

Chapter 5: Propositions and Inference

- An interpretation is an assignment of values to all variables.
- A model is an interpretation that satisfies the constraints.
- Often we don't want to just find a model, but want to know what is true in all models.
- A proposition is statement that is true or false in each interpretation.

Why propositions?

- Specifying logical formulae is often more natural than filling in tables
- It is easier to check correctness and debug formulae than tables
- We can exploit the Boolean nature for efficient reasoning
- We need a language for asking queries (of what follows in all models) that may be more complicated than asking for the value of a variable
- It is easy to incrementally add formulae
- It can be extended to infinitely many variables with infinite domains (using logical quantification)

Human's view of semantics

Step 1 Begin with a task domain.

Step 2 Choose atoms in the computer to denote propositions. These atoms have meaning to the KB designer.

Step 3 Tell the system knowledge about the domain.

Step 4 Ask the system questions.

— the system can tell you whether the question is a logical consequence.

— You can interpret the answer with the meaning associated with the atoms.

In computer:

$light1_broken \leftarrow sw_up$
 $\quad \wedge power \wedge unlit_light1.$

$sw_up.$

$power \leftarrow lit_light2.$

$unlit_light1.$

$lit_light2.$

In user's mind:

- $light1_broken$: light #1 is broken
- sw_up : switch is up
- $power$: there is power in the building
- $unlit_light1$: light #1 isn't lit
- lit_light2 : light #2 is lit

Conclusion: $light1_broken$

- The computer doesn't know the meaning of the symbols
- The user can interpret the symbol using their meaning

Simple language: propositional definite clauses

- An **atom** is a symbol starting with a lower case letter
- A **body** is an atom or is of the form $b_1 \wedge b_2$ where b_1 and b_2 are bodies.
- A **definite clause** is an atom or is a rule of the form $h \leftarrow b$ where h is an atom and b is a body.
- A **knowledge base** is a set of definite clauses

- An **interpretation** I assigns a truth value to each atom.
- A body $b_1 \wedge b_2$ is true in I if b_1 is true in I and b_2 is true in I .
- A rule $h \leftarrow b$ is false in I if b is true in I and h is false in I .
The rule is true otherwise.
- A knowledge base KB is true in I if and only if every clause in KB is true in I .

- A **model** of a set of clauses is an interpretation in which all the clauses are *true*.
- If KB is a set of clauses and g is a conjunction of atoms, g is a **logical consequence** of KB , written $KB \models g$, if g is *true* in every model of KB .
- That is, $KB \models g$ if there is no interpretation in which KB is *true* and g is *false*.

Simple Example

$$KB = \begin{cases} p \leftarrow q. \\ q. \\ r \leftarrow s. \end{cases}$$

	<i>p</i>	<i>q</i>	<i>r</i>	<i>s</i>
<i>l</i> ₁	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>
<i>l</i> ₂	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>
<i>l</i> ₃	<i>true</i>	<i>true</i>	<i>false</i>	<i>false</i>
<i>l</i> ₄	<i>true</i>	<i>true</i>	<i>true</i>	<i>false</i>
<i>l</i> ₅	<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>

model?

Simple Example

$$KB = \begin{cases} p \leftarrow q. \\ q. \\ r \leftarrow s. \end{cases}$$

	<i>p</i>	<i>q</i>	<i>r</i>	<i>s</i>	model?
<i>l</i> ₁	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	is a model of <i>KB</i>
<i>l</i> ₂	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	not a model of <i>KB</i>
<i>l</i> ₃	<i>true</i>	<i>true</i>	<i>false</i>	<i>false</i>	is a model of <i>KB</i>
<i>l</i> ₄	<i>true</i>	<i>true</i>	<i>true</i>	<i>false</i>	is a model of <i>KB</i>
<i>l</i> ₅	<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	not a model of <i>KB</i>

Simple Example

$$KB = \begin{cases} p \leftarrow q. \\ q. \\ r \leftarrow s. \end{cases}$$

	<i>p</i>	<i>q</i>	<i>r</i>	<i>s</i>	model?
<i>l</i> ₁	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	is a model of <i>KB</i>
<i>l</i> ₂	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	not a model of <i>KB</i>
<i>l</i> ₃	<i>true</i>	<i>true</i>	<i>false</i>	<i>false</i>	is a model of <i>KB</i>
<i>l</i> ₄	<i>true</i>	<i>true</i>	<i>true</i>	<i>false</i>	is a model of <i>KB</i>
<i>l</i> ₅	<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	not a model of <i>KB</i>

Which of *p*, *q*, *r*, *s* logically follow from *KB*?

Simple Example

$$KB = \begin{cases} p \leftarrow q. \\ q. \\ r \leftarrow s. \end{cases}$$

	<i>p</i>	<i>q</i>	<i>r</i>	<i>s</i>	model?
<i>l</i> ₁	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	is a model of <i>KB</i>
<i>l</i> ₂	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	not a model of <i>KB</i>
<i>l</i> ₃	<i>true</i>	<i>true</i>	<i>false</i>	<i>false</i>	is a model of <i>KB</i>
<i>l</i> ₄	<i>true</i>	<i>true</i>	<i>true</i>	<i>false</i>	is a model of <i>KB</i>
<i>l</i> ₅	<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	not a model of <i>KB</i>

Which of *p*, *q*, *r*, *q* logically follow from *KB*?

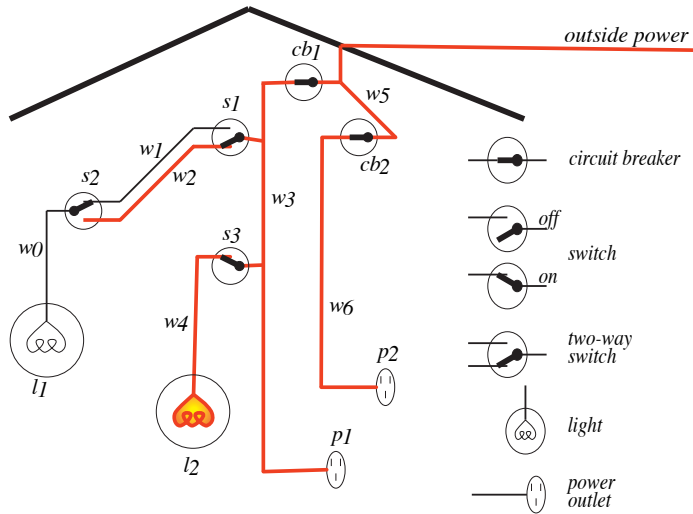
$KB \models p$, $KB \models q$, $KB \not\models r$, $KB \not\models s$

User's view of Semantics

1. Choose a task domain: **intended interpretation.**
2. Associate an atom with each proposition you want to represent.
3. Tell the system clauses that are true in the intended interpretation: **axiomatizing the domain.**
4. Ask questions about the intended interpretation.
5. If $KB \models g$, then g must be true in the intended interpretation.
6. Users can interpret the answer using their intended interpretation of the symbols.

