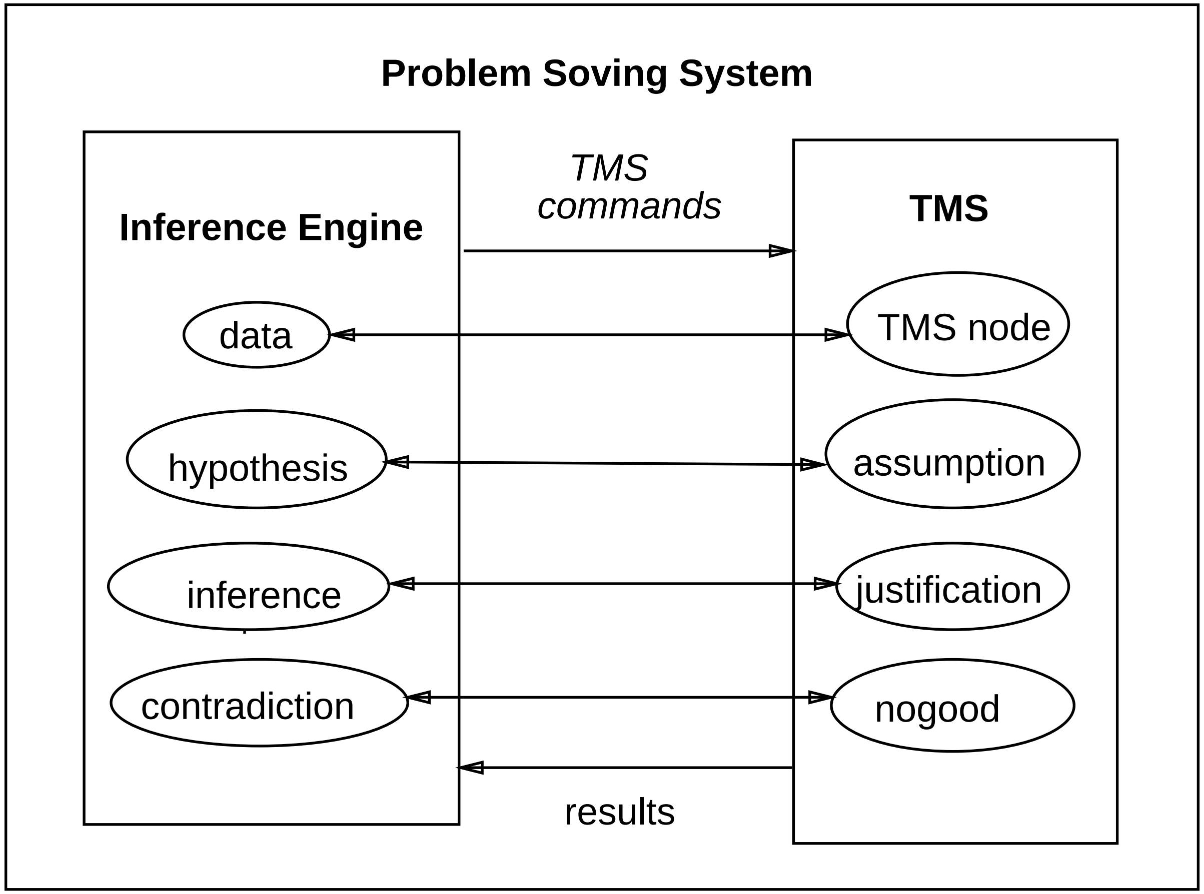
1. 問題解決システム (Problem Solving System)
   - Forbus, K. and de Kleer, J.: *Building Problem Solvers*, MIT Press, 1993. *Lisp* コードは公開 *ftp://beta.xerox.com/pub/bps/BPS*.{tar.Z,sit,zip}*
   - 奥乃: 人工知能学会誌, 5(3) '90, 6 (1) '91, 11(3)
2. 多重文脈推論 (Multiple Context Reasoning)

