Corrected Version

- Reasoning Maintenance
- Doyle's Truth Maintenance

Integrating new information

We have already looked at the problem of keeping track of a dynamic (i.e. changing) KB, when new information is a candidate for being added to the KB.

Although we talk about "Knowledge Bases", in practice there are many assumptions and guesses included in a KB, many of which might be wrong, casting doubt on conclusions we get from the KB. If we are using a non-monotonic reasoning engine (e.g. the CWA is involved), then adding new information can invalidate earlier information, even though the KB remains consistent.

How can we keep track of these dependencies?

A quick fix

We can tag any deduced statement with a time tag (e.g. a sequence number).

If a statement deduced at time T is invalidated, for whatever reason, then throw it away, and all subsequent reasoning: go back to time $T-1$ and redo as much as goes through of the previous reasoning.

This is wasteful (but may be fine depending on the speed of the machines, the size of the KB, and the complexity of the reasoning required).

Example

Time T: add "earth is flat"
Time T+1: add "Edinburgh is the capital of Scotland"
Time T+2: contradict "earth is flat"

So throw away second statement unnecessarily!

A better approach

In a truth (or reasoning) system (TMS/RMS):

- Dependencies are represented by a set of *nodes* with *justifications*;
- Each *node* is labelled with a statement in, or deduced from, the KB;
- Each node has zero or more *justifications*, i.e. records of how the statement came to be believed.

There should be only one node for any given statement; thus we build a system of nodes and interlocking justifications.

We can mark some initial statements as *facts* and others as *assumptions* – just means that we look among latter for problem cases.

Two versions of TMS

Doyle's TMS

Suppose our reasoning engine (separate from the TMS) signals that the current information is contradictory (e.g. logically inconsistent). Then something must change in the KB.

- The TMS gives us a choice of which bits of the network might be to blame.
- TMS (or user) picks one of the possibilities.
- TMS updates the network by removing justifications — but keeps a record to allow these to be put back if the blame is shifted later on.

de Kleer's version

Here the TMS works harder

- build up *all possible* justification patterns by forward chaining
- record for a node all the possible justifications
- when there is inconsistency, get choice of assumptions to blame, as before
- allows efficient propagation of new valid justifications.

Today, we look at Doyle's TMS.

Doyle's TMS

- Justifications are marked *valid* or *invalid*;
- Nodes are marked *IN* if they have 1 or more valid justifications, otherwise marked *OUT*;

Practically, can omit OUT nodes after some time to save space.
Sometimes OUT nodes are important, because a justification for a node being IN may be that another node is OUT (as in CWA).

Kinds of Justifications

In this sort of TMS:

support list Justify by `SL(INlist, OUTlist)`:
this says that the statement associated with the node is justified provided all the nodes in the `INlist` are IN, and all the nodes in the `OUTlist` are OUT.

conditional proof Justify by `CP(node, INlist, OUTlist)`:
this says that the mentioned `node` is IN, provided all the nodes in the `INlist` are IN, and all the nodes in the `OUTlist` are OUT.

The IN/OUT here tells us whether node's statement is currently *believed* to be true.

- A statement can be made a fact by attaching the justification $\mathtt{SL}([],[])$ which means it is always IN.
- Nodes with non-empty OUTlists are called *assumptions* — they can be invalidated.
- Conditional proof justifications are useful in supporting deduced rules:

Node	*Statement*	*Justifications*
n_1	A	??
n_2	B	??
n_3	$A \rightarrow B$	$\mathtt{CP}(\mathtt{n}_2,[\mathtt{n}_1],[\,])$
n_4	C	??
n_5	$A \wedge C \rightarrow B$	$\mathtt{CP}(\mathtt{n}_2,[\mathtt{n}_1,\mathtt{n}_4],[\,])$

Using these

A TMS is *not* a reasoning system – it depends on a reasoning system to to inference.

This supports a form of default reasoning. Suppose that n_1, n_2, n_3 correspond to $bird(tweety)$, $ostrich(tweety)$, $fly(tweety)$. Then justify n_3 with

$$\mathtt{SL}([\mathtt{n}_1],[\mathtt{n}_2])$$

Then the justification will be applicable only if n_2 is OUT, i.e. there is no current justification for n_2.

