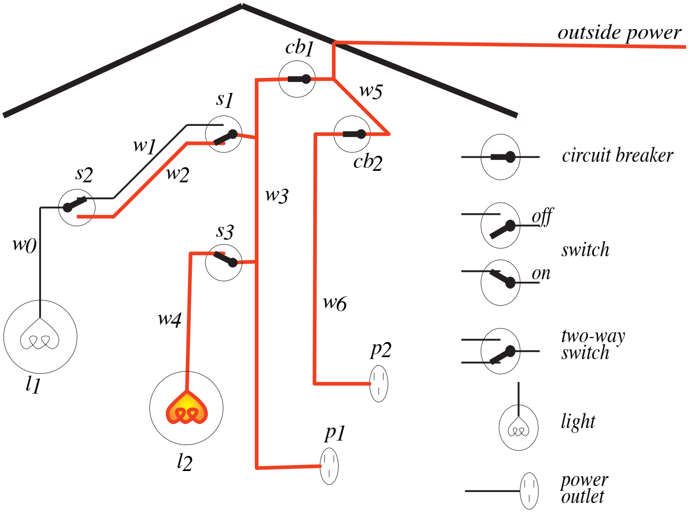
Simple language: propositional definite clauses

Propositional definite clauses are a resticited form of propostions that can't represent disjunction of atoms:

- A body is either
  - an atom or
  - the form $b_1 \wedge b_2$ where b_1 and b_2 are bodies.
- A definite clause is either
  - an atomic fact: an atom or
  - a rule: of the form $h \leftarrow b$ where h is an atom and b is a body.

  An atomic fact is treated as a rule with an empty body.
- A knowledge base or logic program is a set of definite clauses.
- A qeury is a body that is asked of a knowledge base.

Electrical Environment



Representing the Electrical Environment

$light\_l_1.$

$light\_l_2.$

$down\_s_1.$

$up\_s_2.$

$up\_s_3.$

$ok\_l_1.$

$ok\_l_2.$

$ok\_cb_1.$

$ok\_cb_2.$

$live\_outside.$

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

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

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

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

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

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

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

$live\_p_1 \leftarrow live\_w_3.$

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

$live\_p_2 \leftarrow live\_w_6.$

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

$live\_w_5 \leftarrow live\_outside.$

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 $KB \models g$.
  - If a sound proof procedure produces a result, the result is correct.
- A proof procedure is complete if $KB \models g$ implies $KB \vdash g$.
  - A complete proof procedure can produce all results.

Aside: Gödel's incompleteness theorem

Gödel's incompleteness theorem [1930]:
No proof system for a sufficiently rich logic can be both sound and complete.
sufficiently rich = can represent arithmetic

Proof sketch:

Consider the statement "this statement cannot be proven".

- If it is true then system is incomplete.
- If it is false then system is unsound.
- The alternative is that statement cannot be represented.
- the state of a computer can be seen as a (big) integer, and all operations as arithmetic operations
- We can write a proof system that can represent that statement in a computer.

Bottom-up 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.
(An atomic fact is treated as a clause with empty body ($m = 0$).)

Bottom-up proof procedure

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

$C := \{\}$;
repeat
    select fact h or rule "$h \leftarrow b_1 \wedge \ldots \wedge b_m$" 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.
- h was added to C, so there must be a clause in KB

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

  where each b_i is in C, and so true in I.
  h is false in I (by assumption)
  So this clause is false in I.
  Therefore I isn't a model of KB.
- Contradiction. Therefore there cannot be such a g.

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.
- Claim: 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

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

An answer clause is of the form:

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

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

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

An atomic fact in the knowledge base is considered as a clause where $p = 0$.

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
  - 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

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

Query: $?a$

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

Example: failing derivation

$a \leftarrow b \wedge c.$ $\quad a \leftarrow e \wedge f.$ $\quad b \leftarrow f \wedge k.$

$c \leftarrow e.$ $\quad d \leftarrow k.$ $\quad e.$

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

Query: $?a$

$\gamma_0 : \quad yes \leftarrow a$

$\gamma_1 : \quad yes \leftarrow b \wedge c$

$\gamma_2 : \quad yes \leftarrow f \wedge k \wedge c$

$\gamma_3 : \quad yes \leftarrow c \wedge k \wedge c$

$\gamma_4 : \quad yes \leftarrow e \wedge k \wedge c$

$\gamma_5 : \quad yes \leftarrow k \wedge c$

$a \leftarrow b \wedge c.$

$a \leftarrow h.$

$b \leftarrow k.$

$d \leftarrow p.$

$sf \leftarrow p.$

$g \leftarrow f.$

$h \leftarrow m.$

$?a \wedge d$

$a \leftarrow g.$

$b \leftarrow j.$

$d \leftarrow m.$

$f \leftarrow m.$

$g \leftarrow m.$

$k \leftarrow m.$

$p.$