問題解決システムの一般的な構成



推論エンジン (Inference Engine): 問題の解決

真偽維持システム (Truth Maitenance System): 推論管理

なぜ真偽維持システム (TMS) を使用するのか

6 つの必要な機能

1. 陳述間の論理的な関係の遵守

   $\Longleftarrow$ 組合せ探索問題の解を見つけるのに必要

2. 結論に対する根拠の同定

   $\Longleftarrow$ 結論に対する説明を生成するのに必要

3. 失敗原因の同定

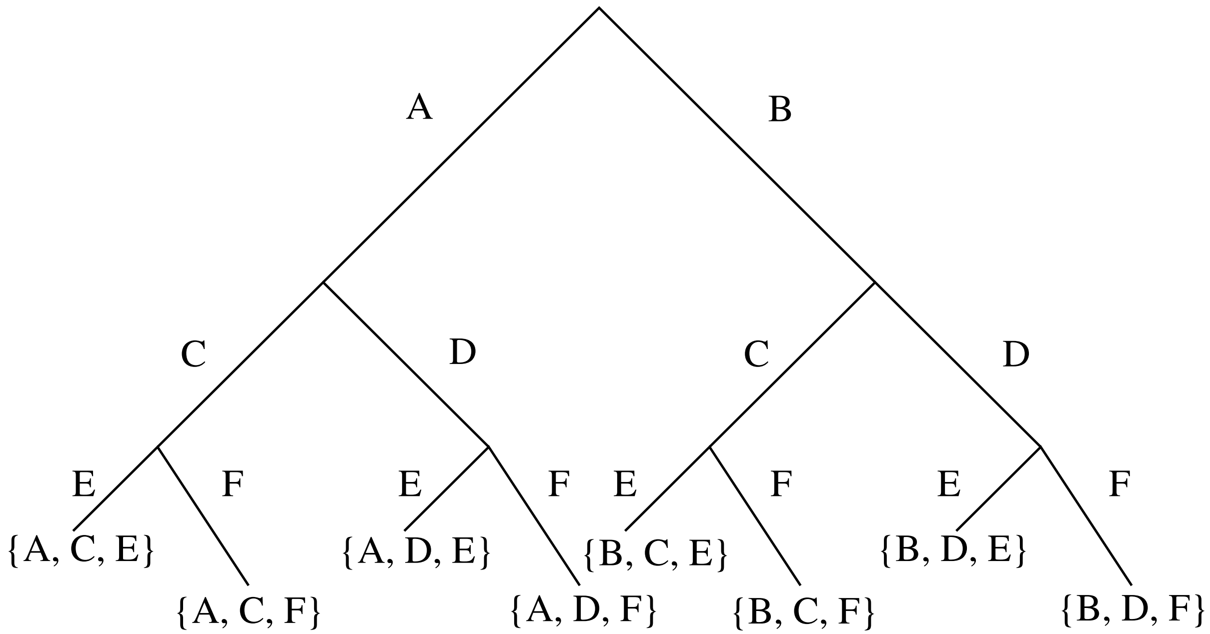
   $\Longleftarrow$ 矛盾状態から回復するのに必要

4. 推論結果のキャッシュ維持
5. 後ろ戻りのガイド
6. デフォールト推論

例：組合せ探索問題

select$\{$A,B$\}$, select$\{$C,D$\}$, select$\{$E,F$\}$

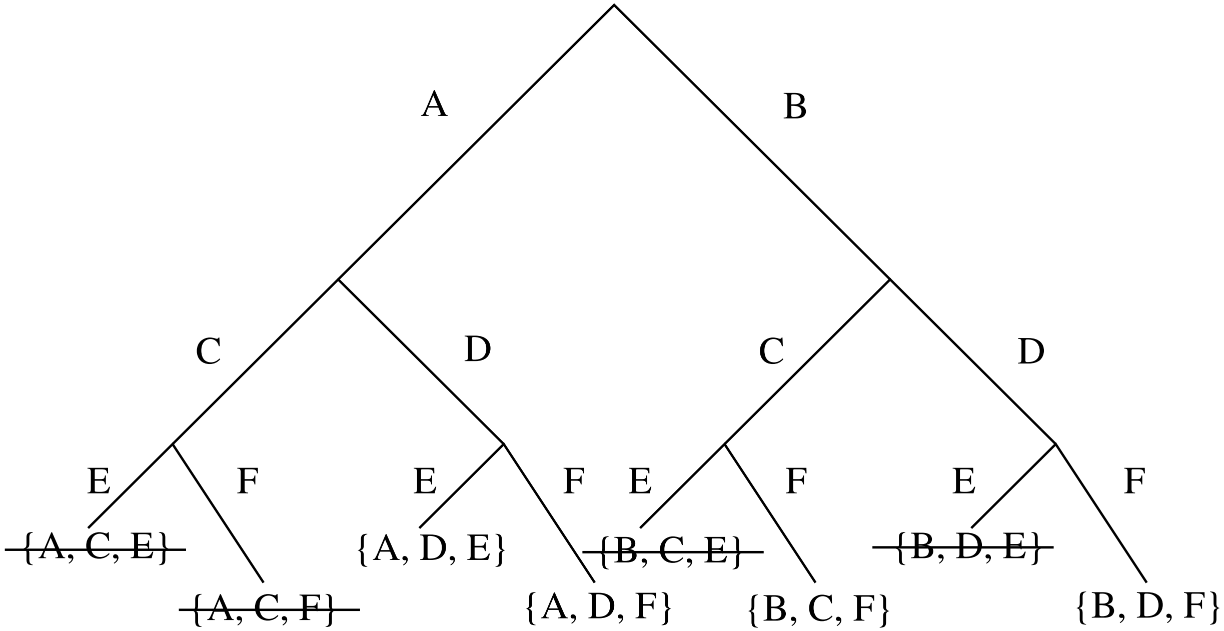
ただし, A $\wedge$ C $\longrightarrow \perp$, B $\wedge$ E $\longrightarrow \perp$



全知全能の神様がこの問題を見ると

$$\text{select}\{A,B\},\ \text{select}\{C,D\},\ \text{select}\{E,F\}$$

ただし, $A \wedge C \longrightarrow \perp$, $B \wedge E \longrightarrow \perp$



$\Longrightarrow$ $\boxed{}$ 個の潜在的な解がある.