- The computer doesn't have access to the intended interpretation.
- All it knows is the knowledge base.
- The computer can determine if a formula is a logical consequence of KB.
- If $KB \models g$ then g must be true in the intended interpretation.
- If $KB \not\models g$ then there is a model of KB in which g is false. This could be the intended interpretation.

Electrical Environment



Representing the Electrical Environment

light_l1.

light_l2.

down_s1.

up_s2.

up_s3.

ok_l1.

ok_l2.

ok_cb1.

ok_cb2.

live_outside.

lit_l1 \leftarrow *light_l1* \wedge *live_w0* \wedge *ok_l1*

live_w0 \leftarrow *live_w1* \wedge *up_s2.*

live_w0 \leftarrow *live_w2* \wedge *down_s2.*

live_w1 \leftarrow *live_w3* \wedge *up_s1.*

live_w2 \leftarrow *live_w3* \wedge *down_s1.*

lit_l2 \leftarrow *light_l2* \wedge *live_w4* \wedge *ok_l2.*

live_w4 \leftarrow *live_w3* \wedge *up_s3.*

live_p1 \leftarrow *live_w3.*

live_w3 \leftarrow *live_w5* \wedge *ok_cb1.*

live_p2 \leftarrow *live_w6.*

live_w6 \leftarrow *live_w5* \wedge *ok_cb2.*

live_w5 \leftarrow *live_outside.*

- A **proof** is a mechanically derivable demonstration that a formula logically follows from a knowledge base.
- Given a proof procedure, $KB \vdash g$ means g can be derived from knowledge base KB .
- Recall $KB \models g$ means g is true in all models of KB .
- A proof procedure is **sound** if $KB \vdash g$ implies $KB \models g$.
- A proof procedure is **complete** if $KB \models g$ implies $KB \vdash g$.

One **rule of derivation**, a generalized form of *modus ponens*:

If " $h \leftarrow b_1 \wedge \dots \wedge b_m$ " is a clause in the knowledge base, and each b_i has been derived, then h can be derived.

This is **forward chaining** on this clause.

(This rule also covers the case when $m = 0$.)

Bottom-up proof procedure

$KB \vdash g$ if $g \in C$ at the end of this procedure:

$C := \{\}$;

repeat

select clause " $h \leftarrow b_1 \wedge \dots \wedge b_m$ " in KB such that

$b_i \in C$ for all i , and

$h \notin C$;

$C := C \cup \{h\}$

until no more clauses can be selected.

Example

$$a \leftarrow b \wedge c.$$

$$a \leftarrow e \wedge f.$$

$$b \leftarrow f \wedge k.$$

$$c \leftarrow e.$$

$$d \leftarrow k.$$

$$e.$$

$$f \leftarrow j \wedge e.$$

$$f \leftarrow c.$$

$$j \leftarrow c.$$

Soundness of bottom-up proof procedure

If $KB \vdash g$ then $KB \models g$.

- Suppose there is a g such that $KB \vdash g$ and $KB \not\models g$.
- Then there must be a first atom added to C that isn't true in every model of KB . Call it h . Suppose h isn't true in model I of KB .
- There must be a clause in KB of form

$$h \leftarrow b_1 \wedge \dots \wedge b_m$$

Each b_i is true in I . h is false in I . So this clause is false in I . Therefore I isn't a model of KB .

- Contradiction.

- The C generated at the end of the bottom-up algorithm is called a **fixed point**.
- Let I be the interpretation in which every element of the fixed point is true and every other atom is false.
- I is a model of KB .
Proof: suppose $h \leftarrow b_1 \wedge \dots \wedge b_m$ in KB is false in I . Then h is false and each b_j is true in I . Thus h can be added to C .
Contradiction to C being the fixed point.
- I is called a **Minimal Model**.

If $KB \models g$ then $KB \vdash g$.

- Suppose $KB \models g$. Then g is true in all models of KB .
- Thus g is true in the minimal model.
- Thus g is in the fixed point.
- Thus g is generated by the bottom up algorithm.
- Thus $KB \vdash g$.

Top-down Definite Clause Proof Procedure

Idea: search backward from a query to determine if it is a logical consequence of KB .

An **answer clause** is of the form:

$$yes \leftarrow a_1 \wedge a_2 \wedge \dots \wedge a_m$$

The **SLD Resolution** of this answer clause on atom a_i with the clause:

$$a_i \leftarrow b_1 \wedge \dots \wedge b_p$$

is the answer clause

$$yes \leftarrow a_1 \wedge \dots \wedge a_{i-1} \wedge b_1 \wedge \dots \wedge b_p \wedge a_{i+1} \wedge \dots \wedge a_m.$$

- An **answer** is an answer clause with $m = 0$. That is, it is the answer clause $\text{yes} \leftarrow$.
- A **derivation** of query “ $?q_1 \wedge \dots \wedge q_k$ ” from KB is a sequence of answer clauses $\gamma_0, \gamma_1, \dots, \gamma_n$ such that
 - ▶ γ_0 is the answer clause $\text{yes} \leftarrow q_1 \wedge \dots \wedge q_k$,
 - ▶ γ_i is obtained by resolving γ_{i-1} with a clause in KB , and
 - ▶ γ_n is an answer.

To solve the query $?q_1 \wedge \dots \wedge q_k$:

$ac := \text{"yes"} \leftarrow q_1 \wedge \dots \wedge q_k$

repeat

select atom a_i from the body of ac ;

choose clause C from KB with a_i as head;

 replace a_i in the body of ac by the body of C

until ac is an answer.

