
⇨ **Beispiel: Ausgangssituation**



Nogoods blockieren die Label-Propagierung

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

Z.B.: Nogood([A])



Gliederung

- ⇨ Einleitung
- ⇨ Default Logic
- ⇨ Truth Maintenance Systems (TMS)
- ⇨ Justification-based TMS
- ⇨ Assumption-based TMS
- ⇨ Zusammenfassung

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

Lernziele:

Verstehen, worum es in der KI geht, was ihre Wurzeln sind und wie sie sich entwickelt hat.

Den Begriff des Agenten kennen lernen und die damit verknüpften Konzepte der Rationalität, Umgebung und Agentenarchitektur.

Einen Überblick über die inhaltliche und organisatorische Struktur dieser LV bekommen.

noch ergänzen:

#3 (Obermayers Themen)

#4 (Lernziele bestätigen oder ggfs. überarbeiten)

Änderungen zur vorherigen Version:

1. zielorientiert und nutzenorientiert anstelle von zielgetriebenund nutzengetrieben.
2. Folienkommentare den Architekturfolien zugefügt.

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

- **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.**

Durch eine Default-Theorie (D,W) werden „Überzeugungsmengen“ S induziert.

In der Logik bezeichnet man eine Menge von Formeln *F* als **deduktiv abgeschlossen**, wenn die Menge aller Formeln, die aus einer der Formeln von *F* logisch folgen, gerade die Menge *F* ergeben. Demnach ist F nicht weiter erweiterbar.

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

Aus ∅ lässt sich alles folgern

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

Unfortunately, this algorithm is NP-complete

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