Truth Maintenance Systems

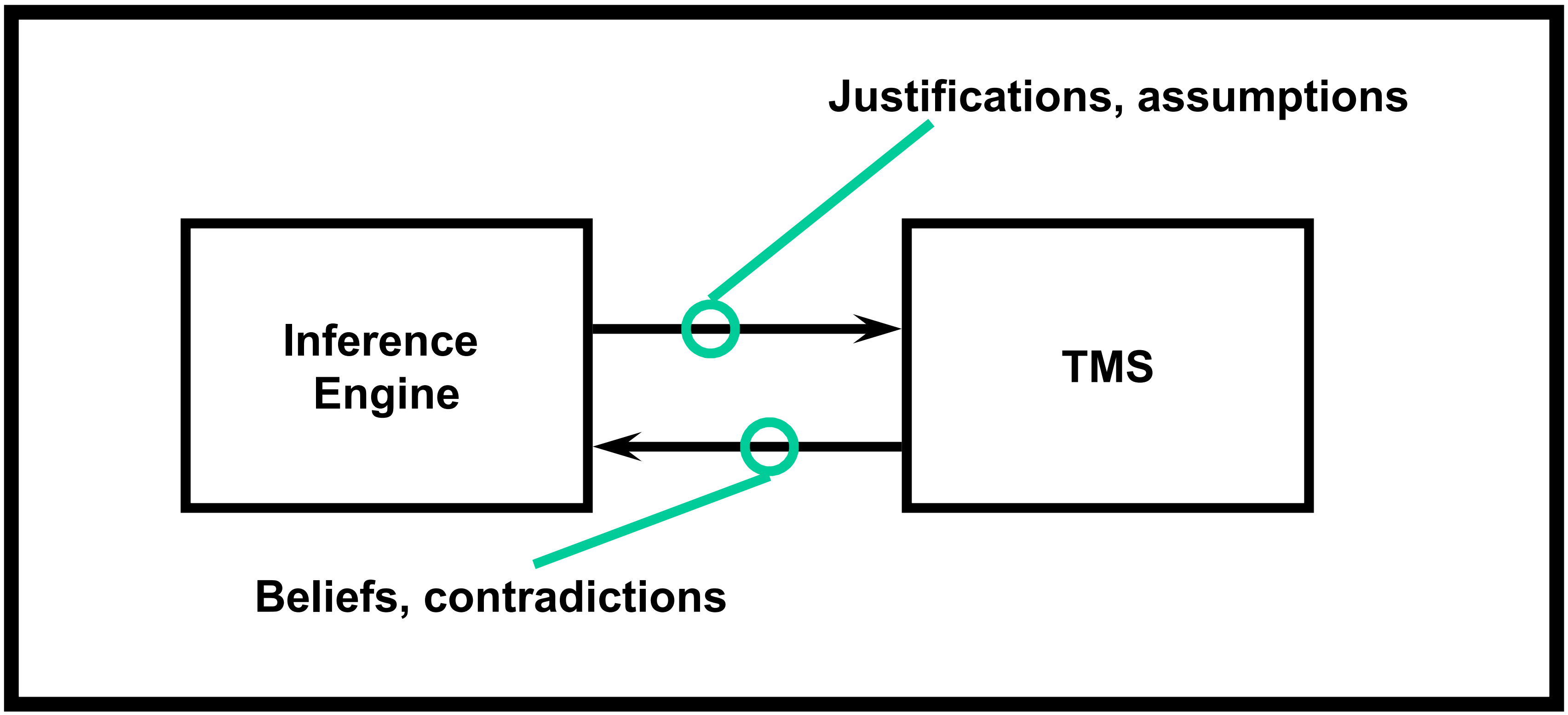
EECS 344
Winter 2008

Outline

- **What is a TMS?**
- **Basic TMS model**
- **Justification-based TMS**

What is a TMS?

- A useful problem-solver module



How using a TMS helps

- Identify responsibility for decisions
- Recover from inconsistencies
- Maintain and update cache of beliefs
- Guide backtracking
- Support default reasoning

Advantages of a TMS

- Meets the desiderata
- Frees designer to work on domain issues
- Avoids reinventing the wheel
- Avoids reinventing the wheel badly
- Change underlying implementation as needed

Desiderata 1
Identify Responsibility

- Providing answers is not enough
  - `Cut the patient's heart out`
  - `That design won't work`
- Explanations are needed
  - `Radical bypass surgery is required because...`
  - `No material will stand the projected stresses.`

Desiderata 2
Recover from Inconsistencies

- **Data can be wrong**
  - The patient's temperature is 986 degrees.
- **Constraints can be impossible**
  - Our new computer should
    - » Run off batteries for 8 hours
    - » Fit in an earphone
    - » Run faster than a Cray MP-X

Desiderata 3
Maintain and Update Cache

- **All AI problem solvers search**
- **Changing assumptions requires updating consequences of beliefs**
- **Rederivation can be expensive**
  - **Large, complex calculations (e.g., computational fluid dynamics)**
  - **Physical experiments**

Desiderata 4
Guide Backtracking

- **Avoid rediscovering contradictions**
- **Avoid throwing away useful results**

Example

Choose in sequence:

