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)

奥乃 博 (okuno@i.kyoto-u.ac.jp)

OHP :

http://winnie.kuis.kyoto-u.ac.jp/∼okuno/Lecture/AI/

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

- the set of Justifications — *monotonic increase*
- the set of Enabled Assumptions — *retracted!*

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:

- Node n is $IN \Longleftrightarrow$ n follows from $A \cup J$
- Node n is OUT otherwise

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

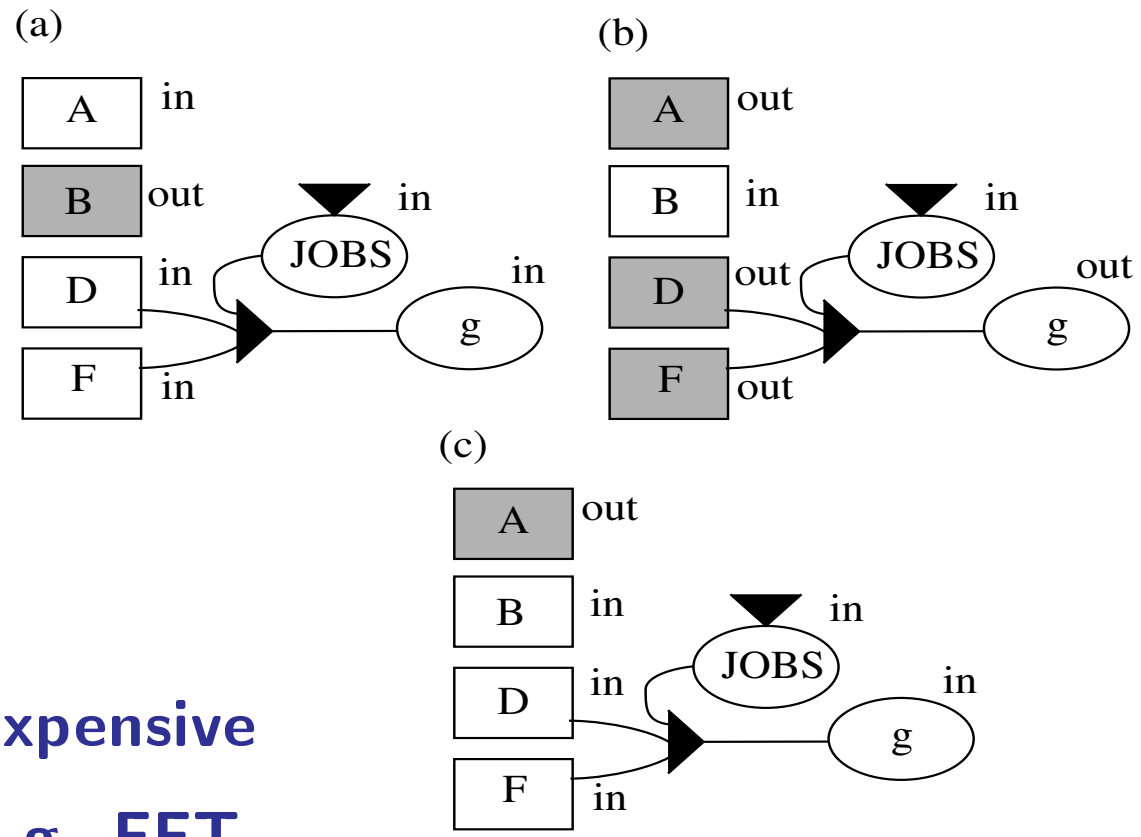
JTMS cannot deduce the Negation of Data

$\Longrightarrow$ Encoding the Negation of Data

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

How Justifications Save Inference Engine Work

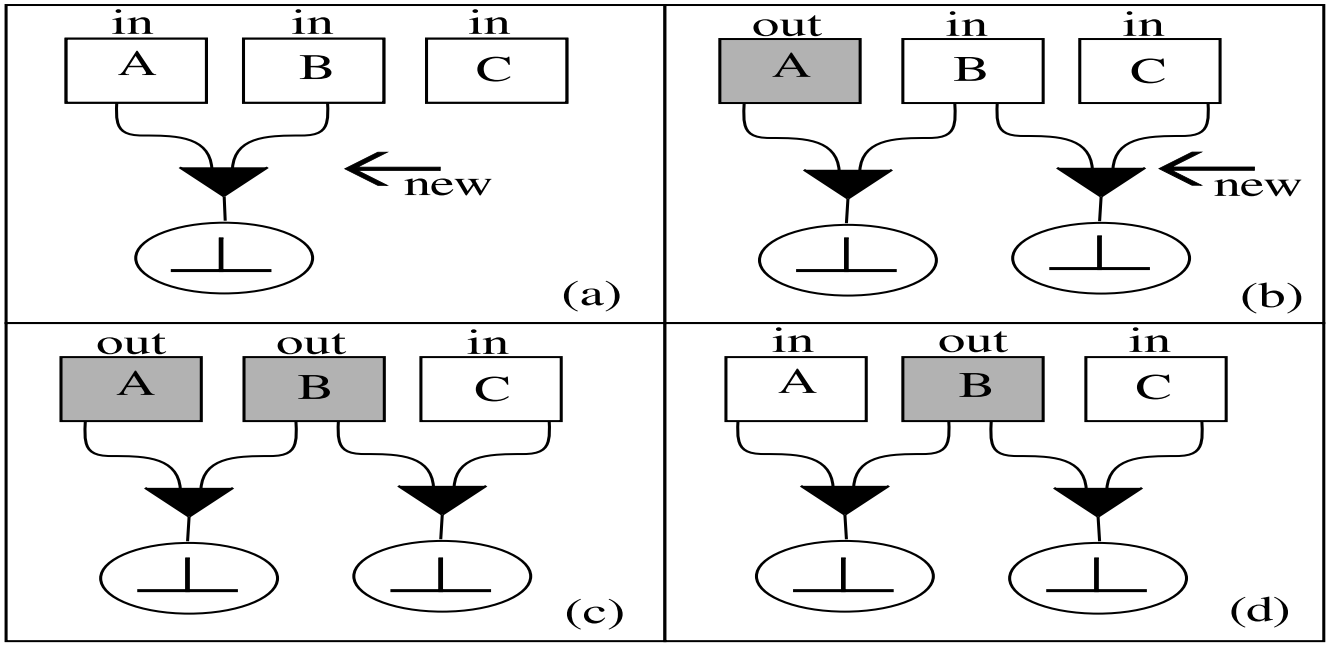
Maintaining a Justification Cache



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)

JTMS Interface [1]

`change-jtms`, `create-jtms` : create jtms

`tms-create-node`, `assume-node`

`enable-assumption`, `retract-assumption`

`justify-node`, `make-contradiction`

`in-node?`, `out-node?` : ask a belief status of a node