If there are many anomalous cases, we can collect them together with a new node, justified by each particular case.

Looking for inconsistency

Note that checking for contradiction (inconsistency) is the job of the reasoning engine, not the TMS.

However, the TMS can help the reasoning system in adapting the KB because it has a record of the dependencies in the reasoning patterns.

Next, look at the algorithm used by this form of TMS.

TMS algorithm

Suppose that node N is currently IN and causing problems (e.g. it is a contradiction).

- Trace the dependencies backward from N (which is currently IN and causing trouble) to find the set of assumptions that support it. Find the maximal ones — those that do not support other assumptions. Suppose that these are nodes $A_1, \ldots, A_n$.
- Create a new "No Good" node NG to represent that the A_i together are cause a problem; label with "not $(A_1 \wedge \cdots \wedge A_n)$", and give it justification

$$\mathtt{CP}(\mathtt{N},[\mathtt{A}_1,\ldots,\mathtt{A}_n],[\,])$$

Algorithm ctd

- Pick one of the As at random, say A_i. We want to change things so that N becomes out.
- A_i has a valid justification with a non-empty OUTlist (since it is an assumption). Find the nodes $D_1, \ldots, D_k$ which are currently OUT, and which would make A_i OUT if any of the Ds became IN. Pick one, say D_j.
- Give D_j a new justification, so it becomes IN, e.g.

$$\mathtt{SL}([\mathtt{NG}, \mathtt{A_1}, \ldots, \mathtt{A_n}], [\mathtt{D_1}, \ldots, \mathtt{D_{j-1}}, \mathtt{D_{j+1}}, \ldots, \mathtt{D_k}])$$

Now, D_j is IN, so A_i is OUT, so N is OUT.

Note that:

- the node NG records the contradiction that we got into, and it remains in the TMS to watch out for the problem.
- If any D comes IN, or A goes out, then D_j will be OUT, so we can later overrule the arbitrary choices made here.

Example

Suppose we have time-tabling problem, given the following:

Node	*Statement*	*Justifications*
n_1	time = 10am	$\mathtt{SL}([\,], [\mathtt{n_2}])$
n_2	time $\neq$ 10am	...
n_3	place = FH	$\mathtt{SL}([\,], [\mathtt{n_4}])$
n_4	place = AT	...

The current IN nodes are $\{ n_1, n_3 \}$. Suppose that the reasoning system spots a problem:

n_5	the snag	$\mathtt{SL}([\mathtt{n_1}, \mathtt{n_3}], [\,])$

i.e. the combination of the currently IN nodes leads to something not possible.
The IN set is now $\{ n_1, n_3, n_5 \}$.

Example ctd

The maximal assumptions for n_5 are n_1 and n_3.
We now introduce the "no good" node:

n_6	not n_1 and n_3	$\mathtt{CP}(\mathtt{n_5}, [\mathtt{n_1}, \mathtt{n_3}], [\,])$

Pick on n_3: it has one OUT node n_4, so we bring it IN, by adding the justification:

n_4	place = AT	$\mathtt{SL}([\mathtt{n_6}, \mathtt{n_1}], [\,])$

The IN set is now $\{ n_1, n_4, n_6 \}$ and the problem is cured.

Choosing the culprit

Rather than picking nodes at random, the reasoning system may help the TMS to make a more rational choice. We will look at some belief revision ideas later to see examples of this, in a KB extended with preferences between statements saying which should be given priority during reasoning maintenance.

There can be problems of looping here – making one node OUT can result in it coming back IN via a different justification, and vice versa. (This is like the problem of making sense of logic programs that use negation outside the situation prescribed by the CWA.)

Summary

We looked at the general problem of Reasoning Maintenance.

In particular, Doyle's TMS:

- Nodes IN/OUT, with justifications
- Support list and Conditional justifications
- Algorithm to repair problematic KB
- Add record of problem situation