全解を Prolog で求めよう.

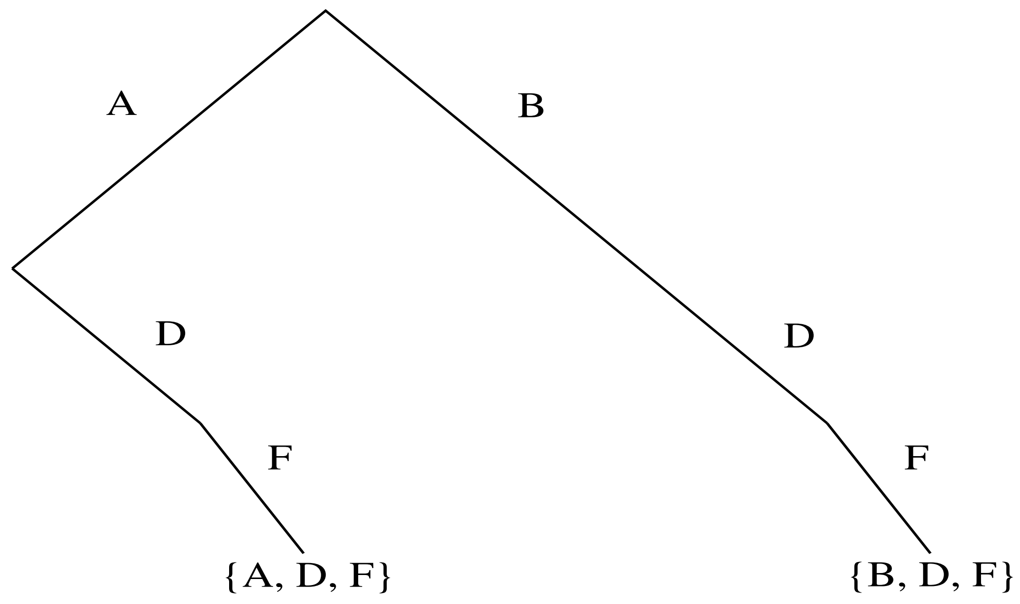
```
solve :- first(F), second(S), third(T),
   constraint(F,S,T), print([F,S,T]), fail.
first(a).
first(b).
second(c).
second(d).
third(e).
third(f).
constraint(a,c,_) :- !, fail.
constraint(b,_,e) :- !, fail.
constraint(_,_,_).
```

実行方法：

時間的な後ろ戻り (*Chronological Backtracking*)

