人工知能特論： 真偽維持システム (TMS)　　第 7 回

単一文脈の真偽維持システム (TMS)

1. JTMS (Justification-based Truth Maintenance System)
2. LTMS (Logic-based Truth Maintenance System)

多重文脈の真偽維持システム (TMS)

1. ATMS (Assumption-based Truth Maintenance System)
2. CMS (Clause Management System)

奥乃 博 (okuno@i.kyoto-u.ac.jp)

OHP：

JTMS Node Properties

- **Premise — Node.Justification has no antecedents.**
- **Contradiction — Node.Contradictory? is set.**

  A contradictory node become believed

  $\Longrightarrow$ JTMS informs IE.
- **Assumption — Node.Assumption? is set.**
  - *enabled* — IE chose to believe it
  - *retracted* — otherwise, treated as any other node.
- **(Normal) Nodes — otherwise.**

Q: *A Single or Multiple* **Contradiction Node(s)**

Propositional Specification of JTMS

$$\begin{cases} \textbf{the set of Justifications} \text{ — } \textit{monotonic increase} \\ \textbf{the set of Enabled Assumptions} \text{ — } \textit{retracted!} \end{cases}$$

JTMS node $\Longrightarrow$ a propositional symbol

A: {**symbols for enabled assumption nodes**}

J: {**Justification $\Leftrightarrow$ a propositional (definite) clause**}

$\Longrightarrow$ **JTMS returns the current status of belief:**

$$\begin{cases} \textbf{Node } n \textbf{ is } IN \Longleftrightarrow n \textbf{ follows from } A \cup J \\ \textbf{Node } n \textbf{ is } OUT \text{ otherwise} \end{cases}$$

When a Contradiction Node becomes Believed,

what happens?

JTMS signals a contradiction to IE.

- **TMS only maintains the labels.**
- **IE must retract assumptions to remove the contradiction.**

A Well-Founded Support (WFS) for Node n

"a Sequence of Justifications, $J_1, \ldots, J_k$" such that

- J_k justifies node n.
- All the antecedents of J_i

  $\left\{ \begin{array}{l} \text{are justified earlier in the sequence, or} \\ \text{enabled assumptions} \end{array} \right.$

- No node has more than 1 justifications in the seq.

Node has a WFS $\Leftrightarrow$ n follows from $A \cup J$

Exponential number of WFS

select one $\Longrightarrow$ *a Supporting Justification*

In and Out vs. True and False

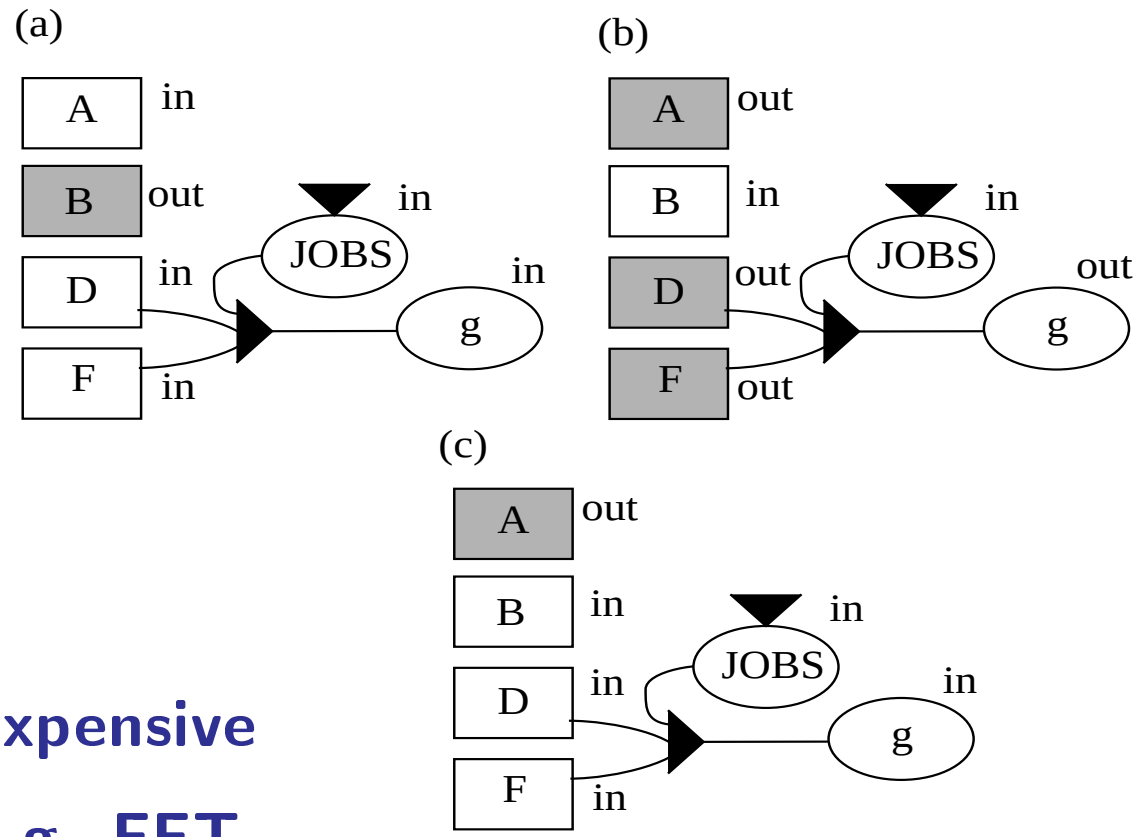
JTMS cannot deduce the Negation of Data

$\Longrightarrow$ Encoding the Negation of Data

		P: in	P: out
$\neg P$	in	Contradiction	$\neg P$
	out	P	Don't know

How Justifications Save Inference Engine Work

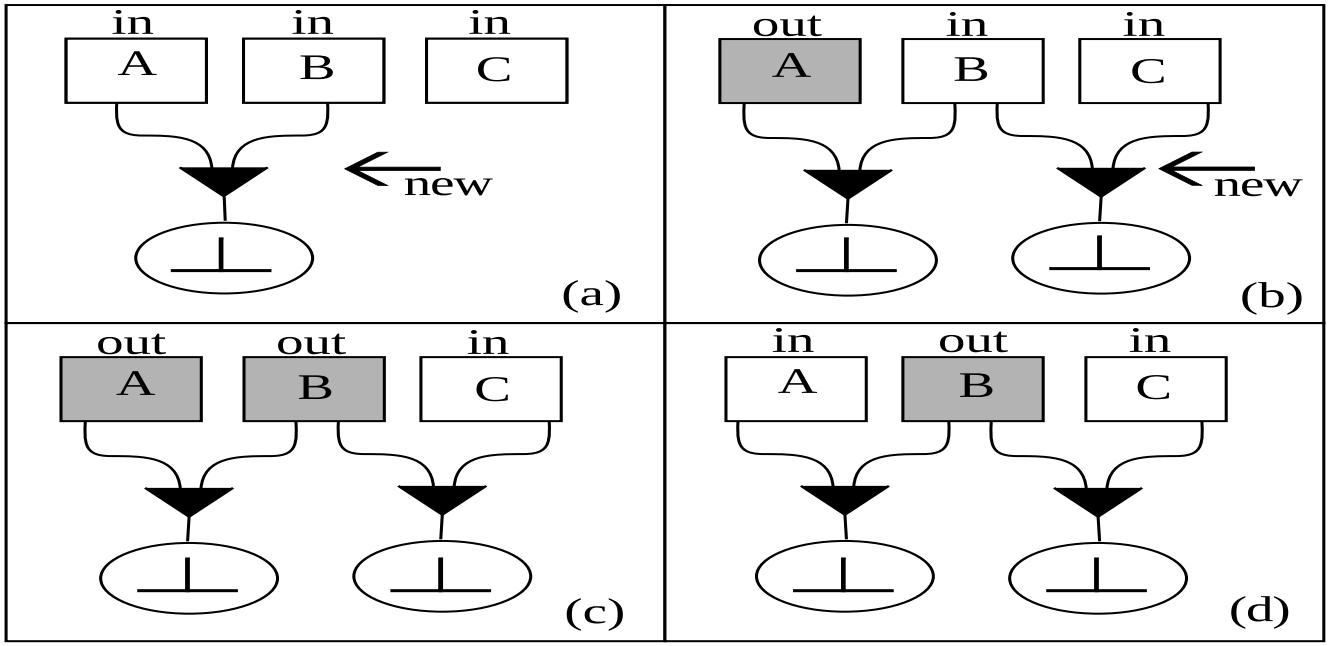
Maintaining a Justification Cache



JOBS involves expensive computations. e.g. FFT.

How Justifications Enable Default Reasoning

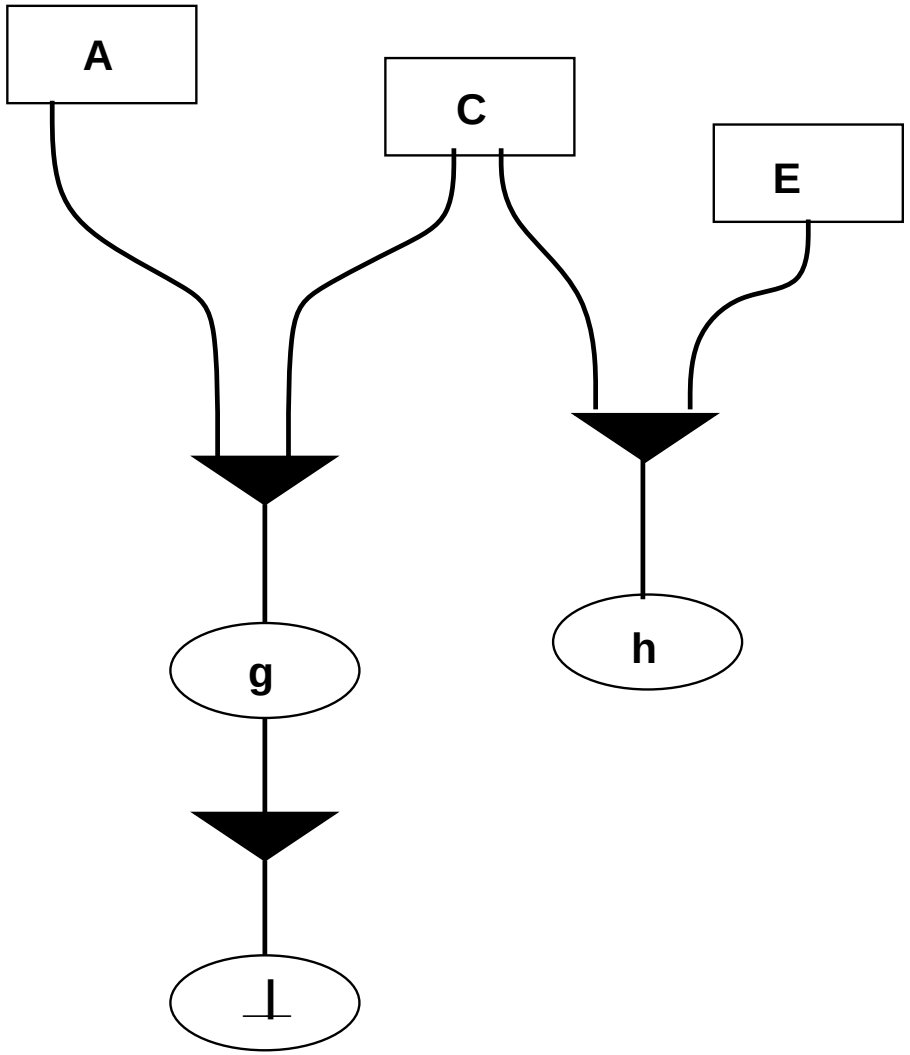
Maintaining the Semantics of Default – IN unless ⊥



(a) **Retract A or B** (b) **Retract B or C**

(c) **A violates the Semantics of Default** ⇒ (d)

Dependency-Network for the Simple Example



JTMS Algorithms — 3 Candidates for Basic Design

1. **Context Approach**:

   JTMS maintains an explicit set of nodes currently believed with their supporting justifications

2. **Lazy Approach**:

   JTMS computes the belief status of a node when requested by IE

3. **Labeling Approach**:

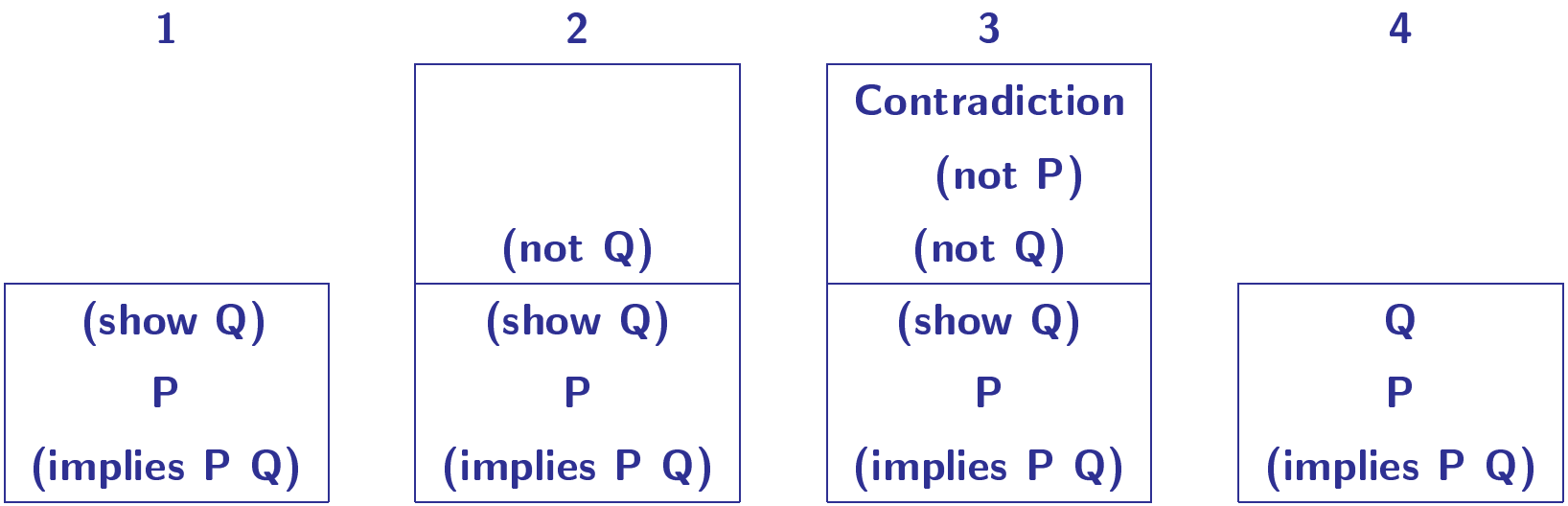
   each node is extended to have its belief status with supporting justifications

Context Approach Example:

Stack-Oriented Context Mechanism

Given **(implies P Q) and P,** *Prove* **Q**

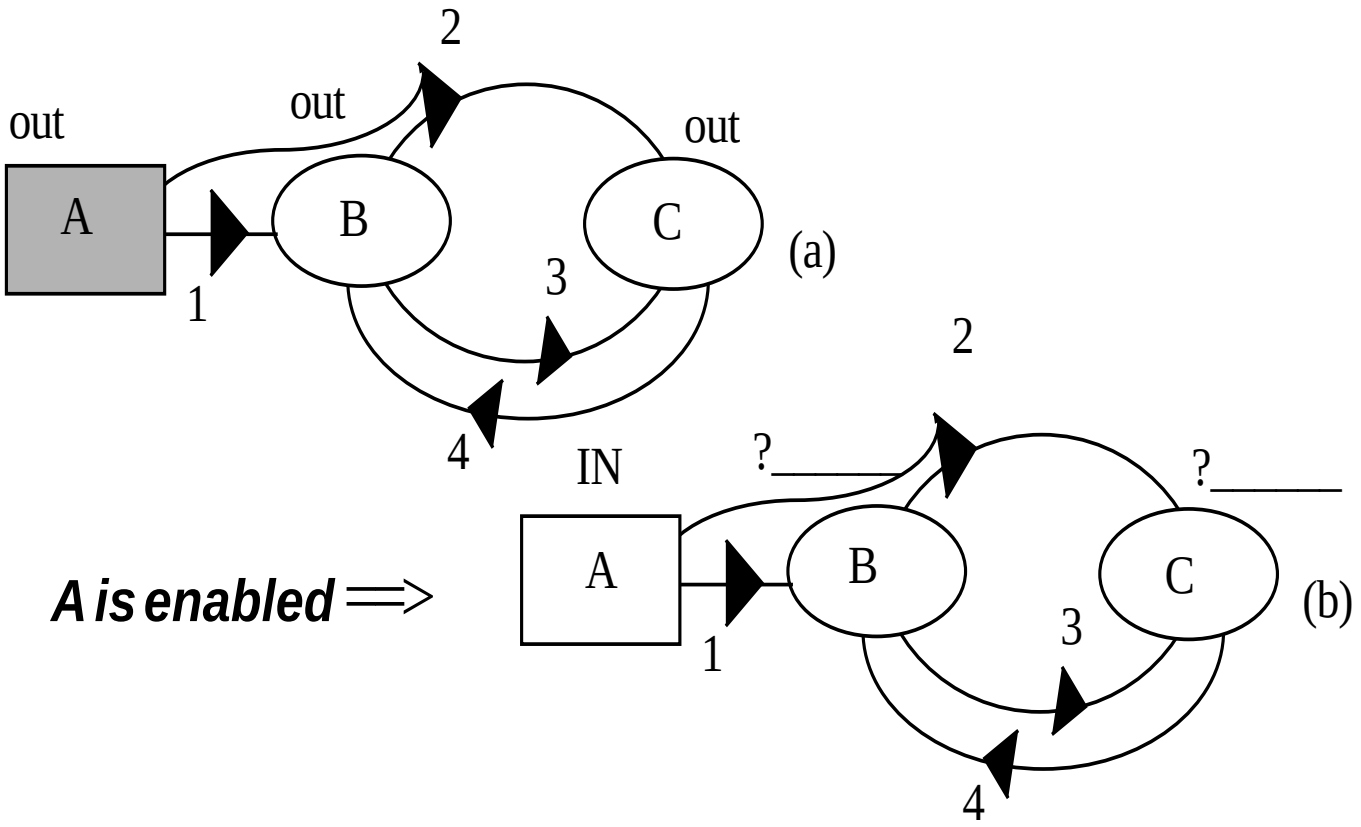
by Reductio Ad Absurdum



Enabling an Assumption A

1. Mark A as enabled, remove its current supporting justifications, and mark A as supported by itself.
2. If A is already in, return control to IE.
3. If any justification with A as its antecedent becomes satisfied (all antecedents are labeled in), label the consequent in and make the justification the supporting justification for it.
4. Check the consequent recursively.