- **A or B**
- **C or D**
- **E or F**

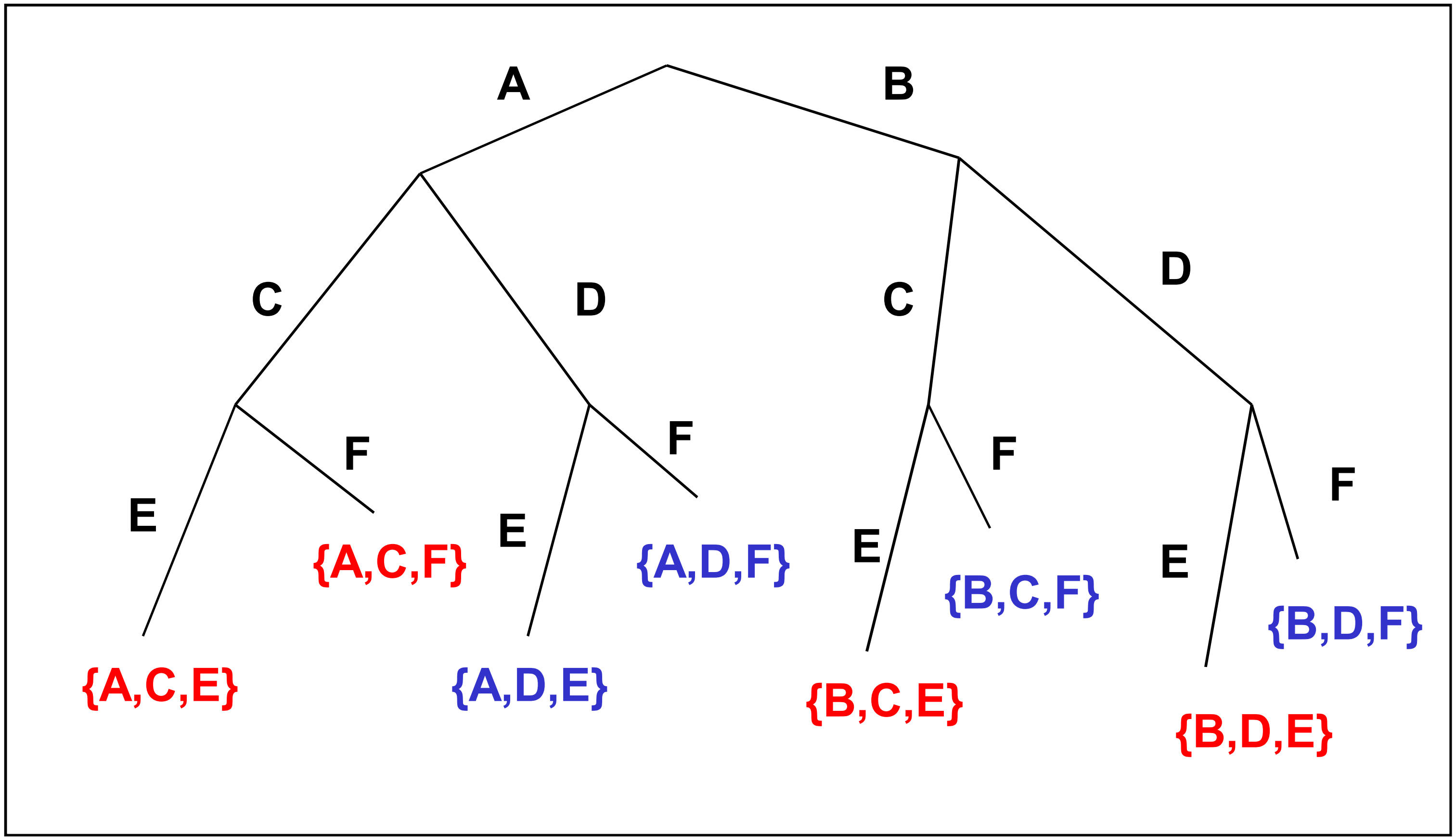
Given: A and C cannot hold together

Given: B and E cannot hold together

Assume we want all consistent solutions

Assume that we cannot test until every choice has been made

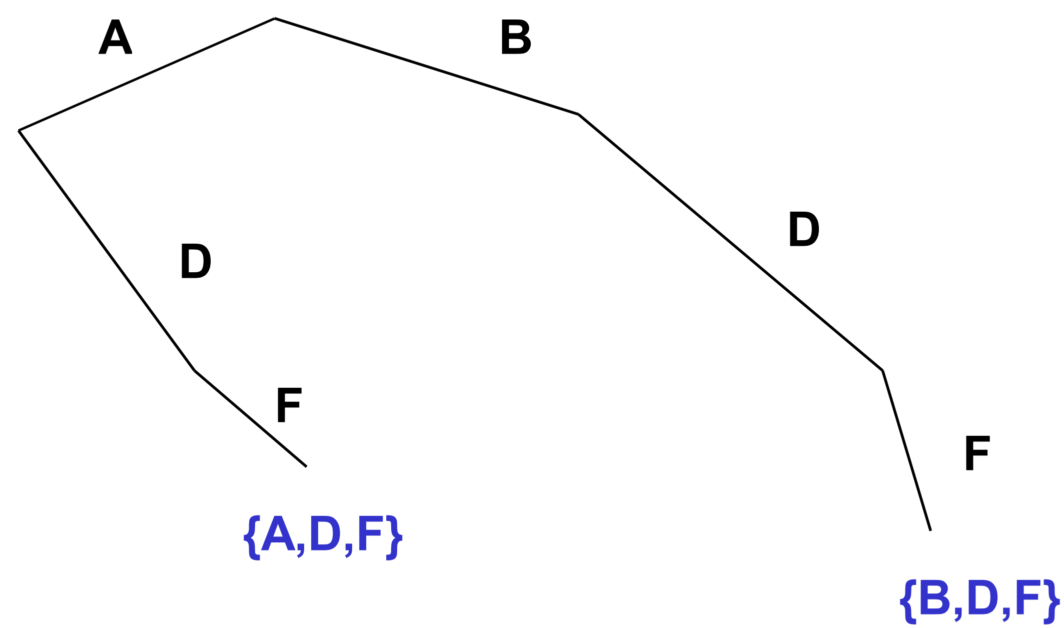
Example Search Space (global view)



Chronological Backtracking

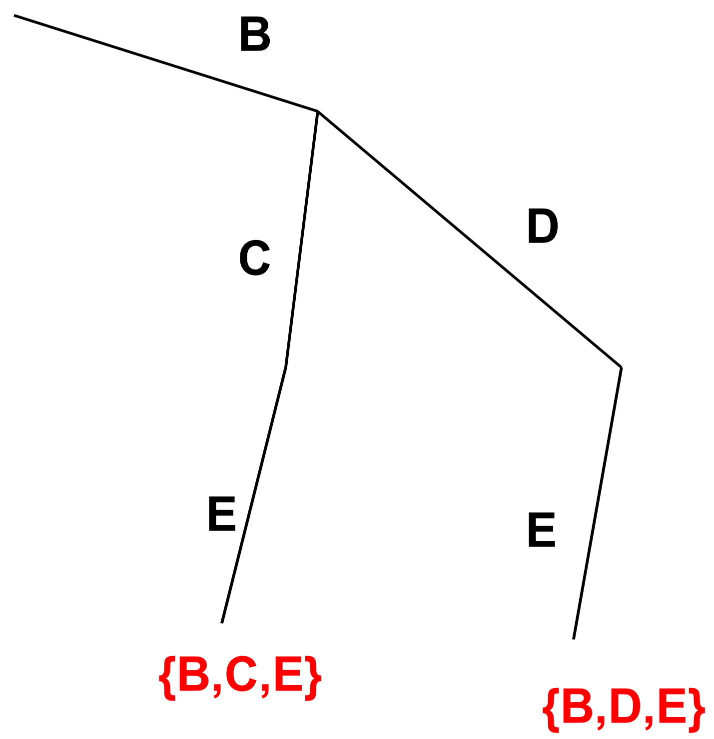
- **Often wastes computation**

Example: Suppose D and F together cause lots of work. Popping context loses this work

