For when I am presented with a false theorem, I do not need to examine or even to know the demonstration, since I shall discover its falsity *a posteriori* by means of an easy experiment, that is, by a calculation, costing no more than paper and ink, which will show the error no matter how small it is. . .

And if someone would doubt my results, I should say to him: "Let us calculate, Sir," and thus by taking to pen and ink, we should soon settle the question.

—Gottfried Wilhelm Leibniz [1677]

Learning Objectives

At the end of the class you should be able to:

- explain how symbols can have meaning
- represent a problem in a simple logic
- prove soundness and completeness of a proof procedure
- debug a logic program without knowing the how inference works
- use negation-as-failure where appropriate
- use assumption-based reasoning for a simple domain

Outline

Propositions and Semantics

Proofs
- Bottom-up Proof Procedure
- Top-down Proof Procedure

Ask-the-user and Knowledge-level Debugging

Complete Knowledge Assumption

Assumption-based Reasoning
- Proof by Contradiction
- Abduction

Why propositions?

A proposition is a statement that is either true or false.
Propositions can be built using logical connectives.

- Specifying proposition is often a natural specification
- Correctness can be checked locally
- The answer depends on the semantics, not how it is implemented
- Debugging can use the semantics of propositions
- We choose inference method to be efficient
- It provides a language for asking queries
- It is easy to incrementally add formulae
- It can be extended to infinite domains (using quantification)

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

Definite Clauses

Which of the following are definite clauses?

(a) $happy \leftarrow sad$

(b) $blimsy$

(c) $old \wedge wise \leftarrow teenager$

(d) $happy \wedge sad$

(e) $glad \leftarrow happy \wedge sad$

(f) $green \vee blue \leftarrow \neg red$

(g) $glad \leftarrow happy \wedge sad \wedge mad \wedge bad$

(h) $glad \leftarrow happy \wedge rad \leftarrow sad \wedge mad \wedge bad$

(i) $happy \leftarrow happy$

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 will answer whether the question is a logical consequence.

Step 4 Interpret the answers with the meaning associated with the atoms.

Role of semantics

In computer:

$$light1\_broken \leftarrow sw\_up \wedge power \wedge unlit\_light1.$$

$$sw\_up.$$

$$power \leftarrow lit\_light2.$$

$$unlit\_light1.$$

$$lit\_light2.$$

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

Semantics

- 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, and is *false* otherwise.
- 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.

Models and Logical Consequence

- 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 body, 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 ...

Simple Example

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

	p	q	r	s	model?
I_1	*true*	*true*	*true*	*true*	
I_2	*false*	*false*	*false*	*false*	
I_3	*true*	*true*	*false*	*false*	
I_4	*true*	*true*	*true*	*false*	
I_5	*true*	*true*	*false*	*true*	

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

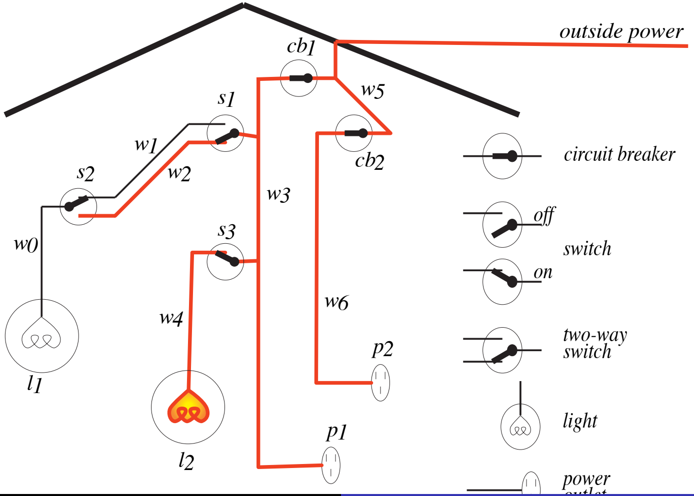
User's view of Semantics

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.