`tms-node-datum`, `tms-node-in-rules`, `tms-node-out-rules`

`just-antecedent`, `just-consequence`, `just-informant`

`supporting-justification-for-node`

: ask a well-founded explanation for a node

`assumptions-of-node`

: ask the set of enabled assumptions for WFE for a node

JTMS Interface [2]

```
(create-jtms title
       &key (node-string 'default-node-string)
            (debugging nil)
            (checking-contradictions t)
            (contradiction-handler 'ask-user-handler)
            (enqueue-procedure nil))

(change-jtms jtms &key node-string
       debugging
       checking-contradictions
       contradiction-handler
       enqueue-procedure)
```

JTMS Interface [3]

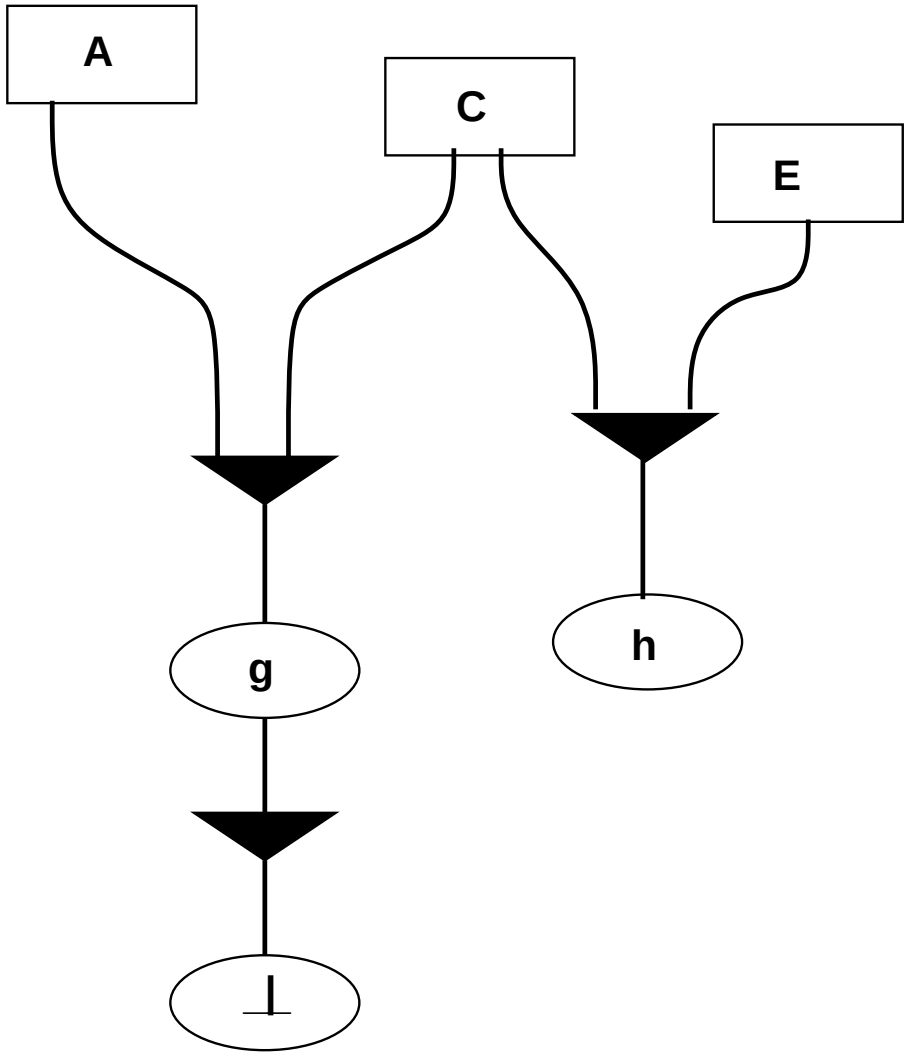
```
(tms-create-node jtms datum
                 &key assumptionp contradictoryp)
(enable-assumption node)
(retract-assumption node)
(make-contradiction node)
(assume-node node)
(in-node? node)
(out-node? node)
(justify-node informant consequent antecedents)
(supporting-justification-for-node node)
(assumptions-of-node node)
```