Nondeterministic Choice

- **Don't-care nondeterminism** If one selection doesn't lead to a solution, there is no point trying other alternatives. **select**
- **Don't-know nondeterminism** If one choice doesn't lead to a solution, other choices may. **choose**

Example: successful derivation

$a \leftarrow b \wedge c.$

$c \leftarrow e.$

$f \leftarrow j \wedge e.$

$a \leftarrow e \wedge f.$

$d \leftarrow k.$

$f \leftarrow c.$

$b \leftarrow f \wedge k.$

$e.$

$j \leftarrow c.$

Query: ?a

$\gamma_0 : \text{yes} \leftarrow a$

$\gamma_1 : \text{yes} \leftarrow e \wedge f$

$\gamma_2 : \text{yes} \leftarrow f$

$\gamma_3 : \text{yes} \leftarrow c$

$\gamma_4 : \text{yes} \leftarrow e$

$\gamma_5 : \text{yes} \leftarrow$

Example: failing derivation

$a \leftarrow b \wedge c.$

$c \leftarrow e.$

$f \leftarrow j \wedge e.$

$a \leftarrow e \wedge f.$

$d \leftarrow k.$

$f \leftarrow c.$

$b \leftarrow f \wedge k.$

$e.$

$j \leftarrow c.$

Query: ?a

$\gamma_0 : \text{yes} \leftarrow a$

$\gamma_1 : \text{yes} \leftarrow b \wedge c$

$\gamma_2 : \text{yes} \leftarrow f \wedge k \wedge c$

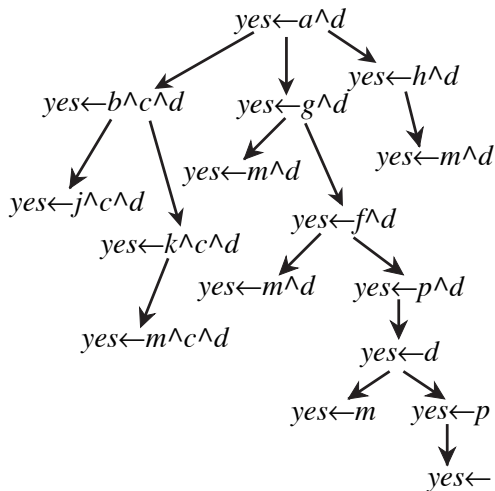
$\gamma_3 : \text{yes} \leftarrow c \wedge k \wedge c$

$\gamma_4 : \text{yes} \leftarrow e \wedge k \wedge c$

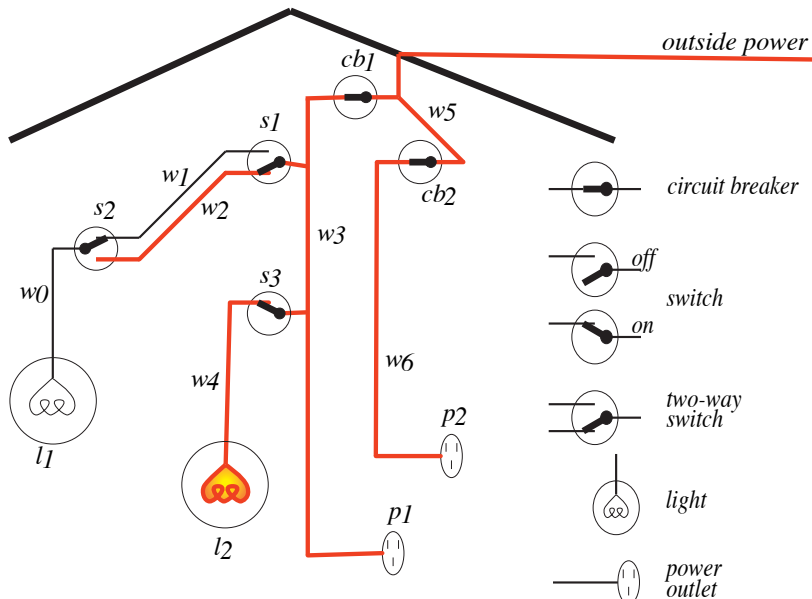
$\gamma_5 : \text{yes} \leftarrow k \wedge c$

Search Graph for SLD Resolution

$a \leftarrow b \wedge c.$	$a \leftarrow g.$
$a \leftarrow h.$	$b \leftarrow j.$
$b \leftarrow k.$	$d \leftarrow m.$
$d \leftarrow p.$	$f \leftarrow m.$
$f \leftarrow p.$	$g \leftarrow m.$
$g \leftarrow f.$	$k \leftarrow m.$
$h \leftarrow m.$	$p.$
$?a \wedge d$	



Electrical Domain



Representing the Electrical Environment

light_l1.

light_l2.

down_s1.

up_s2.

up_s3.

ok_l1.

ok_l2.

ok_cb1.

ok_cb2.

live_outside.

lit_l1 \leftarrow *light_l1* \wedge *live_w0* \wedge *ok_l1*

live_w0 \leftarrow *live_w1* \wedge *up_s2.*

live_w0 \leftarrow *live_w2* \wedge *down_s2.*

live_w1 \leftarrow *live_w3* \wedge *up_s1.*

live_w2 \leftarrow *live_w3* \wedge *down_s1.*

lit_l2 \leftarrow *light_l2* \wedge *live_w4* \wedge *ok_l2.*

live_w4 \leftarrow *live_w3* \wedge *up_s3.*

live_p1 \leftarrow *live_w3.*

live_w3 \leftarrow *live_w5* \wedge *ok_cb1.*

live_p2 \leftarrow *live_w6.*

live_w6 \leftarrow *live_w5* \wedge *ok_cb2.*

live_w5 \leftarrow *live_outside.*

- In the electrical domain, what should the house builder know?
- What should an occupant know?

- In the electrical domain, what should the house builder know?
- What should an occupant know?
- Users can't be expected to volunteer knowledge:
 - ▶ They don't know what information is needed.
 - ▶ They don't know what vocabulary to use.

- Users can provide observations to the system. They can answer specific queries.
- **Askable** atoms are those that a user should be able to observe.
 - ▶ instead of trying to prove them, the system asks the user
- There are 3 sorts of goals in the top-down proof procedure:
 - ▶ Goals for which the user isn't expected to know the answer.
 - ▶ Askable atoms that may be useful in the proof.
 - ▶ Askable atoms that the user has already provided information about.

- Users can provide observations to the system. They can answer specific queries.
- **Askable** atoms are those that a user should be able to observe.
 - ▶ instead of trying to prove them, the system asks the user
- There are 3 sorts of goals in the top-down proof procedure:
 - ▶ Goals for which the user isn't expected to know the answer.
 - ▶ Askable atoms that may be useful in the proof.
 - ▶ Askable atoms that the user has already provided information about.
- The top-down proof procedure can be modified to ask users about askable atoms they have not already provided answers for.
- With the bottom-up proof procedure care has to be taken to only ask relevant questions.

User: ask *lit_l1*.