Computer's view of semantics

- 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

$light\_l_1.$

$light\_l_2.$

$down\_s_1.$

$up\_s_2.$

$up\_s_3.$

$ok\_l_1.$

$ok\_l_2.$

$ok\_cb_1.$

$ok\_cb_2.$

$live\_outside.$

$lit\_l_1 \leftarrow live\_w_0 \wedge ok\_l_1$

$live\_w_0 \leftarrow live\_w_1 \wedge up\_s_2.$

$live\_w_0 \leftarrow live\_w_2 \wedge down\_s_2.$

$live\_w_1 \leftarrow live\_w_3 \wedge up\_s_1.$

$live\_w_2 \leftarrow live\_w_3 \wedge down\_s_1.$

$lit\_l_2 \leftarrow live\_w_4 \wedge ok\_l_2.$

$live\_w_4 \leftarrow live\_w_3 \wedge up\_s_3.$

$live\_p_1 \leftarrow live\_w_3.$

$live\_w_3 \leftarrow live\_w_5 \wedge ok\_cb_1.$

$live\_p_2 \leftarrow live\_w_6.$

$live\_w_6 \leftarrow live\_w_5 \wedge ok\_cb_2.$

Outline

Proofs

- 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 $KB \vdash g$.

Bottom-up Ground Proof Procedure

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

> *If "$h \leftarrow b_1 \wedge \ldots \wedge b_m$" 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 "$h \leftarrow b_1 \wedge \ldots \wedge b_m$" 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:

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:

$$h \leftarrow b_1 \land \ldots \land b_m$$

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:

$$h \leftarrow b_1 \wedge \ldots \wedge b_m$$

  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.

Fixed Point

- 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:

Fixed Point

- 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 $h \leftarrow b_1 \wedge \ldots \wedge b_m$ in KB is false in I. Then h is false and each b_i is true in I. Thus h can be added to C. Contradiction to C being the fixed point.
- I is called a Minimal Model.

Completeness

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

A query is a body that we want to determine if it is a logical consequence of KB.

Idea: search backward from the query.

- An answer clause is of the form:

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

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

  $$a_i \leftarrow b_1 \wedge \ldots \wedge b_p$$

  is the answer clause

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

Derivations

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

Top-down definite clause interpreter

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

> ac := "$yes \leftarrow q_1 \wedge \ldots \wedge q_k$"
> **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

$$
\begin{array}{lll}
a \leftarrow b \land c. & a \leftarrow e \land f. & b \leftarrow f \land k. \\
c \leftarrow e. & d \leftarrow k. & e. \\
f \leftarrow j \land e. & f \leftarrow c. & j \leftarrow c.
\end{array}
$$

Query: $?a$

Example: successful derivation

$a \leftarrow b \wedge c.$ $a \leftarrow e \wedge f.$ $b \leftarrow f \wedge k.$
$c \leftarrow e.$ $d \leftarrow k.$ $e.$
$f \leftarrow j \wedge e.$ $f \leftarrow c.$ $j \leftarrow c.$

Query: $?a$

$\gamma_0 : \quad yes \leftarrow a$
$\gamma_1 : \quad yes \leftarrow e \wedge f$
$\gamma_2 : \quad yes \leftarrow f$
$\gamma_3 : \quad yes \leftarrow c$
$\gamma_4 : \quad yes \leftarrow e$
$\gamma_5 : \quad yes \leftarrow$

Example: failing derivation

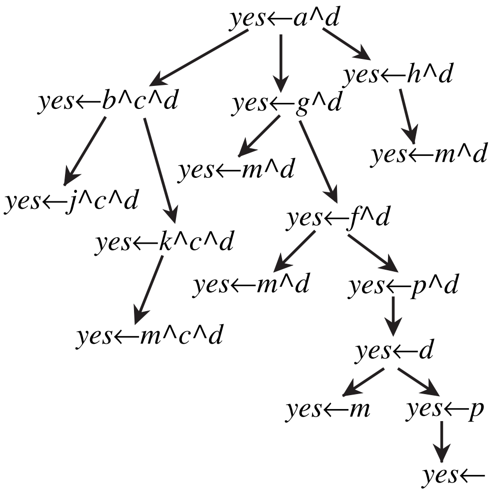
$$
\begin{array}{lll}
a \leftarrow b \wedge c. & a \leftarrow e \wedge f. & b \leftarrow f \wedge k. \\
c \leftarrow e. & d \leftarrow k. & e. \\
f \leftarrow j \wedge e. & f \leftarrow c. & j \leftarrow c.
\end{array}
$$