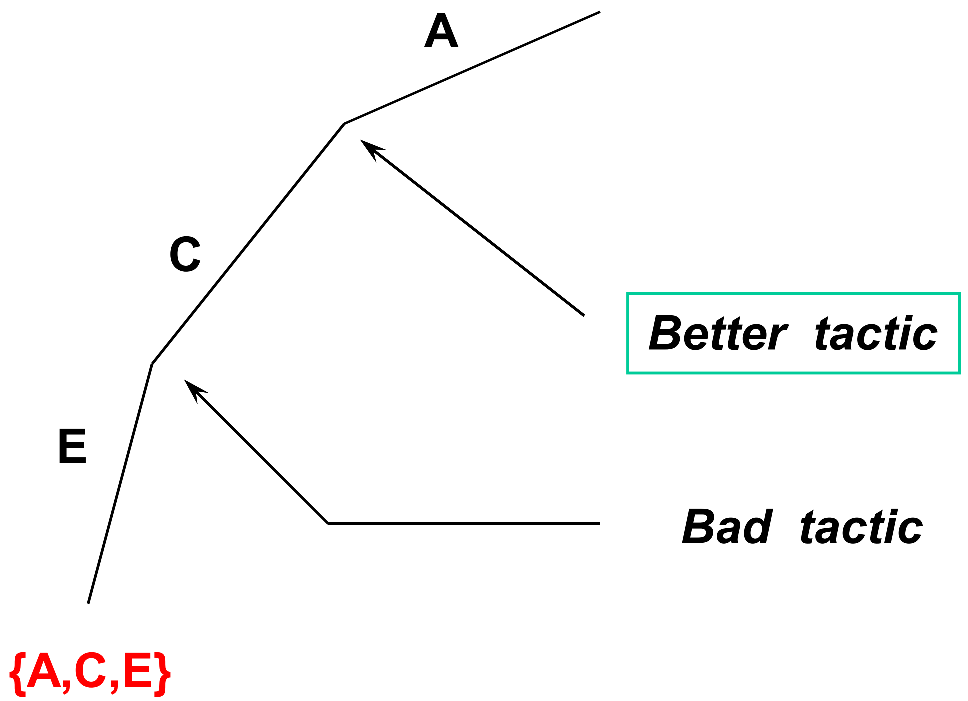


Chronological Backtracking Rediscovers Contradictions

Example: Useless to try B and E together more than once



Dependencies can guide backtracking



Desiderata 5
Support default reasoning

- **Simple defaults**

```
Bird(Tweety) implies Can-Fly(Tweety)
  unless Broiled(Tweety)
```

- **Closed-world assumptions**

  The design can use either NMOS or CMOS

  The only possible bugs are in the fuel pump or the carburetor

How does the TMS do it?

- ***Justifications*** **express relationships between beliefs**
  - **Justifications for a belief provide explanations, ability to pinpoint culprits**
- **Belief in an assertion expressed via its *label***
  - **`P` being in database no longer is the same as believing `P`**
  - **Assertions and justifications serve as cache**
  - **Rules/other computations need only be executed once**
- **Justifications can be used to record inconsistencies**
  - ***Dependency-directed backtracking***
- **Defaults can be represented via explicit assumptions**

Justification-based TMS

- **One element of a TMS design space**
  - **JTMS = *justification-based* TMS**
  - **LTMS = *logic-based* TMS**
  - **ATMS = *assumption-based* TMS**
- **Simplest**
- **Good model for most "embedded" dependency systems**
- **Can quickly focus on how to use it**

JTMS nodes

- Each belief is represented by a TMS node
- Typically, TMS nodes are associated 1:1 with assertions
- The label of a node represents the belief status of the corresponding problem solver fact.
- The relationships between beliefs are expressed by the justifications it participates in.

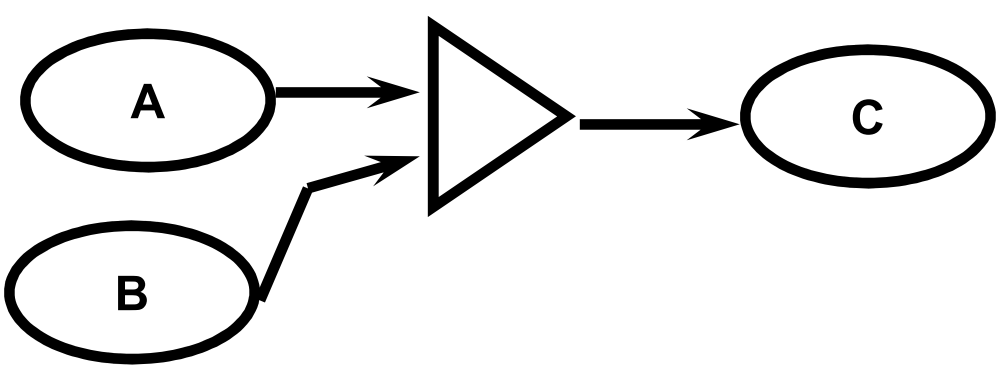
JTMS Labels

- Every assertion is either IN or OUT
  - IN = “believed”
  - OUT = “not believed”
- Warning: IN does not mean TRUE

	`P` in	`P` out
`(not P)` in	Contradiction	`(not P)` true
`(not P)` out	`P` true	Don’t know