System: Is *up_s1* true? [yes,no,unknown,why,help]

User: no.

System: Is *down_s1* true? [yes,no,unknown,why,help]

User: yes.

System: Is *down_s2* true? [yes,no,unknown,why,help]

User: yes.

System: *lit_l1* is true.

Knowledge-Level Explanation

- **HOW** questions can be used to ask how an atom was proved. It gives the rule used to prove the atom. You can then ask HOW an element of the body of that rules was proved. This lets the user explore the proof.

User: ask *lit_l2*.

System: *lit_l2* is true.

User: how

System: $lit_l_2 \leftarrow light_l_2 \wedge live_l_2 \wedge ok_l_2.$

User : how 2.

System: $live_l_2 \leftarrow live_w_4.$

User: how 1.

System: $live_w_4 \leftarrow live_w_3 \wedge up_s_3.$

. . .

Knowledge-Level Explanation

- **WHY** questions can be used to ask why a question was asked. It provides the rule with the asked atom in the body. You can ask WHY the rule in the head was asked.

User: ask lit_l_1 .

System: Is up_s_1 true?

User: why.

System : up_s_1 is used in the rule
 $live_w_1 \leftarrow live_w_3 \wedge up_s_1$.

User: why.

System: $live_w_1$ is used in the rule
 $live_w_0 \leftarrow live_w_1 \wedge up_s_2$.

User: why.

System: $live_w_0$ is used in the rule
 $live_l_1 \leftarrow live_w_0$.

...

There are four types of non-syntactic errors that can arise in rule-based systems:

- An incorrect answer is produced: an atom that is false in the intended interpretation was derived.
- Some answer wasn't produced: the proof failed when it should have succeeded. Some particular true atom wasn't derived.
- The program gets into an infinite loop.
- The system asks irrelevant questions.

Debugging incorrect answers

- Suppose atom g was proved but is false in the intended interpretation.
- There must be a rule $g \leftarrow a_1 \wedge \dots \wedge a_k$ in the knowledge base that was used to prove g .
- Either:
 - ▶ one of the a_i is false in the intended interpretation or
 - ▶ all of the a_i are true in the intended interpretation.
- Incorrect answers can be debugged by only answering yes/no questions.

If atom g is true in the intended interpretation, but could not be proved, either:

- There is no appropriate rule for g .
- There is a rule $g \leftarrow a_1 \wedge \dots \wedge a_k$ that should have succeeded.

If atom g is true in the intended interpretation, but could not be proved, either:

- There is no appropriate rule for g .
- There is a rule $g \leftarrow a_1 \wedge \dots \wedge a_k$ that should have succeeded.
 - ▶ One of the a_i is true in the interpretation but could not be proved.

Integrity Constraints

- In the electrical domain, what if we predict that a light should be on, but observe that it isn't?
What can we conclude?

Integrity Constraints

- In the electrical domain, what if we predict that a light should be on, but observe that it isn't?
What can we conclude?
- We will expand the definite clause language to include **integrity constraints** which are rules that imply *false*, where *false* is an atom that is false in all interpretations.
- This will allow us to make conclusions from a contradiction.
- A definite clause knowledge base is always consistent. This will no longer be true with rules that imply *false*.

- An **integrity constraint** is a clause of the form

$$false \leftarrow a_1 \wedge \dots \wedge a_k$$

where the a_i are atoms and *false* is a special atom that is false in all interpretations.

- A **Horn clause** is either a definite clause or an integrity constraint.

Negative Conclusions

- Negations can follow from a Horn clause KB.
- The negation of α , written $\neg\alpha$ is a formula that
 - ▶ is true in interpretation I if α is false in I , and
 - ▶ is false in interpretation I if α is true in I .
- **Example:**

$$KB = \left\{ \begin{array}{l} \text{false} \leftarrow a \wedge b. \\ a \leftarrow c. \\ b \leftarrow c. \end{array} \right\} \quad KB \models \neg c.$$

Disjunctive Conclusions

- Disjunctions can follow from a Horn clause KB.
- The disjunction of α and β , written $\alpha \vee \beta$, is
 - ▶ true in interpretation I if α is true in I or β is true in I (or both are true in I).
 - ▶ false in interpretation I if α and β are both false in I .
- Example:

$$KB = \left\{ \begin{array}{l} \text{false} \leftarrow a \wedge b. \\ a \leftarrow c. \\ b \leftarrow d. \end{array} \right\} \quad KB \models \neg c \vee \neg d.$$

- An **assumable** is an atom whose negation you are prepared to accept as part of a (disjunctive) answer.
- A **conflict** of KB is a set of assumables that, given KB imply *false*.
- A **minimal conflict** is a conflict such that no strict subset is also a conflict.

Conflict Example

Example: If $\{c, d, e, f, g, h\}$ are the assumables

$$KB = \left\{ \begin{array}{l} \text{false} \leftarrow a \wedge b. \\ a \leftarrow c. \\ b \leftarrow d. \\ b \leftarrow e. \end{array} \right\}$$

- $\{c, d\}$ is a conflict
- $\{c, e\}$ is a conflict
- $\{c, d, e\}$ is a conflict
- $\{c, d, e, h\}$ is a conflict

Using Conflicts for Diagnosis

- Assume that the user is able to observe whether a light is lit or dark and whether a power outlet is dead or live.
- A light can't be both lit and dark. An outlet can't be both live and dead:

false \leftarrow *dark*_{l1} & *lit*_{l1}.

false \leftarrow *dark*_{l2} & *lit*_{l2}.

false \leftarrow *dead*_{p1} & *live*_{p2}.

- Assume the individual components are working correctly:

*assumable ok*_{l1}.

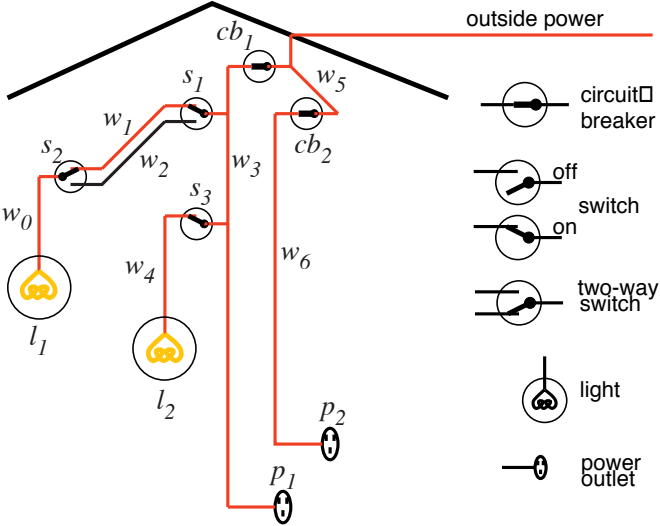
*assumable ok*_{s2}.