Query: $?a$

$$
\begin{array}{ll}
\gamma_0 : & yes \leftarrow a \\
\gamma_1 : & yes \leftarrow b \wedge c \\
\gamma_2 : & yes \leftarrow f \wedge k \wedge c \\
\gamma_3 : & yes \leftarrow c \wedge k \wedge c \\
\gamma_4 : & yes \leftarrow e \wedge k \wedge c \\
\gamma_5 : & yes \leftarrow k \wedge c
\end{array}
$$

Search Graph for SLD Resolution

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



Outline

Propositions and Semantics

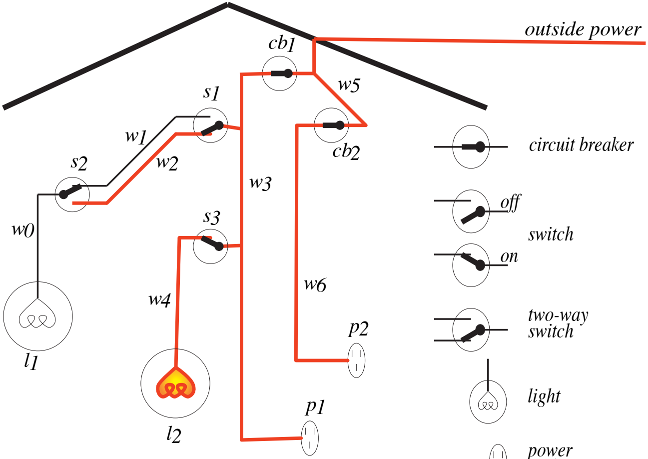
Proofs
- Bottom-up Proof Procedure
- Top-down Proof Procedure

Ask-the-user and Knowledge-level Debugging

Complete Knowledge Assumption

Assumption-based Reasoning
- Proof by Contradiction
- Abduction

Electrical Domain



Users

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

Users

- 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.

Ask-the-user

- Users can provide observations to the system. They can answer specific queries.
- Askable atoms are those that a user should be able to observe.
- 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.

Ask-the-user

- Users can provide observations to the system. They can answer specific queries.
- Askable atoms are those that a user should be able to observe.
- 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.

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 the ask HOW an element of the body of that rules was proved.
  This lets the user explore the proof.
- 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.

Knowledge-Level Debugging

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

Knowledge-Level Debugging

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

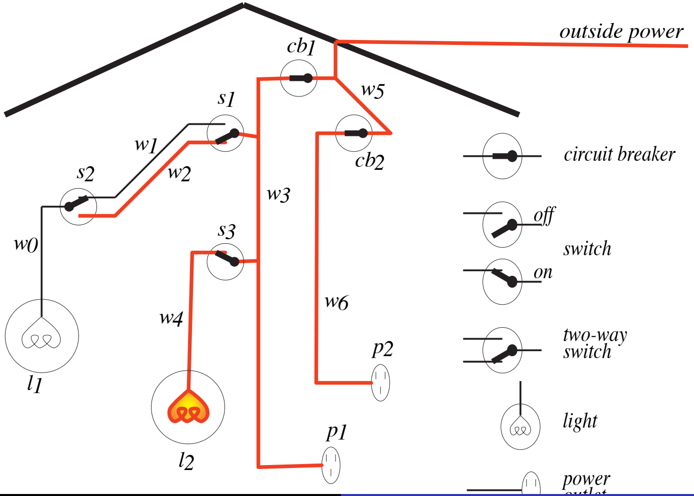
Debugging incorrect answers

- Suppose atom g was proved but is false in the intended interpretation.
- There must be a rule $g \leftarrow a_1 \wedge \ldots \wedge a_k$ in the knowledge base that was used to prove g.
- Either:

Debugging incorrect answers

- Suppose atom g was proved but is false in the intended interpretation.
- There must be a rule $g \leftarrow a_1 \wedge \ldots \wedge a_k$ 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.

