

Refinement operators [Procedural counterpart of $\geq$]

Recall

LUB

$l_{gg} / r_{lgg} / l_{gi} / r_{gvi}$



Gets well with Occam's razor

Downward covers

refinement: reverse along cover links

About refining clauses

$p(c) = 0$

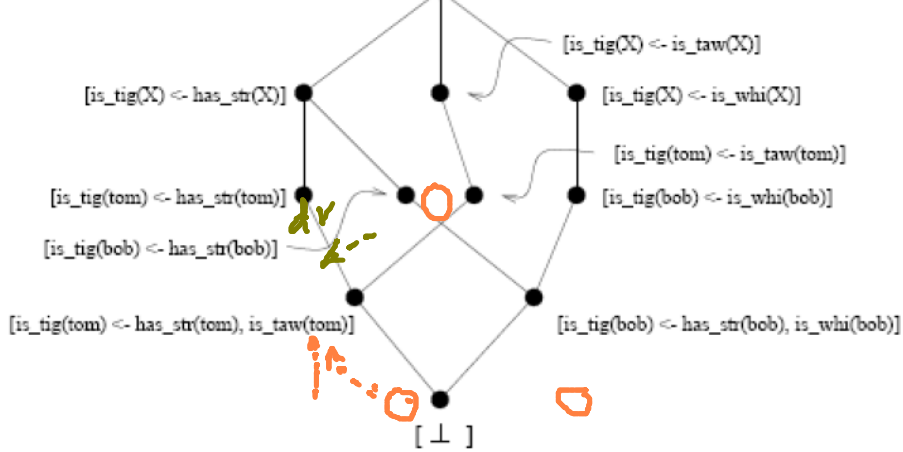
Generality of clause: $g(H) = \sum_{x \in X, H \models x, x \text{ is clause}} p(x)$

GLB: Very nondeterministic



Upward covers: Too many steps to generalize





ur. ↑

Recall.

Note: Taking

2 examples at

a time can

enable large

steps: It may not

be robust noise

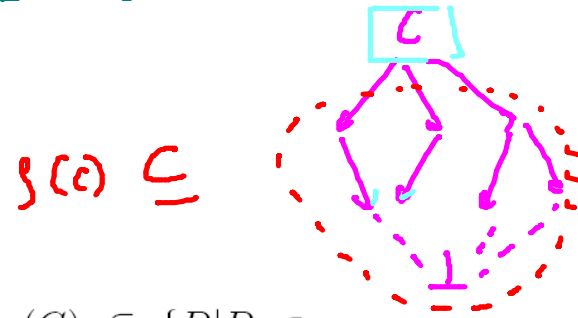
Lub(vlgg) takes the

examples too serious

Refinement operators (also how to incorporate background knowledge)

Example downward refinement:- $P(x) \rightarrow P(f(x))$

Top down search algos for ILP use downward refinement



- ρ is a downward refinement operator if $\forall C \in S : \rho(C) \subseteq \{D \mid D \in S \text{ and } C \succeq D\}$
- δ is an upward refinement operator if $\forall C \in S : \delta(C) \subseteq \{D \mid D \in S \text{ and } D \succeq C\}$

$$l = \begin{cases} V = W & \text{where } V, W \text{ occur in } C \\ V = f(X_1, X_2, \dots, X_{n_f}) & \text{where } V \text{ occurs in } C \\ & \text{and } f/n_f \in \mathcal{L} \text{ and} \\ & \text{the } X_i \text{ are distinct} \\ q(X_1, X_2, \dots, X_{n_q}) & \text{where } q/n_q \in \mathcal{L} \\ & \text{and the } X_i \text{ occur in } C \end{cases}$$

} $\mathcal{L}$ is language

eg: of $\mathcal{P}(C)$: Assume $=/2$ is an equality relation

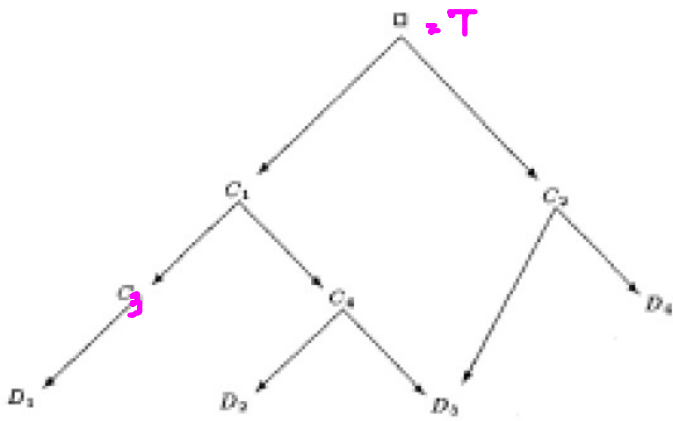
Assume, $V, W, f(x_1 \dots x_{n_f})$, are terms in G

$q(x_1 \dots x_{n_q})$ is a predicate

→ Note: $D \subseteq (\cup \{l\})$ is also another $\mathcal{P}(C)$

→ Thus many $\mathcal{P}(C)$ are possible. Q: Which one to choose

↳ For eg, this $\mathcal{P}(C)$ does not guarantee reachability



[Q: can $\mathcal{S}^*(\square)$ help you reach Σ]

So many different $\mathcal{S}(C)$ are possible. Which to choose?

Say:- $\Sigma = \{D_2, D_3, D_4\}$ is correct theory [could be that D_2, D_3 & D_4 cover Σ^+ & D_1 covers Σ^-]
 $\mathcal{S}(\square) = \{C_1, C_2\}$, $\mathcal{S}(C_1) = \{C_3, C_4\}$, $\mathcal{S}(C_2) = \{D_3, D_4\}$
 $\mathcal{S}(C_3) = \{D_1\}$, $\mathcal{S}(C_4) = \{D_2, D_3\}$

$\mathcal{S}$: chosen such that theories covering correct examples can be reached (as hopefully as quickly as possible)

Roughly.. $\mathcal{S}$ handles recall, "eval" measure handles precision
 (can bias choice from)

Want $\{ s.t. S^*(\text{top clause}) \ni \text{correct theory} \}$.