...

- Suppose switches *s*₁, *s*₂, and *s*₃ are all up:

*up*_{s1}. *up*_{s2}. *up*_{s3}.

Electrical Environment



Representing the Electrical Environment

light_l1.

light_l2.

up_s1.

up_s2.

up_s3.

live_outside.

lit_l1 \leftarrow *light_l1* \wedge *live_w0* \wedge *ok_l1*.

live_w0 \leftarrow *live_w1* \wedge *up_s2* \wedge *ok_s2*.

live_w0 \leftarrow *live_w2* \wedge *down_s2* \wedge *ok_s2*.

live_w1 \leftarrow *live_w3* \wedge *up_s1* \wedge *ok_s1*.

live_w2 \leftarrow *live_w3* \wedge *down_s1* \wedge *ok_s1*.

lit_l2 \leftarrow *light_l2* \wedge *live_w4* \wedge *ok_l2*.

live_w4 \leftarrow *live_w3* \wedge *up_s3* \wedge *ok_s3*.

live_p1 \leftarrow *live_w3*.

live_w3 \leftarrow *live_w5* \wedge *ok_cb1*.

live_p2 \leftarrow *live_w6*.

live_w6 \leftarrow *live_w5* \wedge *ok_cb2*.

live_w5 \leftarrow *live_outside*.

- If the user has observed l_1 and l_2 are both dark:

$dark_{-l_1}. dark_{-l_2}.$

- There are two minimal conflicts:

$\{ok_{cb_1}, ok_{s_1}, ok_{s_2}, ok_{l_1}\}$ and

$\{ok_{cb_1}, ok_{s_3}, ok_{l_2}\}.$

- You can derive:

$\neg ok_{cb_1} \vee \neg ok_{s_1} \vee \neg ok_{s_2} \vee \neg ok_{l_1}$

$\neg ok_{cb_1} \vee \neg ok_{s_3} \vee \neg ok_{l_2}.$

- Either cb_1 is broken or there is one of six double faults.

$$\neg ok_{cb_1} \vee$$

$$\neg ok_{s_1} \wedge \neg ok_{s_3} \vee$$

$$\neg ok_{s_1} \wedge \neg ok_{l_2} \vee$$

$$\neg ok_{s_2} \wedge \neg ok_{s_3} \vee$$

$$\neg ok_{s_2} \wedge \neg ok_{l_2} \vee$$

$$\neg ok_{l_1} \wedge \neg ok_{s_3} \vee$$

$$\neg ok_{l_1} \wedge \neg ok_{l_2} \vee$$

- A **consistency-based diagnosis** is a set of assumables that has at least one element in each conflict.
- A **minimal diagnosis** is a diagnosis such that no subset is also a diagnosis.
- Intuitively, one of the minimal diagnoses must hold. A diagnosis holds if all of its elements are false.
- **Example:** For the preceding example there are seven minimal diagnoses: $\{ok_cb_1\}$, $\{ok_s_1, ok_s_3\}$, $\{ok_s_1, ok_l_2\}$, $\{ok_s_2, ok_s_3\}, \dots$

To solve the query $?q_1 \wedge \dots \wedge q_k$:

$ac := \text{“yes} \leftarrow q_1 \wedge \dots \wedge q_k\text{”}$

repeat

select atom a_i from the body of ac ;

choose clause C from KB with a_i as head;

 replace a_i in the body of ac by the body of C

until ac is an answer.

Implementing conflict finding: top down

- Query is *false*.
- Don't select an atom that is assumable.
- Stop when all of the atoms in the body of the generalised query are assumable:
 - ▶ this is a conflict

Example

$false \leftarrow a.$

$a \leftarrow b \& c.$

$b \leftarrow d.$

$b \leftarrow e.$

$c \leftarrow f.$

$c \leftarrow g.$

$e \leftarrow h \& w.$

$e \leftarrow g.$

$w \leftarrow f.$

assumable $d, f, g, h.$

- **Conclusions** are pairs $\langle a, A \rangle$, where a is an atom and A is a set of assumables that imply a .
- Initially, conclusion set $C = \{\langle a, \{a\} \rangle : a \text{ is assumable}\}$.
- If there is a rule $h \leftarrow b_1 \wedge \dots \wedge b_m$ such that for each b_i there is some A_i such that $\langle b_i, A_i \rangle \in C$, then $\langle h, A_1 \cup \dots \cup A_m \rangle$ can be added to C .
- If $\langle a, A_1 \rangle$ and $\langle a, A_2 \rangle$ are in C , where $A_1 \subset A_2$, then $\langle a, A_2 \rangle$ can be removed from C .
- If $\langle \text{false}, A_1 \rangle$ and $\langle a, A_2 \rangle$ are in C , where $A_1 \subseteq A_2$, then $\langle a, A_2 \rangle$ can be removed from C .

```
C := {⟨a, {a}⟩ : a is assumable };  
repeat  
  select clause “ $h \leftarrow b_1 \wedge \dots \wedge b_m$ ” in  $T$  such that  
    ⟨ $b_i, A_i$ ⟩ ∈ C for all  $i$  and  
    there is no ⟨ $h, A'$ ⟩ ∈ C or ⟨false,  $A'$ ⟩ ∈ C  
      such that  $A' \subseteq A$  where  $A = A_1 \cup \dots \cup A_m$ ;  
  C := C ∪ {⟨ $h, A$ ⟩}  
  Remove any elements of C that can now be pruned;  
until no more selections are possible
```

Complete Knowledge Assumption

- Often you want to assume that your knowledge is complete.
- **Example:** you can state what switches are up and the agent can assume that the other switches are down.
- **Example:** assume that a database of what students are enrolled in a course is complete.
- The definite clause language is **monotonic:** adding clauses can't invalidate a previous conclusion.
- Under the complete knowledge assumption, the system is **non-monotonic:** adding clauses can invalidate a previous conclusion.