Electrical Environment



Missing Answers

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 $g \leftarrow a_1 \wedge \ldots \wedge a_k$ that should have succeeded.

Missing Answers

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 $g \leftarrow a_1 \wedge \ldots \wedge a_k$ that should have succeeded.
  - One of the a_i is true in the interpretation and could not be proved.

Outline

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 $a \leftarrow b_1 \vee \ldots \vee b_n$.
- Under the Complete Knowledge Assumption, if a is true, one of the b_i must be true:

  $$a \rightarrow b_1 \vee \ldots \vee b_n.$$
- Under the CKA, the clauses for a mean Clark's completion:

  $$a \leftrightarrow b_1 \vee \ldots \vee b_n$$

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 $a \leftrightarrow \mathit{false}$.
- 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 "$h \leftarrow b_1 \wedge \ldots \wedge b_m$"
      $b_i \in C$ for all i, and
      $h \notin C$;
    $C := C \cup \{h\}$
  or
    select h such that for every rule "$h \leftarrow b_1 \wedge \ldots \wedge b_m$" $\in KB$
        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:

$$
\begin{aligned}
a &\leftarrow b_1 \\
&\vdots \\
a &\leftarrow b_n
\end{aligned}
$$

  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 $p \leftarrow p$.

Default reasoning

- Birds fly.
- Emus and tiny birds don't.
- Hummingbirds are exceptional tiny birds.

Default reasoning

- Birds fly.
- Emus and tiny birds don't.
- Hummingbirds are exceptional tiny birds.

$$flies \leftarrow bird \wedge \sim ab\_flying.$$
$$ab\_flying \leftarrow emu \wedge \sim ab\_emu.$$
$$ab\_flying \leftarrow tiny \wedge \sim ab\_tiny.$$
$$ab\_tiny \leftarrow hummingbird \wedge \sim ab\_hummingbird.$$

Outline

Assumption-based Reasoning

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

- In proof by contradiction an agent makes assumptions which are shown to be false.
- 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.

Design and Recognition

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.

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 won't be true with the rules that imply *false*.

Horn clauses

- An integrity constraint is a clause of the form

  $$false \leftarrow a_1 \wedge \ldots \wedge a_k$$

  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:

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

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:

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

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:

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

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:

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

Questions and Answers in Horn KBs

- 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

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

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

  $false \leftarrow dark\_l_1 \;\&\; lit\_l_1.$

  $false \leftarrow dark\_l_2 \;\&\; lit\_l_2.$

  $false \leftarrow dead\_p_1 \;\&\; live\_p_2.$

- Assume the individual components are working correctly:

  $assumable\ ok\_l_1.$

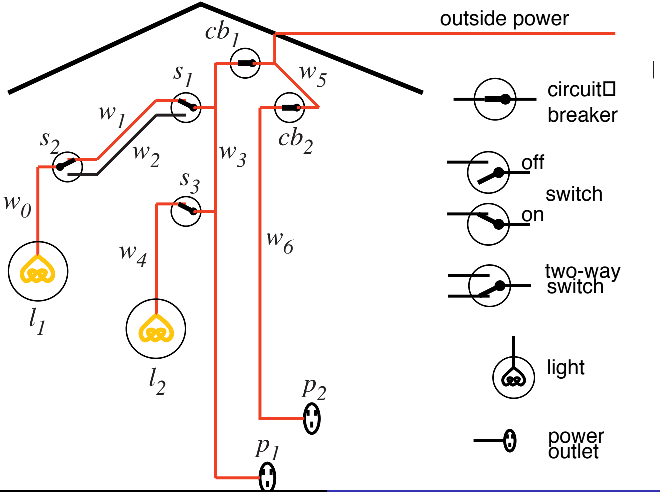
  $assumable\ ok\_s_2.$

  ...