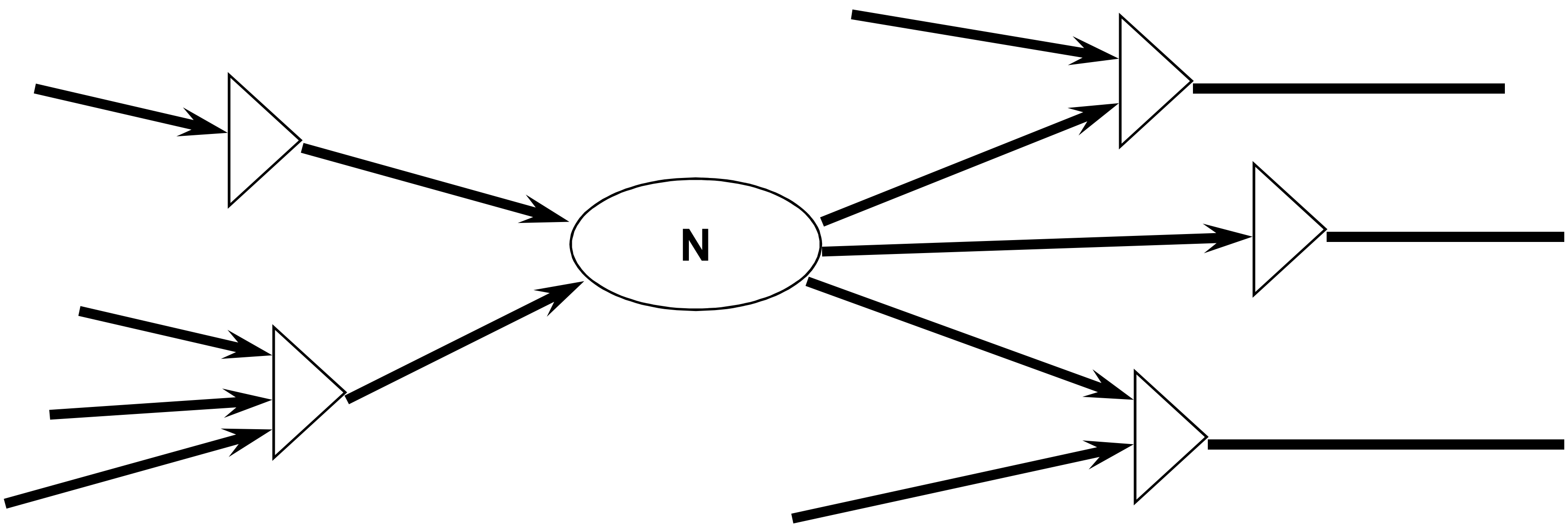
JTMS Justifications

- Must be Horn clauses
- Nomenclature
  - *Consequent* is the node whose belief is supported by a the justification
  - *Antecedents* are the beliefs which, when IN, support the consequent
  - *Informant* records information from external systems



Dependency Networks

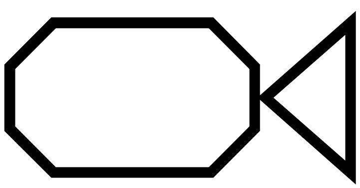
- **Each node has:**
  - *Justifications* **= the justifications which have it as the consequent**
  - *Consequences* **= justifications which use it as an antecedent**
  - *Support* **= a single justification taken as the reason for it being IN, if any.**



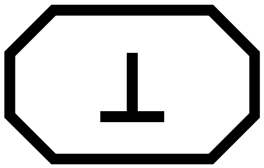
Special states of JTMS nodes

- *Assumptions* are IN if enabled.
- *Premises* are always IN.
- *Contradictions* should never hold.

Assumption



Premise



Contradiction

Enforcing constraints between beliefs

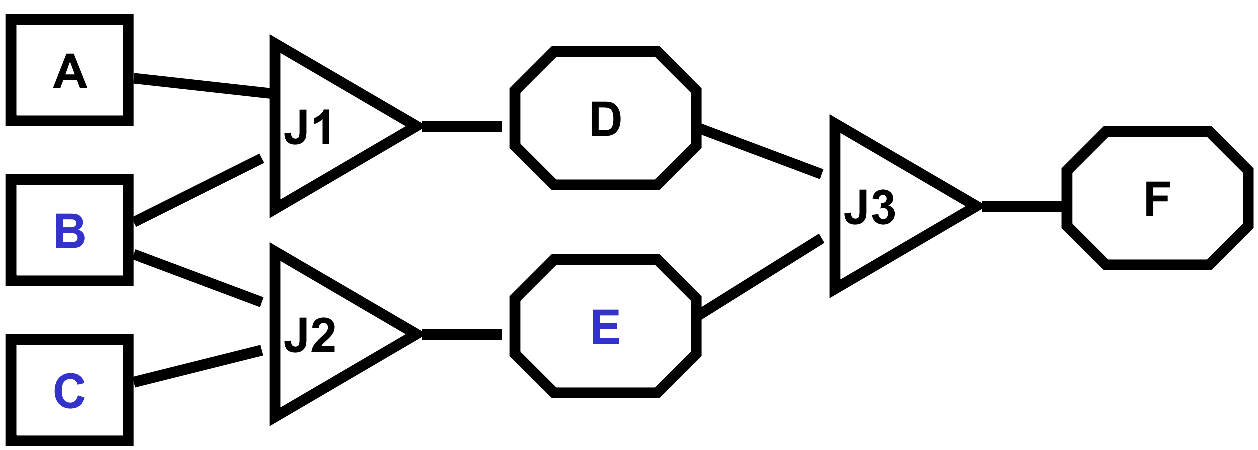
- **A node is IN when either:**
  1. **It is an enabled assumption or premise**
  2. **There exists a justification for it whose antecedents are all IN**
- **Assumptions underlying a belief can be found by backchaining through supporting justifications**
- **JTMS operations must preserve *well-founded support.***

A TMS operates incrementally

- **At any time, the inference engine can add**
  - **new justifications**
  - **declare a statement to be a premise or contradiction (permanently)**
  - **Assume a statement**
- **In all cases,**
  1. **Set the directly affected node, if any.**
  2. **Propagate the consequences (`propagate-inness`)**