$\hookrightarrow$ If top clause = $T = \square$, then you want every clause to be reachable.

$\hookrightarrow$ If top clause = " $\text{gfather}(X, Y) \leftarrow$ ", you want every definition of gfather reachable.



Notations

$S(C)$

$S^k(C)$

$S^*(C) = \bigcup_{i=1}^{\infty} S^i(C)$

S chain: from C to D is a sequence $C = C_0, C_1, \dots, C_n = D$ & $C_i \in S(C_{i-1})$ for $i=1 \dots n$

$\Rightarrow$ Gives you a "refinement graph"

Want: - C covering $E^+ \in$ refinement graph

Some desirable properties of ρ (and dually δ) are that they be:

1. **Locally Finite:** $\forall C \in \mathcal{C}$: $\rho(C)$ is finite and computable. Otherwise ρ will be of limited use in practice.
2. **Complete:** $\forall C \succ D$: $\exists E \in \rho^*(C)$ s.t. $E \sim D$. That is, every specialization should be reachable by a finite number of applications of the operator.
3. **Proper:** $\forall C \in \mathcal{C}$: $\rho(C) \subseteq \{D \mid D \in S \text{ and } C \succ D\}$. That is, it is better only to compute proper generalizations of a clause, for otherwise repeated application of the operator might get stuck in a sequence of equivalent clauses (such as the application of inverse reduction in Section 2.3), without ever achieving any real specialization.

$E \succ \rho(C) \not\sim D \not\succ E$

} if all
3 props,
called
IDEAL

[ie if $E \sim C$ then

$\rho(C) \ni E$



Your steps are non-trivial

$\Rightarrow$ A ref graph.
 - finite # edges starting at each node (1)
 - a path of finite length from C to some $E \in \rho^*(C)$ (2)
 - no cycles (3)

ideal ref operators exist for atoms

Definition 7 Let A be the set of atoms in a language. The downward refinement operator ρ_A for A is defined as follows:

1. For every variable z in an atom A and every n -ary function symbol f in the language, let $x_1, \dots, x_n$ be distinct variables not appearing in A . Let $\rho_A(A)$ contain $A\{z/f(x_1, \dots, x_n)\}$.
2. For every variable z in A and every constant a in the language let $\rho_A(A)$ contain $A\{z/a\}$.
3. For every two distinct variables x and z in A , let $\rho_A(A)$ contain $A\{z/x\}$.

eg: ① $p(x, y, z) \rightarrow \begin{cases} p(x, y, f(x_1, x_2)) \\ p(f(x_1, x_2), y, z) \\ p(x, f(x_1, x_2), z) \end{cases} \quad \left. \begin{array}{l} \text{for every } n\text{-ary} \\ \text{fun.} \end{array} \right\}$

② special case of ① with 0-ary fun $\Rightarrow$ constant

③ $p(x, y, z) \rightarrow \begin{cases} p(x, x, z) \\ p(x, z, z) \\ p(x, y, x) \end{cases} \quad \left. \begin{array}{l} \exists c_2 \end{array} \right\} \text{ (think of it as adding a new predicate } \hat{x} = z \text{)}$

An ideal upward refinement for atoms ($\delta(A)$)

Definition 8 Let A be the set of atoms in a language. The upward refinement operator δ_A for A is defined as follows:

1. For every $t = f(x_1, \dots, x_n)$ in A , for which $x_1, \dots, x_n$ are distinct variables and each occurrence of some x_i in A is within an occurrence of t , $\delta_A(A)$ contains an atom obtained by replacing all occurrences of t in A by some new variable z not in A .
2. For every constant a in A and every non-empty subset of the set of occurrences of a in A , $\delta_A(A)$ contains an atom obtained by replacing those occurrences of a in A by some new variable z not in A .
3. For every variable x in A and every non-empty proper subset of the set of occurrences of x in A , $\delta_A(A)$ contains an atom obtained by replacing those occurrences of x in A by some new variable z not in A . Note that in this last item, we cannot replace all occurrences of x by a new variable z , for then we would get a variant of A . For instance, $P(z, a, z)$ is a variant of $A = P(x, a, x)$.

Locally finite
Complete
Proper

$$\textcircled{1} \quad P(f(x_1, x_2), y, z) \rightarrow P(z_1, y, z)$$

$$P(f(\underline{x}_1, \underline{x}_2), \underline{x}_1, z) \rightarrow P(z_1, z_1, z)$$

$$P(f(x_1, x_2), f(x_1, x_2), z)$$

$\textcircled{2}$ Special case of 1 for 0-ary fns.

$$\textcircled{3} \quad P(x, f(x), y, g(z, r(x))) \rightarrow P(\underline{x}, f(\underline{x}), y, g(z, r(\underline{x})))$$

$$\rightarrow P(z', f(x), y, g(z, r(z')))$$

$$\textcircled{4} \rightarrow P(z', f(z'), y, g(z, r(z')))$$

$\therefore$ Renaming $\Rightarrow$ Improper

For general clauses, ideal refinements ops

do not exist

↳ Recall:- $C_n = \{p(x_i, x_j) \mid i \neq j \text{ \& } 1 \leq i, j \leq n\} \quad n \geq 2$
& $C = \{p(x_1, x_2)\}$

can show! $C_2 > C_3 > C_4 \dots > C_n \dots > C$

$\Rightarrow C$ does not have an upward cover.