時間的な後ろ戻りの欠点 (1) 無駄な計算

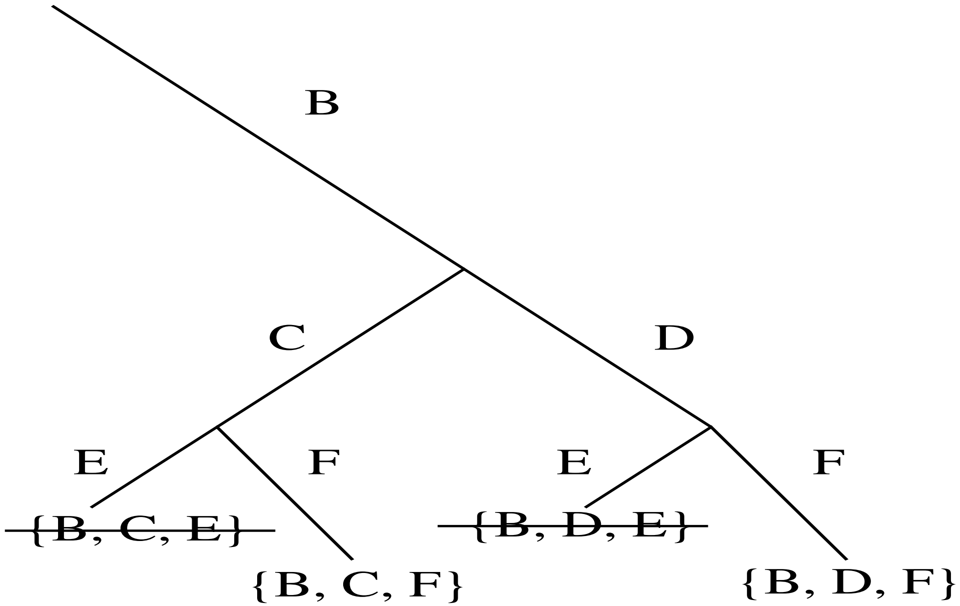
D と E では計算時間が膨大にかかるとすると,



改善策 $\Longrightarrow$ 推論や計算の結果をキャッシュに

時間的な後ろ戻りの欠点 (2) 同じ矛盾に遭遇

「B ∧ E ⟶⊥」 を思い出そう

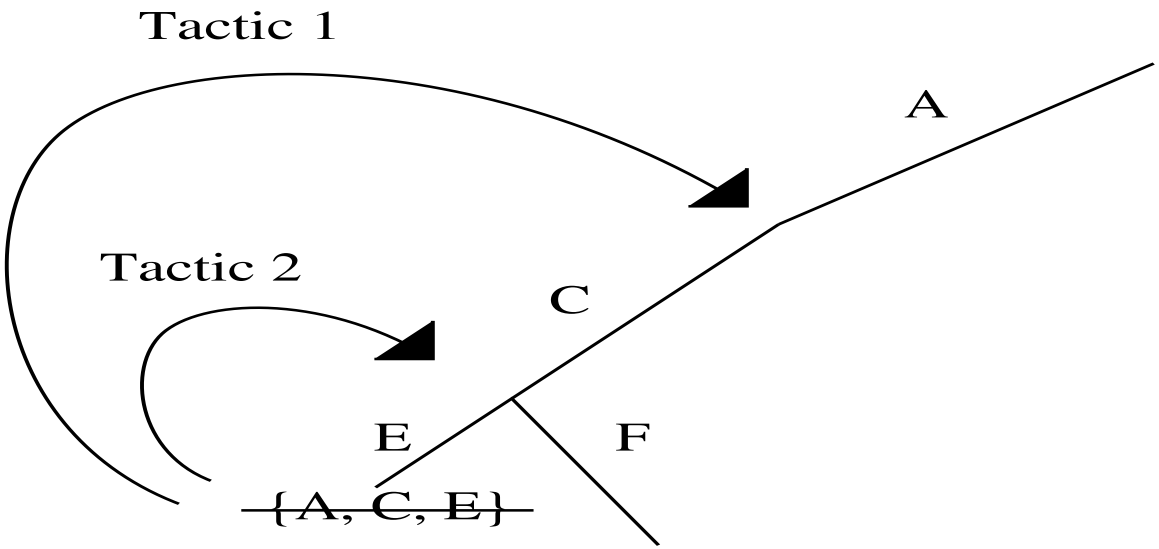


改善策 ⟹ 推論結果 (矛盾) をキャッシュに

後ろ戻りのガイド

どちらの後ろ戻り方法の方がよいか: $\boxed{}$

「A ∧ C $\longrightarrow \perp$」 を思い出そう



方法 1: $\Longrightarrow$ 依存関係による後ろ戻り
($Dependency\text{-}Directed\ Backtracking,\ DDB$)

デフォルト推論

『反例がない限り x が成立することを仮定する』

[例 1] 仮定「**Since Tweety is a bird, Tweety can fly.**」

「**Tweety can fly.**」を証明するためには,

問題解決システムは以下のことを示す必要がある:

$$\left\{\begin{array}{l}\bullet\ \textbf{Tweety is not a penguin, an ostrich, a kiwi,}\ldots \\ \bullet\ \textbf{Tweety is not dead, boiled,}\ldots \\ \ldots\end{array}\right.$$

これは $\square$ 可能，それとも $\square$ 不可能

Credo ut intelligam. (I believe in order to understand.)

デフォールト推論 (続き)

[例 2] 閉世界仮説 (*Closed World Assumption, CWA*)

『集合のメンバは自分が知っているものだけである』

- 論理型プログラミング:

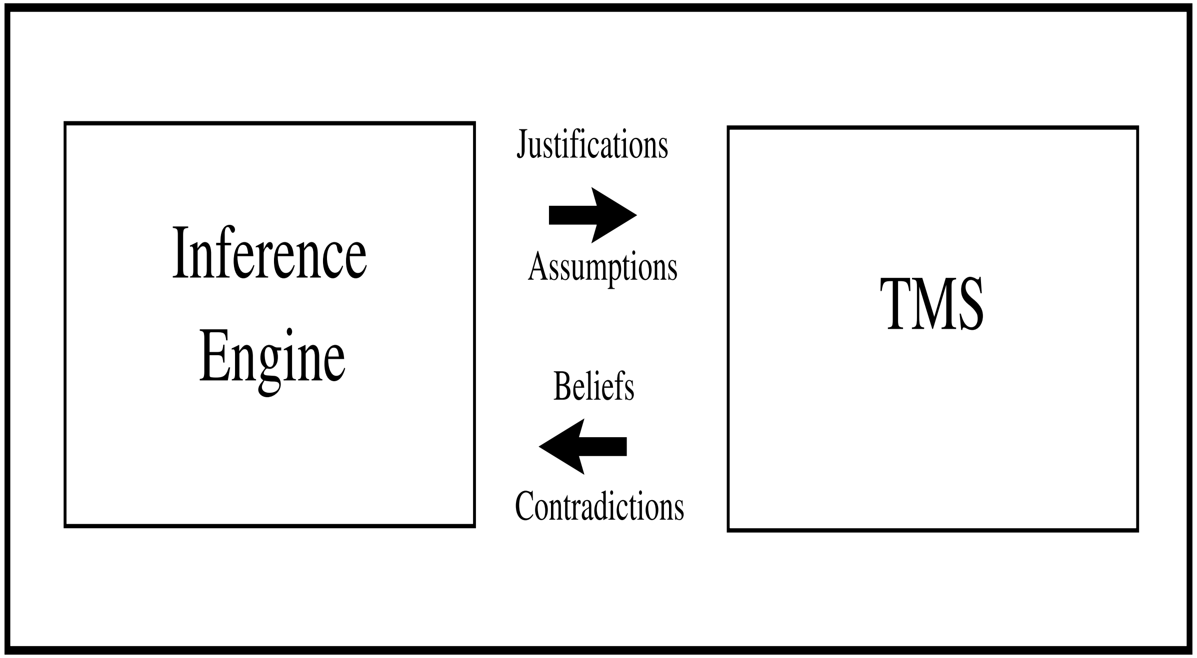
  失敗による否定の実現　(**Negation as Failure**)

```
constraint(a,_,c) :- !, fail.
constraint(_,b,e) :- !, fail.
constraint(_,_,_).
```