Enabling an Assumption



A well-founded support for B: [ ] , for C: [ ]

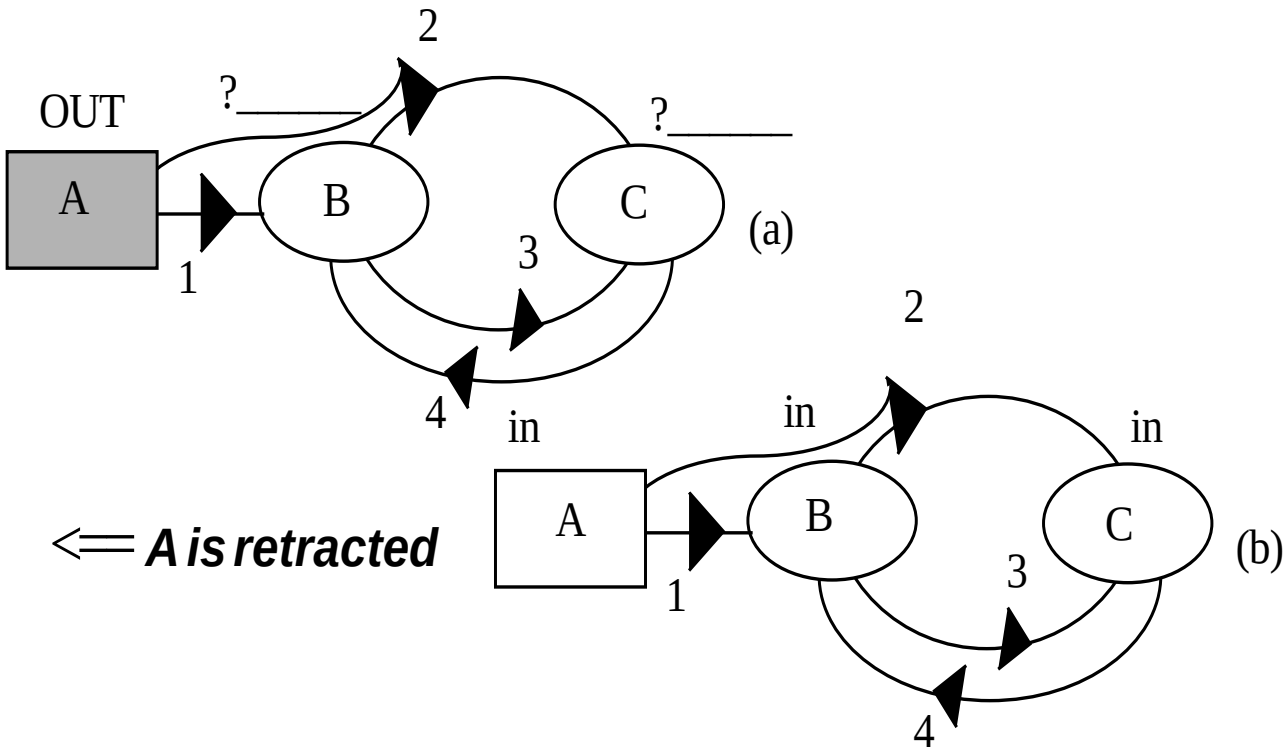
Adding a Justification J

1. Add J to the dependency network
2. If the consequent of J is in, return control to IE.
3. Check whether J is satisfied.
4. If so, label the consequent in and make J the supporting justification. Then check it recursively.

Retracting an Assumption: Why Difficult?

"Look for an alternative satisfied justification."

Is this algorithm Correct?



Node A is retracted, *but* Node B and C are still in.

Retracting an Assumption A

1. Label all nodes whose WFE contains A out.

2. Check whether every node labeled out in 1st phase has an alternative supporting justification. If so, label such node in and set the new supporting justification. Do it recursively as in the assumption enabling.

Solving N-Queens

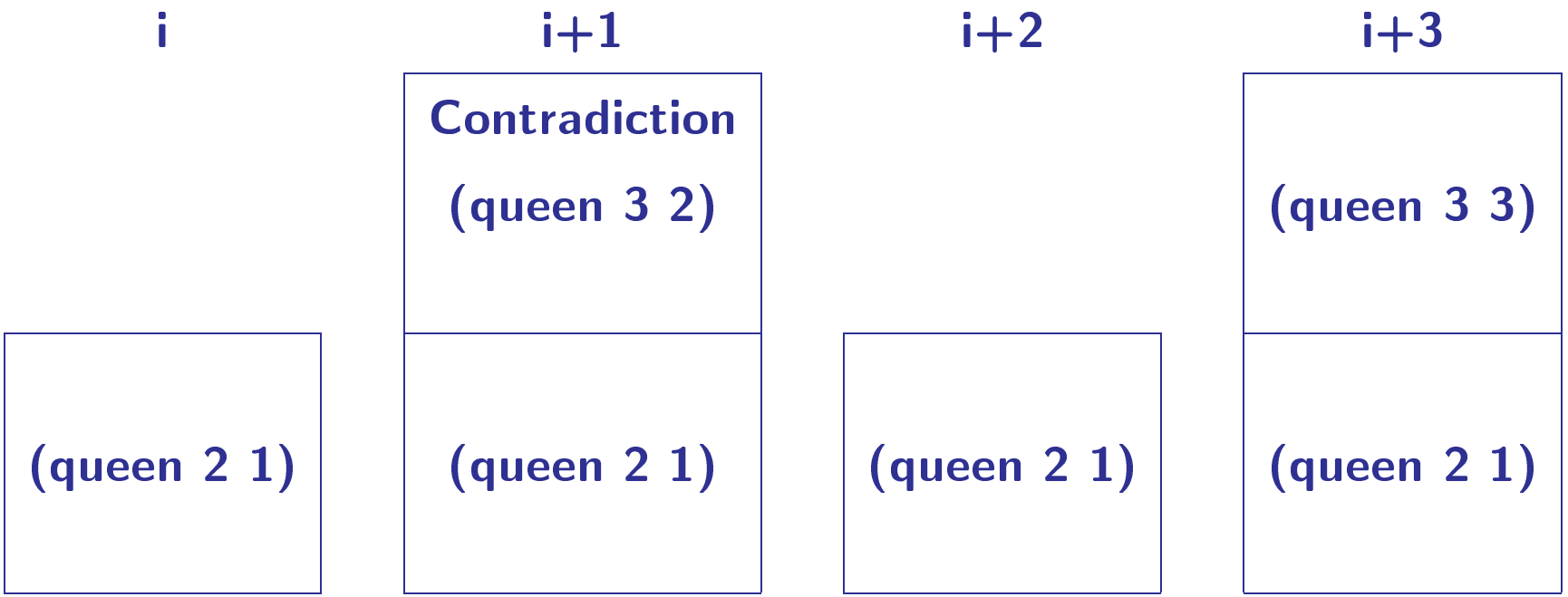
1. Chronological Search with Stack-Oriented Context Mechanism
2. JTMS-based Dependency-Directed Search

```
(defun chrono-search (choice-sets)
  (if (null choice-sets) (record-solution)
      (dolist (choice (first choice-sets))
         (choose-one-by-one choice
            (if (consistent?)
                (chorno-search (rest choice-sets)) )))))
```

choice-set $\Rightarrow$ **elements of each row**

Stack-Oriented Context Mechanism

Push a new context at each selection



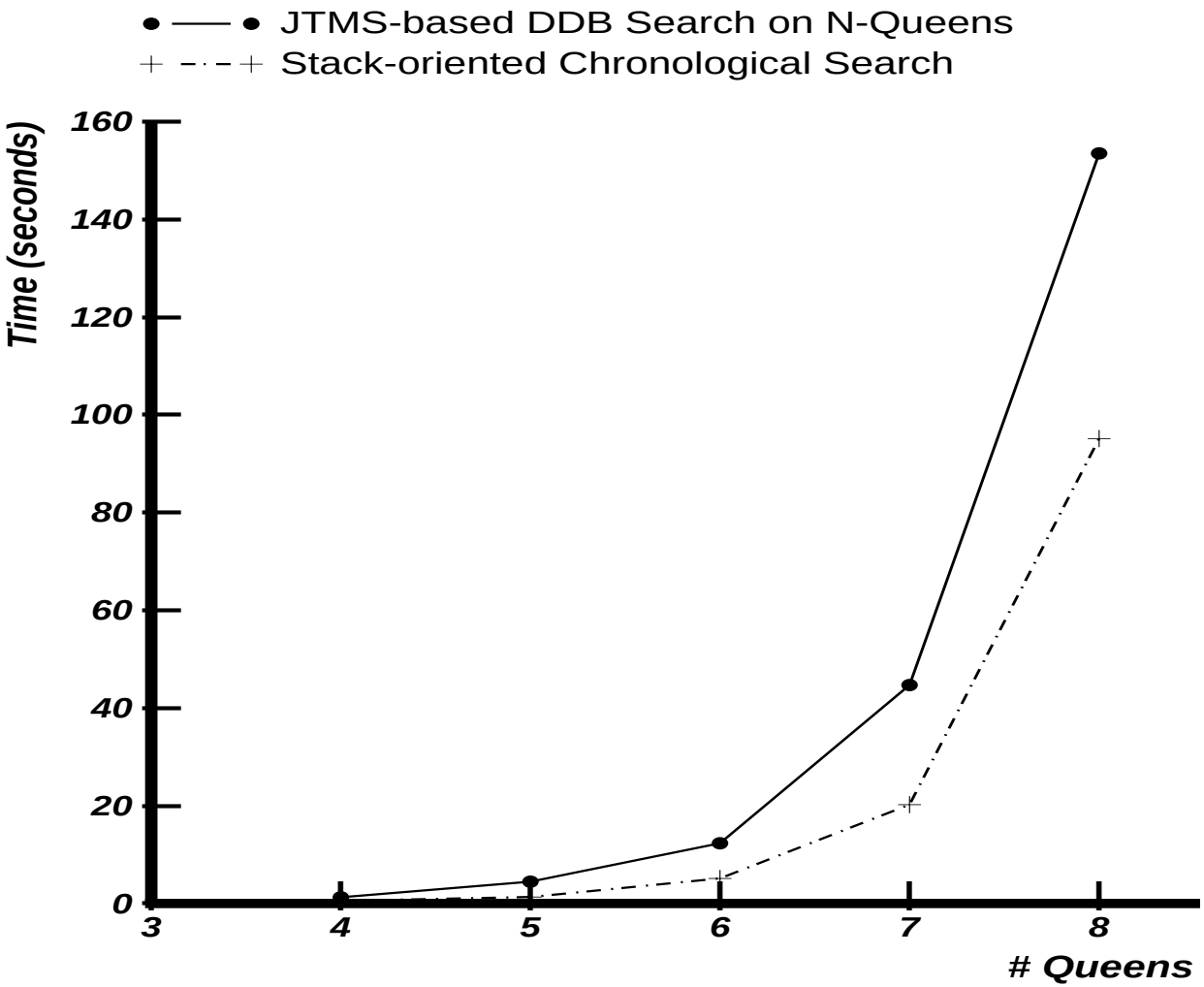
Dependency-Directed Search

```
(defun DDS (choice-sets)
  (if (null choice-sets) (record-solution)
      (dolist (choice (first choice-sets))
         (unless (nogood? choice)
            (choose-one-by-one choice
               (if (consistent?)
                   (DDS (rest choice-sets))
                   (record-nogood choice) ))))))
```

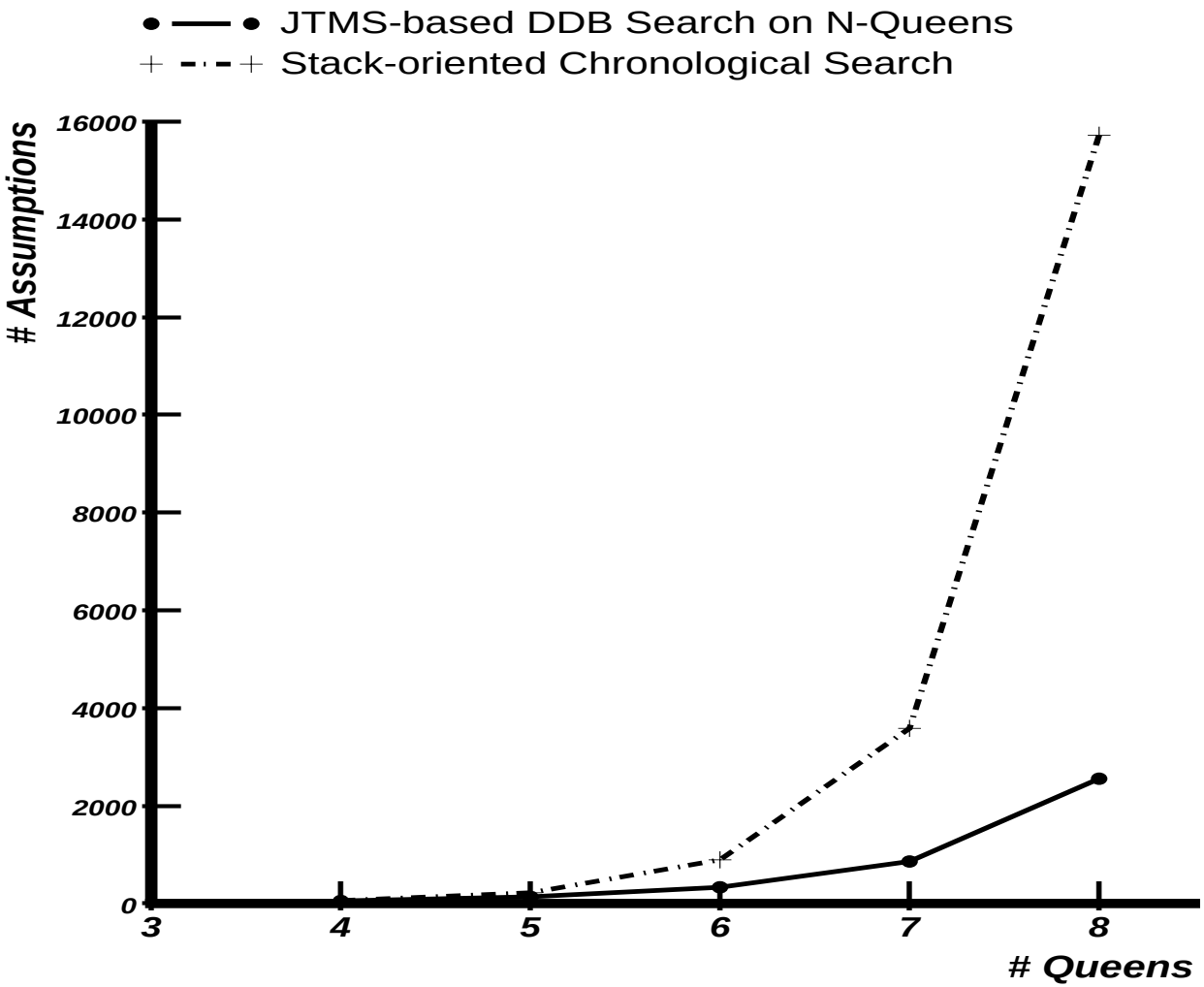
((queen 2 1) (queen 3 2)) is *inconsistent*

$\Rightarrow$ *Record the pair as a nogood*

Comparative run times for N-Queens



Assumptions required for N-Queens



Implications of N-queens with 2 Search Strategies

- Why Chronological Search is faster than DDS ?
  1. Problem Space is well understood $\Rightarrow$ Optimal ordering of choice sets (Last assumption made is always to be retracted.)
  2. Testing a combination of assumptions is cheap.
- DDS requires less assumptions than CS.