$\Rightarrow$ completeness not possible. If properness is imposed (completeness may require redundancy)

C_1
 C_2
 $\vdots$
 C_n
 $\vdots$
 C

↓ infinite elements just above C

Compromising

① finiteness = insensible

② completeness is valuable

③ properness is least important

④ finite & complete $\mathcal{S}_L$ is possible :- $\begin{cases} \text{by applying elem. subst} \\ \text{adding literals (most general)} \end{cases}$

Definition 9 Let C be a clausal language. The locally finite and complete downward refinement operator ρ_L for $\langle C, \succeq \rangle$ is defined as follows:

(a) Apply a substitution to a clause C in one of the following ways:

- For every variable z in a clause C and every n -ary function symbol f in the language, let $x_1, \dots, x_n$ be distinct variables not appearing in C . Let $\rho_L(C)$ contain $C\{z/f(x_1, \dots, x_n)\}$.
- For every variable z in C and every constant a in the language, let $\rho_L(C)$ contain $C\{z/a\}$.
- For every two distinct variables x and z in C , let $\rho_L(C)$ contain $C\{z/x\}$.

} Similar to $\mathcal{S}(A)$

Not proper! check
on $C = \{P(x)\}$ & $D = \{P(a), P(y)\}$
but $\vdash$

(b) Add a literal to a clause C : For every n -ary predicate symbol P in the language, let $x_1, \dots, x_n$ be distinct variables not appearing in C . Then ρ_L contains both $C \vee \{P(x_1, \dots, x_n)\}$ and $C \vee \{\neg P(x_1, \dots, x_n)\}$. Note that the literals $P(x_1, \dots, x_n)$ and $\neg P(x_1, \dots, x_n)$ that are added to C by the fourth item in the definition are most general with respect to C .

} additional part . . .

$\hookrightarrow x_1 \dots x_n$ do not appear in C

ex: $\{P(x)\} \dashrightarrow \{P(a), P(b)\}$

Definition 10 Let C be a clausal language. The locally finite and complete upward refinement operator δ_u for $\langle C, \succeq \rangle$ is defined as follows:

(a) Apply one of the following inverse substitution operations:

- i. For every $t = f(x_1, \dots, x_n)$ in C , for which all x_i are distinct variables and each occurrence of x_i in a clause C is within an occurrence of t , $\delta_u(C)$ contains the clause obtained by replacing all occurrences of t in C by some new variable z not previously in C .
- ii. For every constant a in C and every non-empty subset of the set of occurrences of a in $\overline{C} = \text{dup}(C, a)$, if $\overline{D}$ is the ordered clause obtained by replacing those occurrences of a in $\overline{C}$ by the new variable z , then $\delta_u(C)$ contains D , where D is the set of literals in the ordered clause $\overline{D}$.
 $\text{dup}(C, t)$ is defined as follows. If $C = \{L_1, \dots, L_n\}$ is a clause and t a term occurring in C and suppose t occurs k_1 times in L_1 , k_2 times in L_2 , etc. Then $\text{dup}(C, t) = \overline{C}$ is an ordered clause consisting of 2^{k_1} copies of L_1 , 2^{k_2} copies of L_2 , $\dots$ and 2^{k_n} copies of L_n . Note that if some $L \in C$ does not contain the term t , then $\overline{C}$ contains L $2^0 = 1$ times, as it should.
- iii. For every variable x in C and every non-empty proper subset of the set of occurrences of x in $\overline{C} = \text{dup}(C, x)$, if $\overline{D}$ is the ordered clause obtained by replacing those occurrences of x in $\overline{C}$ by the new variable z , then $\delta_u(C)$ contains D .

(b) **Remove a literal from the body of a clause:** If $C = D \cup \{L\}$ and L is a most general literal with respect to D , then $\delta_u(C)$ contains D .

Complete &
finite upward
refinement.

Compromising requirement of completeness

MIS, FOIL, CLAUDIEN, LINUS, PROGOL

↳ For reasons of efficiency

FOIL

$p(x,y) \leftarrow$

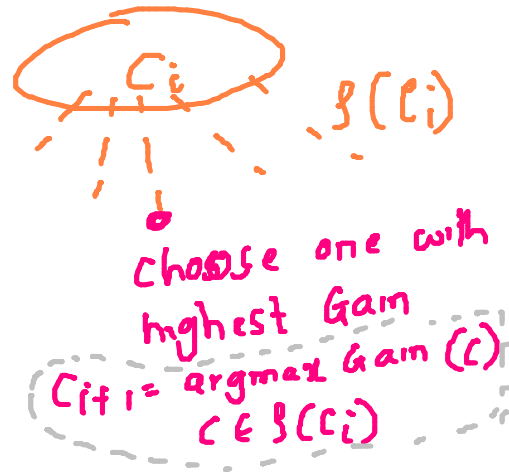
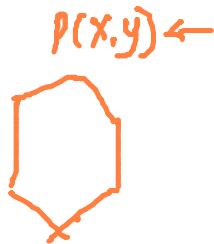
① head clause

② considers only addition of literals (L_i)

$g(x_1^{new}, x_2^{new}, \dots, x_1^{old}, \dots, x_2^{old})$ OR

$x_1^{old} = x_2^{old}$

(No function / constant substs)



Training examples	Background knowledge	
daughter(mary, ann). $\oplus$	parent(ann, mary).	female(ann).
daughter(eve, tom). $\oplus$	parent(ann, tom).	female(mary).
daughter(tom, ann). $\ominus$	parent(tom, eve).	female(eve).
daughter(eve, ann). $\ominus$	parent(tom, ian).	