- 設計型システムへの応用:

  $\Longrightarrow$ 候補集合を指定する (『選択集合』)

問題解決システム ＝ 推論エンジン ＋ TMS



エキスパートシステムシェル (KEE, ART), 定性シミュレーション, 設計システム, 故障診断システム, 時間的推論, 知的 CAI システム, 制約言語, 非単調推論, . . .

推論エンジンと真偽維持システムの対応関係

推論エンジン	⟺	真偽維持システム
データ (**Datum**)	⟺	ノード (**Node**)
推論過程 (**Inference**)	⟺	正当化 (**Justification**)
仮説 (**Hypothesis**)	⟺	仮定 (**Assumption**)
矛盾 (**Contradiction**)	⟺	**Nogood**, あるいは, 後ろ戻りによる矛盾解消
信念 (**Belief**)	⟺	ラベル (**Label**)

推論エンジン (IE)・真偽維持システム (TMS) 間の通信

$$\frac{\textsf{Graduate-Student}(x) \longrightarrow \textsf{Underpaid}(x) \wedge \textsf{Overworked}(x) \qquad \textsf{Graduate-Student(Tanaka)}}{\textsf{Underpaid(Tanaka)} \wedge \textsf{Overworked(Tanaka)}}$$

ノード N0001：　推論エンジンのルールを表現

```
(rule (Graduate-Student ?x)
   (assert (and (Underpaid ?x) (Overworked ?x))))
```