JTMS 使用法の簡単な例

```
(setq *jtms* (create-jtms "Simple Example"))

(setq assumption-a (tms-create-node *jtms* "A" :ASSUMPTIONP t)
      assumption-c (tms-create-node *jtms* "C" :ASSUMPTIONP t)
      assumption-e (tms-create-node *jtms* "E" :ASSUMPTIONP t))
(enable-assumption assumption-a)
(enable-assumption assumption-c)
(enable-assumption assumption-e)

(setq node-h (tms-create-node *jtms* "h"))
(justify-node "R1" node-h (list assumption-c assumption-e))

(setq node-g (tms-create-node *jtms* "g"))
(justify-node "R2" node-g (list assumption-a assumption-c))
(setq contradiction (tms-create-node *jtms* 'CONTRA :CONTRADICTORYP t))
(justify-node "R3" contradiction (list node-g))
```

```
Contradiction found: CONTRADICTION
1 C
2 A
Call (TMS-ANSWER <number>) to retract assumption.
Break: JTMS contradiction break
```

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**:

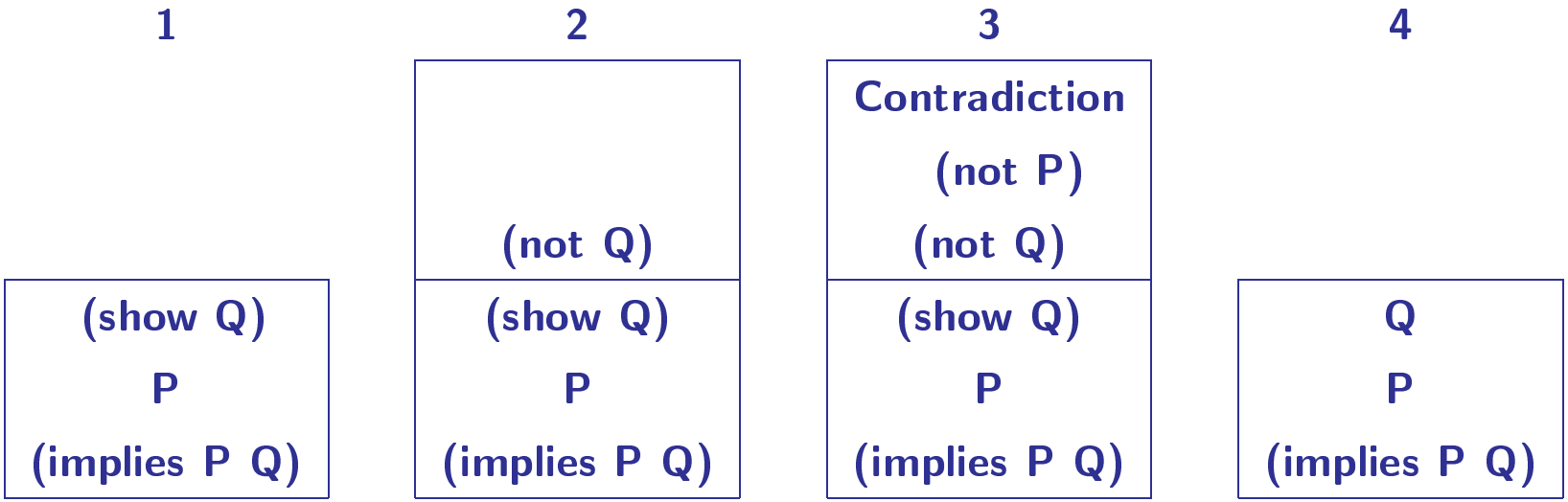
   each node is extended to have its belief status with supporting justifications

Context Approach Example:

Stack-Oriented Context Mechanism

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

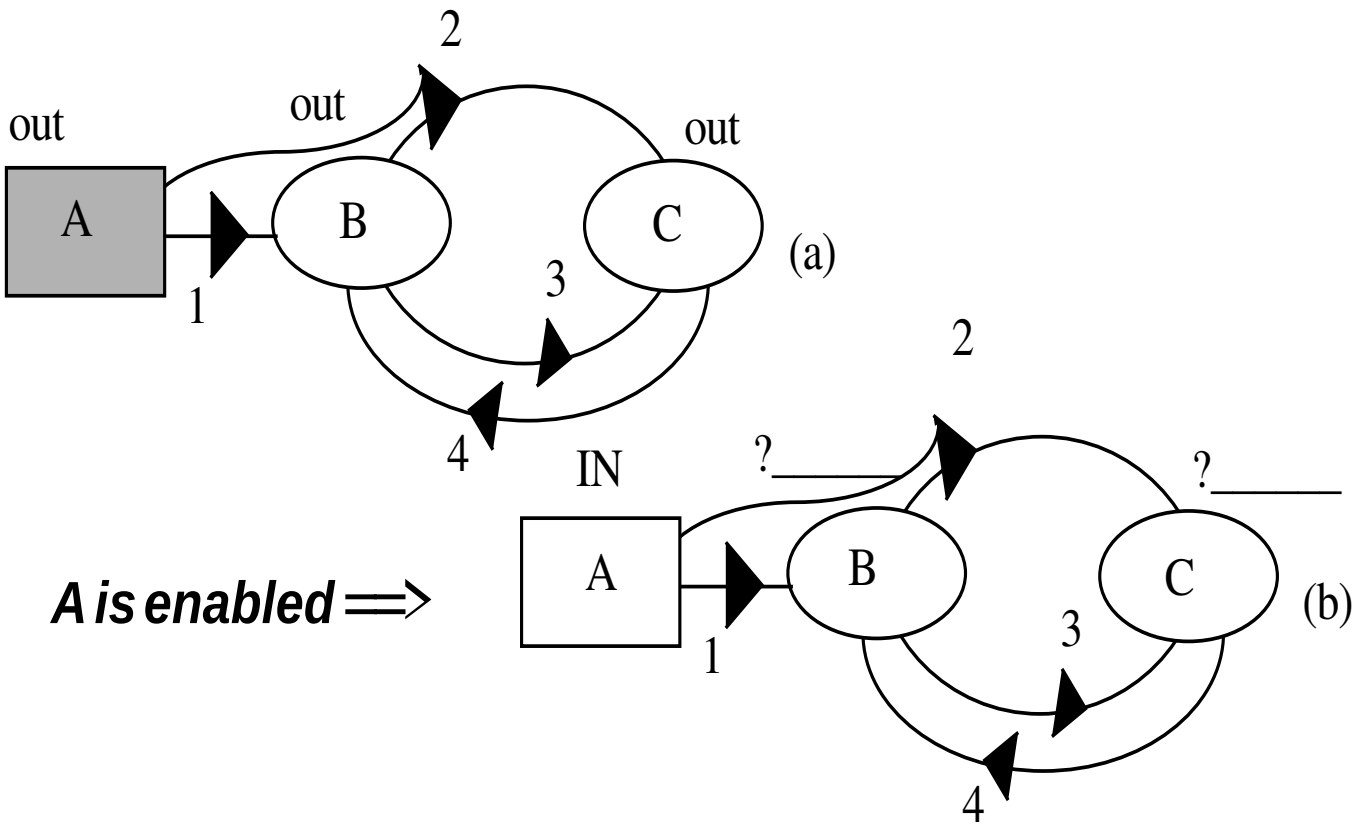
by Reductio Ad Absurdum



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: [ ]