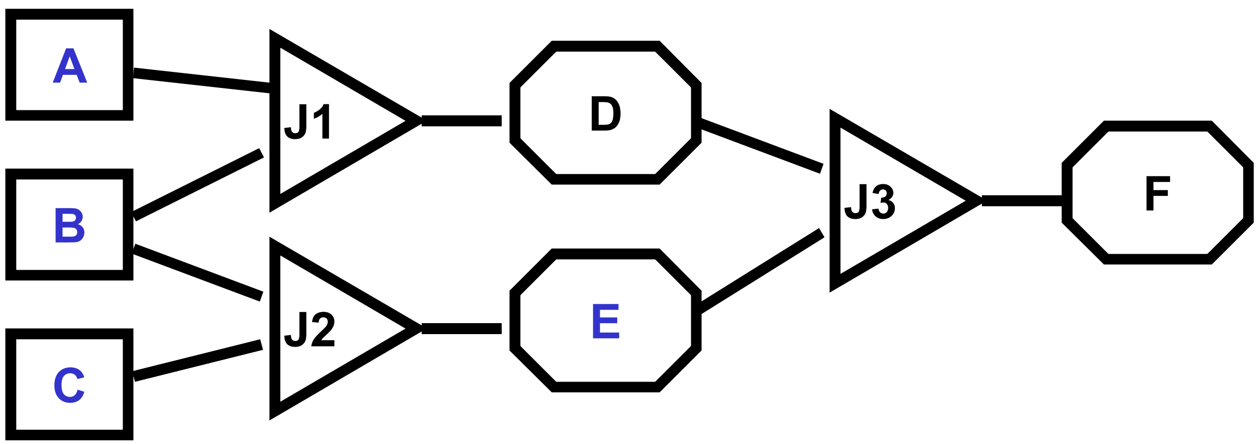
Propagation of Belief Example

Initial state of dependency network



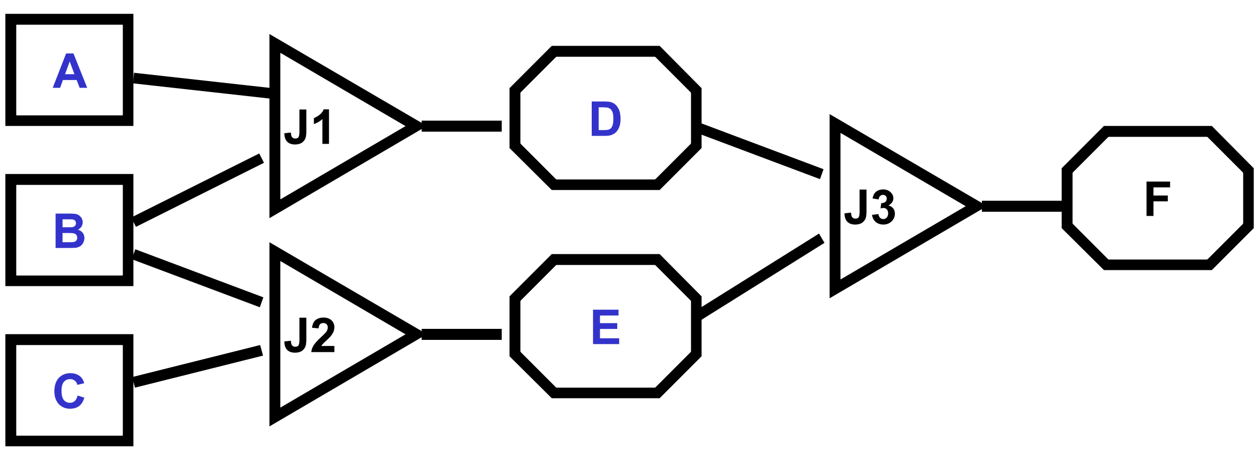
Example of `Propagate-inness`

Suppose inference engine enables A:



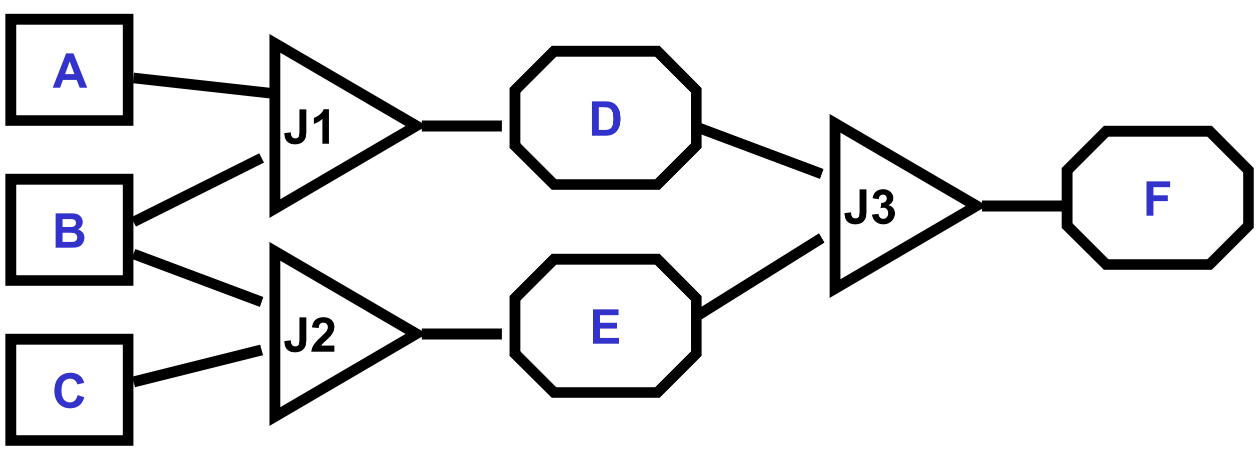
Example of `Propagate-inness`

D becomes believed via J1:



Example of `Propagate-inness`

F becomes believed via J3:

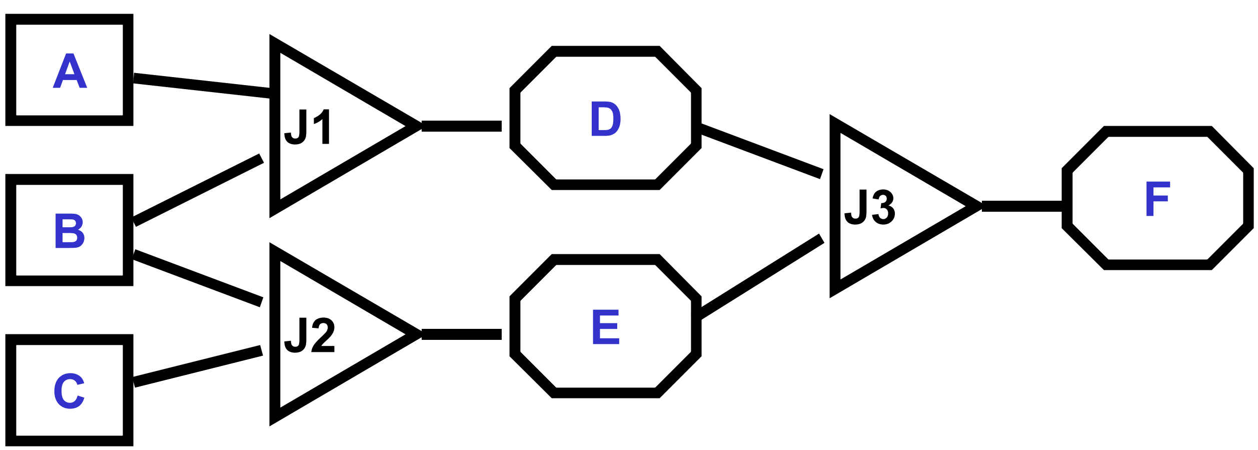


Retracting information

- **Premises, contradictions cannot be retracted**
- **Justifications cannot be retracted**
  - **They comprise the problem solver's cache**
  - **Rules need only be run once for each set of matching data**
- **Assumptions can be retracted**
- **Algorithm:**
  1. **Make assumption OUT**
  2. **Retract all nodes which rely on it (`propagate-outness`)**
  3. **Find alternate support for newly OUT nodes.**

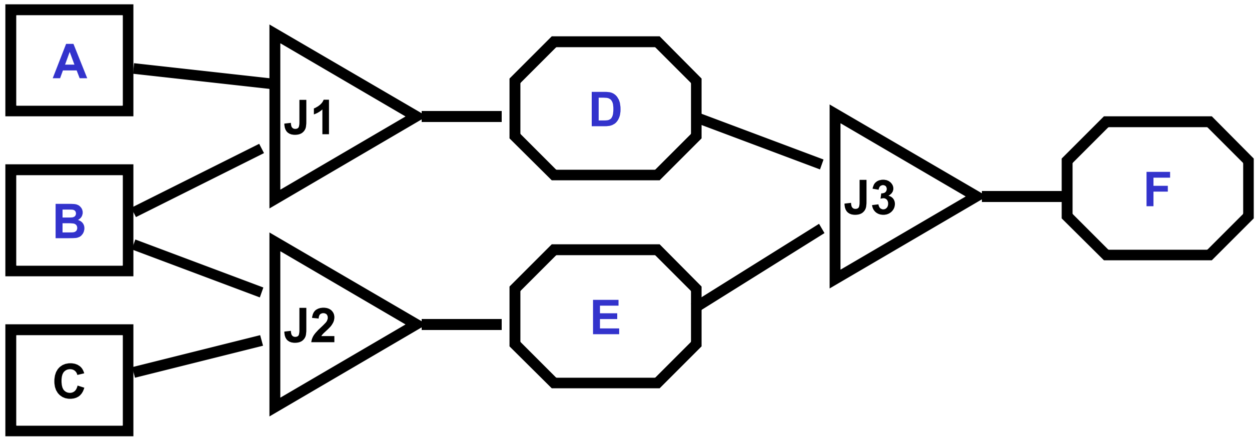
Retraction Example

Initial state:



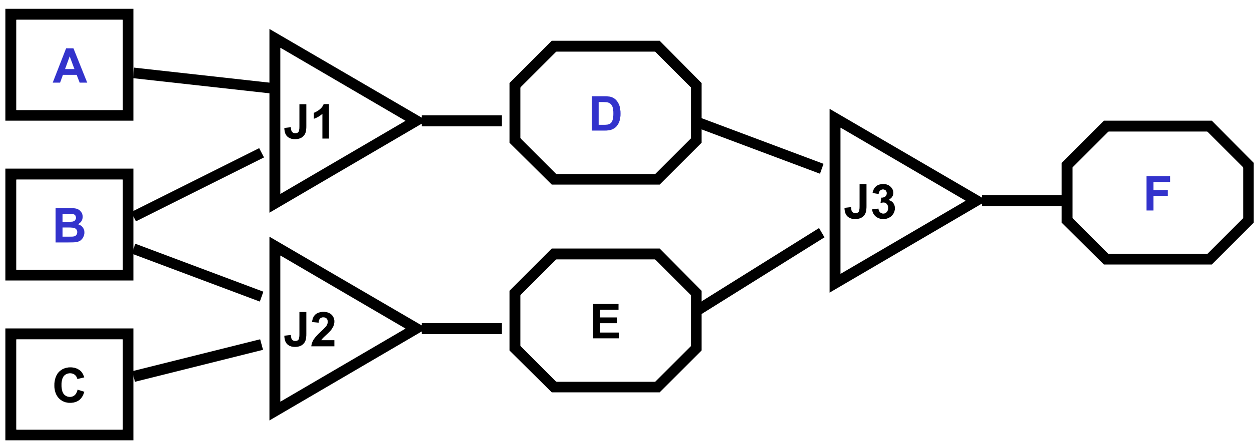
Retraction Example, cont

Retract C:



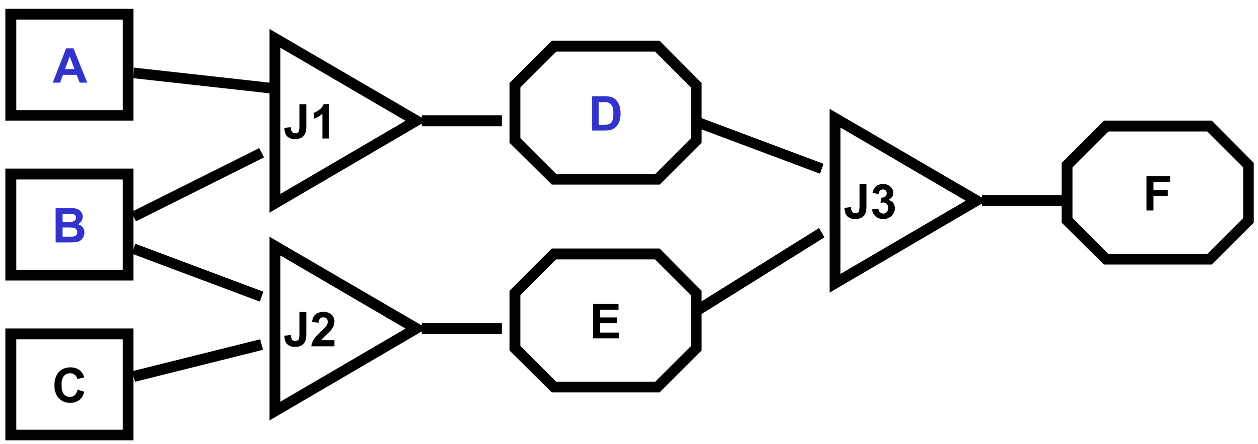
Retraction Example, cont.