ノード N0002：　言明（assertion）を表現

```
(Graduate-Student Tanaka)
```

ノード N0003：　推論結果を表現

```
(and (Underpaid Tanaka) (Overworked Tanaka)
```

正当化 J0001：　ルール適用（inference）を表現

```
(N0003 MODUS-PONENS N0002 N0001)
```

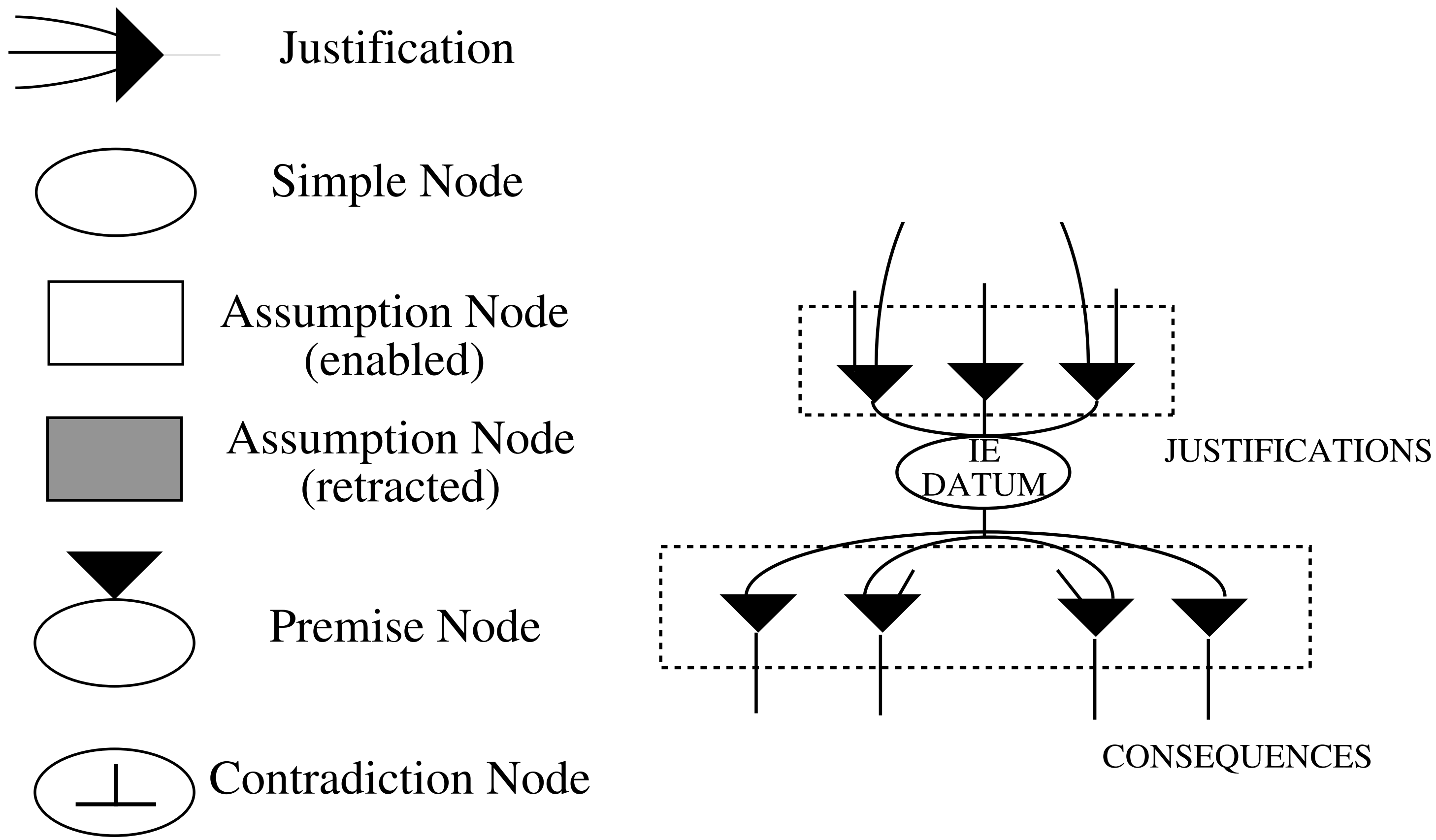
ノードの種類と正当化

- 前提 (**premise**) — 恒真であるデータ
- 矛盾 (**contradiction**) — 恒偽であるデータ
- 仮定 (**assumption**) — 推論エンジンが一時的に成立すると選択したデータ．後で取り消されることもある
- 通常のノード — 上記以外

正当化 は (< 後件 > < 情報部 > ・ < 前件 >) で表現.

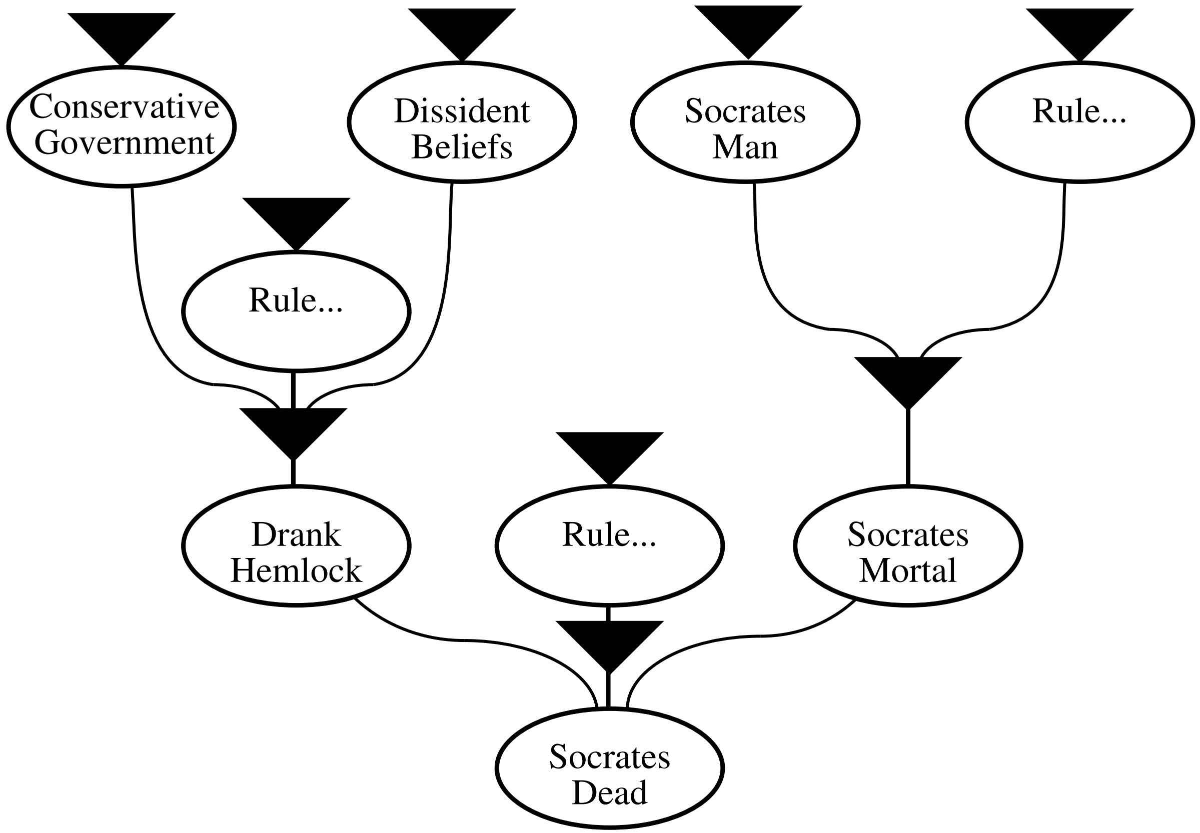
- 後件 (**Consequent**) ： ノード.
- 前件 (**Antecedents**) ： ノード.
- 情報部 (**Informant**) ：何でも可． **TMS** は未使用.