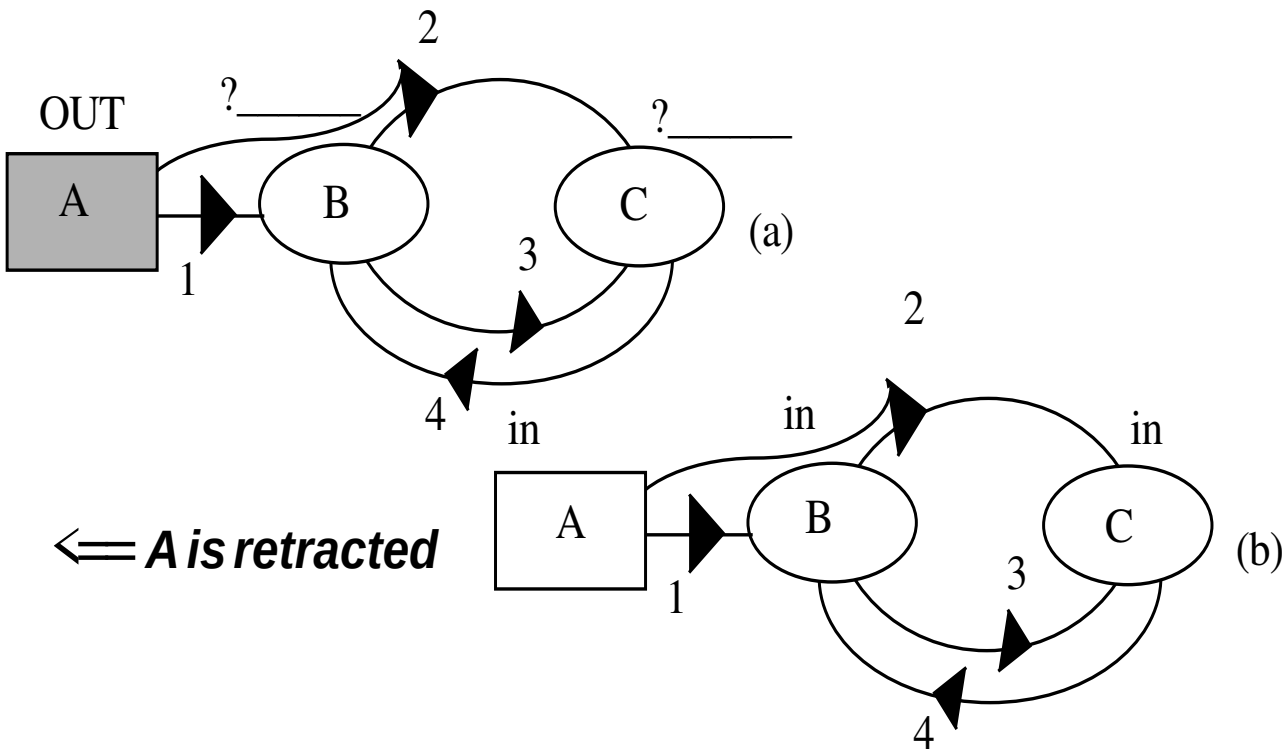
Adding a Justification J

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

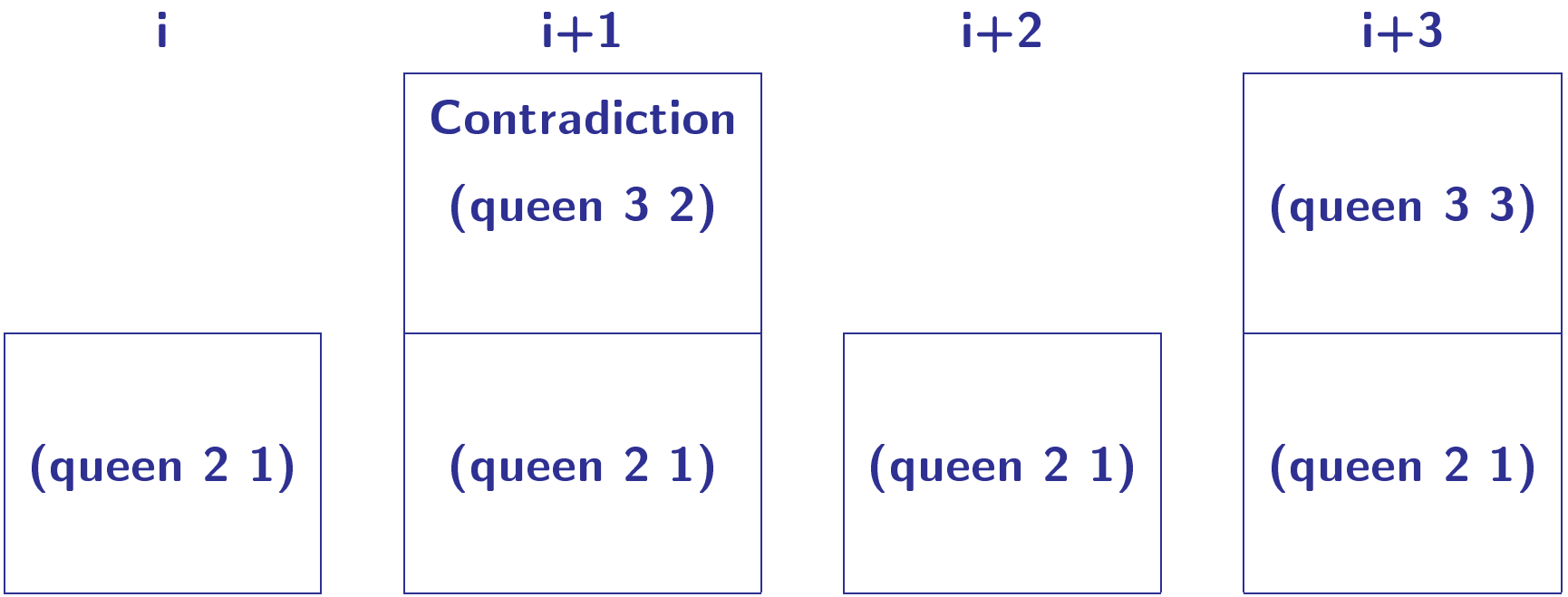
1. Chronological Search with Stack-Oriented Context Mechanism
2. JTMS-based Dependency-Directed Search

```
(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



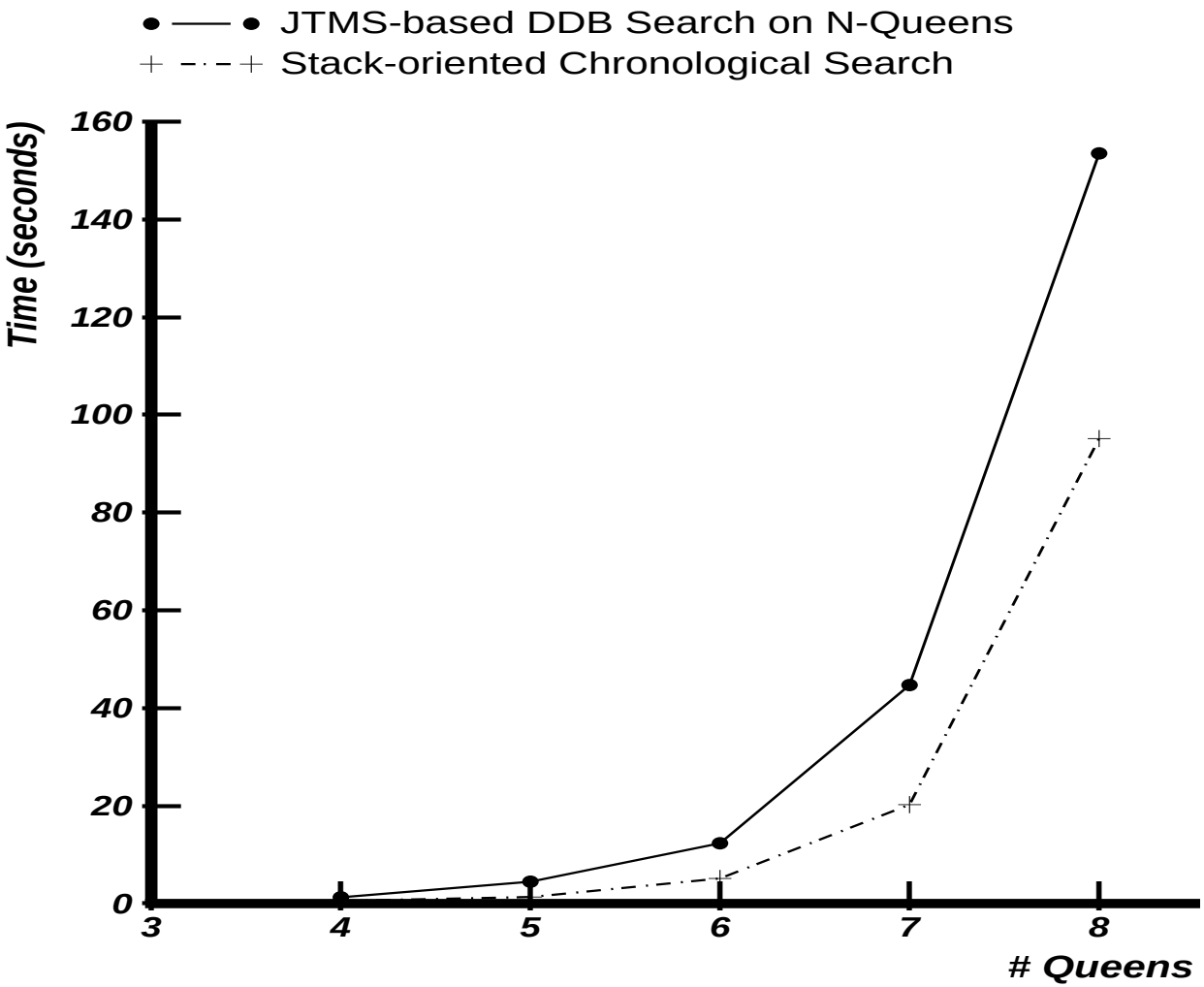
Dependency-Directed Search