  In most real problems:
  1. Overhead of testing assumptions is much larger.
  2. Problem space is not well understood.

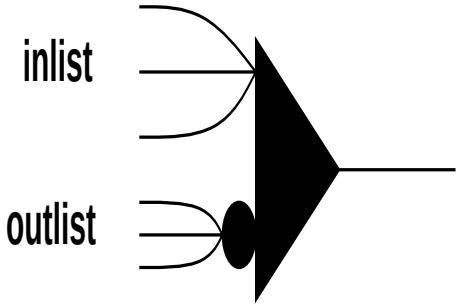
  However, DDS *postpones* a combinatorial explosion.

Doyle's JTMS

Support-List Justification

(SL $<$ consequent $>$ $<$ inlist $>$ $<$ outlist $>$)

A SL J. is valid iff each node in inlist is in, and each node in outlist is out.



Premise	(SL node {} {})
Assumption	(SL node {} {...})
normal node	(SL node {...} {})

Default Reasoning

"Assume x unless there is some evidence to the contrary."

1. **Reiter's formulation:**

$$\frac{a \quad : Mb}{b}$$

2. **Doyle's Encoding**

   Use a non-monotonic justification:

   (SL b ($\mathsf{a}_1 \ldots \mathsf{a}_i$) ($\mathsf{o}_1 \ldots \mathsf{o}_i$))

Odd Loop in Non-monotonic Reasoning

an odd # of justifications leads to the starting node

1. **(SL F () (F))**

2. **(SL A (B) ())**
   (SL B () (A))

JTMS falls into an infinite-loop.

Even Loop in Non-monotonic Reasoning

(SL A () (B))

(SL B () (A))

JTMS can get either A in **and B** out**, or A** out **and B** out**, but not both.**

Nixon Example

$$\frac{\text{REPUBLICAN} : \text{MHAWK}}{\text{HAWK}}$$

$$\frac{\text{QUAKER} : \text{MDOVE}}{\text{DOVE}}$$

$$\text{DOVE} \wedge \text{HAWK} \supset \bot$$

If Nixon is a quaker
and a republican,
what happens?

JTMS can get either a context with HAWK and ¬DOVE, or the other context with DOVE and ¬HAWK, but not both.

Working Example — Exceptions of exceptions

$$\frac{\mathbf{WORKDAY : M\neg EXCUSE}}{\mathbf{WORK}}$$

$$\frac{\mathbf{ILL : M\neg COLD}}{\mathbf{EXCUSE}}$$

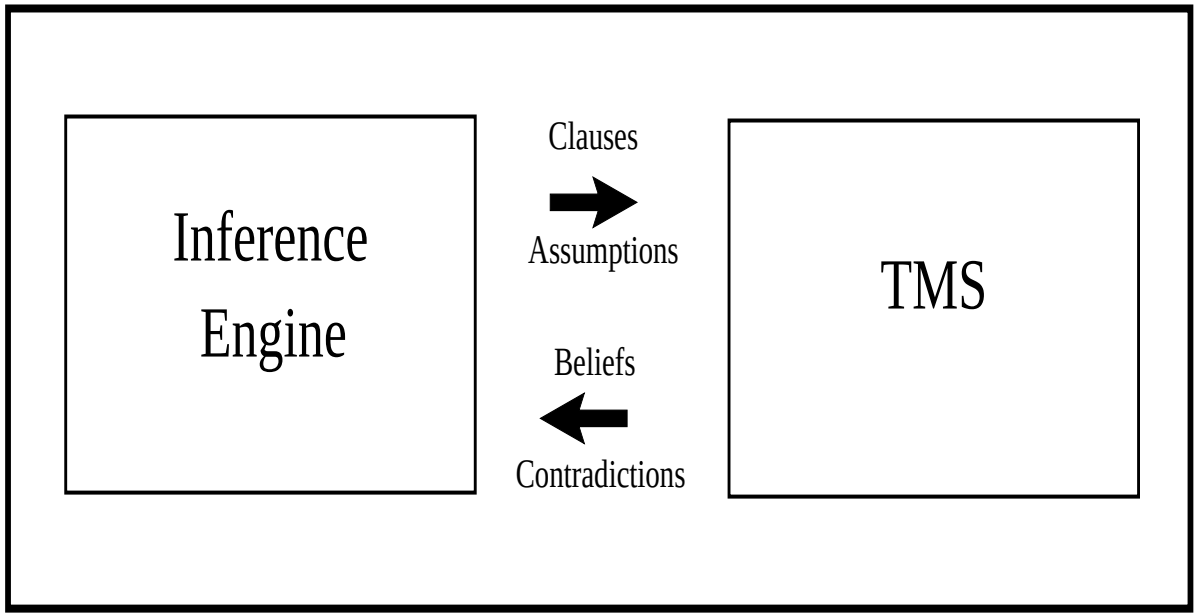
$$\mathbf{COLD \supset ILL}$$

If Prof. Okuno is ill,
what happnes?

JTMS can work well.

LTMS: Logic-based TMS

Change from JTMSs to LTMSs



Justification (Horn and Definite Clauses)

$\Rightarrow$ **Clauses** (Any Propositional Clauses including OR)

Logical Weakness of JTMS

Justification for JTMS:

$x_1 \wedge \cdots \wedge x_n \Rightarrow c$ **where** x_i **and** c **are TMS nodes.**

cannot express:

- *"If x is true, then y is false."*
- *"Either x or y holds."*

Representing Clauses Approximately by Encoding Tricks

$$A \vee B \equiv (\neg A \supset B) \wedge (\neg B \supset A)$$

But in LTMS $A \vee B \vee C$

Redundancies in Encoding with Negations

Introducing a node $\overline{A}$ for $\neg A$ s.t.

$\overline{A}$ **is labeled** :IN **iff** A **is false.**

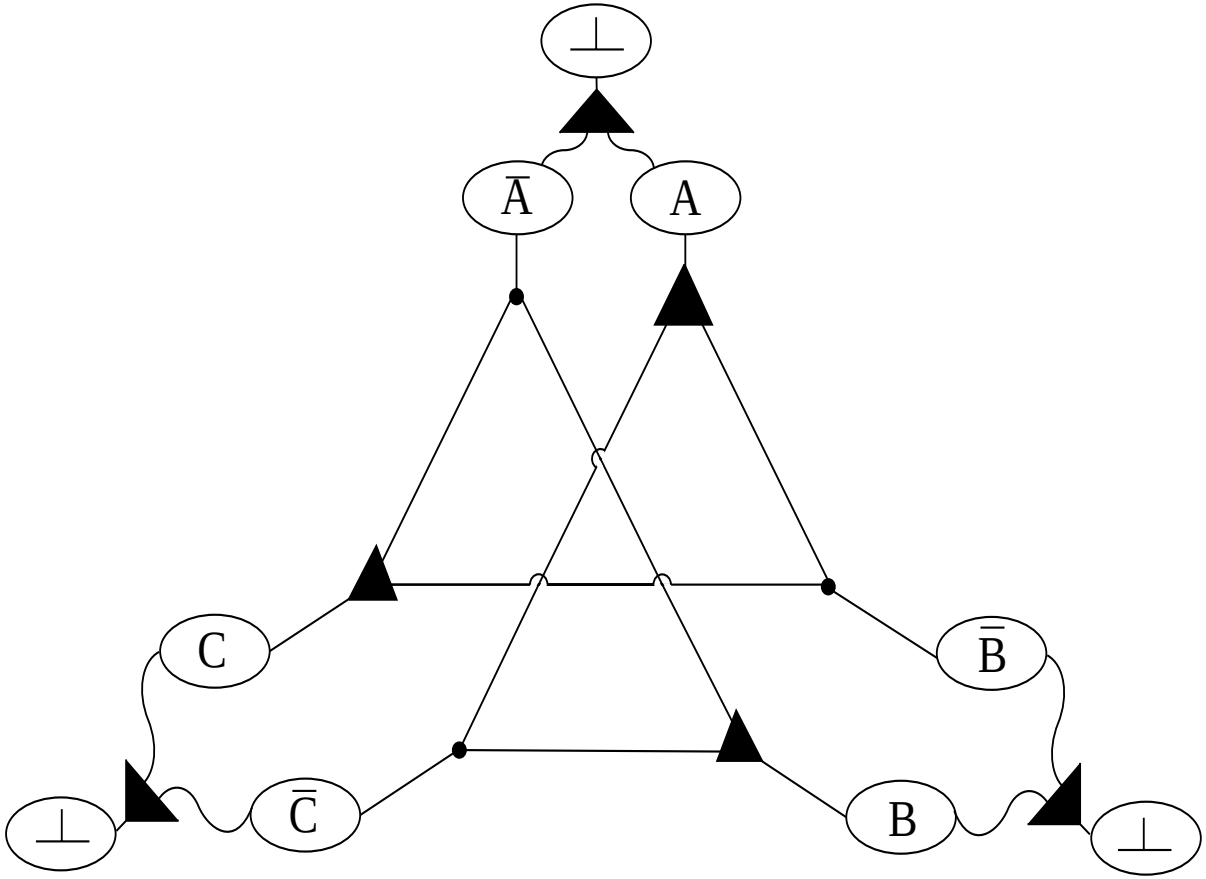
Consider: Encoding $A \vee B \vee C$ **for JTMS**

$$\left.\begin{array}{ll} A \wedge \overline{A} \Rightarrow \bot & \overline{A} \wedge \overline{B} \Rightarrow C \\ B \wedge \overline{B} \Rightarrow \bot & \overline{B} \wedge \overline{C} \Rightarrow A \\ C \wedge \overline{C} \Rightarrow \bot & \overline{C} \wedge \overline{A} \Rightarrow B \end{array}\right\} \begin{array}{l} 6 \text{ justifications} \\ 3 \text{ additional} \\ \text{nodes} \end{array}$$

[Note] Don't forget encoding all clauses:

E.g., for $A \supset B$, $A \Rightarrow B$ **and** $\overline{B} \Rightarrow \overline{A}$

Encoding A ∨ B ∨ C for JTMS



Guiding Backtracking with JTMS

Suppose that $A_1, \ldots A_n$ underlie a contradiction.

If IE enables a set of assumptions that includes $A_1, \ldots A_n$, JTMS signals a contradiction before any more IE operations take place.

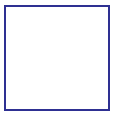
But more processing may be possible.

$A_1, \ldots A_n$ underlie a contradiction.

$$\Longrightarrow \neg A_1 \vee \cdots \vee \neg A_n \qquad \textbf{(called } nogood\textbf{)}$$

This cannot be represented or used in JTMS.

$A_1, \ldots, A_{m-1}, A_{m+1}, \ldots, A_n$ **hold** $\Longrightarrow \neg A_m$ **?**



Special Case for Guiding Backtracking

If search order through assumptions fixed and A_n is the last assumption (as N-Queens problem), the following encoding is enough:

$$A_1 \wedge \cdots \wedge A_{n-1} \Rightarrow \overline{A_n}$$

$$A_n \wedge \overline{A_n} \Rightarrow \bot$$

In N-Queens Problem:

Suppose queens of A_i and A_n capture each other.

$$A_i \Rightarrow \overline{A_n}$$

$$A_n \wedge \overline{A_n} \Rightarrow \bot$$

This is what chronological search does.

Guiding Backtracking with Negation

Suppose: $\left\{ \begin{array}{c} \neg Q \vee \neg A \\ \neg Q \vee \neg B \\ Q \text{ is determined} \\ \text{a choice of } \{A, B, C\} \end{array} \right\}$ **, then** $\square$

How is this done?

- **JTMS Encodes** $\{A, B, C\}$

  # nogoods is large $\Longrightarrow$ **combinatorial explosion**

- **LTMS Represents** $\{A, B, C\}$ **directly.**

Labels in LTMS

$$\begin{cases} C : \{\textbf{clauses supplied by Inference Engine}\} \\ A : \{\textbf{enabled assumption nodes}\} \end{cases}$$

$$\Downarrow$$

Node n is labeled:

$$\begin{cases} \texttt{:TRUE} & \Longleftrightarrow n \textbf{ is derivable from } A \cup C \\ \texttt{:FALSE} & \Longleftrightarrow \neg n \textbf{ is derivable from } A \cup C \\ \texttt{:UNKNOWN} & \textbf{otherwise} \end{cases}$$

Mapping JTMS labels for P and $\overline{P}$ to LTMS labels for P

$\overline{P}$ \ P	:IN	:OUT
:IN	**(Impossible)**	:FALSE
:OUT	:TRUE	:UNKNOWN

LTMS Node Properties

- **Premise — not specified explicitly**
- **Contradiction — no such a notion**
- **Assumption — belief may be changed by IE**

  $\left\{\begin{array}{l} enabled \text{ — IE choosed to be } \texttt{:TRUE} \text{ or } \texttt{:FALSE} \\ retacted \text{ — otherwise. treated as any other node} \end{array}\right.$

- **(Normal) Nodes — otherwise.**

Why LTMS does not require a notion of Contradiction Node?

Consider a JTMS justification: $A \wedge B \Rightarrow \bot$

When A **and** B **are enabled assumptions,**

then JTMS signals IE about a contradiction.

$\Longrightarrow$ **Contradiction Handling (IE) may retract** A **or** B**.**

In LTMS a nogood is added:

$$\neg A \vee \neg B \vee \bot \quad \Longrightarrow \quad \neg A \vee \neg B$$

If A **is enabled to** `:TRUE`**, then LTMS labels** B `:FALSE`**.**

Contradiction Handler is still required in LTMS

Consider A and B were enabled assumptions.

Then $\neg A \vee \neg B$ is added.

$$\Downarrow$$

Contradiction Handler should decide

which to retract, A or B.

Logical Specification for LTMS [1]

S: {propositional symbols}

A: {assumption literals}

C: {IE-supplied Clauses }

$$\begin{cases} x \in A \text{ if assumption } x \text{ is initially labeled } : \texttt{TRUE} \\ \neg x \in A \text{ if assumption } x \text{ is initially labeled } : \texttt{FALSE} \end{cases}$$

Three fundamental tasks of LTMS

1. Provide labels for nodes
2. Detect contradictions
3. Provide explanations for labels

Logical Specification for LTMS [2]

LTMS returns current status of belief for a symbol s:

$$\begin{cases} : \texttt{TRUE} \Leftrightarrow E \subset A \cup C \text{ s.t. } E \text{ is satisfiable} \\ \quad \text{and } s \text{ follows propositionally from } E \\ : \texttt{FALSE} \Leftrightarrow E \subset A \cup C \text{ s.t. } E \text{ is satisfiable} \\ \quad \text{and } \neg s \text{ follows propositionally from } E \\ : \texttt{UNKNOWN} \quad \text{otherwise} \end{cases}$$

- **If $A \cup C$ is satisfiable, set E as $A \cup C$.**
- **Otherwise, LTMS may arbitrarily provide either :TRUE or :FALSE, with corresponding explanation.**

Logical Specification for LTMS [3]

- **Detecting contradictions**

  When $A \cup C$ is unsatisfiable, LTMS signal IE.

  LTMS is still expected to provide either `:TRUE` **or** `:FALSE`**, with corresponding explanation.**

- **Providing Explanations**

  Explanation: at least a logical proof

  using E for label.

Boolean Constraint Propagation (BCP) [1]

logically incomplete, but efficient and sound algorithm

- **Input: initial labeling of assumptions A**

  all remaining symbols are initially labeled `:UNKNOWN`
- **Define the label of literals:**
  - **Label of literal x = label of symbol x**
  - **Label of literal $\neg x$** `:FALSE`, `:TRUE`, `:UNKNOWN` **if label of symbol x is** `:TRUE`, `:FALSE`, `:UNKNOWN`, **respectively.**

Boolean Constraint Propagation (BCP) [2]

Every Clause $C \in \mathcal{C}$ is:

- *Satisfied*: $\exists$ **literal** $\in C$ **is labeled** :TRUE.

  e.g., x :TRUE $\Rightarrow x \vee y$ **is satisfied.**

- *Violated*: $\forall$ **literal** $\in C$ **is labeled** :FALSE.

  e.g., x **and** y :FALSE $\Rightarrow x \vee y$ **is violated.**

- *Unit open*: **One literal is labeled** :UNKNOWN **and the remainder** :FALSE.

  e.g., x :TRUE **and** y :UNKNOWN $\Rightarrow \neg x \vee y$ **is unit open.**

  BCP may label the :UNKNOWN **literal** :TRUE.

- *Non-unit open*: **More than one literal is labeled** :UNKNOWN **and the remainder** :FALSE.

Boolean Constraint Propagation (BCP) Algorithm

S : stack of clauses to examine
V : a set of violated clauses

Algorithm BCP

1. Repeat until S is empty.
2. Pop clause C off of S
3. If C is unit open, compute the label of the
   :UNKNOWN literal l which will satisfy C. Call SET(l).
4. Otherwise, do nothing.