依存関係のネットワーク表現



正当化の役割： 結論に対する根拠の同定

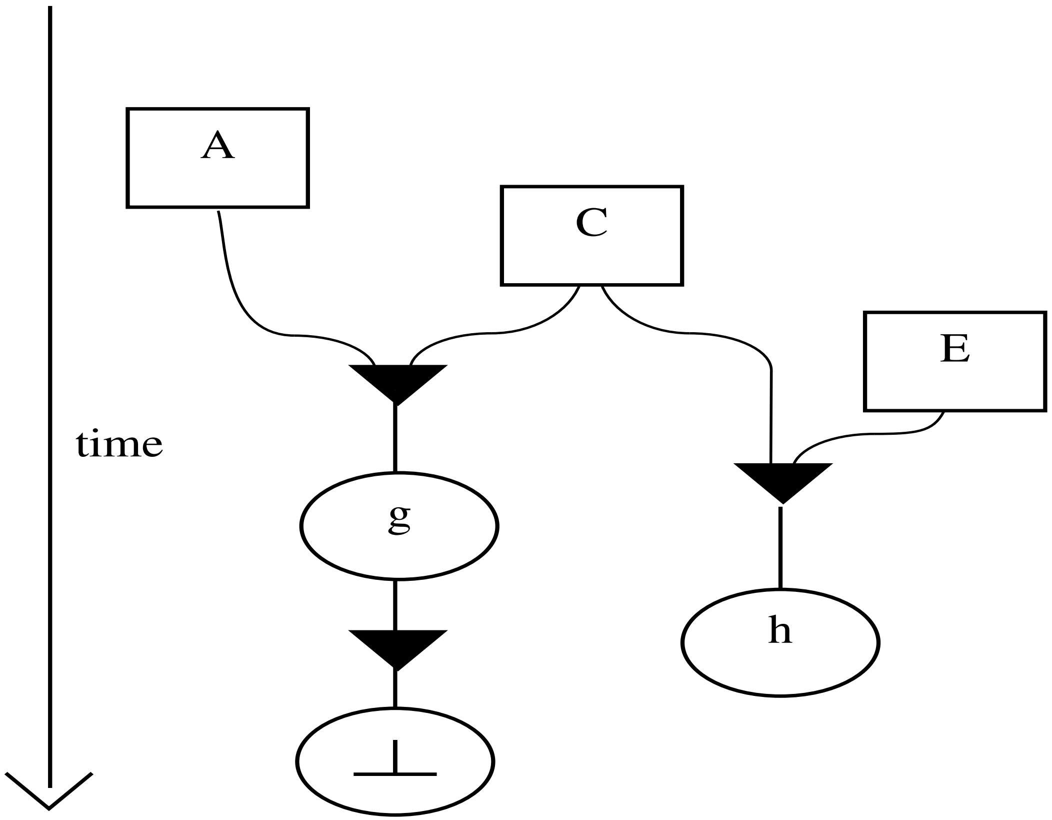
「ソクラテスはなぜ死んだか」の説明生成



正当化の役割： 後ろ戻りのガイド

時間的な後ろ戻りでの戻り先：

依存関係による後ろ戻りでの戻り先：



命題論理による真偽維持システムの記述

- ノード $x_1, \ldots, x_m$ が，ノード n を正当化：

  $\neg x_1 \vee \cdots \vee \neg x_m \vee n$ で表現．

  つまり，$$\overbrace{\underbrace{x_1 \wedge \cdots \wedge x_m}_{\text{前件}} \Rightarrow \underbrace{n}_{\text{後件}}}^{\text{正当化}}$$

  一般には，$$x_1 \wedge \cdots \wedge x_m \Rightarrow y_1 \vee \cdots \vee y_k$$

  $k = 1$ の時，「ホーン節」と呼ぶ．

- 前提 n ： 単一節 (unit clause) n で表現．
- 矛盾 m ： 否定節 (negative clause) $\neg m$ で表現．

ラベル (Label) — ノードの信念状態の表現

IN — 「信念がある」, **OUT** — 「信念がない」

注意: **IN** (信念あり) $\neq$ **TRUE** (恒真)

