A "Greedy" Implementation (given B, E)

1. $h_0 = B, E_0^+ = E^+, i = 0$

2. repeat

   (a) increment i

   (b) Randomly choose a positive example e_i from E_{i-1}^+

   (c) Obtain the most specific clause $\perp(B, e_i)$

   (d) Find the clause D_i that: subsumes $\perp(B, e_i)$; and is consistent with the negative examples; and maximises $p(h_{i-1} \cup \{D_i\} | e_i^+ \cup E^-)$ where e_i^+ are the examples in E^+ made redundant by $h_{i-1} \cup \{D_i\}$

   (e) $h_i = h_{i-1} \cup \{D_i\}$

   (f) $E_i^+ = E_{i-1}^+ \backslash e_i^+$

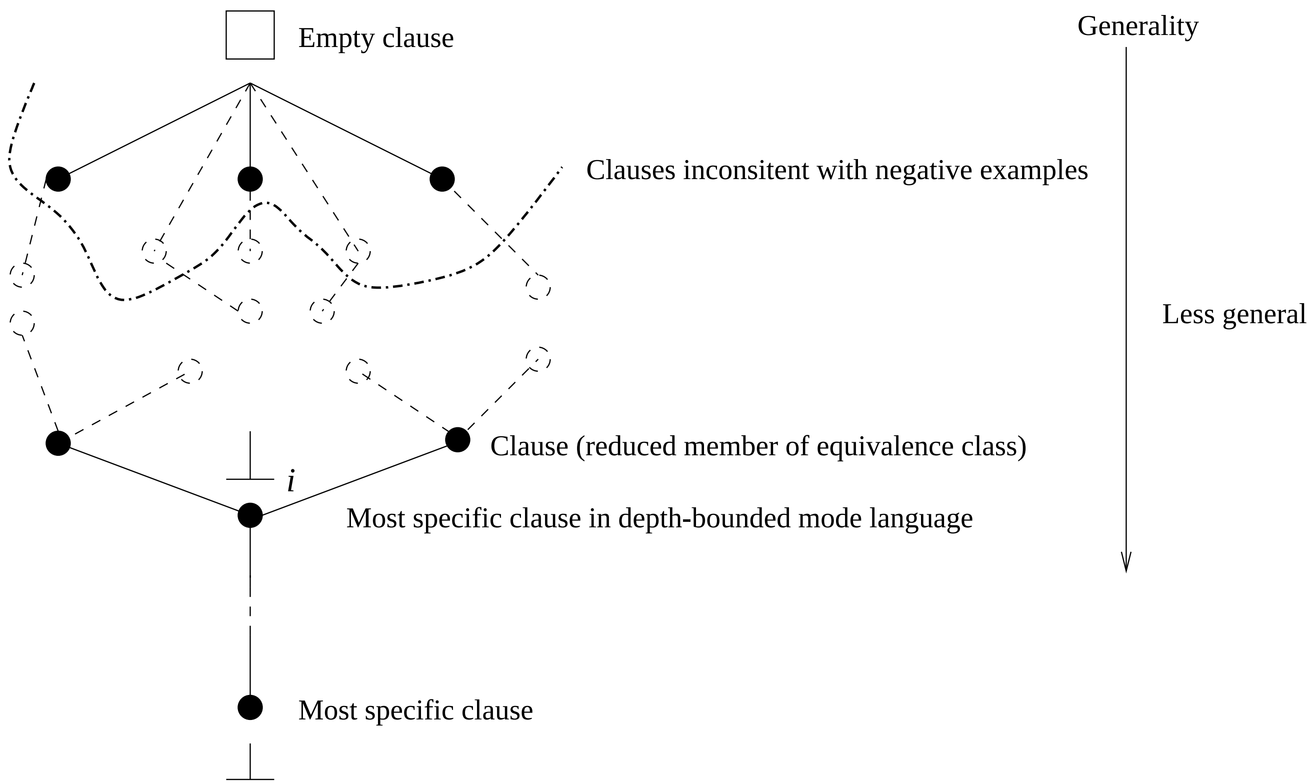
3. until $E_i^+ = \emptyset$

4. return h_i

Search and Redundancy

2 stages in clause-by-clause construction of hypothesis

1. Search



2. Remove redundant clauses once best clause is found

Moving about in the lattice: refinement steps

General-to-specific search: start at $\Box$, and move by

1. Adding a literal drawn from $\bot_i$

   $p(X,Y) \leftarrow q(X)$ becomes $p(X,Y) \leftarrow q(X), r(Y)$

2. Equating two variables of the same type

   $p(X,Y) \leftarrow q(X)$ becomes $p(X,X) \leftarrow q(X)$

3. Instantiate a variable with a general functional term or constant

   $p(X,Y) \leftarrow q(X)$ becomes $p(3,Y) \leftarrow q(3)$

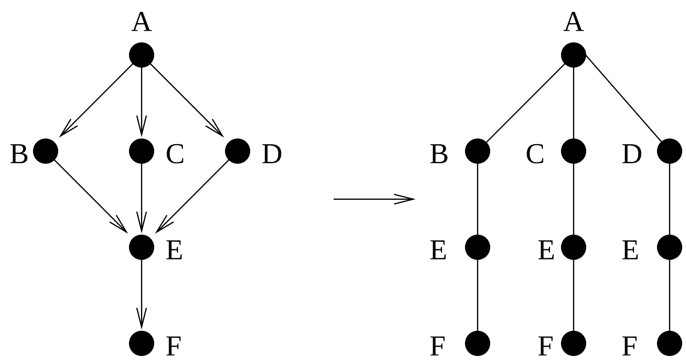
Specific-to-general search: start at $\bot_i$

Each of these is called a "refinement step"

Search Methods

Subsumption lattice can be represented as a directed acyclic graph

Can convert this to a tree. Root is the first node ($\Box$ or $\perp_i$). Children of a node are refinements.



Searching the lattice is therefore equivalent to searching a tree

– 2 basic types of tree search: depth-first (DF) and breadth-first (BF)

– DF and BF are "blind". More guidance at any node s

* g_s: cost of optimal path from root to s
* h_s: estimated cost of optimal path to goal from s

– Different kinds of guided search:

  Hill-climbing: DF with h_s

  Best-first: BF with h_s

  Best-cost: BF with g_s

  A^*: BF with g_s and h_s

An Optimal Search Algorithm: Branch-and-Bound

$bb(i, \rho, f)$: Given an initial element i from a discrete set S; a successor function $\rho : S \rightarrow 2^S$; and a cost function $f : S \rightarrow \Re$, return $H \subseteq S$ such that H contains the set of cost-minimal models. That is for all $h_{i,j} \in H, f(h_i) = f(h_j) = f_{min}$ and for all $s' \in S \backslash H \;\; f(s') > f_{min}$.