Σ_i = examples covered by C_i

Current clause c_1 : daughter(X, Y) $\leftarrow$				
$\mathcal{E}_1$	(mary, ann) $\oplus$		$n_1^{\oplus} = 2$	$I(c_1) = 1.00$
	(eve, tom) $\oplus$		$n_1^{\ominus} = 2$	
	(tom, ann) $\ominus$	$L_1 = \text{female}(X)$		
	(eve, ann) $\ominus$	$\text{Gain}(L_1) = 0.84$	$n_1^{\oplus\oplus} = 2$	
Current clause c_2 : daughter(X, Y) $\leftarrow$ female(X), L_1				
$\mathcal{E}_2$	(mary, ann) $\oplus$		$n_2^{\oplus} = 2$	$I(c_2) = 0.58$
	(eve, tom) $\oplus$		$n_2^{\ominus} = 1$	
	(eve, ann) $\ominus$	$L_2 = \text{parent}(Y, X)$		
		$\text{Gain}(L_2) = 1.16$	$n_2^{\oplus\oplus} = 2$	
Current clause c_3 : daughter(X, Y) $\leftarrow$ female(X), parent(Y, X)				
$\mathcal{E}_3$	(mary, ann) $\oplus$		$n_3^{\oplus} = 2$	$I(c_3) = 0.00$
	(eve, tom) $\oplus$		$n_3^{\ominus} = 0$	

$S(C_1)$
 $S(C_2)$

Σ_2 = set of exs in Σ_1 which satisfy L_1

$\Sigma_{i+1} = \Sigma_i \Delta L_i$

$I(C_i) = -\log_2\left(\frac{n_i^{\oplus}}{n_i}\right)$
info needed for true signal using C_i

$\text{Gain}(L_i) = (I(C_i) - I(C_{i+1})) * n_i^{\oplus\oplus}$ reflectivity of L_i

$n_i^{\oplus\oplus}$ = # of true tuples in Σ_i represented in Σ_{i+1}

What if L_i introduces new vars

Current clause $c_1 : \text{daughter}(X, Y) \leftarrow$				
$\mathcal{E}_1$	(mary, ann)	$\oplus$		$n_1^{\oplus} = 2$
	(eve, tom)	$\oplus$		$n_1^{\ominus} = 2$
	(tom, ann)	$\ominus$	$L_1 = \text{parent}(Y, Z)$	
	(eve, ann)	$\ominus$	Gain: 1	$n_1^{\oplus\oplus} = 2$
Current clause $c_2 : \text{daughter}(X, Y) \leftarrow \text{parent}(Y, Z)$				
$\mathcal{E}_2$	(mary, ann, mary)	$\oplus$		$n_2^{\oplus} = 4$
	(mary, ann, tom)	$\oplus$		
	(eve, tom, eve)	$\oplus$		
	(eve, tom, ian)	$\oplus$		
	(tom, ann, mary)	$\ominus$		$n_2^{\ominus} = 4$
	(tom, ann, tom)	$\ominus$		
	(eve, ann, mary)	$\ominus$		
	(eve, ann, tom)	$\ominus$		

Tom story continues

$$\mathcal{E}_{i+1} = \mathcal{E}_i \bowtie L_i$$

You do not seem to get rid of any $\ominus$ res in $\mathcal{E}_2$

Pruning criterion: Best gain so far $> \underbrace{n_{i,k}^{\oplus\oplus} * I(c_i)}_{\text{By introducing new vars in } L_i}$

Diagram showing a sequence of literals $L_1, L_2, \dots, L_k$ with arrows indicating a flow from L_1 to L_2 and then to L_k . A dashed line connects L_1 and L_2 .

'Overall FOIL'

① $\Sigma_{\text{curr}} = \Sigma$

② $\mathcal{H} = \phi$

③ repeat set covering {

④ - - - - Initialize $C := T \leftarrow$

⑤ - - - Find C_{best}

⑥ - - - $\hat{C} = C_{\text{best}}$

⑦ - - Post process C by removing irrelevant literals to get C'

⑧ - - - $\mathcal{H}' = \mathcal{H} \cup \{C'\}$

⑨ - - - $\Sigma'_{\text{curr}} = \Sigma_{\text{curr}} - \text{covers}(B_1\{C'\}, \Sigma_{\text{curr}})$

⑩ - - - $\Sigma_{\text{curr}} = \Sigma'_{\text{curr}}$, $\mathcal{H} = \mathcal{H}'$

⑪ until $\Sigma_{\text{curr}} = \phi$ or encoding length restriction violated

Mainly determinate literals.

For noise handling

Encoding length for c_i w.r.t Σ_{curr}

$$EL(c_i, \Sigma_{curr}) = \log_2(n_{curr}) + \log_2\left(\underbrace{\binom{n_{curr}}{n^{\oplus}(c_i)}}_{\text{\# of set choices possible}}\right)$$

$\underbrace{\hspace{10em}}$ # of set choices possible

of bits needed to specify the eggs covered by a c_i

(related MDL)

Requirement : $EL(c_i, \Sigma_{curr}) \leq \Delta$ (so that c_i does not become toooooo long)

(FOL 2.0) Determinate Literals:-

A lit is det, if each of its vars that do not appear in preceding lits has only one possible binding, given the bindings of its vars that appear in preceding lits

Eg:

Training examples	
grandmother(ann, bob).	$\oplus$
grandmother(ann, sue).	$\oplus$
grandmother(bob, sue).	$\ominus$
grandmother(tom, bob).	$\ominus$

Background knowledge		
father(zak, tom).	father(pat, ann).	father(zak, jim).
mother(ann, tom).	mother(liz, ann).	mother(ann, jim).
father(tom, sue).	father(tom, bob).	father(jim, dave).
mother(eve, sue).	mother(eve, bob).	mother(jean, dave).

Det lit's help
broaden horizon

grandmother(X, Y) $\leftarrow$ father(Z, Y), mother(X, Z).
grandmother(X, Y) $\leftarrow$ mother(U, Y), mother(X, U).

Both clauses
are determinate

Z in father(Z, Y) has only one
value given a value for Y
"Since every person has only
one father"

If L_i is det $\Rightarrow \eta_{i+1}^{\oplus} = \eta_i^{\oplus}$, $\eta_{i+1}^{\ominus} = \eta_i^{\ominus} \Rightarrow \text{Gain}(L_i) = 0$

While traditional FOIL does greedy hill climbing, it avoids
det lit's (gain being 0) & $\therefore$ may miss good theories

Q: When should you consider adding a det lit?