E becomes out:



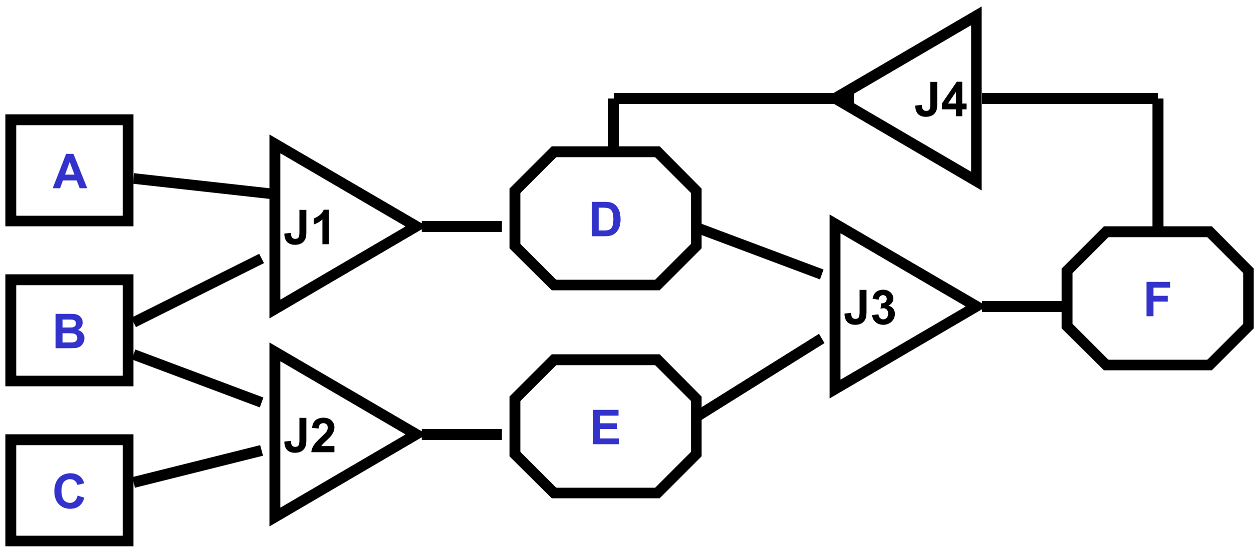
Retraction Example, cont.

F becomes out:

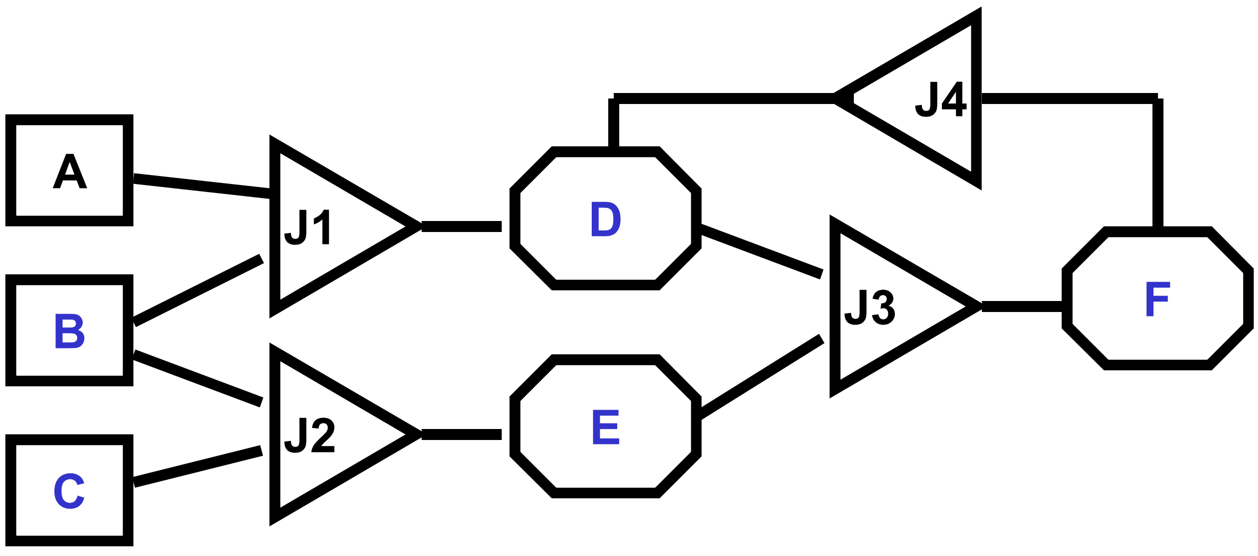


Retract, then Resupport

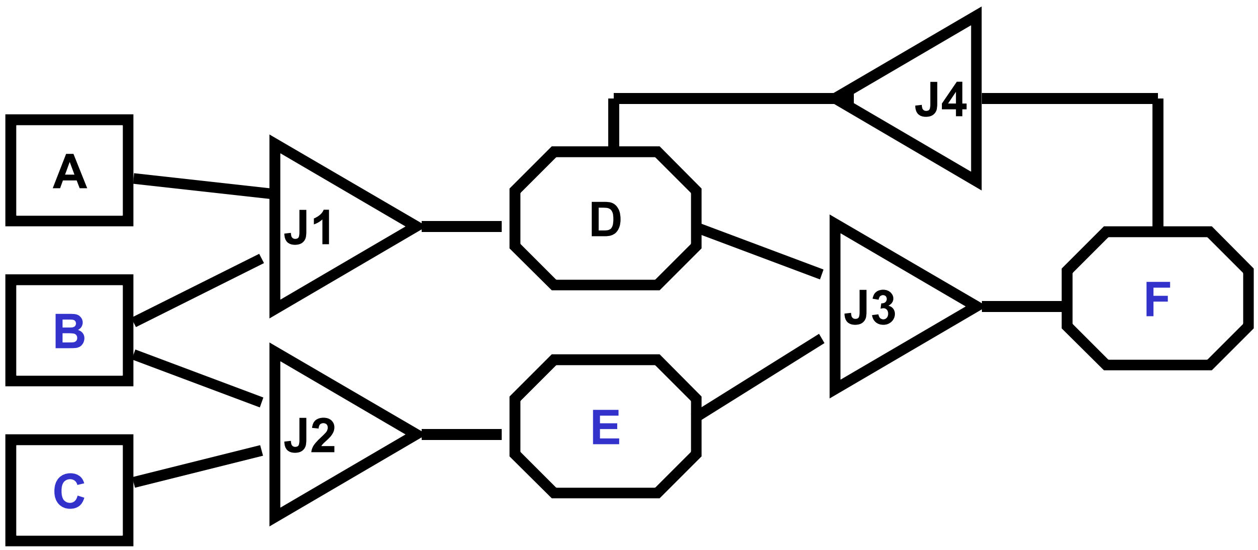
Initial state:



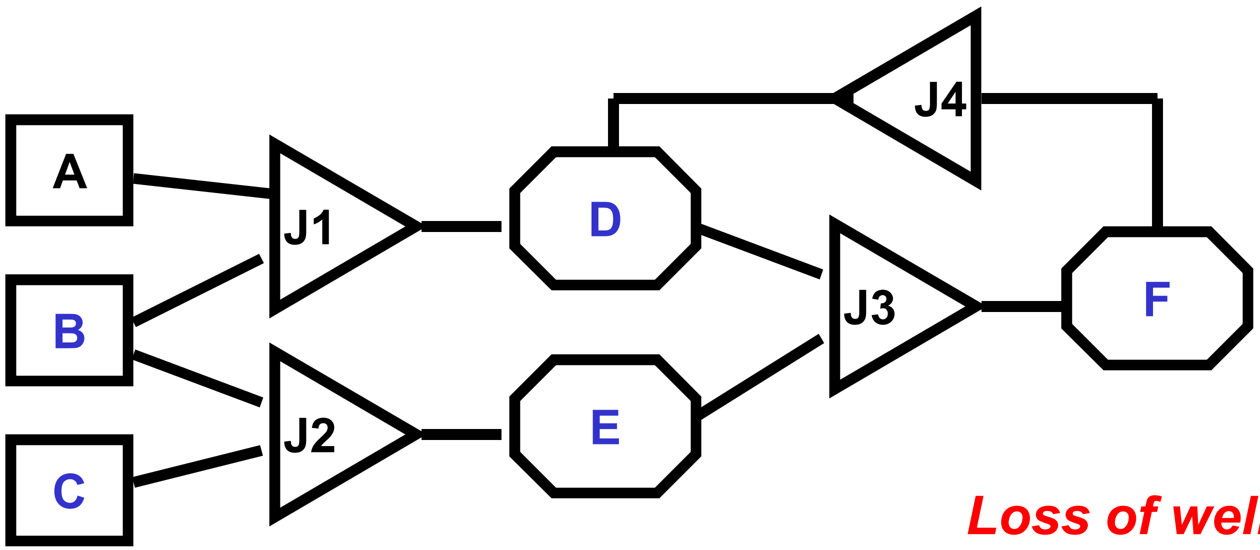
Retract A:



Retract D via J1:



Resupport D via J4:



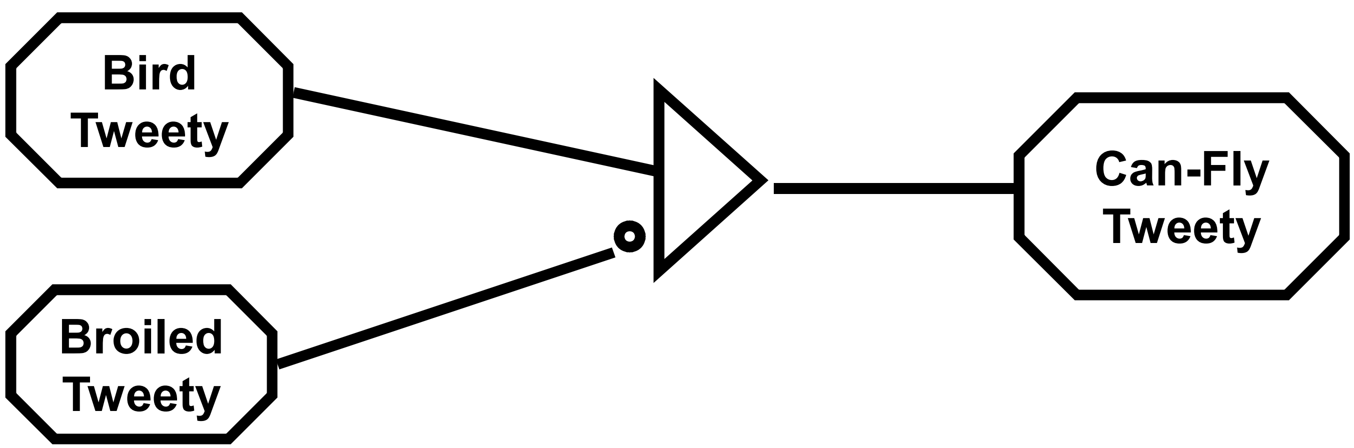
Loss of well-founded support!

Whither Context?

- No explicit representation of context
- Context implicit in union of premises and enabled assumptions
- Advantages
  - Context is often very large
  - Context often changes slowly
- Drawbacks
  - Hard to compare two contexts
  - Context switching can be expensive

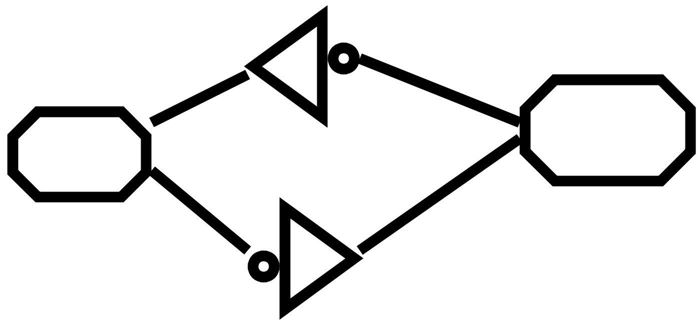
Non-monotonicity

- **Attempt to capture default reasoning**
- **Divide antecedents into *in-list* and *out-list*.**
- **A node is IN if either**
  - **it is an enabled assumption or premise**
  - **at least one justification has all in-list nodes IN and all out-list nodes OUT**



Problems with out-lists

- Beliefs become order-sensitive



- Odd loops don't converge