- Suppose switches s_1, s_2, and s_3 are all up:

  $up\_s_1.\ up\_s_2.\ up\_s_3.$

Electrical Environment



Representing the Electrical Environment

$light\_l_1.$

$light\_l_2.$

$up\_s_1.$

$up\_s_2.$

$up\_s_3.$

$live\_outside.$

$lit\_l_1 \leftarrow live\_w_0 \wedge ok\_l_1.$

$live\_w_0 \leftarrow live\_w_1 \wedge up\_s_2 \wedge ok\_s_2.$

$live\_w_0 \leftarrow live\_w_2 \wedge down\_s_2 \wedge ok\_s_2.$

$live\_w_1 \leftarrow live\_w_3 \wedge up\_s_1 \wedge ok\_s_1.$

$live\_w_2 \leftarrow live\_w_3 \wedge down\_s_1 \wedge ok\_s_1.$

$lit\_l_2 \leftarrow live\_w_4 \wedge ok\_l_2.$

$live\_w_4 \leftarrow live\_w_3 \wedge up\_s_3 \wedge ok\_s_3.$

$live\_p_1 \leftarrow live\_w_3.$

$live\_w_3 \leftarrow live\_w_5 \wedge ok\_cb_1.$

$live\_p_2 \leftarrow live\_w_6.$

$live\_w_6 \leftarrow live\_w_5 \wedge ok\_cb_2.$

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

$$dark\_l_1.\ dark\_l_2.$$

- There are two minimal conflicts:

- 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

  $\{ok\_cb_1, ok\_s_3, ok\_l_2\}$.

- We can derive:

- 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

  $\{ok\_cb_1, ok\_s_3, ok\_l_2\}$.

- We 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.

Diagnoses

- 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 proceeding example there are seven minimal diagnoses: $\{ok\_cb_1\}$, $\{ok\_s_1, ok\_s_3\}$, $\{ok\_s_1, ok\_l_2\}$, $\{ok\_s_2, ok\_s_3\}$,...

Recall: top-down consequence finding

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

ac := "$yes \leftarrow q_1 \wedge \ldots \wedge q_k$"
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.

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 $d, f, g, h.$

Bottom-up Conflict Finding

- 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 $h \leftarrow b_1 \wedge \ldots \wedge b_m$ such that for each b_i there is some A_i such that $\langle b_i, A_i\rangle \in C$, then $\langle h, A_1 \cup \ldots \cup A_m\rangle$ 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 $\langle \mathit{false}, A_1\rangle$ 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.

Bottom-up Conflict Finding Code

$C := \{\langle a, \{a\}\rangle : a \text{ is assumable } \}$;
repeat
    select clause "$h \leftarrow b_1 \wedge \ldots \wedge b_m$" in T such that
        $\langle b_i, A_i \rangle \in C$ for all i and
        there is no $\langle h, A' \rangle \in C$ or $\langle \mathit{false}, A' \rangle \in C$
            such that $A' \subseteq A$ where $A = A_1 \cup \ldots \cup A_m$;
    $C := C \cup \{\langle h, A \rangle\}$
    Remove any elements of C that can now be pruned;
until no more selections are possible

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. Ground instance of the possible hypotheses can be assumed if consistent.

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 $F \cup D \not\models \mathit{false}$.
  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

$a \leftarrow b \wedge c.$
$b \leftarrow e.$
$b \leftarrow h.$
$c \leftarrow g.$
$c \leftarrow f.$
$d \leftarrow g.$
$false \leftarrow e \wedge d.$
$f \leftarrow h \wedge m.$
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

There are two strategies for using the assumption-based framework:

- Default reasoning Where the truth of g is unknown and is to be determined.
  An explanation for g corresponds to an argument for g.
- Abduction Where 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.

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

Computing Explanations

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

> ac := "$yes \leftarrow q_1 \wedge \ldots \wedge q_k$"
> **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 $?q_1 \wedge \ldots \wedge q_k$:

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