Ans:- If no lit has gain close to max, all determinate lits are added.

In the game $\approx \text{max}$, it means $\exists$ a literal which gets rid of all -ve examples

Problem:- Many determinate lits added, may never be expanded on

$\text{grandmother}(X, Y) \leftarrow \text{father}(Z, Y), \text{mother}(X, Z), \text{mother}(U, Y)$
 $\text{grandmother}(X, Y) \leftarrow \text{mother}(U, Y), \text{mother}(X, U).$

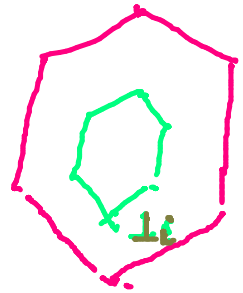
↓
removed thru post processing

If $\text{Gain}(c) \approx \text{Likelihood} \approx n \cdot \text{foil}$.
 $\subseteq$ Highest SVM $F1 \approx k \cdot \text{foil}$

Another example of compromise on completeness of $\mathcal{P}$

Goal is : Find simplest H st

$$\mathcal{B} \wedge H \models \mathcal{E}$$



Simplest $\Rightarrow$ at least one example should be explained. O/w.....

$$\mathcal{B} \wedge \bar{\mathcal{E}}_i \models \bar{H} \quad (\text{deduction thm})$$

Let $\bar{I}_i \equiv MM(\mathcal{B} \wedge \bar{\mathcal{E}}_i) = a_1 \wedge a_2 \dots a_n$ st
 $\mathcal{B} \wedge \bar{\mathcal{E}}_i \models a_k \quad \forall k \in [1, n]$

$$\Rightarrow \mathcal{B} \wedge \bar{\mathcal{E}}_i \models \bar{I}_i \models \bar{H}$$

$$\Rightarrow \forall h, \quad h \models \bar{I}_i$$

B	E	$\perp$
anim(X) $\leftarrow$ pet(X). pet(X) $\leftarrow$ dog(X).	nice(X) $\leftarrow$ dog(X).	nice(X) $\leftarrow$ dog(X), pet(X), anim(X).
hasbeak(X) $\leftarrow$ bird(X). bird(X) $\leftarrow$ vulture(X).	hasbeak(tweety). <i>-ve eg</i>	hasbeak(tweety); bird(tweety); vulture(tweety).
white(swan1).	$\leftarrow$ black(swan1).	$\leftarrow$ black(swan1), white(swan1).
sentence([], []).	sentence([a,a,a], []).	sentence([a,a,a], []) $\leftarrow$ sentence([], []).

$$\textcircled{1} B \wedge \bar{e} = \{ (\text{anim}(x) \leftarrow \text{pet}(x)), (\text{pet}(x) \leftarrow \text{dog}(x)), \\ (\text{dog}(c)), \neg \text{nice}(c) \}$$

↓
Skolemization constant

(Note: Skolem const
is just a convenient)

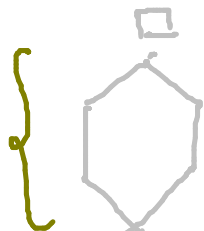
$$MM(B \wedge \bar{e}) = \{ (\text{dog}(c)), \{ \text{pet}(c) \}, \{ \text{anim}(c) \}, \{ \neg \text{nice}(c) \} \}$$

$$\begin{aligned} \overline{MM(B \wedge \bar{e})} &= \{ (\neg \text{dog}(x)), \vee \neg \text{pet}(x), \vee \neg \text{anim}(x), \vee \\ &\quad \text{nice}(x) \} \\ &= \text{nice}(x) \leftarrow \text{dog}(x), \text{pet}(x), \text{anim}(x) \end{aligned}$$

One possibility

look at all h that θ -subsume $\perp_i$

Sublattice
more
bounded
than foil



h st $\exists \theta$ for which

$$h\theta \subseteq \perp_i$$

Since $\perp_i$ already has absorbed B , we need to consider only subsumption (need not consider relative subsumption)

Q: what h will be st $h \geq \perp_i$

① $\exists \theta$ st $h\theta \subseteq \perp_i$... i.e. if $l \in h$, $\exists l' \in \perp_i$ st $l\theta = l'$

Let $\perp_i(h)$ be all such l'

uniquely defined

$$\perp_i(h) = h\theta$$

$$(l_1, l_2, \dots, l_n) \geq \underbrace{(l'_1, \dots, l'_n)}_{\perp_i(h)}$$

② How to get a $L_i(h)$

L_i , A non-deterministic approach

Let $|L_i| = n$

→ Maintain an index j st for each $j \in [1, n]$,
you decide whether to include the j^{th} element of
 L_i in $L_i(h)$.

③ → Program maintains both j & θ

④ L_i may be infinite ∴ Restrictions thru mode decol.

Recall Mode declarations

A var is ^(relatively free) splittable if it is a

+/- type in
mode h

- type in mode b

Prolog refinement op. (say: $l \equiv m \in M$ & say $l \equiv p(u_1, \dots, u_m)$)

Let $1 \leq j \leq n$. Let C be a clause in $L(M)$ &
 θ a substitution s.t. $\theta \in \perp_i$

$\langle C', \theta', j' \rangle \in \rho(\langle C, \theta, j \rangle)$ iff either

① $C' = C \cup \{l\}$, $j' = j$, $\langle l, \theta' \rangle$ is in $\delta(\theta, j)$
 & $C' \in L(M)$ OR

② $C' = C$, $j' = j+1$, $\theta' = \theta$ & $j < n$

where $\langle p(u_1, \dots, u_m), \theta' \rangle$ is in $\delta(\theta, j)$ iff θ' is initialized
 to θ , $l_j \equiv p(u_1, \dots, u_m)$ is the j th literal of $\perp_i$ & for
 each v , $1 \leq v \leq m$

① If u_v is splittable then $v_v / u_v \in \theta'$
 else $v_v / u_v \in \theta$ OR

② If u_v is splittable then u_v is a new
 var not in $\text{dom}(\theta)$ & $\theta' = \theta \cup \{v_v / u_v\}$

$\langle C, \theta \rangle$
 $\perp_i$

$\delta(\theta, j) = \text{set of literals which can be unified using } \theta'.$

Example 28 Applying ρ in list reversal. Suppose M consists of the following mode declarations.