Given: x is labeled :TRUE, and unit open clause $\neg x \vee y$,

$\Longrightarrow$ the label of y is forced to be :TRUE.

BCP Algorithm [2]

Algorithm SET(l)

1. Set label of node of l to make the literal l :TRUE.
2. Push every unit-open clause that contains $\neg l$ on to S.
3. Push $\forall$(voilated clause $\ni \neg l$) on to V.

Algorithm ENABLE-ASSUMPTION(l)

1. Call SET(l)
2. Call BCP
3. If V is not empty, signal IE that $\perp$ occured.

BCP Algorithm [3]

Algorithm ADD-CLAUSE(C)

1. **Add C to $\mathcal{C}$.**
2. **If C is violated, push C on to V and signal IE**
3. **If C is unit open, push C on to S.**
4. **Call BCP.**
5. **If V is not empty, signal IE that $\perp$ coccured.**

$\heartsuit$

BCP is P-$complete$ on # literals in the clausees.

Propositional Satisfiability (3-SAT) is NP-$complete$.

Logical Properties of BCP — its *Incompleteness*

BCP is sound, but is not logically complete.

- *Literal-Incompleteness* : **BCP fails to label a node** `:TRUE`/`:FALSE` **when it should do.**

  e.g., $x \vee y$ **and** $x \vee \neg y$ **when all labels are** `:UNKNOWN`**.**

  $\Longrightarrow$ $\square$

- *Refutation-Incompleteness*: **BCP fails to detect contradictions and so fails to signal IE about** $\bot$**.**

  e.g., $x \vee y$**,** $x \vee \neg y$**,** $\neg x \vee \neg y$ **and** $\neg x \vee y$

  $\Longrightarrow$ **Consistent labeling? Yes** $\square$ **No** $\square$

When to Use BCP-based LTMS vs. JTMS

For Horn clauses, BCP is refutation-complete, and literal-complete for positive literals only.

So, BCP is literal-complete for JTMS definite clauses.

Definite Horn clauses
:TRUE initial labels
$\Longrightarrow$
JTMS is identical to BCP-based LTMS

Computational Complexity

JTMS	BCP-based LTMS
# literals in J	# literals in C

Search to Accomodate Logical Incompleteness

- **No contradiction**
- **Relabeling any** :UNKNOWN **node to be** :TRUE **or** :FALSE **will not cause contradictions.**
- **Some clauses remain non-unit open.**

$\Longrightarrow$ **Invoke Search with adding** *implicate* **of** C **to** C**.**

(Implicate: a clause logically follows from C**.)**

If a contradiction occurs,

$A_1, \ldots, A_n$ **: a set of underlying enabled assumptions**

$\Longrightarrow$ **Add nogood clause** $\neg A_1 \vee \ldots \vee \neg A_n$ **to** C**.**

Example of Search

Consider:

$$\neg a \vee b$$
$$\neg c \vee d$$
$$\neg c \vee e$$
$$\neg b \vee \neg d \vee \neg e$$

All nodes are assumptions labeled initially :UNKNOWN.

1. **Search labels** a :TRUE.
2. **Search labels** c :TRUE, **producing a contradiction. Relabel** c :FALSE, **and record nogood** $\neg a \vee \neg c$.

$$\Downarrow$$

BCP is linear, but this search is *exponential*.

BCP Algorithm Revisited — its Refinement

Each node data structure includes two lists :

- **clauses in which the nodes appears** *positively*
- **clauses in which the nodes appears** *negatively*

Each clause data structure has a counter of # nodes that are potential violators. (0 means violation.)

The potential violators for C**:** $x \vee \neg y$

2 if x **and** y **are :**`UNKNOWN`**.** (x **:**`FALSE`**,** y **:**`TRUE`)

- **1 if** x **becomes labeled :**`FALSE`**.** (C **is not satisfied.**)
- **2 if** x **becomes labeled :**`TRUE`**.** (C **is satisfied.**)

Enabling an Assumption A

1. If A receives label :TRUE, find all clauses in which it appears negatively and decrement their counts. Schedule clauses whose count ≤ 1 for step 3.
2. If A receives label :FALSE, find all clauses in which it appears positively and do the same as the above.
3. If the count of clause is 0, the clause is violated. Contradiction will be signaled.

   If one potential violator $(unit\ open)$, the clause forces a node's label recursively.

Retracting an Assumption A — Phase I

1. If A was labeled :TRUE, find all clauses in which it appears negatively and increment their counts. Schedule clauses whose count > 1 for step 3.
2. If A was labeled :FALSE, find all clauses in which it appears positively and do the same as the above.
3. If the count of clause > 1, the clause whose label has been forced by this clause loses support. If a node has lost support, apply this recursively.

Retracting an Assumption A — Phase II

Look for an alternative support

1. For each node just marked :UNKNOWN, examine all clauses that mention it. If any clause's count is 1, force the node's label and propage as in the case of enabling assumptions.

Adding a Clause C

1. Compute the initial count for C.
2. If it is 0, signals a contradiction.
3. If it is 1, force that node's label and propagate that node's value as in the case of enabling assumptions.

Improving Completeness of TMS :

formula-BCP vs. clausal-BCP

Consider $(x \supset (y \vee z)) \wedge (x \vee y \vee z))$

From y **labeled** :FALSE,
can BCP conclude that z **is labeled** :TRUE**?**

- **formula-BCP: If** x **were labeled** :TRUE, z **would have to be labeled** :TRUE**. If** x **were labeled** :FALSE**, the same holds.** $\Longrightarrow$ z **should be labeled** :TRUE**.**
- **clausal-BCP on** $\neg x \vee y \vee z$ **and** $x \vee y \vee z$
  $\Longrightarrow$ **fail to labels** z :TRUE**.**

Use *Prime Implicates* **to Bridge derived clauses**

Basic Terminology Definition

[Def 1] For every symbol s, s and $\neg s$ is *Compilimentary literals*. **A clause is a disjunction of literals with no literal repeated and no complimentary literals.**

[Def 2] An *implicate* **of a formula or a set of formulae C is a clause entailed by C.**

Consider $(x \supset (y \vee z)) \wedge (x \vee y \vee z))$

By reducing to CNF, $\neg x \vee y \vee z$ **and** $x \vee y \vee z$

$y \vee z$ **is also an implicate** $\Longrightarrow$ C**' =** $\{$**3 clause**$\}$

Clausal-BCP on C' achieves formula-BCP.

Basic Terminology Definition [2]

[Def 3] Clause A *subsumes* clause B if all the literals of A appear in B.

[Th 1] Suppose C is a set of clauses and $C' \subset C$ are the clauses of C not subsumed by any other. BCP on C produces the same labels as BCP on C'.

[Def 4] A *prime implicate* of a formular C is an implcate of C which is not subsumed bu any other implcate of C.

$(x \supset (y \vee z)) \wedge (x \vee y \vee z))$ **has a unique P.I.:** $y \vee z$

Basic Terminology Definition [3]

[Th 2] Given a set of formulae F and the union, I, of the prime implicates of each individual formula of F, then BCP on F produces the same labeling as BCP on I.

[Th 3] Suppose that the set of clauses I is the set of prime implicates of some set of formulae, then BCP on I is logically complete.

ATMS: Assumption-based TMS

1. **Possible States**

   JTMS: **a single consistent state** ATMS: **multiple**

2. **Contradiction Handling — e.g., "$A \wedge B \supset \bot$"**

   JTMS: **often "either A or B"** ATMS: **exactly**

3. **Context Switch or Comparison**

   JTMS: **cumbersome** ATMS: **easy**

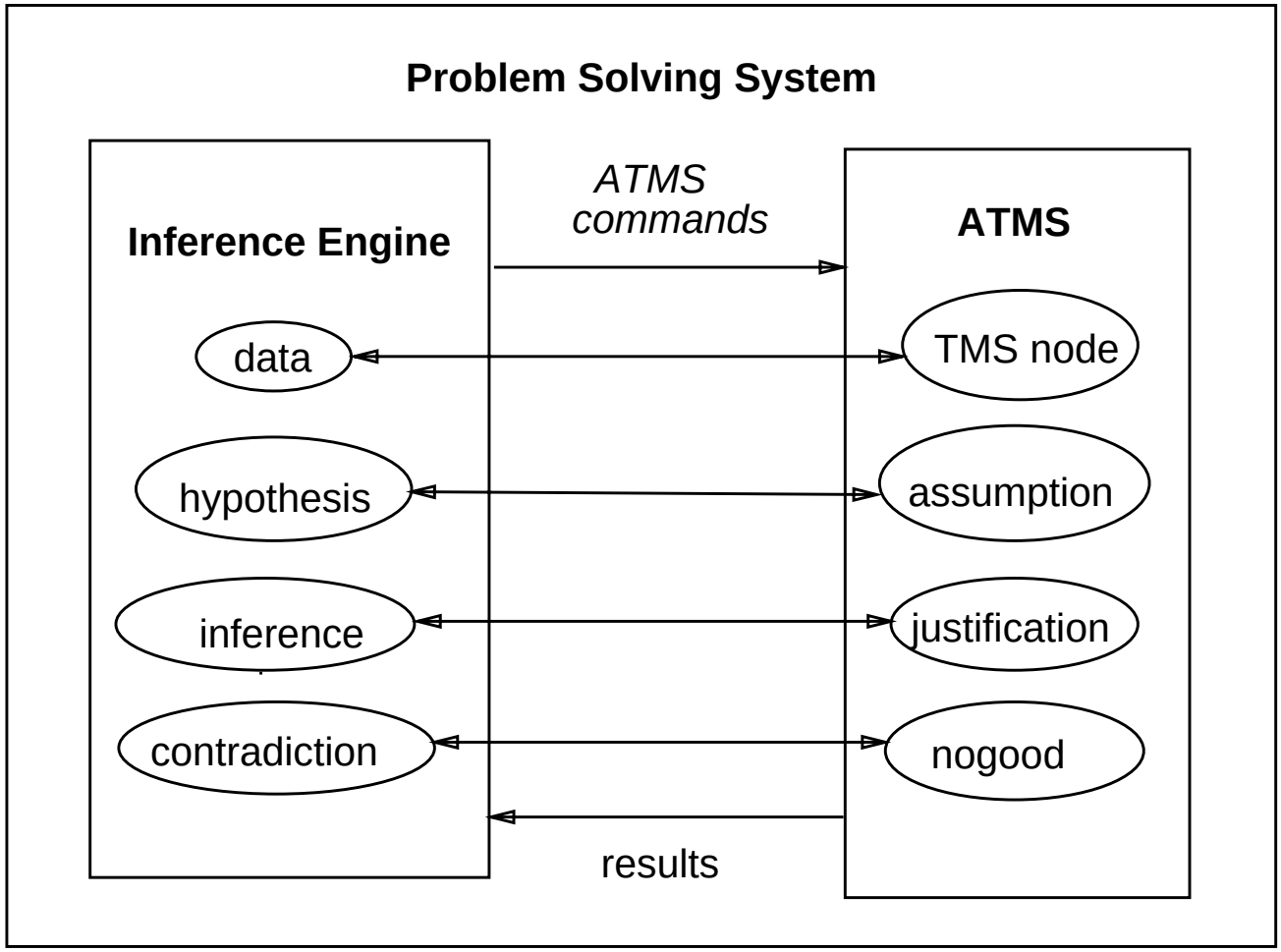
4. **Backtracking —** JTMS: **Yes.** ATMS: **No.**

5. **Redandunt Computations**

   JTMS: **sometimes unavoidable** ATMS: **No**

Departure from *"Single Current Context" Mechanism*

Problem Solving with ATMS



ATMS Node Properties

1. **Premise — Node.Justification has no antecedents.**
2. **Contradiction — Node.Contradictory? is set.**

   A contradictory node becomed believed

   $\not\Longrightarrow$ **JTMS informs IE.**
3. **Assumption — Node.Assumption? is set.**
4. **(Normal) Nodes — otherwise.**

Justification

$$(\langle consequent \rangle \ \langle antecedents \rangle \ \langle informant \rangle)$$

When a Contradiction Node becomes Believed, what happens?

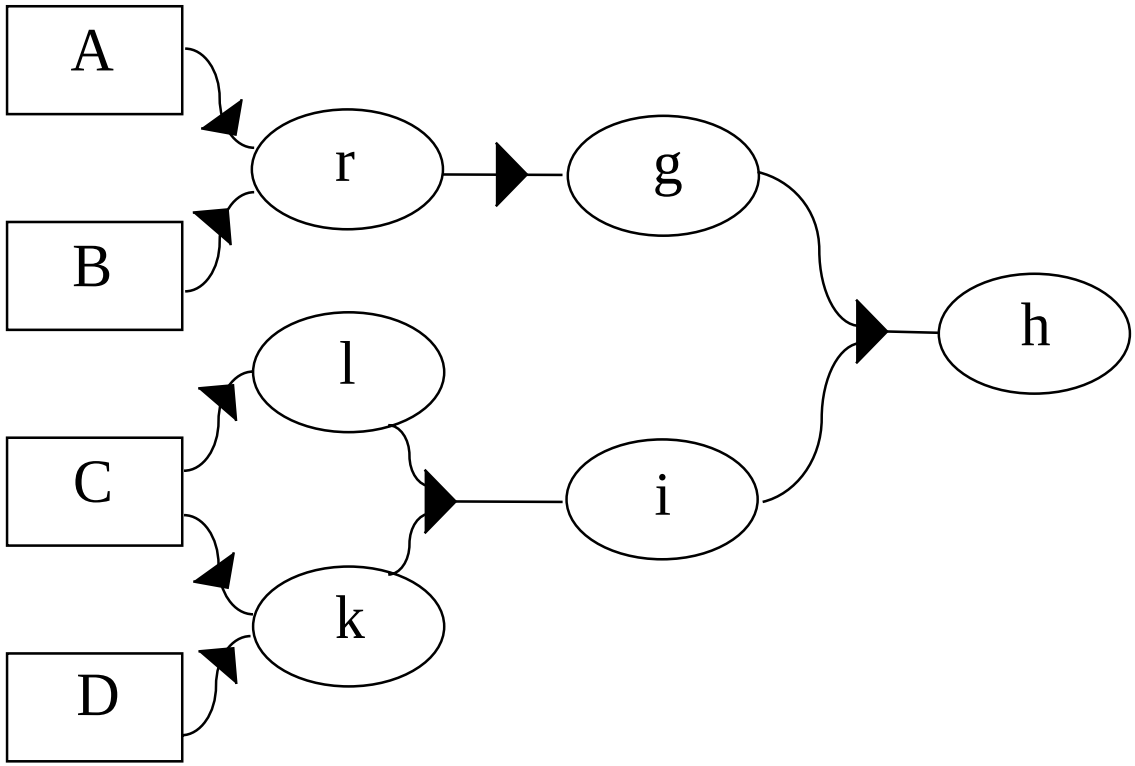
1. **ATMS does not signal a contradiction to IE.**
2. **ATMS ensures that the set of assumptions underlying contradictions will not be considered.**

The more contradictions,
The less assumptions,

$\Longrightarrow$ **the less potential solutions**

$\Longrightarrow$ **the better.**

Multiple Contexts for h (1)



h *follows from*

{A, C}

{A, B, C}

{A, C, D}

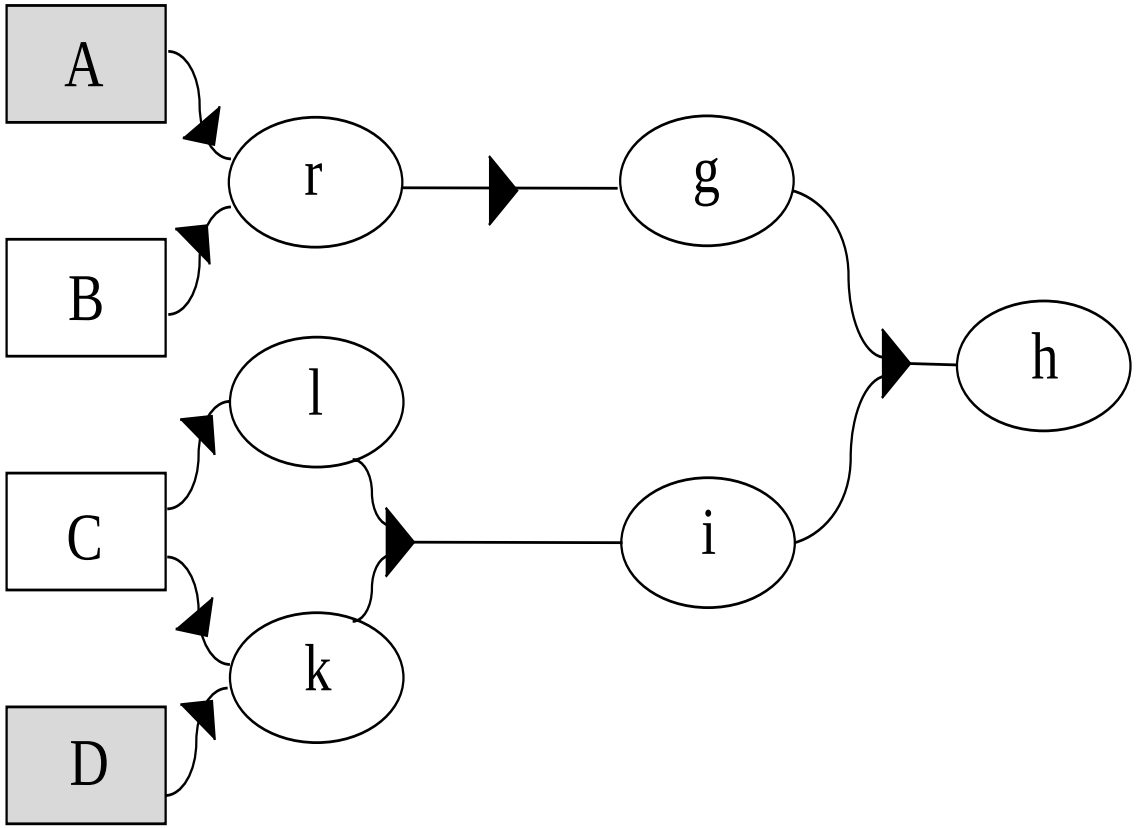
{A, B, C, D}

{B, C}

{B, C, D}

Multiple Context for h (2)

After Assumptions A and D are retracted



h *follows from*

{A, C} ?

