Restricted Languages

- definite clauses

and their

Specialized Proof Systems

- SLD resolution

Selecting an atom using a Linear strategy

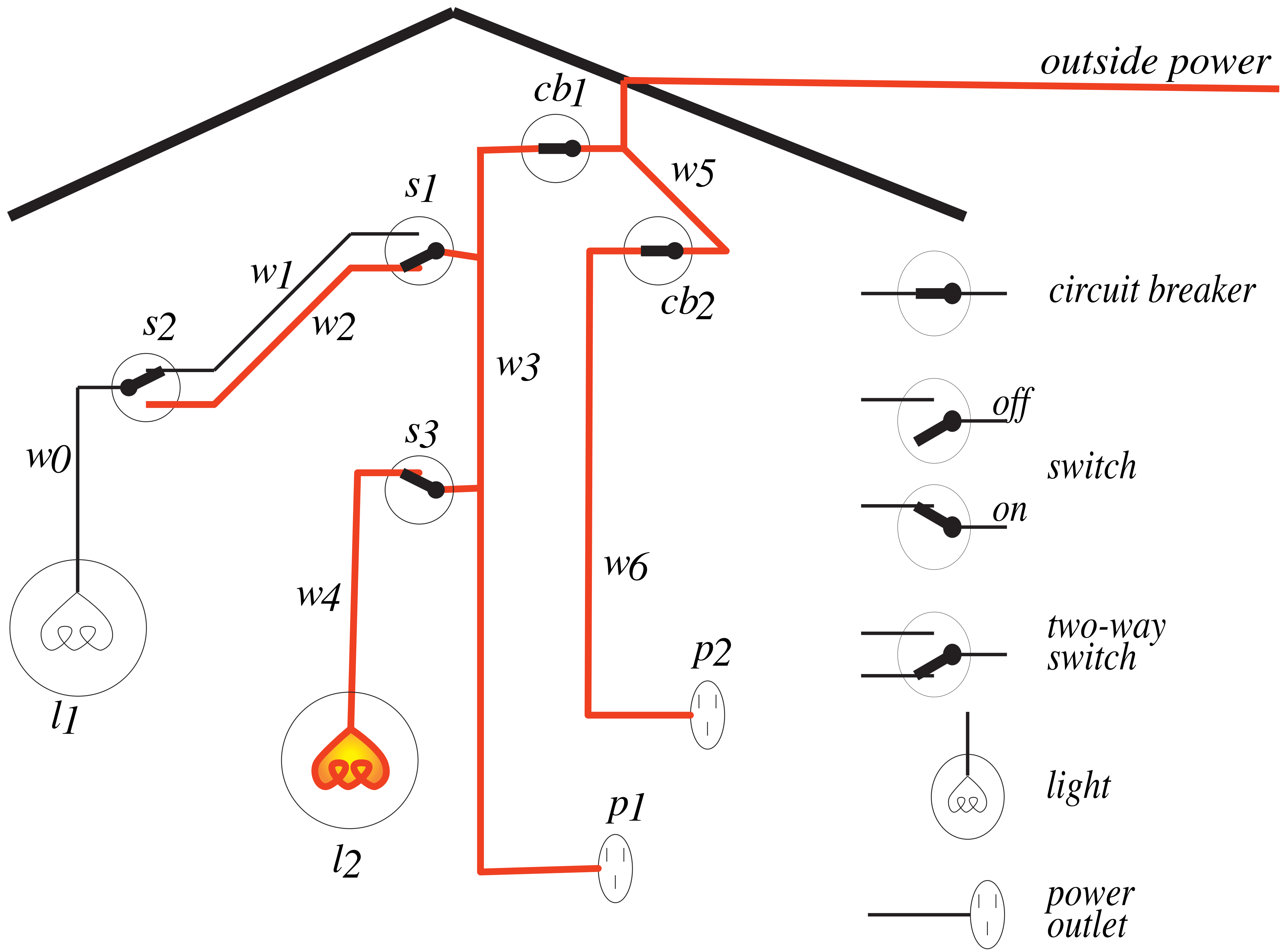
Definite clauses

(backward chaining, top-down reasoning)

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

Electrical Environment



Representing the Electrical Environment

$light\_l_1.$

$light\_l_2.$

$down\_s_1.$

$up\_s_2.$

$up\_s_3.$

$ok\_l_1.$

$ok\_l_2.$

$ok\_cb_1.$

$ok\_cb_2.$

$live\_outside.$

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

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

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

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

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

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

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

$live\_p_1 \leftarrow live\_w_3.$

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

$live\_p_2 \leftarrow live\_w_6.$

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

$live\_w_5 \leftarrow live\_outside.$

Definite Clause

$$b_1 \wedge b_2 \wedge \dots \wedge b_m \rightarrow h$$

rule

h (when $m = 0$)

fact

$$\equiv \neg(b_1 \wedge b_2 \wedge \dots \wedge b_m) \vee h$$

$$\equiv \underbrace{\neg b_1 \vee \neg b_2 \vee \dots \vee \neg b_m \vee h}_{\text{clausal form}}$$

clausal form

Definite clauses are clauses with exactly one positive literal.

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

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

  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: 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

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

SLD Resolution

$$\left.\begin{array}{l} a_i \leftarrow b_1 \wedge b_2 \wedge \dots \wedge b_k \\ yes \leftarrow a_1 \wedge \dots \wedge a_i \wedge \dots \wedge a_n \end{array}\right\} \quad yes \leftarrow a_1 \wedge \dots \wedge b_1 \wedge b_2 \wedge \dots \wedge b_k \wedge \dots \wedge a_n$$

$$a_i \vee \neg b_1 \vee \neg b_2 \vee \dots \vee \neg b_k \qquad yes \vee \neg a_1 \vee \dots \vee \neg a_i \vee \dots \vee \neg a_n$$

$$yes \vee \neg a_1 \vee \dots \vee \neg b_1 \vee \neg b_2 \vee \dots \vee \neg b_k \vee \dots \vee \neg a_n$$

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.

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

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

Query: $?a$

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

Search Graph for SLD Resolution

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

$?a \wedge d$