Completion of a knowledge base

- Suppose the rules for atom a are

$$a \leftarrow b_1.$$

\vdots

$$a \leftarrow b_n.$$

equivalently $a \leftarrow b_1 \vee \dots \vee b_n.$

- Under the Complete Knowledge Assumption, if a is true, one of the b_i must be true:

$$a \rightarrow b_1 \vee \dots \vee b_n.$$

- Under the CKA, the clauses for a mean **Clark's completion:**

$$a \leftrightarrow b_1 \vee \dots \vee b_n$$

Clark's Completion of a KB

- Clark's completion of a knowledge base consists of the completion of every atom.
- If you have an atom a with no clauses, the completion is $a \leftrightarrow \text{false}$.
- You can interpret negations in the body of clauses.
 $\sim a$ means that a is false under the complete knowledge assumption
This is **negation as failure**.

Bottom-up negation as failure interpreter

```
C := {};  
repeat  
  either  
    select  $r \in KB$  such that  
       $r$  is " $h \leftarrow b_1 \wedge \dots \wedge b_m$ "  
       $b_i \in C$  for all  $i$ , and  
       $h \notin C$ ;  
       $C := C \cup \{h\}$   
  or  
    select  $h$  such that for every rule " $h \leftarrow b_1 \wedge \dots \wedge b_m$ "  $\in KB$   
      either for some  $b_i, \sim b_i \in C$   
      or some  $b_i = \sim g$  and  $g \in C$   
       $C := C \cup \{\sim h\}$   
until no more selections are possible
```

Negation as failure example

$p \leftarrow q \wedge \sim r.$

$p \leftarrow s.$

$q \leftarrow \sim s.$

$r \leftarrow \sim t.$

$t.$

$s \leftarrow w.$

Top-Down negation as failure proof procedure

- If the proof for a fails, you can conclude $\sim a$.
- Failure can be defined recursively:
Suppose you have rules for atom a :

$$a \leftarrow b_1$$

$$\vdots$$

$$a \leftarrow b_n$$

If each body b_i fails, a fails.

A body fails if one of the conjuncts in the body fails.

Note that you need *finite* failure. Example $p \leftarrow p$.

Often we want our agents to make assumptions rather than doing deduction from their knowledge. For example:

- In **abduction** an agent makes assumptions to explain observations. For example, it hypothesizes what could be wrong with a system to produce the observed symptoms.
- In **default reasoning** an agent makes assumptions of normality to make predictions. For example, the delivery robot may want to assume Mary is in her office, even if it isn't always true.

Two different tasks use assumption-based reasoning:

- **Design** The aim is to design an artifact or plan. The designer can select whichever design they like that satisfies the design criteria.
- **Recognition** The aim is to find out what is true based on observations. If there are a number of possibilities, the recognizer can't select the one they like best. The underlying reality is fixed; the aim is to find out what it is.

Compare: Recognizing a disease with designing a treatment.
Designing a meeting time with determining when it is.

The Assumption-based Framework

The assumption-based framework is defined in terms of two sets of formulae:

- F is a set of closed formula called the **facts**.
These are formulae that are given as true in the world.
We assume F are Horn clauses.
- H is a set of formulae called the **possible hypotheses** or **assumables**.

Making Assumptions

- A **scenario** of $\langle F, H \rangle$ is a set D of ground instances of elements of H such that $F \cup D$ is satisfiable.
- An **explanation** of g from $\langle F, H \rangle$ is a scenario that, together with F , implies g .
 D is an explanation of g if $F \cup D \models g$ and $F \cup D \not\models \text{false}$.
A **minimal explanation** is an explanation such that no strict subset is also an explanation.
- An **extension** of $\langle F, H \rangle$ is the set of logical consequences of F and a maximal scenario of $\langle F, H \rangle$.

Example

$a \leftarrow b \wedge c.$

$b \leftarrow e.$

$b \leftarrow h.$

$c \leftarrow g.$

$c \leftarrow f.$

$d \leftarrow g.$

$\text{false} \leftarrow e \wedge d.$

$f \leftarrow h \wedge m.$

assumable $e, h, g, m, n.$

- $\{e, m, n\}$ is a scenario.
- $\{e, g, m\}$ is not a scenario.
- $\{h, m\}$ is an explanation for a .
- $\{e, h, m\}$ is an explanation for a .
- $\{e, g, h, m\}$ isn't an explanation.
- $\{e, h, m, n\}$ is a maximal scenario.
- $\{h, g, m, n\}$ is a maximal scenario.

Default Reasoning and Abduction

Using the assumption-based framework for

- **Default reasoning**: The truth of a goal g is unknown and is to be determined.
An explanation for g corresponds to an **argument** for g .
- **Abduction**: A goal g is given, and we are interested in explaining it. g could be an observation in a recognition task or a design goal in a design task.

Given observations, we typically do abduction, then default reasoning to find consequences.

To find assumables to imply the query $?q_1 \wedge \dots \wedge q_k$:

$ac := \text{“yes} \leftarrow q_1 \wedge \dots \wedge q_k\text{”}$

repeat

select non-assumable atom a_i from the body of ac ;

choose clause C from KB with a_i as head;

 replace a_i in the body of ac by the body of C

until all atoms in the body of ac are assumable.

To find an explanation of query $?q_1 \wedge \dots \wedge q_k$:

- find assumables to imply $?q_1 \wedge \dots \wedge q_k$
- ensure that no subset of the assumables found implies *false*

Default Reasoning

- When giving information, we don't want to enumerate all of the exceptions, even if we could think of them all.
- In default reasoning, we specify general knowledge and modularly add exceptions. The general knowledge is used for cases we don't know are exceptional.
- Classical logic is **monotonic**: If g logically follows from A , it also follows from any superset of A .
- Default reasoning is **nonmonotonic**: When we add that something is exceptional, we can't conclude what we could before.

Defaults as Assumptions

Default reasoning can be modeled using

- H is normality assumptions
- F states what follows from the assumptions

An explanation of g gives an **argument** for g .

Default Example

A reader of newsgroups may have a default:
“Articles about AI are generally interesting”.

$$H = \{int_ai\},$$

where *int_ai* means *X* is interesting if it is about AI.

With facts:

$$interesting \leftarrow about_ai \wedge int_ai.$$

$$about_ai.$$

$\{int_ai\}$ is an explanation for *interesting*.

