Optimizing Resolution – Most General Unifiers

The most efficient way to avoid unnecessary search in a first-order derivation is to keep the search as general as possible.

For example:

the clause c_1 contains the literal $P(g(x),f(x),z)$

the clause c_2 contains the literal $\neg P(y,f(w),a)$

For unification, we may have the substitution

$\theta_1=\{x/b,y/g(b),z/a,w/b\}$

or

$\theta_2=\{x/f(z),y/g(f(z)),z/a,w/f(z)\}$

or …

These slides are mainly based on Ronald J. Brachman, Hector J. Levesque. Knowledge representation and reasoning, Morgan Kaufmann, 2004.

Most General Unifiers

$P(g(x),f(x),z)\,\theta_1 = P(y,f(w),a)\,\theta_1 \qquad \theta_1=\{x/b, y/g(b), z/a, w/b\}$

We can try to derive the empty clause using θ_1; if it doesn't work, we can try with θ_2 and so on.

θ_1 and θ_2 are more specific than they should be (it is not necessary to give a value for x).

The substitution $\theta_3=\{y/g(x), z/a, w/x\}$ unifies c_1 and $\bar{c}_2$ without making unnecessary arbitrary choices that might exclude a path to the empty clause.

θ_3 is a most general unifier (MGU).

It may not be unique – for example $\theta_4=\{y/g(w), z/a, x/w\}$ is also an MGU.

Most General Unifiers

Def. A most general unifier θ of literals ρ_1 and ρ_2 is a unifier that has the property that for any other unifier θ', there is a substitution θ^* such that $\theta'=\theta\cdot\theta^*$.

By $\theta\cdot\theta^*$ we mean that we first apply θ and then apply θ^* to the result.

For example, from θ_3 we can get to θ_1 by further applying x/b:

$\rho_1=P(g(x),f(x),z)$

$\rho_2=P(y,f(w),a)$

$$\{\underbrace{y/g(x),z/a,w/x}_{\theta_3}\}\cdot\{x/b\} \Rightarrow \theta_1=\{x/b,y/g(b),z/a,w/b\}$$

Similarly, from θ_3 to θ_2 by applying x/f(z); and to θ_4 by applying x/w.

By limiting resolution to MGUs, the completeness is maintained and the number of resolvents is drastically reduced.

Most General Unifiers

The procedure for computing an MGU:

Input: literals ρ_1 and ρ_2

Output: a substitution θ

1. $\theta = \{\}$
2. If $\rho_1\theta=\rho_2\theta$ then exit
3. Determine the disagreement set DS, which is the pair of terms in the first place (from left to right) where the two literals disagree. For example:

   If $\rho_1\theta=P(a,f(a,\underline{g(z)},\ldots)$

   $\rho_2\theta=P(a,f(a,\underline{u},\ldots)$ then $DS=\{u,g(z)\}$

4. Find a variable $v\in DS$ and a term $t\in DS$ not containing v; if none then fail.
5. Otherwise, set $\theta=\theta\cdot\{v/t\}$ and go to 2.

Most General Unifiers

The procedure for computing an MGU:

Input: literals ρ_1 and ρ_2

Output: a substitution θ

1. $\theta = \{\}$
2. If $\rho_1\theta=\rho_2\theta$ then exit
3. Determine the disagreement set DS, which is the pair of terms in the first place (from left to right) where the two literals disagree. For example:

   If $\rho_1\theta=P(a,f(a,\underline{g(z)},\ldots)$
   $\rho_2\theta=P(a,f(a,\underline{u},\ldots)$ then $DS=\{u,g(z)\}$

4. Find a variable $v \in DS$ and a term $t \in DS$ not containing v; if none then fail.
5. Otherwise, set $\theta=\theta\cdot\{v/t\}$ and go to 2.

Example

$\theta=\{\}$; $\rho_1\theta=P(\underline{x},f(a,g(z)))$ $\Rightarrow$ $\theta=\{x/h(y)\}$; $\rho_1\theta=P(h(y),f(\underline{a},g(z)))$ $\Rightarrow$ $\theta=\{x/h(a),y/a\}$; $\rho_1\theta=P(h(a),f(a,\underline{g(z)}))$ $\Rightarrow$ $\theta=\{x/h(a),y/a,u/g(z)\}$

$\rho_2\theta=P(\underline{h(y)},f(y,u))$ $\rho_2\theta=P(h(y),f(\underline{y},u))$ $\rho_2\theta=P(h(a),f(a,\underline{u})$ Output

The procedure is very efficient in practice. All resolution-based systems use MGUs.

Resolution – other optimizations to improve search

Clause elimination

There are types of clauses that do not participate in the (shortest) derivation to the empty set:

-*pure clauses* – contain some literal ρ such that $\bar{\rho}$ does not appear anywhere else;

-*tautologies* – contain both ρ and $\bar{\rho}$ and they can be bypassed in any derivation;

-***subsumed clauses*** – clauses for which there already exists another clause with a subset of the literals, possible after a substitution.