$\text{modeb}(*, \text{reverse}(+list, -list))$ $\text{modeb}(*, +list = [-int] - list)$
 $\text{modeb}(*, +any = \#any)$ $\text{modeb}(*, \text{reverse}(+list, -list))$
 $\text{modeb}(*, \text{append}(+list, [+int], -list))$

The types and other background knowledge are defined as follows.

$B = \begin{cases} \text{any}(\text{Term}) \leftarrow \\ \text{list}([]) \leftarrow \\ \text{list}([H|T]) \leftarrow \text{list}(T) \\ \text{Term} = \text{Term} \leftarrow \\ \text{reverse}([], []) \leftarrow \\ \text{append}([], X, X) \leftarrow \\ \text{append}([H|T], L1, [H|L2]) \leftarrow \text{append}(T, L1, L2) \end{cases}$

Verify
 $C' \theta \subseteq L_i$

start from:-
 $C = \langle [], \epsilon, 1 \rangle$
 $\downarrow \delta(C)$

anti unification

C'	θ'	k'
$\text{reverse}(D, E) \leftarrow$	$\{D/A, E/A\}$	1
$\text{reverse}(D, D) \leftarrow$	$\{D/A\}$	1
$\square$	$\emptyset$	2

move to 2nd lit

C'	θ'	k'
$\text{reverse}(D, E) \leftarrow D = [F G], \text{reverse}(G, G)$	θ	6
$\text{reverse}(D, E) \leftarrow D = [F G], \text{reverse}(G, H)$	$\theta \cup \{H/C\}$	6
$\text{reverse}(D, E) \leftarrow D = [F G]$	θ	7

Figure 3: Two applications of ρ .

Let $h = 30$ and $i = 3$ and let the example be as below.

of depth resolution

$e = \text{reverse}([1], [1]) \leftarrow$

steps st

$B \wedge \perp \wedge \bar{e} \vdash_h \square$

Note: $\perp_i = \text{reverse}(A, A) \leftarrow A = [i], A = [B|C],$

$B = [], C = [],$

$\text{reverse}(C, C),$
 $\text{append}(C, [B], A)$

Refinement operators on theories (theoretical interest)

Definition 11 Let $\mathcal{C}$ be a clausal language, containing only a finite number of constants, function symbols, and predicate symbols. Let $\mathcal{S}$ be the set of finite subsets of $\mathcal{C}$. The downward refinement operator $\rho_{\mathcal{I}}$ for $\langle \mathcal{S}, \models \rangle$ is defined as follows:

1. $(\Sigma \cup \{R \mid R \text{ is a resolvent of } C_1, C_2 \in \Sigma\}) \in \rho_{\mathcal{I}}(\Sigma)$.
2. If $\Sigma = \{C_1, \dots, C_n\}$, then $(\Sigma \cup \rho_{\mathcal{I}}(C_i)) \in \rho_{\mathcal{I}}(\Sigma)$, for each $1 \leq i \leq n$.
3. If $\Sigma = \{C_1, \dots, C_n\}$, then $(\Sigma \setminus \{C_i\}) \in \rho_{\mathcal{I}}(\Sigma)$, for each $1 \leq i \leq n$.

} Assume you have a ref op $\rho_{\mathcal{I}}$ over clauses.

3 types of elements of

$\rho_{\mathcal{I}}(\Sigma)$: It is complete by subsumption thm

INVERSE RESOLUTION (More of a hack)



Inverting resolution

step $\Rightarrow$ Theoretical foundations need to be studied

Induction $\longleftrightarrow$ Deduction

Inv res. $\longleftrightarrow$
(Generalization approach)

Resolution
(Specialization)

$\Sigma \models C$
.....

↓
Non-deterministic
(Like principle of induction)

↓
Deterministic

2 operators $\begin{matrix} \nearrow \vee \\ \searrow \wedge \end{matrix}$

V-operator

$$\{C_1; R\} \xrightarrow{\text{generalises}} \{E_1, C_2\}$$

----- $\rightarrow$ s.t

R is an instance of
resolvent of C_1 & C_2

eg. $C_1 = \underline{P(x)} \vee \neg Q(f(x))$

L_2

$$R = Q(g(y)) \vee \neg Q(f(g(y)))$$

Recall TRS:

$$\begin{array}{cc} C_1 & C_2 \\ C_1' \vee L_1 & L_2 \vee C_2' \end{array}$$

$$\begin{array}{c} \theta_1 \quad \theta_2 \\ L_2 \theta_2 \vee L_1' \theta_1 \\ \downarrow \\ R = C_1' \theta_1 \vee C_2' \theta_2 \end{array}$$

Minimal C_2 we can
speculate

Assuming $C_1' \theta_1$ & $C_2' \theta_2$
don't overlap:-
 $C_2 = (R - C_1' \theta_1) \vee \neg L_1 \theta_1$

INPUT: Horn clauses $C_1 = L_1 \vee C'_1$ and R , where $C'_1\theta_1 \subseteq R$ for some θ_1 .

OUTPUT: A Horn clause C_2 , such that R is an instance of a resolvent of C_1 and C_2 .