{A, B, C} ?

{A, C, D} ?

{A, B, C, D} ?

{B, C} ?

{B, C, D} ?

Terminology for ATMS Label

Complex Data structure — not :IN, :OUT

1. *Environment* — a set of assumptions

   A node holds in an env E if it is labeled :IN in a JTMS when all assumptions of E are enabled.

2. *Nogood* — an env where a contradiction holds.

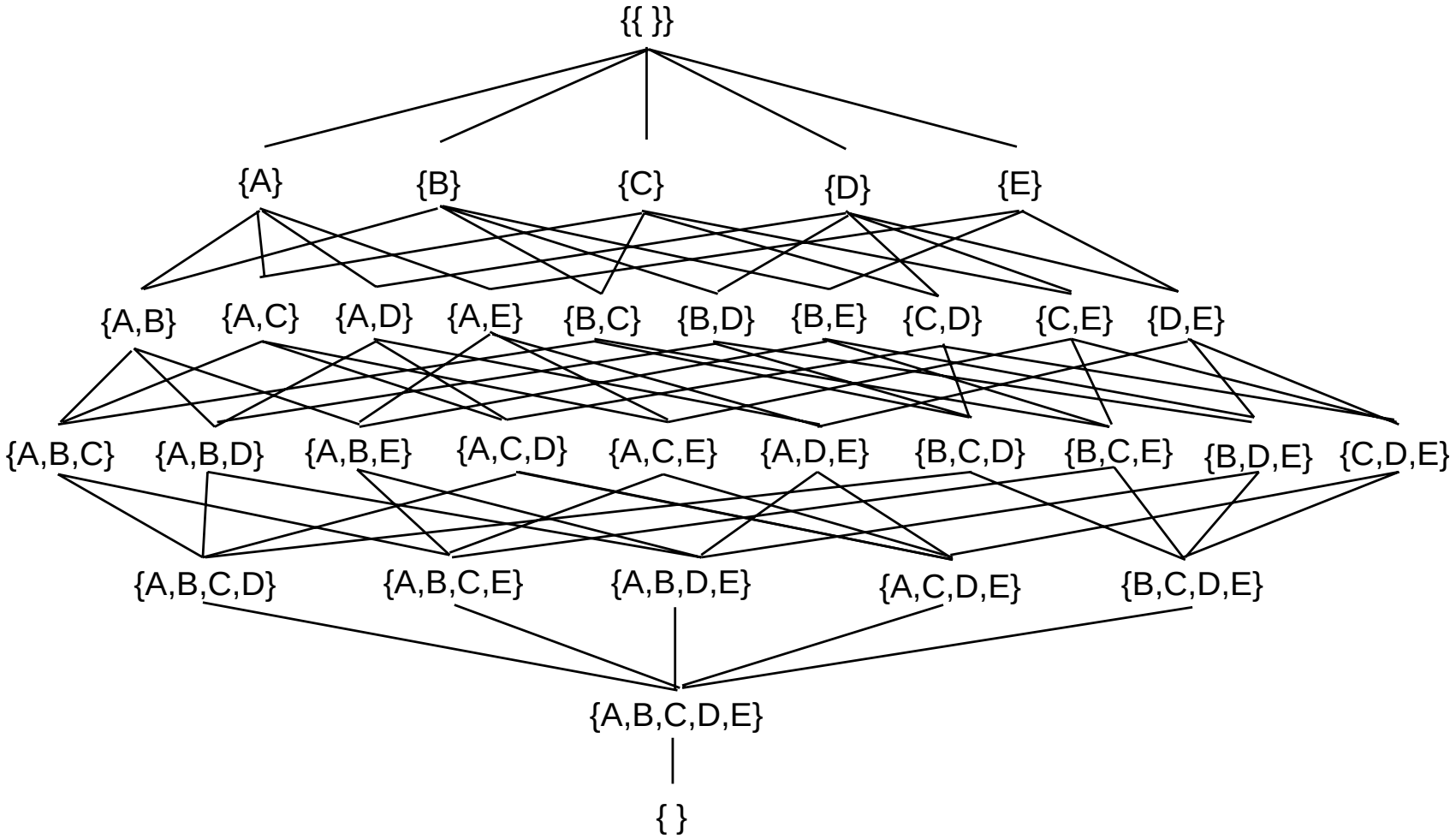
   Otherwise, the environment is *consistent*

3. *Context* of an env —

   the set of nodes which hold in the env.

How to know if a node holds in some environment.

Environment Lattice



ATMS Label

Node: $<datum,\ label>$

Label = a set of environments.

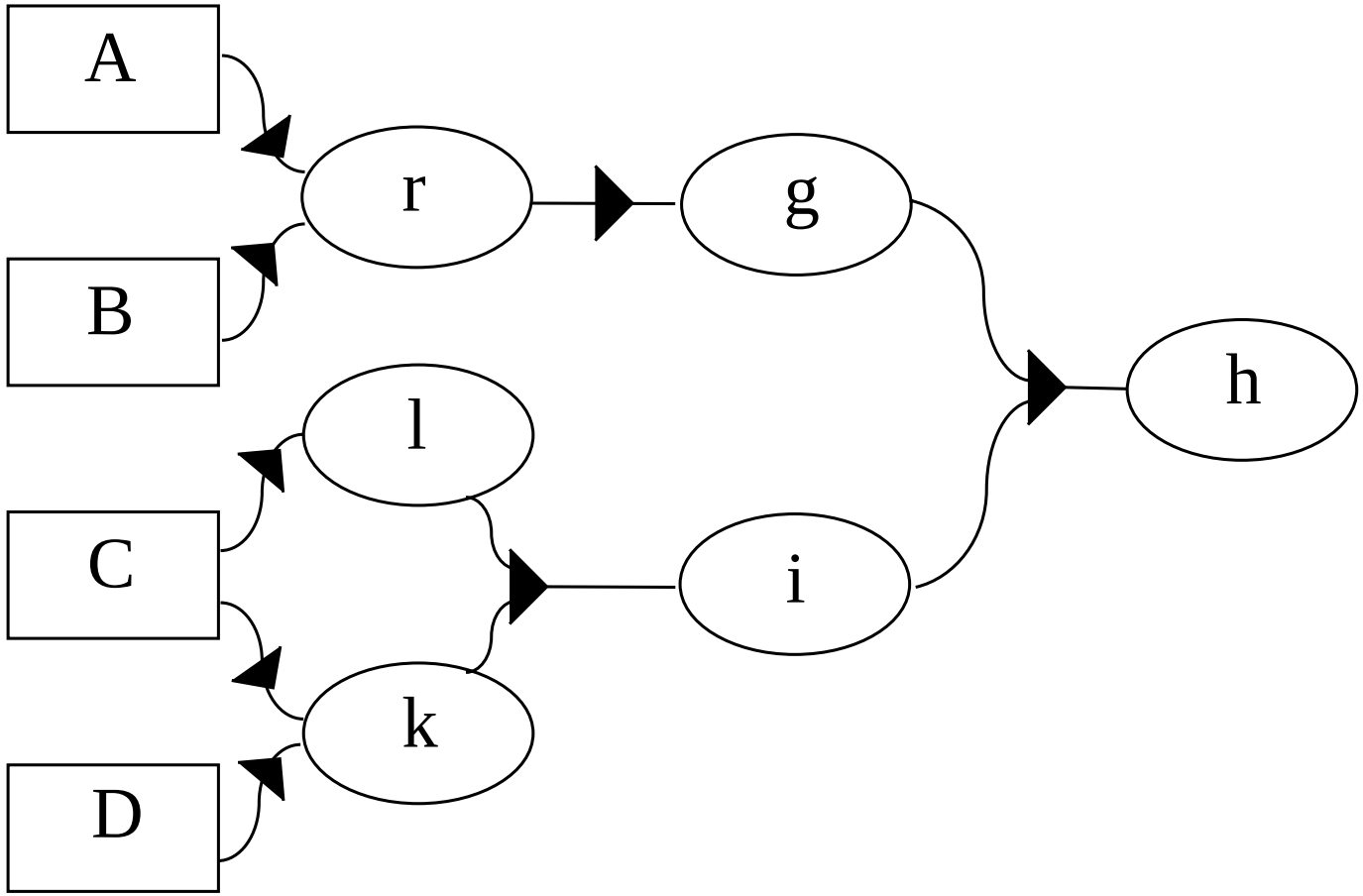
ATMS Node Properties revisited

Premise:	**<p,** $\{\{\ \}\}$**>**
Assumption:	**<A,** $\{\{A\}\}$**>**
Derived Node	**<data,** $\{\{A, B, E\}\ \{C, D\}\}$**>**
Contradiction:	$<\ \bot,\ \{\}>$

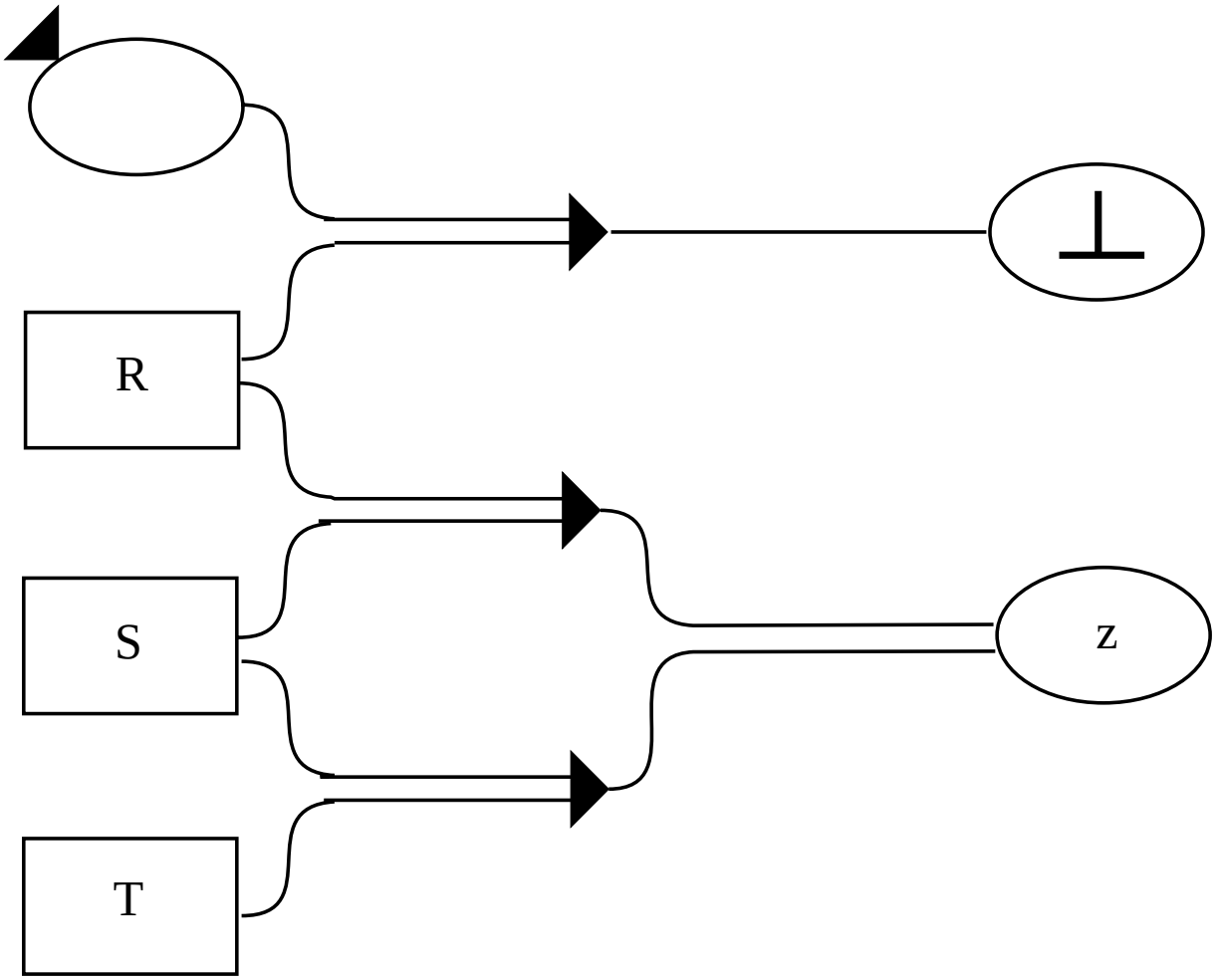
Note

- **<d,** $\{\}>$ **doesn't mean a contradiction.**
- **<d,** $\{\}>$ **doesn't mean** $\neg$ **d holds in** $\forall$**env.**

Exercise: Label all nodes including assumptions (1)



Exercise Label all nodes including assumptions (2)



Logical Specification for ATMS

S: {propositional symbols}
A: {assumption literals} s.t. $A \subset S$
C: {IE-supplied Clauses }

every contradiction node n. $\Longrightarrow$ a unit clause $\neg n$.

An environment E : $E \subset A$

n holds in E if $n \overset{\text{propositionally}}{\Longleftarrow} E$ with C.

Nogood N is an env of assumption literals s.t.

an empty caluse ($\bot$) $\overset{\text{propositionally}}{\Longleftarrow} N$ with C.

A nogood N is $minimal$ if $\forall E \subset N$, E is not nogood.

ATMS Label Properties

Node n has the *label*, a set of envs $\{E_1, \ldots, E_k\}$:

- **[Soundness] n holds in each E_i.**
- **[Consistency] $\bot$ cannot be derived from any E_i with C.**
- **[Completeness] Every consistent env E in which n holds is a superset of some E_i.**
- **[Minimality] No E_i is a proper set of any other.**

n holds in $E \Longleftrightarrow E$ is a superset of some E_i.