For example, if [P(x)] ∈ KB then we do not need [P(a),Q(b)] for the shortest derivation of [].

If we have [p,r], we don't need [p,q,r].

Resolution – other optimizations to improve search

Ordering strategies

-choose a predefined order to perform resolution to maximize the chance of deriving the empty clause.

-the best strategy up-to-date is “unit preference”, that is, to use unit clauses first.

A unit clause + a clause with k literals $\Rightarrow$ a clause of length k-1…

Resolution – other optimizations to improve search

Special treatment of equality

The explicit use of the axioms of equality can generate many resolvents. A way to avoid this is by introducing a second rule of inference in addition to resolution, called Paramodulation.

We are given two clauses:

c_1 U t=s where t and s are terms

c_2 U ρ[…,t′,…] containing the term t′ as argument

If necessary, we rename the variables in the two clauses to be distinct.

We assume that there is a substitution θ such that tθ = t′θ.

Then we can infer the clause (c_1Uc_2Uρ[…,s,…])θ, which eliminates =, replaces t′ by s and perform substitution θ.

Resolution – other optimizations to improve search

Example (no. 4 in the previous course)

KB
- $\forall x.Married(father(x),mother(x))$
- $father(john)=bill$

Question: Married(bill,mother(john))

$$[\underbrace{father(john)}_{t}=\underbrace{bill}_{s}] \quad [\underbrace{Married}_{\rho}(\underbrace{father(x)}_{t'},mother(x))]$$

$c_1=[]$ $c_2=[]$

$\theta=\{x/john\}$

We can derive [Married(bill,mother(john)] in a single Paramodulation step.

Horn clauses

Horn clauses are a subset of FOL, where the resolution procedure works well. This subset is sufficiently expressive for many problems.

In a resolution-based system, the clauses are used for two purposes:

- To express disjunctions like [Rain,Sleet,Snow] to represent incomplete knowledge;
- To express a conditional – disjunctions like [¬Child,¬Male,Boy] – although it can be read as “someone is not a child, or is not a male, or is a boy”, it is more natural to be understood as a conditional “if someone is a child and a male then is a boy”.

Horn clauses

Def. A Horn clause contains at most one positive literal. A clause with no positive literals is called a negative Horn clause.

Obs. The empty clause is a negative Horn clause.

The positive Horn clause $[\neg p_1,\ldots,\neg p_n,q]$ can be read

“if p_1 and …and p_n then q”.

It is called “rule” and it is written as $p_1 \wedge \ldots \wedge p_n \Rightarrow q$ to emphasize the conditional.

Resolution derivations with Horn clauses

Obs. Two negative clauses cannot resolve together.

A negative and a positive clause produce a negative clause by resolution.

Two positive clauses produce a positive clause.

Resolution over Horn clauses involves always a positive clause.

Prop. Given S a set of Horn clauses and S $\vdash$ c, where c is a negative clause, then there exists a derivation of c where all the new clauses is the derivation (i.e., clauses not in S) are negative.

Resolution derivations with Horn clauses

Proof.

[$c_1,\ldots,c_n=c$ is a derivation iff $c_i \in S$ or c_i is a resolvent of two previous clauses in the sequence]

Suppose that we have a derivation with new positive clauses. Let c′ be the last one (from left to right):

$$\underbrace{c_1,\ldots,c'}_{\text{positive}},\underbrace{\ldots,c_n}_{\text{negative}}=c$$

Instead of producing negative clauses using c′, we will generate these negative clauses using the positive parents of c′.

Resolution derivations with Horn clauses

p positive literal

$\cap$

$q \in c_i$ positive clause

Res c'

$p \in c_j$ positive clause

$\cup$

$\bar{q}$

Res c_{neg}

d negative clause

$\cup$

$\bar{p}$

c_i Res $c_j \to c'$

c' Res d $\to c_{neg}$

Replaced by (c_j Res d) Res $c_i \to c_{neg}$

The derivation still produces c_{neg}, but without using c′. We remove c′ from the derivation and repeat this for every new positive clause introduced.

Thus, we eliminate all of them.

Resolution derivations with Horn clauses

Prop. Given S a set of Horn clauses and $S \vdash c$, where c is a negative clause, then there exists a derivation of c, where each new clause derived is negative and is a resolvent of the previous negative one in the derivation and a clause from S.

Proof. $c_1,\ldots,c',\ldots,c_n=c$

new negative clause

c_i positive $\in S$

c_j negative $\xrightarrow{\text{Res}} c'$

$S \ni c_k$ negative

c'_k positive $\in S$ $\xrightarrow{\text{Res}} \cdots$ c_2 negative

c'_2 positive $\in S$ $\xrightarrow{\text{Res}}$ c_1 negative

c'_1 positive $\in S$ $\xrightarrow{\text{Res}} c$

and we discard all the clauses that are not in this chain.