Default Example, Continued

We can have exceptions to defaults:

$false \leftarrow interesting \wedge uninteresting.$

Suppose an article is about AI but is uninteresting:

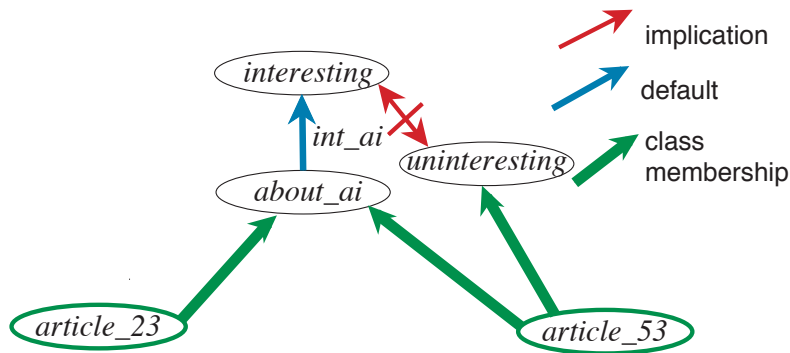
$interesting \leftarrow about_ai \wedge int_ai.$

$about_ai.$

$uninteresting.$

We cannot explain *interesting* even though everything we know about the previous we also know about this case.

Exceptions to defaults



Exceptions to Defaults

“Articles about formal logic are about AI.”

“Articles about formal logic are uninteresting.”

“Articles about machine learning are about AI.”

$about_ai \leftarrow about_fl.$

$uninteresting \leftarrow about_fl.$

$about_ai \leftarrow about_ml.$

$interesting \leftarrow about_ai \wedge int_ai.$

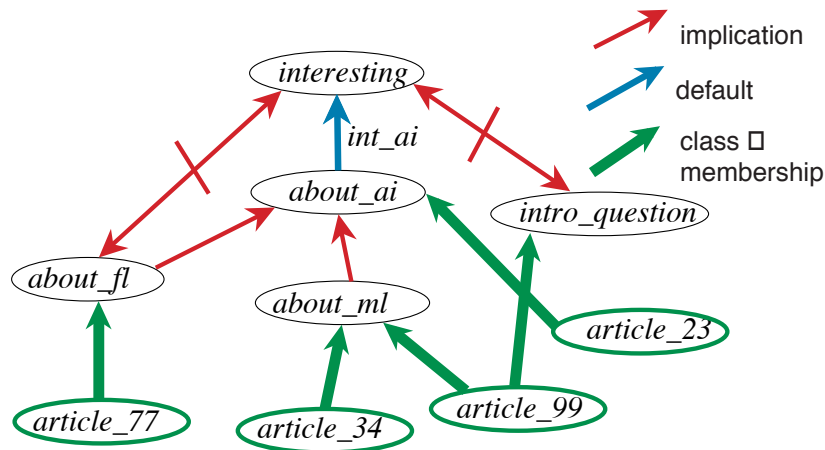
$false \leftarrow interesting \wedge uninteresting.$

$false \leftarrow intro_question \wedge interesting.$

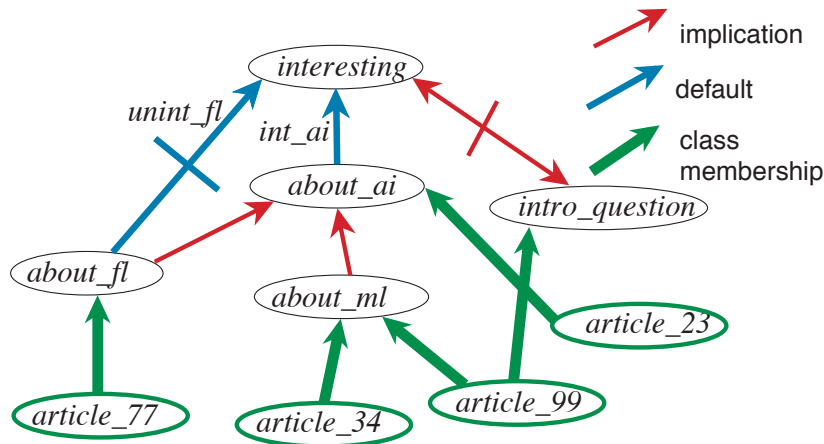
Given $about_fl$, is there explanation for $interesting$?

Given $about_ml$, is there explanation for $interesting$?

Exceptions to Defaults



Formal logic is uninteresting by default



Contradictory Explanations

Suppose formal logic articles aren't interesting *by default*:

$$H = \{unint_fl, int_ai\}$$

The corresponding facts are:

$$interesting \leftarrow about_ai \wedge int_ai.$$

$$about_ai \leftarrow about_fl.$$

$$uninteresting \leftarrow about_fl \wedge unint_fl.$$

$$false \leftarrow interesting \wedge uninteresting.$$

$$about_fl.$$

Does *uninteresting* have an explanation?

Does *interesting* have an explanation?

Overriding Assumptions

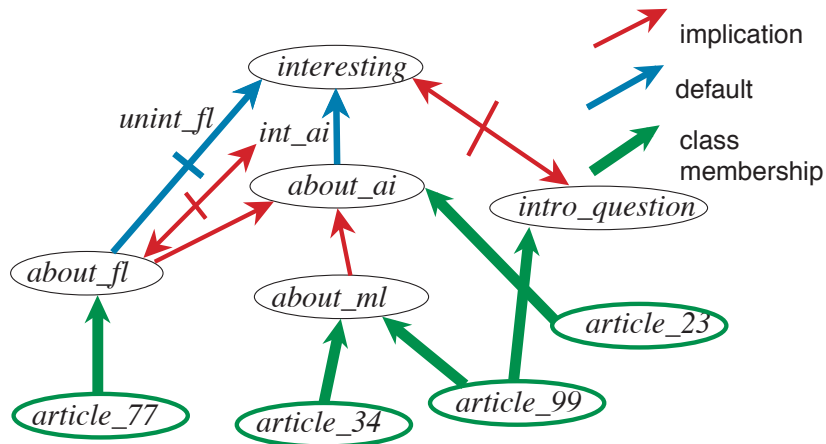
- For an article about formal logic, the argument “it is interesting because it is about AI” shouldn’t be applicable.
- This is an instance of preference for **more specific** defaults.
- Arguments that articles about formal logic are interesting because they are about AI can be defeated by adding:

$false \leftarrow about_fl \wedge int_ai.$

This is known as a **cancellation rule.**

- We can no longer explain *interesting*.