- 1 Choose a substitution θ_1 such that $C'_1\theta_1 \subseteq R$. → det
 - 2 Choose an L_2 and C'_2 such that $L_1\theta_1 = \neg L_2\theta_2$ and $C'_2\theta_2 = R - C'_1\theta_1$, for some θ_2 .
- return $C_2 = L_2 \vee C'_2$.

Algo for determining $\vee$ op.

① $\Rightarrow$ iterates over all $C'_1 \supseteq R$.

② $\Rightarrow$ choice of L_2 is the non-deterministic part.

You need to choose an L_2 s.t

① $L_1\theta_1 = \neg L_2\theta_2$

② For this θ_2 , $C'_2\theta_2 = R - C'_1\theta_1$

Simplest case if- $C_1 = L_1 \Rightarrow$ find L_2 s.t $L_1\theta_1 = \neg L_2\theta_2$
 & $C'_2\theta_2 = R$.

$\Rightarrow$ Most specific $C_2 = \neg L_1 \vee R$
 ($\theta_1 = \theta_2 = \epsilon$)

Notes: θ_1 & θ_2 need not be equal



eg. $C_1 = \underbrace{P(x)}_{L_1} \vee \underbrace{\neg Q(f(x))}_{C'_1}$

$$R = Q(g(y)) \vee \neg Q(f(g(y)))$$

(Step 1): find θ_1 s.t. $C'_1 \theta_1 \subseteq R$.

$$\theta_1 = \{x / g(y)\}$$

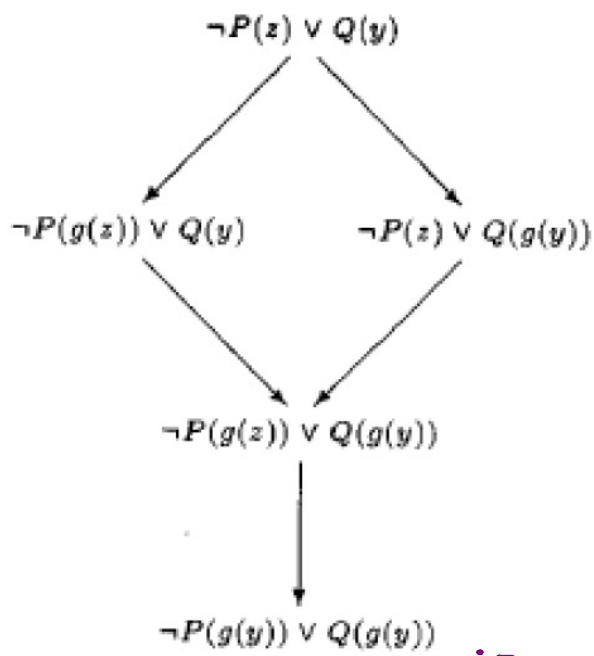
{ May have
to ping pong
between
steps 1 & 2 }

(Step 2): $\underline{C'_2} \theta_2 = Q(g(y))$
 $\underline{L_2} \theta_2 = \neg L_1 \theta_1 = \neg P(g(y))$

Sort of inverse subst on $Q(g(y))$ & $\neg P(g(y))$

Simplest but most
specific $\theta_2 = E$

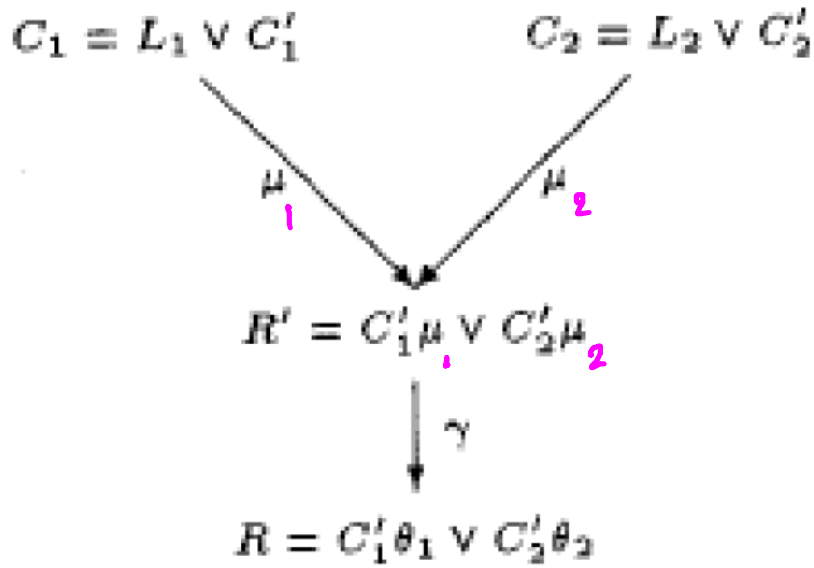
$\underline{C'_1} = Q(g(y))$ $\underline{L_2} = \neg P(g(y))$
 Most specific. Can be generalized
 CAREFULLY using
 upward refinement



All possible $C_2 = L_2 \vee C_2'$
 from top to bottom
 in decreasing order
 of generality

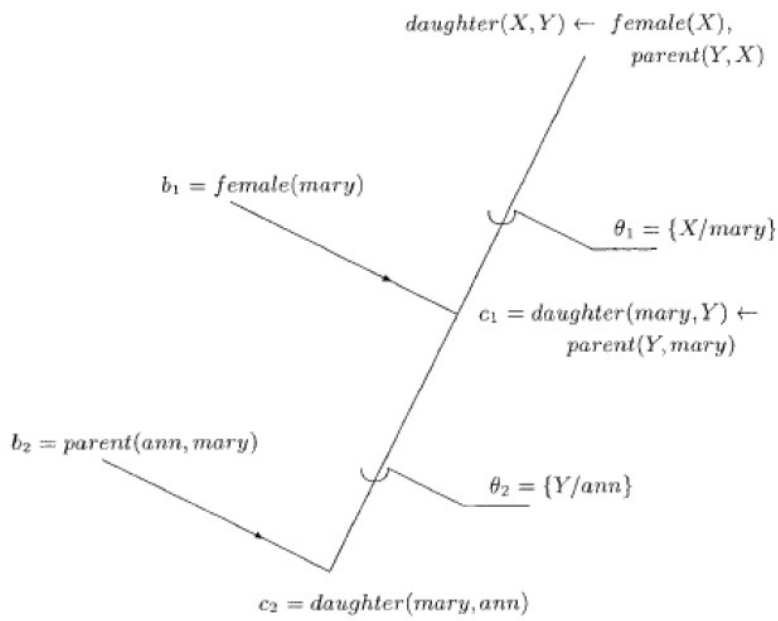
always obtained by $\theta_2 = \epsilon$ → Minimal choice for C_2