```
(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) ))))))
```

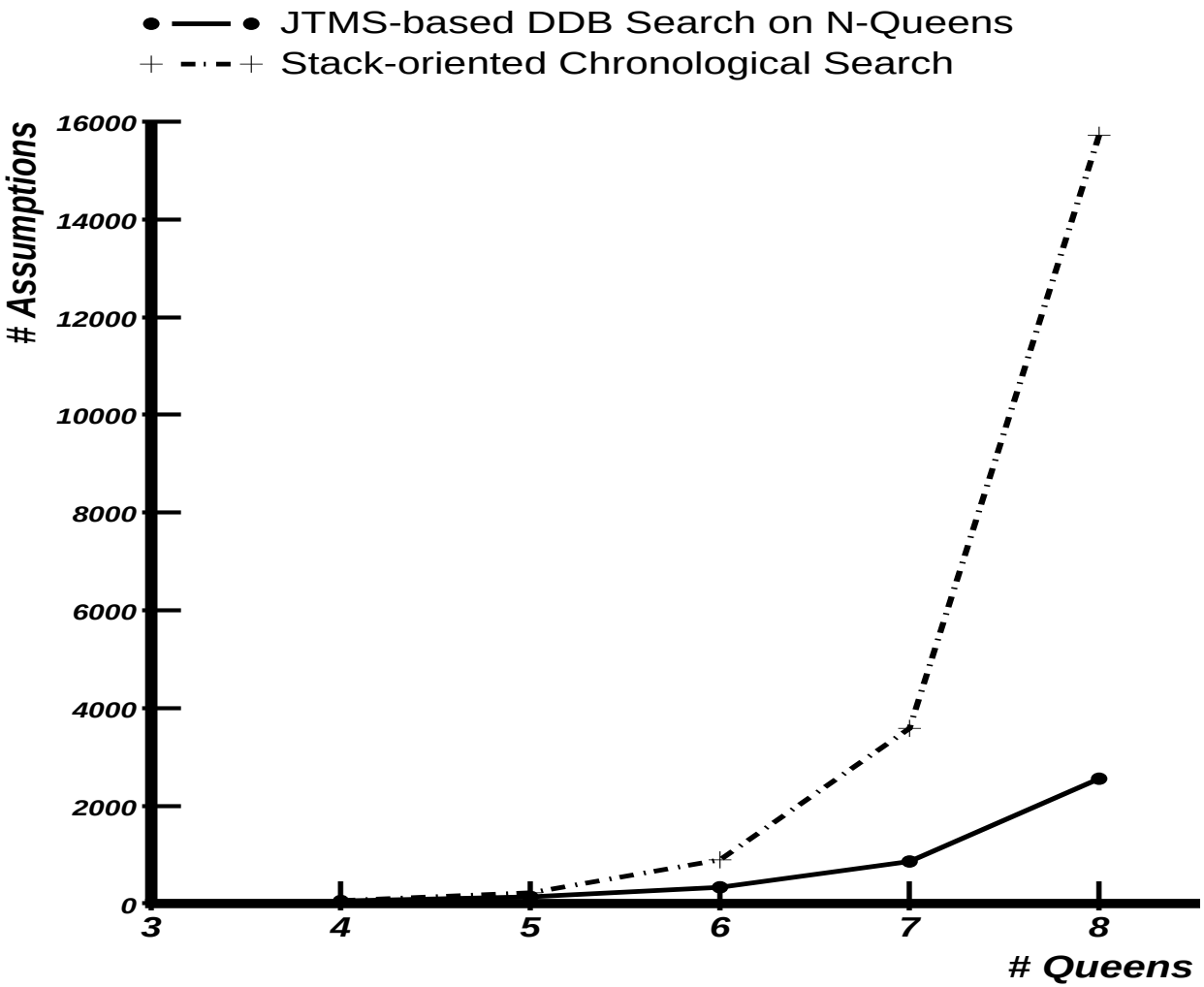
((queen 2 1) (queen 3 2)) is *inconsistent*

$\Rightarrow$ *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.

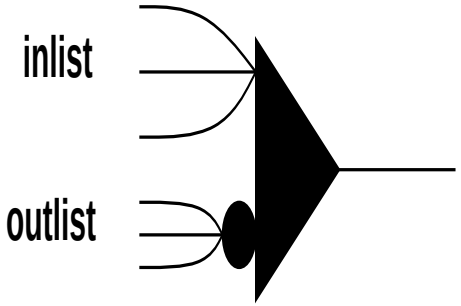
  However, DDS *postpones* a combinatorial explosion.

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

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

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

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

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

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

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

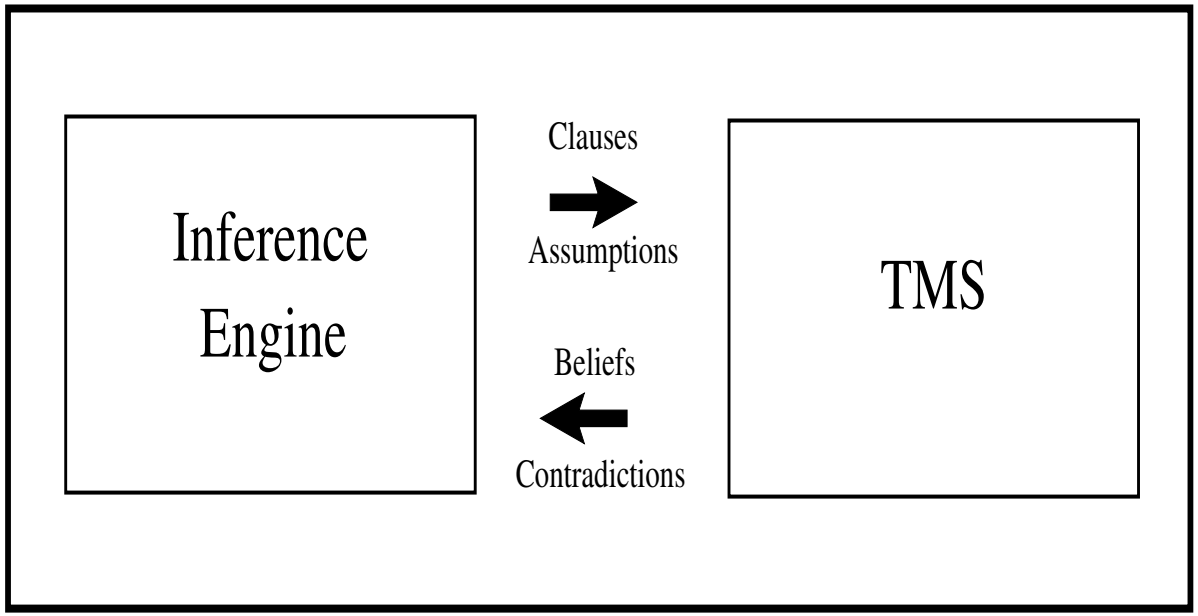
$$\mathbf{COLD} \supset \mathbf{ILL}$$

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.**

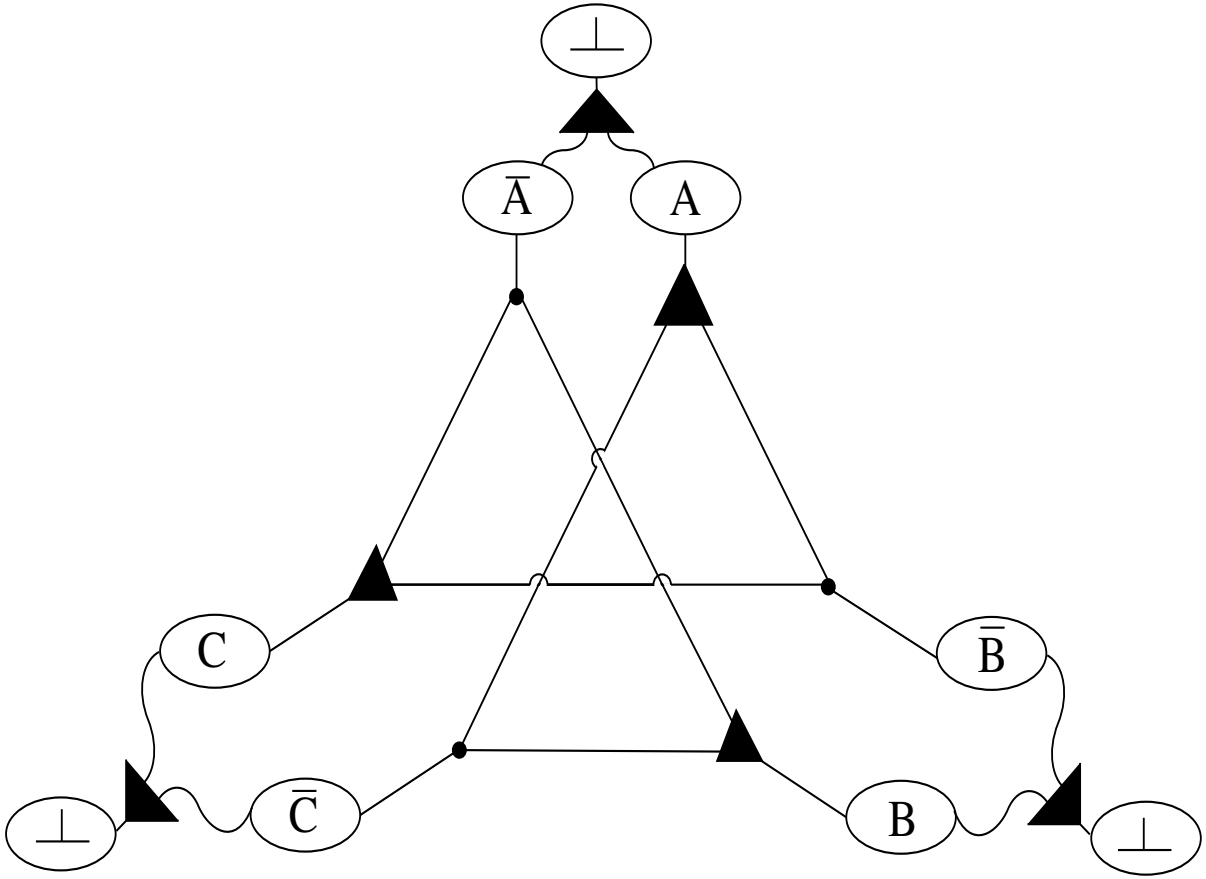
Consider: Encoding $A \vee B \vee C$ **for JTMS**

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

[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.

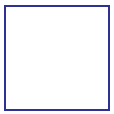
But more processing may be possible.

$A_1, \ldots A_n$ 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**

  - *enabled* — **IE choosed to be** `:TRUE` **or** `:FALSE`
  - *retacted* — **otherwise. treated as any other node**

- **(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-$complete$.

Trace of BCP Algorithm [1]

Given:

$y \vee s \vee z$	**(1)**	$\neg y \vee q \vee r$	**(2)**
$x \vee \neg y$	**(3)**	$r \vee \neg s$	**(4)**
$r \vee z$	**(5)**		

ENABLE-ASSUMPTION$(\neg x)$

1. **Label** x :`FALSE`
2. **(3) is now unit open, so** $S = \{3\}$
3. **BCP labels** y :`FALSE` **and halts**

Then

Trace of BCP Algorithm [2]

$y \vee s \vee z$	**(1)**	$\neg y \vee q \vee r$	**(2)**
$x \vee \neg y$	**(3)**	$r \vee \neg s$	**(4)**
$r \vee z$	**(5)**		

ENABLE-ASSUMPTION$(\neg r)$

1. **Label** r :FALSE
2. **(4) and (5) are now unit open, so** $S = \{4,5\}$
3. **Processing (4), BCP labels** s :FALSE.
   (1) is now unit open, so $S = \{1,5\}$