ATMS Algorithms (Rough Sketch)

L_{ik} : label of ith node of kth justification for node n

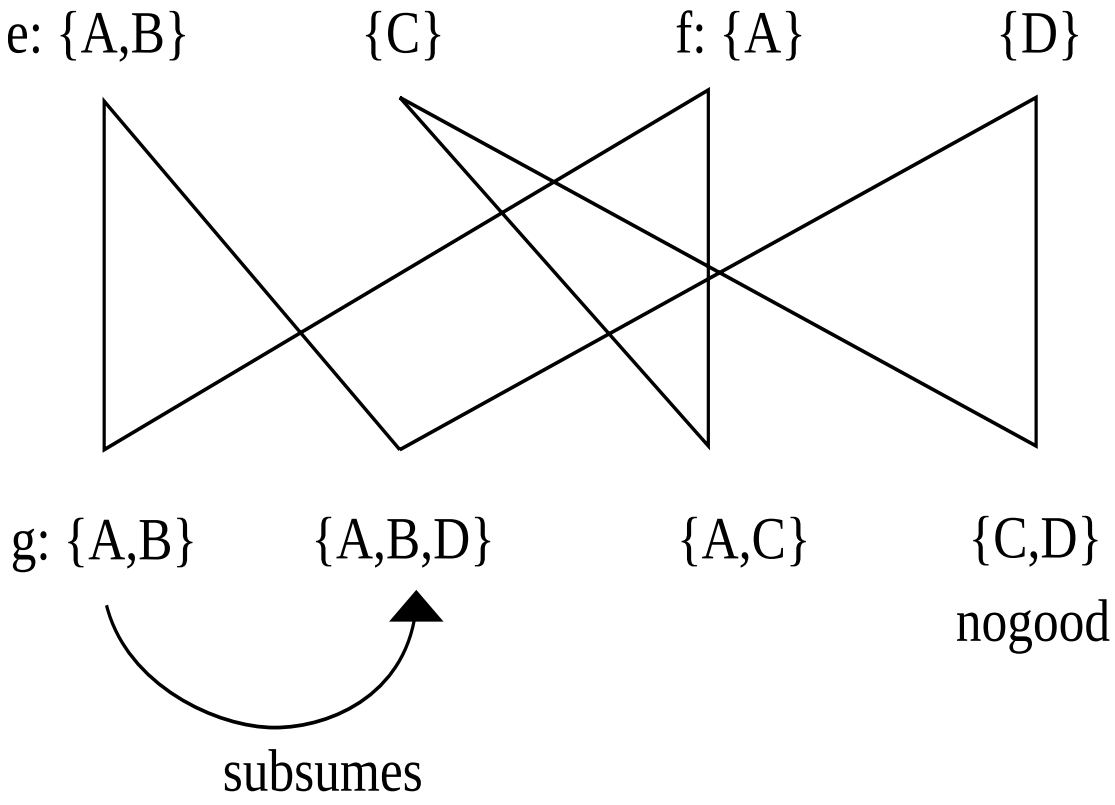
kth justification $(n \ (n_1, .., n_i, ..) \ \langle informant \rangle)$

1. Compute a tentative label $L' = \{\bigcup_i e_i | e_i \in L_{ik}\}$
2. Remove all nogoods and supersets of others L'.
3. If label has not changed, then return.
4. If n is contradiction node,
   (a) Mark all envs of L' nogood.
   (b) Remove all new nogoods from all node labels.
5. Otherwise, recursively update all n's consequents

ATMS Algorithms (Illusitrated)

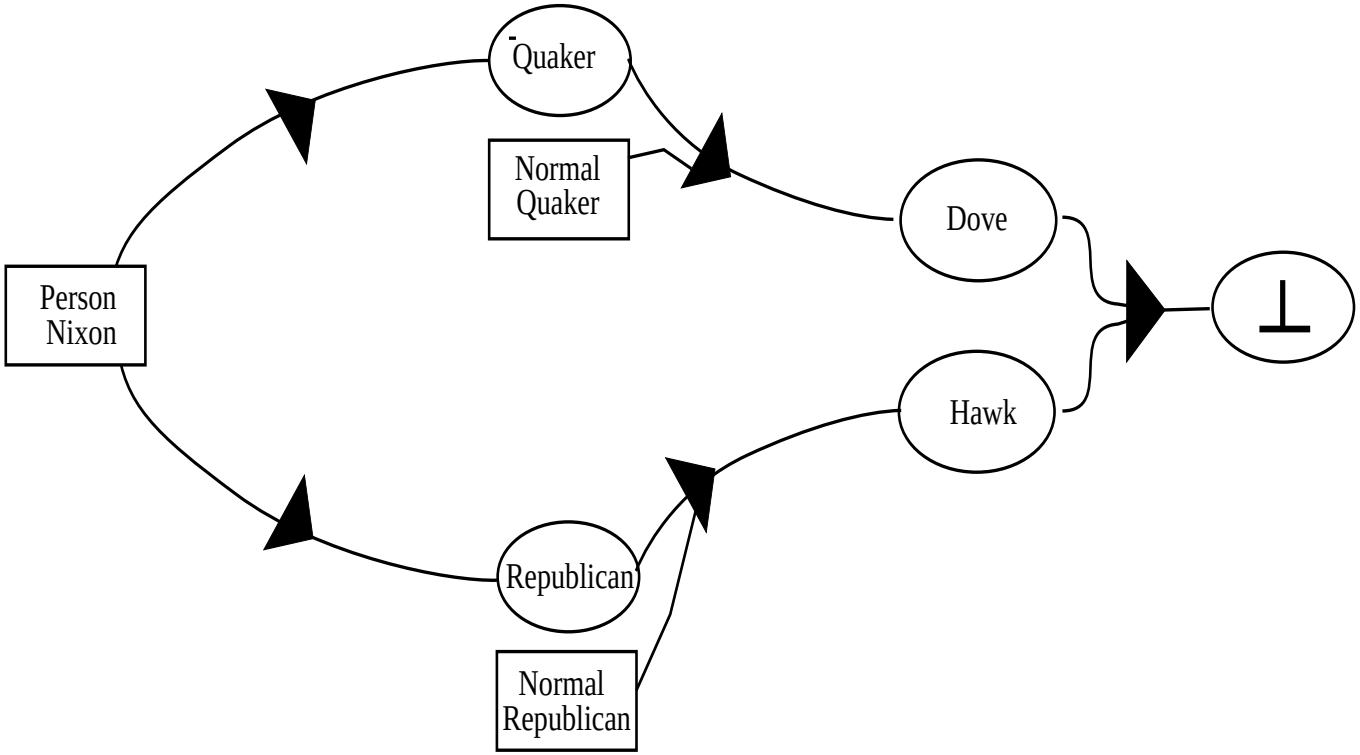
Given $e \wedge f \Rightarrow g$, Compute g's label

$< e, \{\{A, B\}\{C\}\} >, < f, \{\{A\}\{D\}\} >, nogood\{C, D\}$



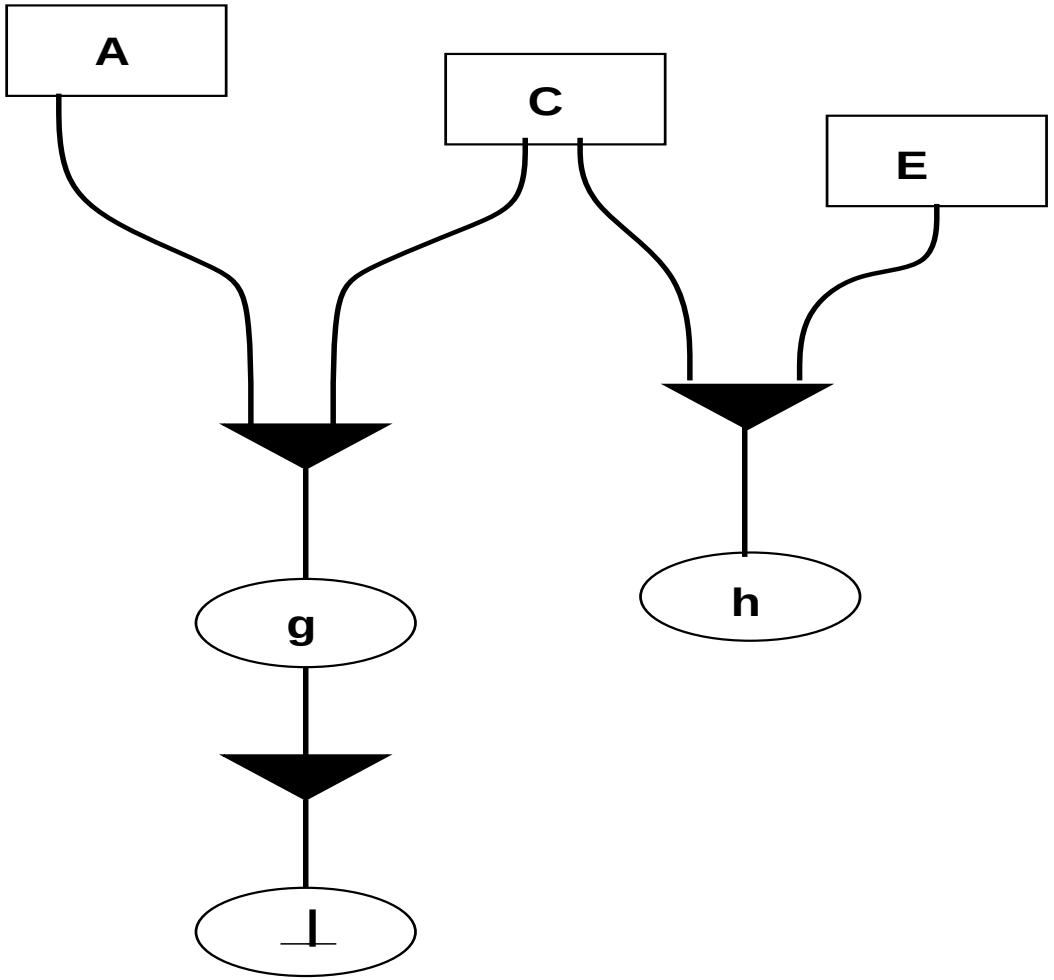
Default Reasoning with ATMS

Treat assumptions as Defaults



$nogood\{Person\text{-}Nixon, Normal\text{-}Quaker, Normal\text{-}Republican\} \Longrightarrow$
$\{Person\text{-}Nixon, Normal\text{-}Quaker\}, \{Person\text{-}Nixon, Normal\text{-}Republican\},$
$\{Normal\text{-}Quaker, Normal\text{-}Republican\}$

Dependency-Network for the Simple Example



Implementation Techiques

- **Environment $\Longleftrightarrow$ Bit vector**
- **Assumption $\Longleftrightarrow$ Unique position in Bit vector**

$E_1 \cup E_2 \Longrightarrow$ **bitor of** V_{E_1}, V_{E_2}

E_1 **is a superset of** $E_2 \Longrightarrow$ **bitand of** $\neg V_{E_1}, V_{E_2} = 0$

# of assumptions in $E \Longrightarrow$ **bitcount of** V_E
(called $dimension$**)**

binary nogood (env) — nogood (env) of two ass

n**-ary nogood (env) — nogood (env) of** n **ass**

Implementation Techiques for BITCOUNT

Usually BITCOUNT in Lisp is very slow.

- **Scan every bit**
- **Divide a set of bytes, and add byte-wise bit count by consulting bit-weight table for byte**

$\Longrightarrow$ **10 times speed-up on TAO/ELIS system.**

Implementation Techniques for Nogood Check

- Assumption has a bitvector consisting of opponent assumptions of binary nogood.

  If a single nogood, the vector is -1.
- Assumption has n which is the least number of n-ary nogoods containing it.
- Environment has its dimension.
- Environment Hash Table
- Non-Nogood Env Table indexed by n-ary.
- Nogood Table for n-ary nogoods ($n > 2$)

N-Queens by Label Update

1. Make assumptions $Queen_{i,j}$ for each position of $n \times n$ board.
2. Make nogoods for capturing Position pair on different rows.
3. Create nodes for 1st-row Queens $Pos_{i,1}$ and Justify it with its position: $Queen_{i,1} \Rightarrow Pos_{i,1}$.
4. Repeat for $2 \leq k \leq n$,

   $Pos_{i,k}, Queen_{j,k-1} \Rightarrow Pos_{i,k}$
5. Gather labels of $Queen_{i,n}$ for n. $\Longrightarrow$ solutions.

N-Queens by Label Update

```
(defun n-queens (n &aux goal goals last-goals classes
           class classes-backup assumption solutions )
  (setq classes (make-class n))    ; 仮定の作成
  (detect-capturing-pair classes) ; nogood の作成
  (setq classes-backup classes)
  (dotimes (i n)
    (setq goals nil)
    (setq class (pop classes-backup))
    (dotimes (j n)
      (setq assumption (pop class))
      (setq goal (tms-create-node *atms* (list 'queen i j)))
      (if (null last-goals)              ; 第 1 行目か
          (justify-node 'first-row goal (list assumption))
          (dolist (previous-goal last-goals)
            (justify-node 'compose goal
                          (list previous-goal assumption)) ))
      (push goal goals) )
    (setq last-goals goals) )
  (setq solutions
      (mapcan #'(lambda (x) (copy-tree (tms-node-label x))) goals))
  (length solutions) )                ; 最終行のラベル答が
```

N-Queens by Label Update

```
(defun make-class (n &aux node class classes)
  (dotimes (row n)
    (setq class nil)
    (dotimes (column n)
      (push (setq node (tms-create-node *atms* `(Queen ,row ,column)))
            class )
      (assume-node node) )
    (push (nreverse class) classes) )
  (nreverse classes) )

(defun detect-capturing-pair (classes)
   (do ((class1 (pop classes) (pop classes)))
      ((null classes))
    (dolist (node1 class1)
      (dolist (class2 classes)
        (dolist (node2 class2)
          (if (queens-captured? (node-datum node1) (node-datum node2))
              (nogood-nodes (list node1 node2)) ))))))

(defun queens-captured? (q1 q2)
  (or (= (cadr q1) (cadr q2))
      (= (abs (- (cadr q1) (cadr q2))) (abs (- (caddr q1) (caddr q2))))))
```

N-Queens by Interpretation Construction

1. **Make a set of queens of the same row a choice set**

   $$\{\Gamma_{i,j} | 1 \leq j \leq n\}$$

2. **Construct Interpretations on the choice sets**

   `(interpretations`

   $\{$`k-th choice set`$| 1 \leq k \leq n\}$ `)`

3. **Interpretations $\Longrightarrow$ Solutions**

N-Queens by Interpretation Construction

```
(defun n-queens-by-IC (n &aux classes solutions)
  (setq classes (make-class n))    ; choice set
  (detect-capturing-pair classes) ; nogood
  (setq solutions                  ; 解釈構築
        (interpretations *atms* classes) )
  (length solutions) )
```

Organizing ATMS-based Problem Solver

- *Many-Worlds* **Strategy: Work in all consistent contexts at once, seek possible solutions.**

  Node is `:IN` **if the label is non-empty, and**

  `:OUT` **if the label is empty.**

- *Focused* **Strategy: Work in single context (or small number of contexts) at a time to find a good solution. Switch contexts opportunistically.**

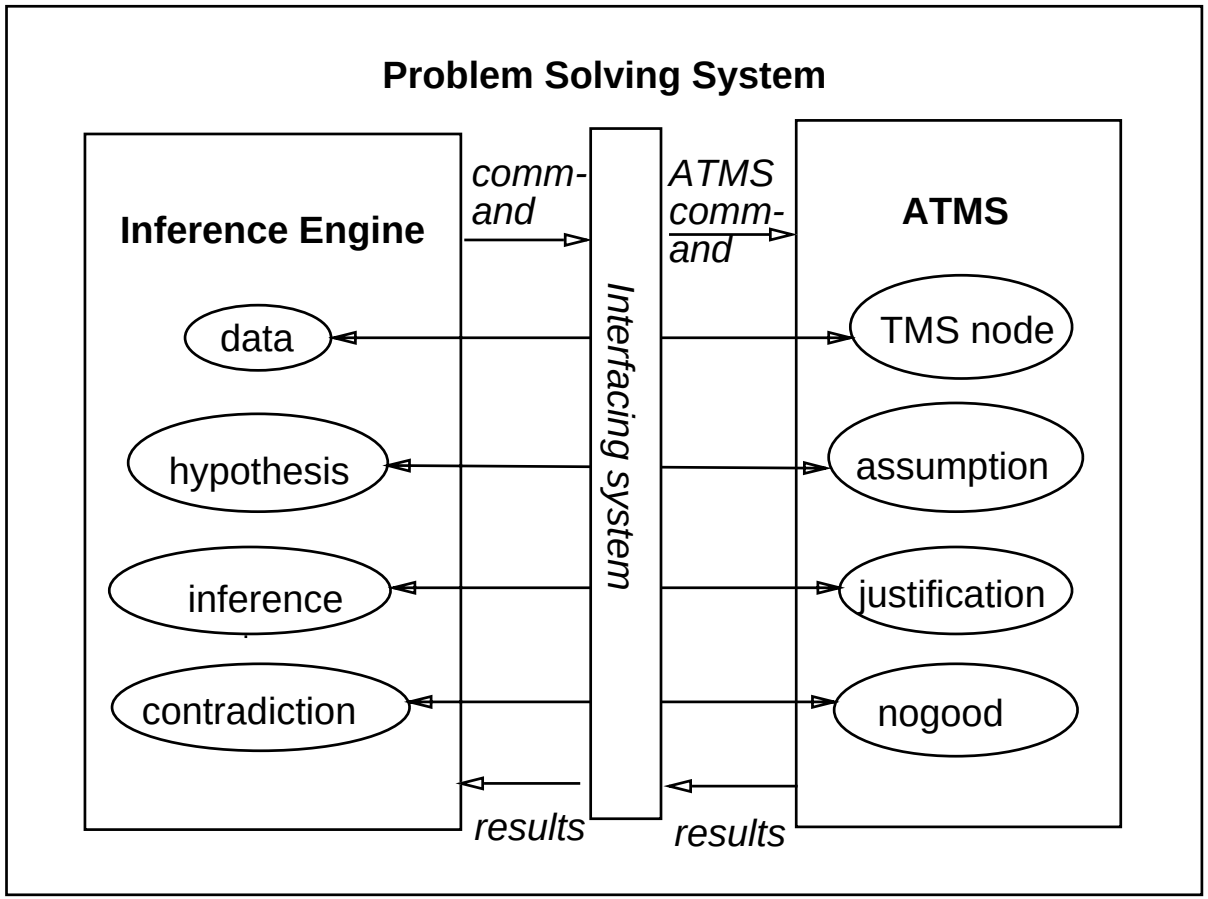
  The environment for context is called *focus*.

  Node is `:IN` **if it is implied by the focus, and**

  `:OUT` **otherwise.**

ATMS の拡張

disjunctive normal form で表現された正当化が扱えるように ATMS を拡張.