1. $Active := \langle i \rangle$.
2. $best :=$ inf
3. $selected := \emptyset$
4. while $Active \neq \langle\rangle$
5. begin
   (a) remove element k from $Active$
   (b) $cost := f(k)$
   (c) if $cost < best$
   (d) begin
      i. $best := cost$
      ii. $selected := \{k\}$
      iii. let $Prune_1 \subseteq Active$ s.t. for each $j \in Prune_1$, $\underline{f}(j) > best$ where $\underline{f}(j)$ is the lowest cost possible from j or its successors

iv. remove elements of $Prune_1$ from $Active$

(e) end

(f) elseif $cost = best$

i. $selected := selected \cup \{k\}$

(g) $Branch := \rho(k)$

(h) let $Prune_2 \subseteq Branch$ s.t. for each $j \in Prune_2$, $\underline{f}(j) > best$ where $\underline{f}(j)$ is the lowest cost possible from j or its successors

(i) $Bound := Branch \backslash Prune_2$

(j) add elements of $Bound$ to $Active$

6. end

7. return $selected$

Different search methods result from specific implementations of $Active$

– Stack: depth-first search

– Queue: breadth-first search

– Prioritised Queue: best-first search

Redundancy 1: Literal Redundancy

Literal l is redundant in clause $C \vee l$ relative to background B iff

$$B \wedge (C \vee l) \equiv B \wedge C$$

Can show The literal l is redundant in clause $C \vee l$ relative to the background B iff

$$B \wedge (C \vee l) \models C$$

The clause C is said to be reduced with respect to background knowledge B iff no literal in C is redundant.

Redundancy 2: Clause redundancy

Clause C is redundant in the $B \wedge C$ iff $B \wedge C \equiv B$.

Can show Clause C is redundant in $B \wedge C$ iff

$$B \models C \equiv B \wedge \overline{C} \models \Box$$

A set of clauses S is said to be reduced iff no clause in S is redundant

Example

$$e_j : \quad gfather(henry, john) \leftarrow$$

$$B : \quad \begin{array}{c} father(henry, jane) \leftarrow \\ father(henry, joe) \leftarrow \\ parent(jane, john) \leftarrow \\ parent(joe, robert) \leftarrow \end{array}$$

$$D_j : \quad gfather(X, Y) \leftarrow father(X, Z), parent(Z, Y)$$

e_j is redundant in $B \wedge D_j \wedge e_j$ since $B \wedge D_j \wedge \overline{e_j} \models \Box$

Implementation Issues

Question. Will the clause-by-clause search method yield the best set of clauses? If no, why not?

Question. Is it possible to do a theory-by-theory search?

Question. Is it possible devise a complete search that is non-redundant? If no, why not?