4. **Processing (1), BCP labels** z :TRUE. $S = \{5\}$
5. **BCP determines (5) is now satisfied, so halts.**

Well-Founded Explanations (WFE) for Node c

"a Sequence of steps, $S_1, \ldots, S_k$" s.t.

$$\textbf{n} \quad \langle conclusion \rangle \quad \langle antecedents \rangle \quad \langle reason \rangle$$

- **n is an integer.**
- $\langle conclusion \rangle$ **is a literal. A symbol or its negation can be in the conclusion of at most one step.**
- $\langle antecedents \rangle$ **is a set of intergers ($<$ n).**
- $\langle reason \rangle$ **is an IE-supplied clause or 'Assumption'.**

Conclusion of S_k must correspond to c's label.

Assumption Steps and Derivation Steps

- **Assumption Steps**

n	$\langle conclusion \rangle$	$\langle antecedents \rangle$	$\langle reason \rangle$
i	x	$\{\}$	**Assumption**

  x **: a literal corresp. to an enabled assumption**

- **Derivation Steps**

n	$\langle conclusion \rangle$	$\langle antecedents \rangle$	$\langle reason \rangle$
i	x	A	C

  x **: any literal,** A **: a set of antecedent steps**

  x **follows from clauses** C **and the consequences of some steps in** A**.**

Well-Founded Explanation

Consider the clauses:

$$x \vee y$$
$$\neg y \vee z$$
$$\neg z \vee r$$

IF x is assumed to be :FALSE,

a WFE for r is:

n	$\langle conclusion \rangle$	$\langle antecedents \rangle$	$\langle reason \rangle$
1	$\neg x$	$\{\}$	**Assumption**
2	y	$\{\mathbf{1}\}$	$x \vee y$
3	z	$\{\mathbf{2}\}$	$\neg y \vee z$
4	r	$\{\mathbf{3}\}$	$\neg z \vee r$

Encoding Propositonal Formulas as Clauses

Formula $\Longrightarrow$ Conjunctive Normal Form $C_1 \wedge \ldots \wedge C_k$

- **Replace $x \equiv y$ with $(x \supset y) \wedge (y \supset x)$.**
- **Replace $x \supset y$ with $\neg x \vee y$.**
- **Apply DeMorgan's laws and Remove double negation.**
- **Apply distributive, associative and commutative laws.**

練習 Transform $(r \wedge (s \supset t)) \supset u)$ to CNF

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`**.**

  $\Longrightarrow$ $\boxed{}$