Not always as straightforward :- Sometimes,
 if $C_1' \theta_1 \cap C_2' \theta_2 \neq \emptyset$, you may have to duplicate
 literals in R .



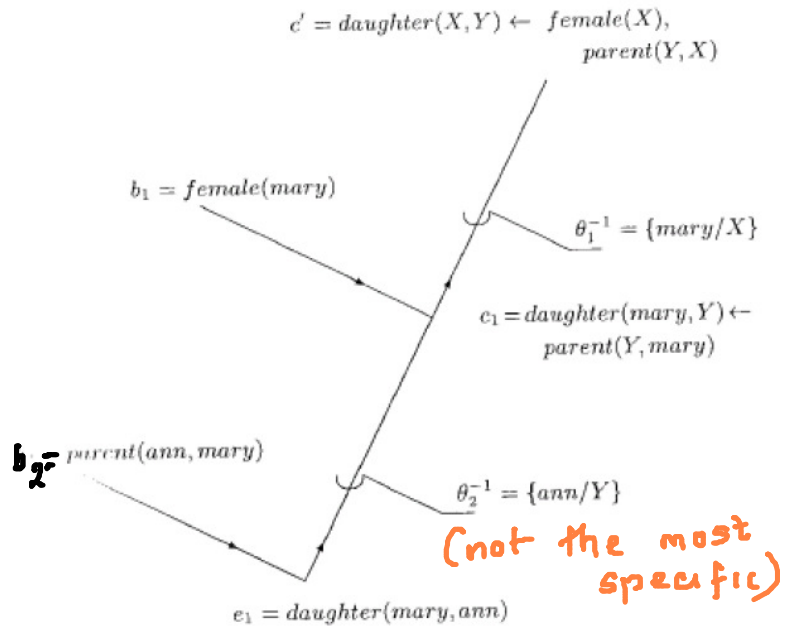
The general setting . . .

Note:- All that we have studied about generality orderings becomes useful (refinement ops in part)



Linear derivation

Inverse linear derivation.



The $\forall$ -operator : Predicate Invention.

In the case of the $\forall$ operator, all predicates possibly appearing in C_2 must appear in C_1 or R .

Predicate Invention:- By putting 2 $\forall$ operators side by side. } Yields succint definitions

$$C_1 = L_1 \vee C_1' \quad C_2 = L_2 \vee C_2' \quad C_3 = L_3 \vee C_3'$$

R_1

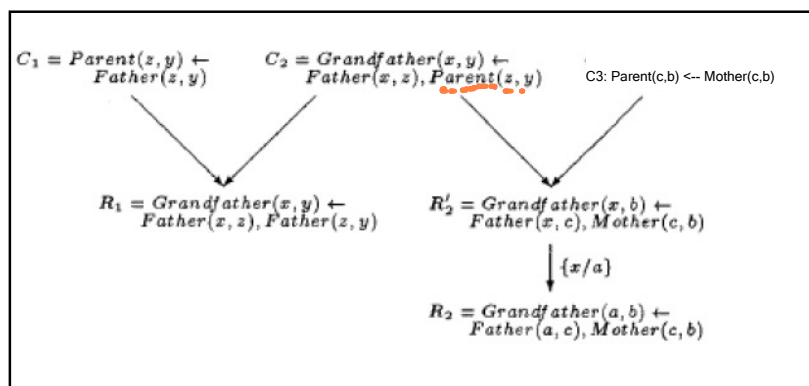
$$C_1' \vee C_2' \vee C_3'$$

R_2

$$C_2' \vee C_3' \vee C_2'$$

Example

Note: Same sign for L_1 & L_3



Sketch of the idea

1. Given R_1 and R_2 , we first try to find a C'_2 such that $C'_2\theta_2 \subseteq R_1$ and $C'_2\sigma_1 \subseteq R_2$, for some θ_2 and σ_1 . Let $D_1 = C'_2\theta_2$ and $D_2 = C'_2\sigma_1$. Clearly, many different C'_2 's can give the same D_1 and D_2 . These C'_2 's can be considered as generalizations of $\{D_1, D_2\}$. We would like to begin with a minimal C'_2 . This motivates the use of the notion of a 'least generalization' of clauses discussed in Section 2.1.1. If we have found a minimal C'_2 , the other possible C'_2 's can be found by taking small generalization steps, starting from the minimal C'_2 . This motivates the use of 'covers' (minimal generalizations or specializations of that clause) and consequently the refinement operator of the clause.

If such a C'_2 cannot be found - which means, intuitively, that R_1 and R_2 have 'nothing in common' - we should let C'_2 be empty.

→ so $C'_2 = \text{lub}(D_1, D_2)$
for some $D_1 \subseteq R_1$
 $D_2 \subseteq R_2$

2. If we have chosen an appropriate C'_2 , we can complete C_2 by choosing also L_2 . In principle, **any** L_2 will do.
3. Once we have decided which clause to take as C_2 , then a C_1 and C_3 can be found *independently*. C_1 can be constructed by the V-operator from C_2 and R_1 , and C_3 can be constructed by the V-operator from C_2 and R_2 .

} Predicate invention

} Reuse V operator...