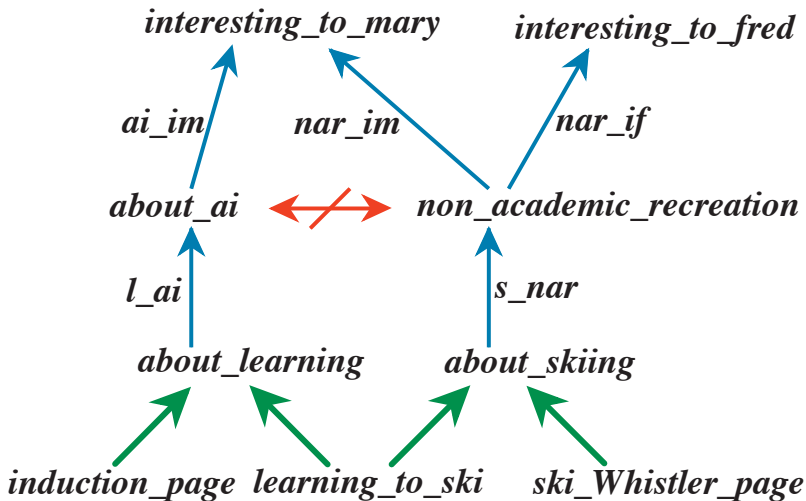
Diagram of the Default Example



Multiple Extension Problem

- What if incompatible goals can be explained and there are no cancellation rules applicable?
What should we predict?
- **For example:** what if introductory questions are uninteresting, by default?
- This is the **multiple extension problem**.
- **Recall:** an **extension** of $\langle F, H \rangle$ is the set of logical consequences of F and a maximal scenario of $\langle F, H \rangle$.

Competing Arguments



Skeptical Default Prediction

- We **predict** g if g is in all extensions of $\langle F, H \rangle$.
- Suppose g isn't in extension E . As far as we are concerned E could be the correct view of the world.
So we shouldn't predict g .
- If g is in all extensions, then no matter which extension turns out to be true, we still have g true.
- Thus g is predicted even if an adversary gets to select assumptions, as long as the adversary is forced to select something. You do not predict g if the adversary can pick assumptions from which g can't be explained.

Evidential and Causal Reasoning

Much reasoning in AI can be seen as **evidential reasoning**, (observations to a theory) followed by **causal reasoning** (theory to predictions).

- **Diagnosis** Given symptoms, evidential reasoning leads to hypotheses about diseases or faults, these lead via causal reasoning to predictions that can be tested.
- **Robotics** Given perception, evidential reasoning can lead us to hypothesize what is in the world, that leads via causal reasoning to actions that can be executed.

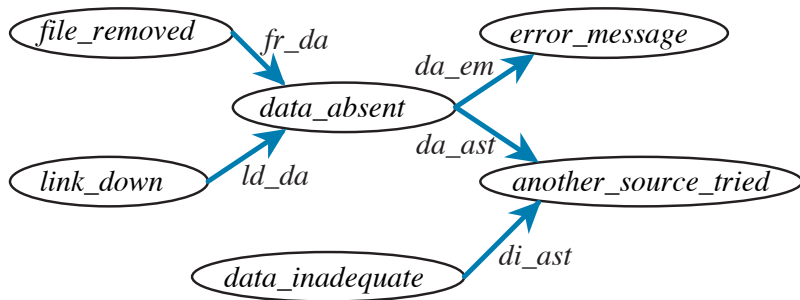
To combine evidential and causal reasoning, you can either

- Axiomatize from causes to their effects and
 - ▶ use abduction for evidential reasoning
 - ▶ use default reasoning for causal reasoning
- Axiomatize both
 - ▶ effects \rightarrow possible causes (for evidential reasoning)
 - ▶ causes \rightarrow effects (for causal reasoning)

use a single reasoning mechanism, such as default reasoning.

- Representation:
 - ▶ Axiomatize causally using rules.
 - ▶ Have normality assumptions (defaults) for prediction
 - ▶ other assumptions to explain observations
- Reasoning:
 - ▶ given an observation, use all assumptions to explain observation (find base causes)
 - ▶ use normality assumptions to predict consequences from base causes (hypotheses to be tested).

Causal Network

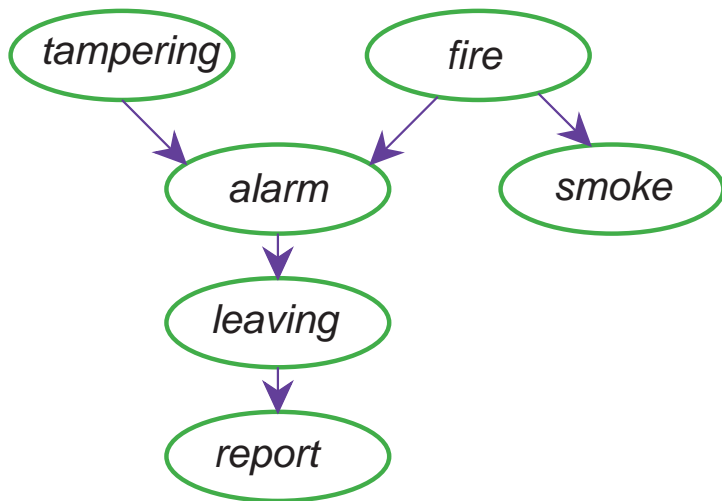


Why is the infobot trying another information source?
(Arrows are implications or defaults. Sources are assumable.)

Code for causal network

```
error_message ← data_absent ∧ da_em.  
another_source_tried ← data_absent ∧ da_ast  
another_source_tried ← data_inadequate ∧ di_ast.  
data_absent ← file_removed ∧ fr_da.  
data_absent ← link_down ∧ ld_da.  
default da_em, da_ast, di_ast, fr_da, ld_da.  
assumable file_removed.  
assumable link_down.  
assumable data_inadequate.
```

Example: fire alarm



Fire Alarm Code

assumable *tampering*.

assumable *fire*.

$alarm \leftarrow tampering \wedge tampering_caused_alarm.$

$alarm \leftarrow fire \wedge fire_caused_alarm.$

default *tampering_caused_alarm*.

default *fire_caused_alarm*.

$smoke \leftarrow fire \wedge fire_caused_smoke.$

default *fire_caused_smoke*.

$leaving \leftarrow alarm \wedge alarm_caused_leaving.$

default *alarm_caused_leaving*.

$report \leftarrow leaving \wedge leaving_caused_report.$

default *leaving_caused_report*.

- If we observe *report* there are two minimal explanations:
 - ▶ one with *tampering*
 - ▶ one with *fire*
- If we observed just *smoke* there is one explanation (containing *fire*). This explanation makes no predictions about tampering.
- If we had observed $report \wedge smoke$, there is one minimal explanation, (containing *fire*).
 - ▶ The smoke **explains away** the tampering. There is no need to hypothesise *tampering* to explain report.