Disjunction のコーディング

choose$\{C_1, C_2, \ldots\}$

- Hyperresolution ルールを導入
- 仮定の否定を導入 — NATMS

disjunction を扱うための 6 つの hyperresolution

A : 仮定 Γ_A “<n, ラベル >”: ノード n のラベル

choose$\{A\}$: 恒真

nogood$\{A\}$: 偽

Hyperresolution ルール

ルール H1

仮定が恒真ならば, nogood から抜く.

$$\frac{\text{choose}\{A\} \quad \text{nogood}[\{A\} \cup \alpha]}{\text{nogood}[\alpha]}$$

ルール H2

仮定が恒真ならば, ラベルから抜く.

$$\frac{\text{choose}\{A\} \quad < n, \{\{A\} \cup \alpha\} \cup \beta >}{< n, \{\alpha\} \cup \beta >}$$

ルール H3

仮定が偽ならば, disjunction から抜く.

$$\frac{\text{nogood}\{A\} \quad \text{choose}\{A, A_1, A_2, \ldots\}}{\text{choose}\{A_1, A_2, \ldots\}}$$

ルール H4

2 元 disjunction と negative 節から,

新たな nogood を生成.

$$\frac{\text{choose}\{A, B\} \quad \text{nogood}[\{A\} \cup \alpha] \text{ where } B \notin \alpha}{\alpha \Rightarrow B}$$

Hyperresolution ルール (続き)

ルール H5 新たな nogood か disjunction が与えられれば, 新たな nogood を生成.

$$\frac{\text{choose}\{A_1, A_2, \ldots\} \quad \text{nogood } \alpha_i \text{ where } A_i \in \alpha_i \text{ and } A_{j\neq i} \notin \alpha_i \text{ for all } i}{\text{nogood} \cup_i [\alpha_i - \{A_i\}]}$$

ルール H6 ラベルが変化するか, 新たな nogood か disjunction が見つかるとラベルを簡単化.

$$\frac{\text{choose}\{A_1, A_2, \ldots\} \quad < \beta, \lambda > \quad \text{nogood}[A_i \cup \alpha_i] \text{ or } \{A_i\} \cup \alpha \in \lambda \text{ and } A_{j\neq i} \notin \alpha_i \text{ for } \forall i}{< \beta, \{\cup_i \alpha_i\} \cup \lambda^* >}$$

ここで, λ^* は λ から $\cup\alpha_i$ の上位集合をすべて除いたもの.

Negated assumption ATMS — 仮定の否定を導入

NATMS の主たる目的: hyperresolution の代用

仮定の否定は, 仮定ではなく, 普通のノード

disjunction 構文 choose{A, B, C} のコーディング:

$$\neg A, \neg B, \neg C \Rightarrow \bot$$

[注意] k 個の否定節 ($\neg A \vee \neg B \vee \neg C$) は k 個の含意

($A \wedge B \rightarrow \neg C$ 等) と論理的に等価.

ラベルの無矛盾性の達成のためのルール

$$\frac{\mathrm{nogood}\{A, A_1, \ldots, A_k\}}{A_1, \ldots, A_k \Rightarrow \neg A}$$

しかし, 完全性は成立しない.

非単調推論の取り扱い

1. choose, control によるコーディング [de Kleer]
2. 非単調 ATMS — **非単調正当化** を導入

   $(a), (b) \Rightarrow c$ (a：IN リスト b：OUT リスト)

   $a_1, a_2, \cdots, OUT(b_1), OUT(b_2), \cdots \Rightarrow c$

   と書く

例 「Tweety が鳥であり，『Tweety が飛べる』が無矛盾である限り，Tweety は飛べる」という命題

- a：「Tweety が鳥である」という命題.
- n：「Tweety は飛べる」という命題.

ATMS での非単調推論のコーディング

仮定 Γ_A は A で, ノード γ_a は a で表現

- 基本 ATMS N' は n の反例がないという仮定

  $$a, N' \Rightarrow n$$

  $$\mathrm{ignore}\{N'\}$$

  ignore で仮定 N' だけで構成される文脈は意味がないことを記述し, 余分な探索を防止する.

- 非単調 ATMS コーディングは以下の通り.

  $$a, OUT(\neg n) \Rightarrow n$$

「例外の例外」であるニクソン問題のコーディング

default logic で表現すると:

$$\frac{Republican \quad : \quad \mathrm{M}\ Hawk}{Hawk}$$

$$\frac{Quaker \quad : \quad \mathrm{M}\ Dove}{Dove}$$

$$Dove \& Hawk \supset False$$

de Kleer のアプローチ:

仮定: $N_1', \ N_2',$

$\{republican, N_1'\} \Rightarrow hawk$

$\{quaker, N_2'\} \Rightarrow dove$

nogood$\{N_1', N_2'\}$

ignore$\{N_1'\}$, **ignore**$\{N_2'\}$

非単調 ATMS によるコーディング

仮定: $OUT(\neg dove),\ OUT(\neg hawk)$

$\{republican, OUT(\neg hawk)\} \Rightarrow hawk$

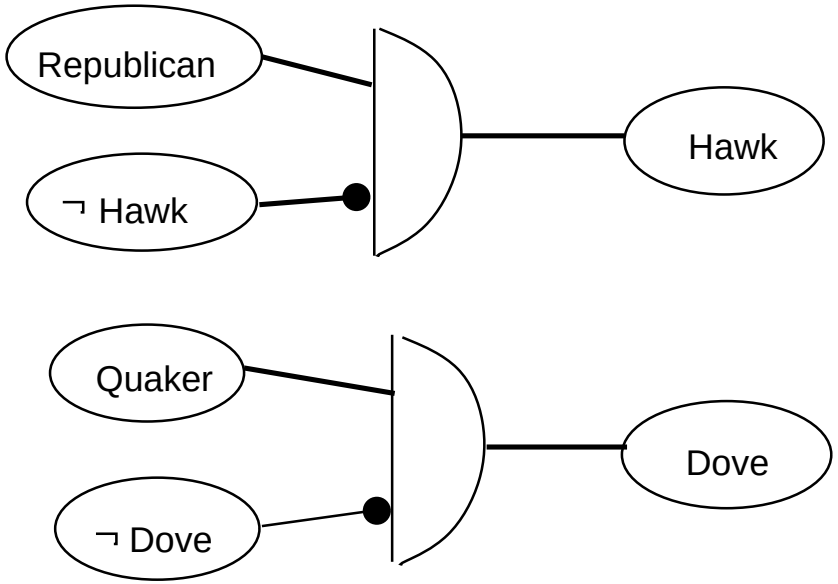
$\{quaker, OUT(\neg dove)\} \Rightarrow dove$

nogood $\{OUT(\neg dove), OUT(\neg hawk)\}$

解釈構築により

$\{OUT(\neg dove)\}$ **と**

$\{OUT(\neg hawk)\}$



ATMS 使用戦略による探索空間の制限

1. **INTERN 戦略** — どれか１つの前件が IN になると，すぐにそのルールを実行．ルール実行が軽く，全空間を探索し，全解を求めるときに有効.
2. **IN 戦略** — すべての前件が矛盾せずに IN になるときまで，ルールの実行を遅延.
3. **ADDB** (*Assumption-based Dependency-Directed Backtracking*) — コントロール構文で実行すべきルールの候補を記述し，IN 戦略を用いてルールを実行．IN 戦略よりは効率がよい．無矛盾なルールはすべて実行.
4. **Implied-By 戦略** — 前件の和集合が現在の focus environment で導き出されるときにかぎり，ルールが実行．focus environment が nogood になると推論エンジンに通知する.

ATMS の応用分野

- 非単調的な信念の翻意 — 頻繁に更新されるデータベース間での無矛盾性のチェックや, 曖昧なデータ, 不完全なデータを用いて推論を行うために，データの信念を真偽値マーキングとして ATMS で管理.
- 論理的な依存関係を利用した探索制御 — 横型探索で，推論結果を保持し，失敗情報を貯えることによって，制約条件を規定し，同じ計算を繰り返さずに最終ゴールおよび部分ゴールへの最適なパスを求めるのに ATMS を使用.
- 多重文脈推論における無矛盾性の管理 — 複数の文脈を同時に推論するときに，各文脈でデータの無矛盾性を保証するために ATMS を使用.

ATMS の応用分野

1. 多重文脈推論: QPE (Qualitative Physics Engine) (Univ. of Northwestern)
2. 多重世界データベース: ATMS の多重文脈推論だけでは, 世界間の関係が記述できる多重世界の機能はなし.
3. 依存関係に基づいた後戻り, 非単調推論: 論理型プログラミング (Oregon State Univ.) circumscription theorem prover (Stanford Univ.), 並列定理証明システム
4. 論理的な依存関係を利用した探索制御: 画像理解, 音声理解 (阪大)
5. 自然言語処理 (Yale Univ., Linköing Univ., NTT)
6. 項書換えシステム (東北大)

ATMS に関連する計算量

disjunction-free な Default ルール n 個 $\quad \dfrac{\alpha : \beta \wedge \gamma}{\beta}$

1. **存在問題:**

   極大無矛盾集合 ($extension$) の存在するか.

2. **メンバーシップ問題 (ゴール指向推論):**

   与えられた命題 (リテラル) が成立する extension があるか.

3. **含意 (entailment) 問題 (スケプティカル推論):**

   与えられた命題 (リテラル) がすべての extension で成立するか.

ATMS に関連する計算量

1. **存在問題:**

   α **が正の単項で** β **が単項である disjunction-free な部分クラス (**$unary$**) は, NP 困難 (**$NP\text{-}hard$**), それより制限の強い部分クラスは** $O(n^2)$**.**

2. **メンバーシップ問題:**

   Horn 節か ordered unary の部分クラスは $O(n)$ **で, それ以上のクラスが NP 困難.**

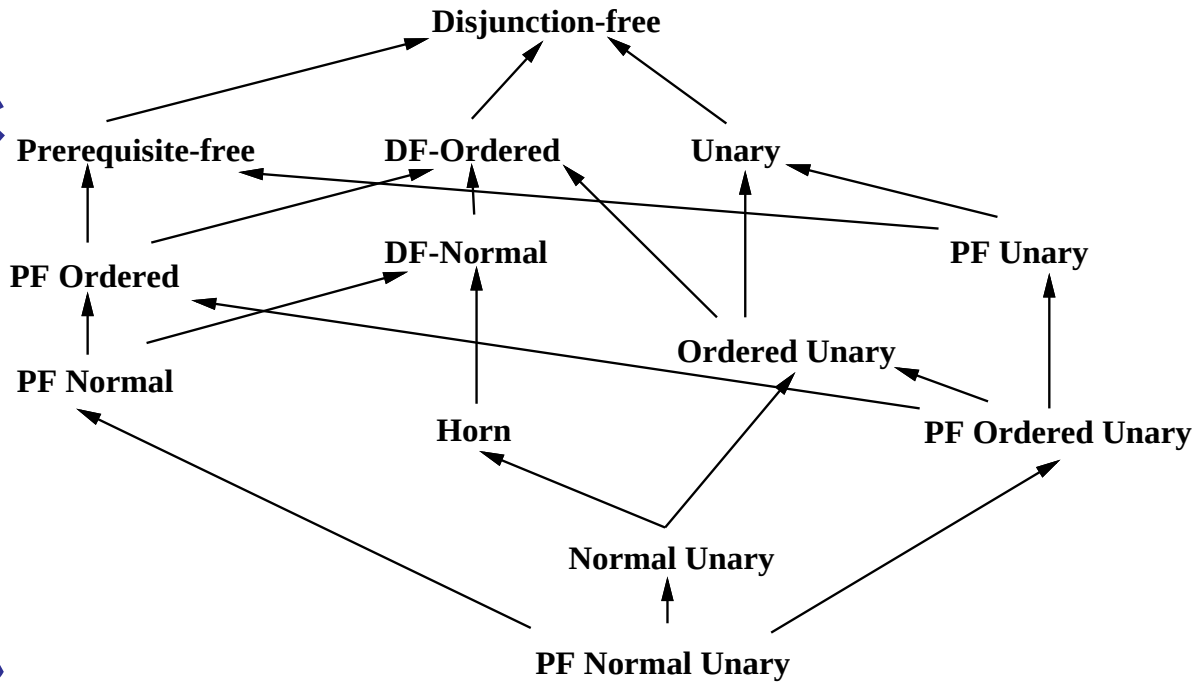
3. **含意 (entailment) 問題:**

   γ **がない unary な部分クラスは** $O(n^3)$ **であり, それより弱い条件の部分クラスは co-NP 困難である.**

メンバーシップ問題の計算量 [Stillman, AAAI-90]

メンバーシップ問題について，命題論理を制限．default ルールとの組合せに対する計算量を調べる．

例．Horn 節命題論理は線形時間で decidable．これにいかなる default ルールを使用してもメンバーシップ問題は NP 完全．



ATMS 問題点 — ATMS の記述能力不足 —

- 問題解決レベルでは，一般の論理関係で表現
- ATMS の正当化は，ホーン節だけ

従来の解決策

1. or，not を choose 述語を用いてエンコード

   ハイパーレゾリューションによる完全性を保証.

   しかし，ハイパーレゾリューションの処理は重い.
2. エンコーディングによる冗長計算

   QPE 同じ解釈構築，ラベル更新を繰り返す

CMS の問題点 — CMS での主項の計算 $\Longleftarrow$ NP- 完全問題

「計算の効率さ」と「完全性」とのトレードオフ

1. 論理関係 $\Longrightarrow$ 節形式に変換 (PROLOG と同じ)

   ブール制約伝播アルゴリズム (BCP, Boolean Constraint Propagation)

   効率はよいが, 完全性が保証されず
2. 主項 (prime implicates) を使用した BCP

   完全性は保証されるが, 効率が悪い ($A \vee \neg A$ も主項)
3. ラベルを二分決定グラフ (Binary Decision Diagram) で表現し, 主項を列挙しないで非明示的に扱う. [奥乃]

二分決定グラフ (BDD) (Bryant, 1986)

1. ブール関数のシャノン展開に基づいたグラフ表現

$$f = (\neg x \cdot f_{|x=0}) \vee (x \cdot f_{|x=1})$$

2. 変数の順序を固定

   $\Longrightarrow$ カノニカル表現 (Unique) となる.

3. 論理演算の多くが効率よく処理できる.

4. ブール関数のコンパクトな表現.

$\Longrightarrow$ VLSI CAD では標準的な技法.

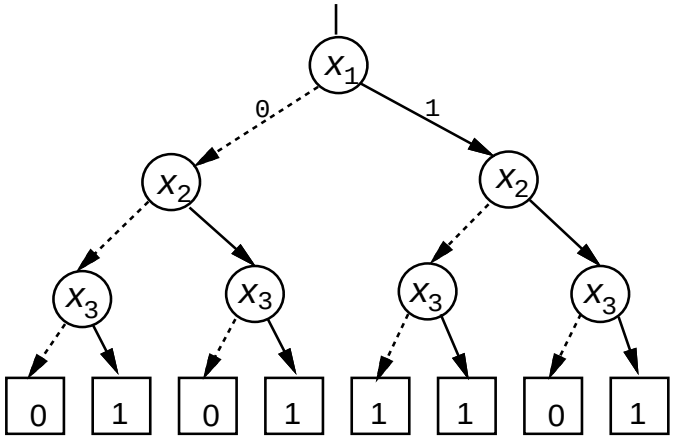
   $\Longrightarrow$ BDD を多重文脈型 TMS に適用

奥乃他: 情処論文誌, 36(8), 35 (5); bit, '97 年 4 月号

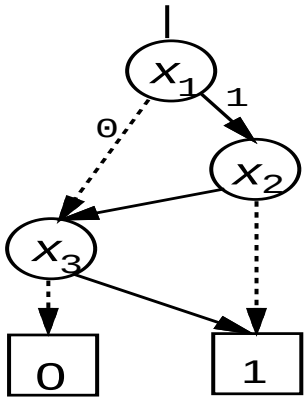
ftp://eda.kuee.kyoto-u.ac.jp/pub/cad/BemII.tar.Z

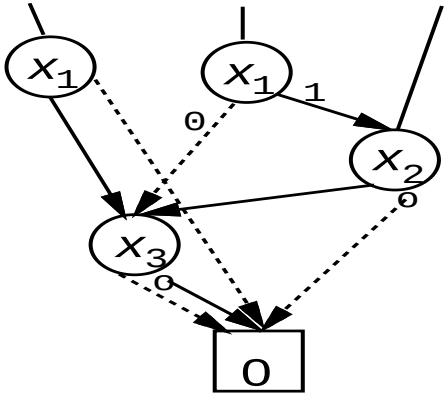
二分決定グラフ (BDD) — $x_1\overline{x_2} + x_3$