	P **IN**	P **OUT**
$\neg P$ **IN**	矛盾 ($Contradiction$)	$\neg P$ **TRUE**
$\neg P$ **OUT**	P **TRUE** (恒真)	未知 ($Unknown$)

[拡張] 節 C のラベルを極小支持集合 **MinSup** で定義：

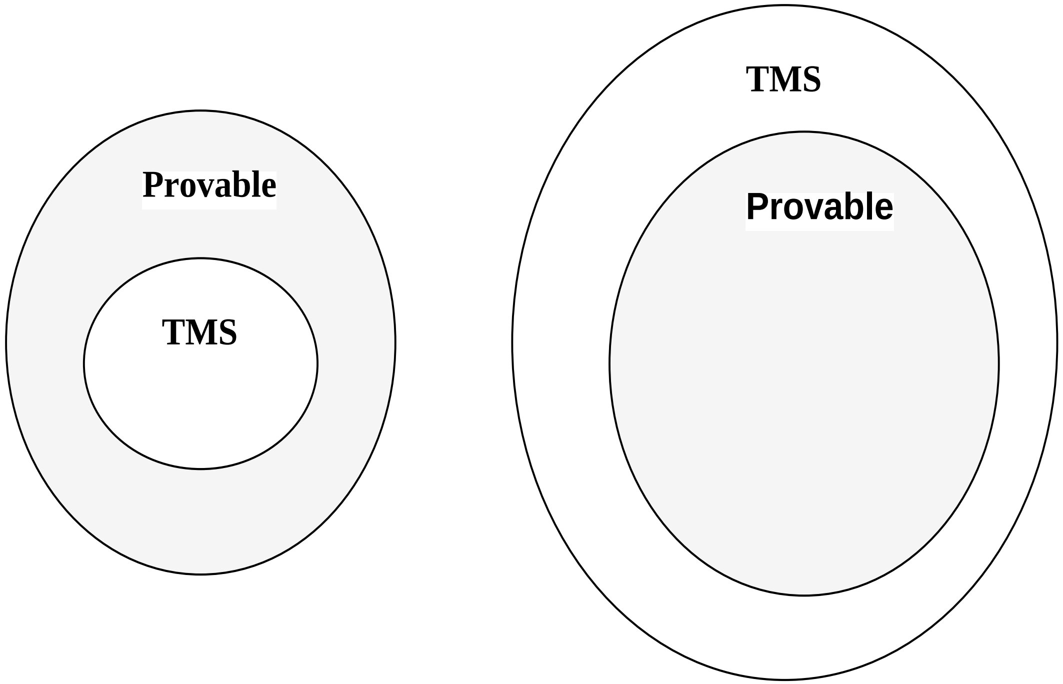
Σ 全正当化の集合とすると,

$$MinSup(C, \Sigma) = \{S | S \in \Delta(C, \Sigma), S: \text{は極小}\}$$

ただし, $\Delta(C, \Sigma) = \{PI - C | PI : \Sigma\text{の主項}, PI \cup C \neq \{\}\}$

真偽維持システムの健全性 と完全性

健全性 $(Soundness)$ 　　完全性 $(Completeness)$



完全性が成立するかどうかは **TMS** のファミリーに依存.

不完全性の例

$$\left.\begin{array}{l} P \Longrightarrow (Q \vee W) \\ Q \Longrightarrow R \\ W \Longrightarrow R \\ \Gamma = \{P\} \end{array}\right\}$$ R は Γ から導かれる.

どのような **TMS** でも R の正しいラベルが求まるか?

☆ ☆ ☆

正当化 $x_1 \wedge \cdots \wedge x_m \Rightarrow y_1 \vee \cdots \vee y_n$ ：

ホーン節 **(n=1)** の確定節 **(**変数を含まない**)** しか表現できない **TMS** では正しいラベルは求まらない．

非単調性 (Non-Monotonicity)

- **単調性 (Monotonicity):** すでに言明済みの他の全事実と無矛盾な新事実が追加されても，矛盾が生ぜず， 既知の真の事実の集合からなにも取り消す必要がない.
- **非単調性 (Non-Monotonicity):** 上記が成立しない．

[非単調性の例] 『鳥は飛ぶ』という知識の下に行われた推論結果の一部が, 『ペンギンは鳥だが, 飛ばない』という知識が加えられたことによって, 成り立たなくなる.

非単調性 (Non-Monotonicity) の例 — Nixon 問題

$$Republican(Nixon) \qquad Quaker(Nixon)$$

$$Quaker(x) \Rightarrow Dove(x) \qquad Republican(x) \Rightarrow Hawk(x)$$

$$Dove(x) \wedge Hawk(x) \Rightarrow \perp$$

TMS の分類

INPUT CONSTRAINT \ LABEL	Simple	Complex
Horn / Definite	JTMS	ATMS
NM	NMJTMS	
Clause	LTMS	CMS

JTMS (Justification-based TMS), ATMS (Assumpition-based TMS), NMJ (Non Monotonic JTMS), LTMS (Logic-based TMS), CMS (Clause Management System)