- *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** $\boxed{}$ **No** $\boxed{}$

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*.

LTMS Interface [1]

change-ltms, create-ltms : **create ltms**
tms-create-node, find-node
tms-node-datum, tms-node-true-rules, tms-node-false-rules
enable-assumption, retract-assumption
true-node?, false-node?, unknown-node?, known-node?
satisfied-clause?, violated-clause?
add-clause, add-nogood
formula is converted to a set of clauses.
add-formula, compile-formula
supporting-justification-for-node, support-for-node
assumptions-of-node, assumptions-of-clause
clause-antecedent,clause-consequence,clause-informant
with-assumptions, with-contradiction-check
explain-node ; **generate a WFE**
some functions for sophisticated problem solving
with-contradiction-check, without-contradiction-check
with-contradiction-handler, ltms-pending-contradictions

LTMS Interface [2]

```
(create-ltms title &key (title nil)
        (node-string 'default-node-string)
        (debugging nil)
        (checking-contradictions T)
        (contradiction-handler 'ask-user-handler)
        (enqueue-procedure nil)
        (complete nil)
        (delay-sat T))

(change-ltms ltms &key node-string
             debugging checking-contradictions
             contradiction-handler enqueue-procedure)
```

Simple Example of LTMS Usage

```
(setq *ltms* (create-ltms "Simple Example"))

(setq x (tms-create-node *ltms* "x" :assumptionp t)
      y (tms-create-node *ltms* "y")
      z (tms-create-node *ltms* "z")
      r (tms-create-node *ltms* "r"))

(add-formula *ltms* `(:OR ,x ,y))                ; x ∨ y
(add-formula *ltms* `(:OR (:NOT ,y) ,z))         ; ¬ y ∨ z
(add-formula *ltms* `(:OR (:NOT ,z) ,r))         ; ¬ z ∨ r

(enable-assumption x :FALSE)

(explain-node r)

   1       (:NOT x)                ()     Assumption
   2       y                       (1)    (:OR x y)
   3       z                       (2)    (:OR (:NOT y z))
   4       r                       (3)    (:OR (:NOT z r))
```

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.