(a) シャノン展開



$x_1x_3 \quad x_1\overline{x_2} + x_3 \quad \overline{x_2} + x_3$





(b) 簡略化 $\Rightarrow$ ROBDD (c) 共有化 $\Rightarrow$ SBDD

木構造の簡約化ルール

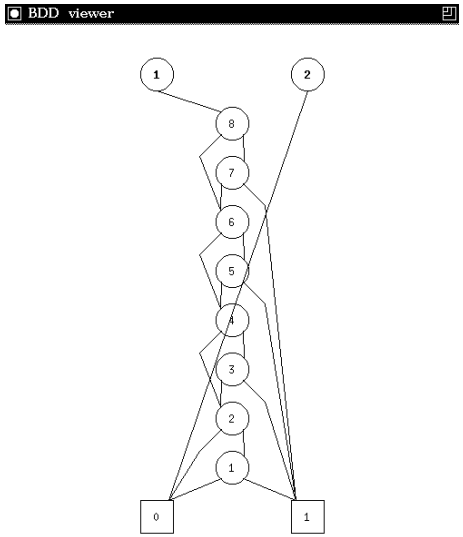
1. **重複終端ノードの除去 — 同じ終端ノードを 1 個にまとめる.**

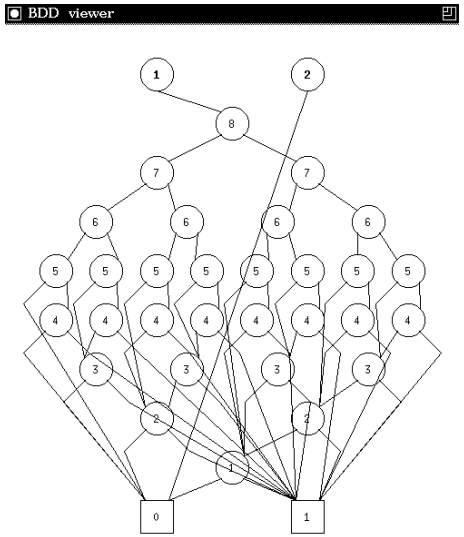
2. **重複非終端ノードの除去 — 2 つの非終端ノード u, v が $var(u) = var(v)$, かつ, $lo(u) = lo(v)$, かつ, $hi(u) = hi(v)$ を満足するならば, 1 つにまとめる.**

3. **冗長テストの除去 — 非終端ノード v が $lo(v) = hi(v)$ ならば, そのノード v を除去し, v への枝をすべて $lo(v)$ に向くように変更する.**

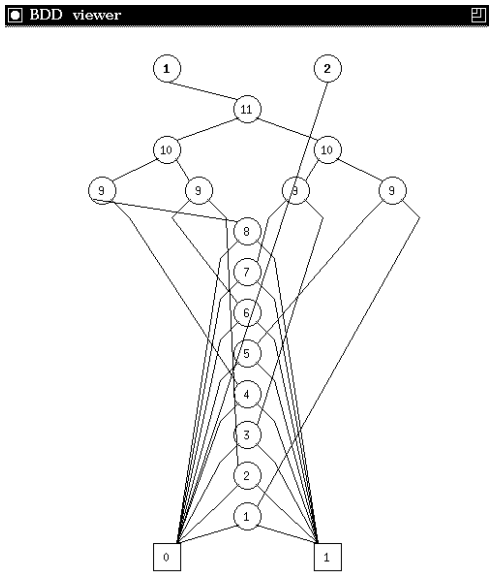
変数順序とノード数

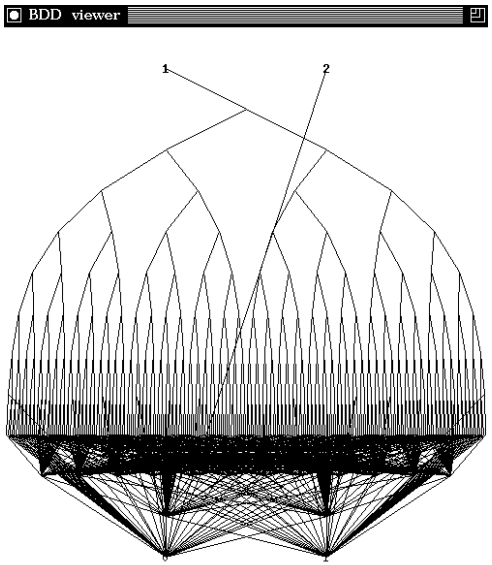
2 段 AND-OR 回路





8 入力選択回路





多重文脈型 TMS に BDD 適用での課題とその一解

1. BDD のサイズを最小にする変数順序 (NP 完全問題)

   既知の問題. ヒューリスティックスが多々提案.

2. 組合せ爆発を防ぐための正当化適用順序

   正当化の適用順序決定ヒューリスティックスの提案.

   変数順序も同時に決定.

3. 主項を使用しない正当化のコーディング・操作

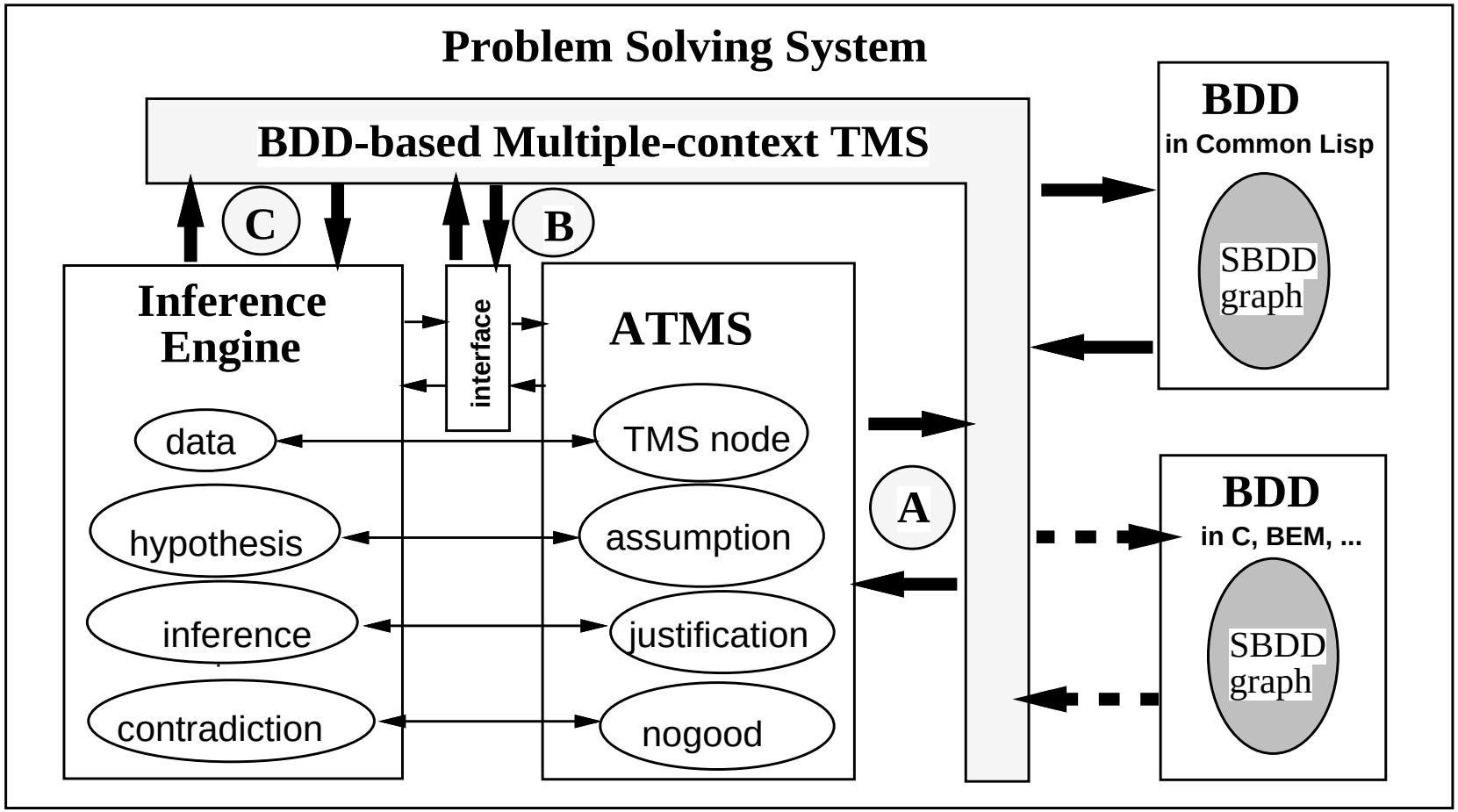
   BDD でラベルを表現.

4. 既存システムとの整合性.

   シームレス・インターフェース「$Plug\ and\ Play$」

BDD に基づいた多重文脈型 TMS (BMTMS)

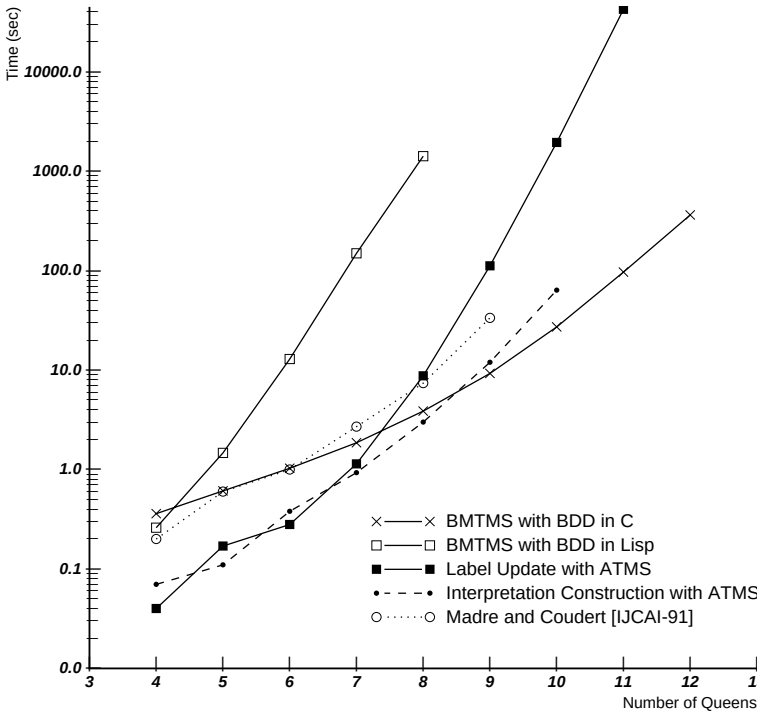
３つのインターフェース (Ⓐ, Ⓑ, Ⓒ)



BMTMS の評価 (Ⓐand Ⓑ)

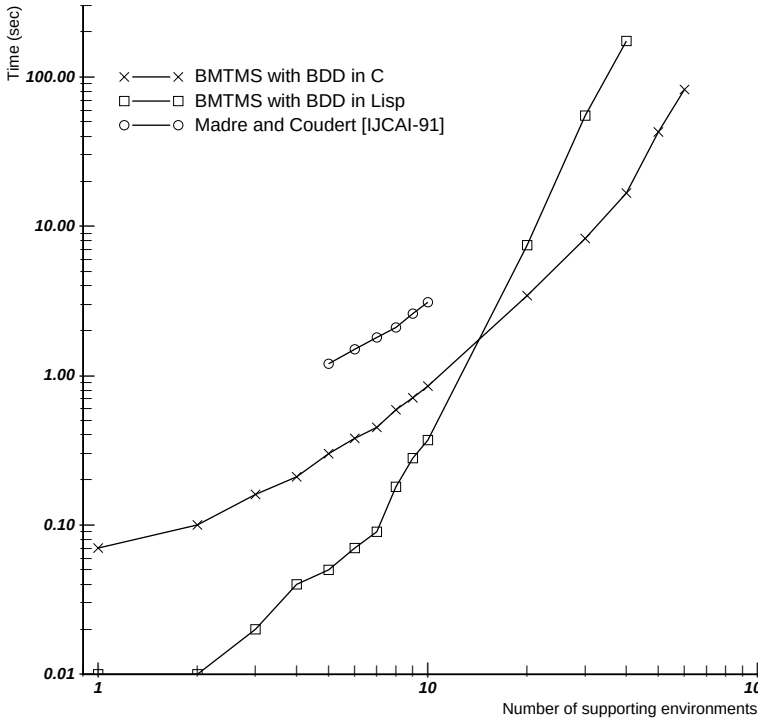
Ⓐ

N 人の女王問題

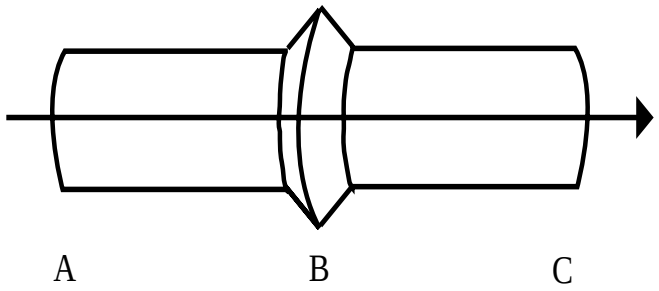


Ⓑ

最小被覆集合



BMTMS の評価 (©) — QPE による定性シミュレーション



$$[dP_A] - [dP_B] = [dQ_{AB}],$$
$$[dP_B] - [dP_C] = [dQ_{BC}],$$
$$[dQ_{AB}] = [dQ_{BC}],$$

$x + y = 0$ のコーディング: $(y_- \wedge \overline{y_+} \wedge \overline{y_0} \wedge \overline{x_-} \wedge x_+ \wedge \overline{x_0}) \vee (\overline{y_-} \wedge y_+ \wedge \overline{y_0} \wedge x_- \wedge \overline{x_+} \wedge \overline{x_0}) \vee (\overline{y_-} \wedge \overline{y_+} \wedge y_0 \wedge \overline{x_-} \wedge \overline{x_+} \wedge x_0)$

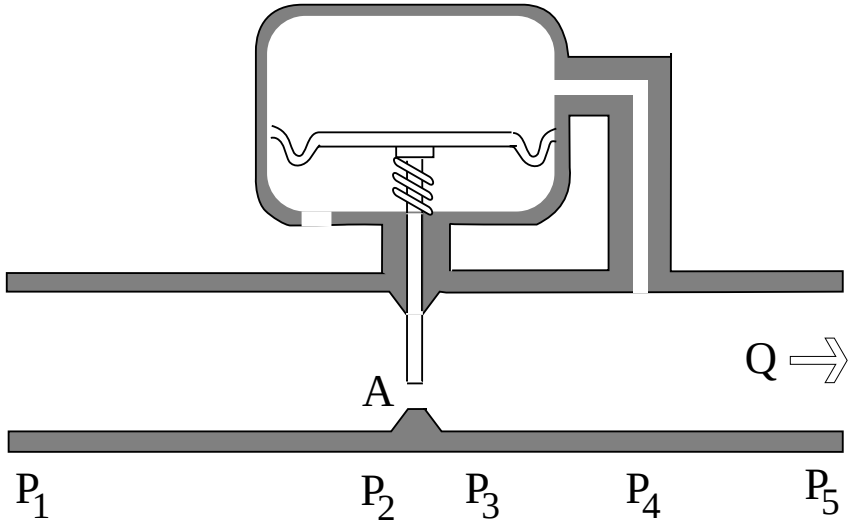
$[dP_A]$ が $+$, $[dP_C]$ が 0 と仮定.

BMTMS は, 主項を列挙せずに以下を証明:

「$[dQ_{AB}]$ と $[dQ_{BC}]$ とが $+$」

BMTMS の評価 (©)

QPE による圧力調整弁の定性シミュレーション



- ATMS を用いた QPE では, 2,814 個の主項が生成. 挙動解析に 50.38 秒.
- BMTMS による QPE では, BDD のサイズが 132. 挙動解析に 0.28 sec.

まとめと今後の課題

1. 問題解決システムのアーキテクチャ
2. 真偽維持システムの概念と機能
3. 多重文脈型真偽維持システムの機能と問題点

   ATMS ：ホーン節に限定, CMS ： 一般節
4. 問題領域に応じたコーディング法 (ホーン節を越える)
5. 多重文脈を用いた高度知的システムの開発
6. 実問題への挑戦 — システムインテグレーション

   量的挑戦：定性シミュレーション, 巨大データベース

   質的挑戦：設計問題, 診断問題, 知的データベース, ...

さらに研究を進めるにあたって

1. **強いフレームワーク**

   表現能力が弱くても，理論的に健全な論理体系を基に，システムを展開.

2. **弱いフレームワーク**

   理論的には危ういが，表現力の富む論理体系を基に，できるだけ大きな問題を効率よく処理できるように対応.

AI 研究でのスケールアップ問題 [北野]

レポート (2) — 多重文脈推論. 8／4提出

1. 御自身の研究に多重文脈推論がどのようにつかえるのかを詳細に検討し，議論する.

   最低 A4 4 枚

2. 随意 BPS をインストールし，何らかの問題に適用してみる.

レポート (2) 随意 8／4

随意 bemII (BDD) を使用する SUN のみ.

http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/02/AI/bemII.tar.g

1. SEND + MORE = MONEY
2. CROSS + ROADS = DANGER
3. FOUR + FIVE = NINE
4. NEWTON + KLEIN = KEPLER
5. MAN + WOMAN = CHILD
6. ONE + TWO + FOUR = SEVEN

随意 AC のどれかを使用して解く.