Resolution derivations with Horn clauses

Prop. Given S a set of Horn clauses and S ⊢ c, where c is a negative clause, then there exists a derivation of c, where each new clause derived is negative and is a resolvent of the previous negative one in the derivation and a clause from S.

Proof. $c_1, \ldots, c', \ldots, c_n = c$

new negative clause

c_i positive $\in S$ Res c'

c_j negative

S ∋ c_k negative Res … c_2 negative Res c_1 negative Res c

c'_k positive $\in$ S c'_2 positive $\in$ S c'_1 positive $\in$ S

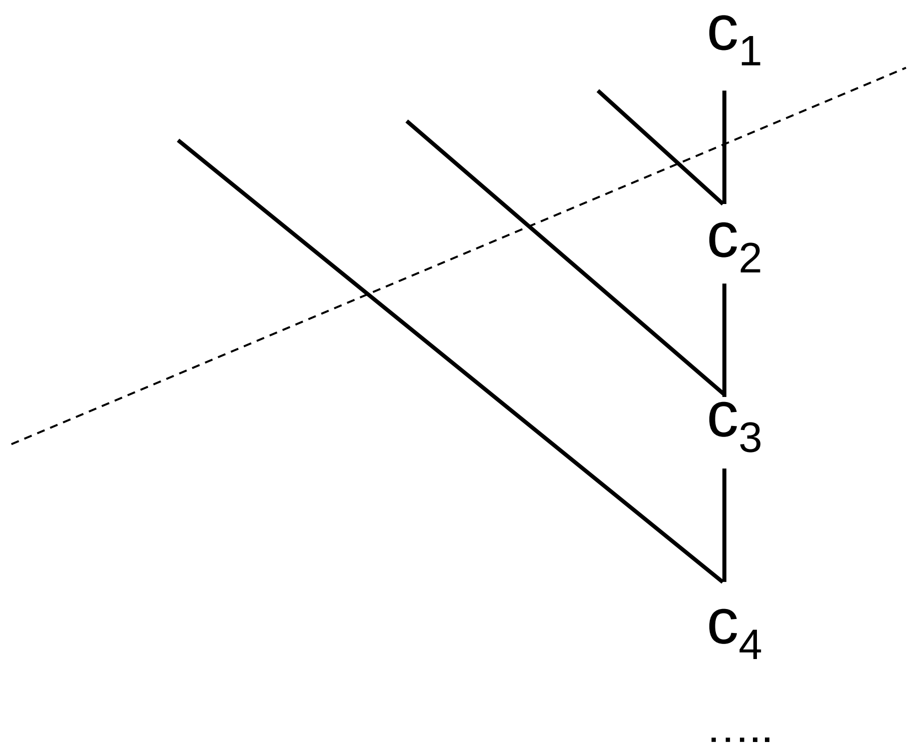
and we discard all the clauses that are not in this chain.

Given S a set of Horn clauses, there is a derivation of a negative clause (including [ ]) iff there is one where each new clause in the derivation is a negative resolvent of the previous negative clause in the derivation and a clause from S.

SLD Resolution – Selected literals, Linear pattern, over Definite clauses

It is a restricted form of resolution, where each new clause is a resolvent of the previous clause and a clause from the original set S. This version of resolution is sufficient for Horn clauses.

Def. If S is a set of clauses (not necessarily Horn), an SLD derivation is a sequence $c_1, \ldots, c_n = c$, where $c_1 \in S$ and c_{i+1} is a resolvent of c_i and a clause in S. We write $S \vdash_{SLD} c$.



Except for c_1, the elements of S are not explicitly mentioned.

SLD Resolution

It is clear that if $S \vdash_{SLD} [\,]$ then $S \vdash [\,]$ but the converse doesn't hold.

For example, for $S=\{[p,q], [\neg p,q], [p, \neg q], [\neg p, \neg q]\}$, we have that $S \vdash [\,]$.

To generate [], the last step in resolution should involve $[\rho]$ and $[\overline{\rho}]$ for some literal ρ. But S does not contain any unit clauses, so there is no element from S in the last step of the resolution. That means that $S \nvdash_{SLD} [\,]$.

But for S a set of Horn clauses, then $S \vdash_{SLD} [\,]$ iff $S \vdash [\,]$.

Moreover, each of the new clauses in the derivation $c_2,\ldots,c_n$ can be assumed to be negative.

c_2 has a negative and a positive parent, so c_1 can be chosen to be the negative one.

Thus, for Horn clauses, SLD derivations of the empty clause begin with a negative clause in S.

SLD Resolution

Toddler

Toddler⊃Child

Infant⊃Child

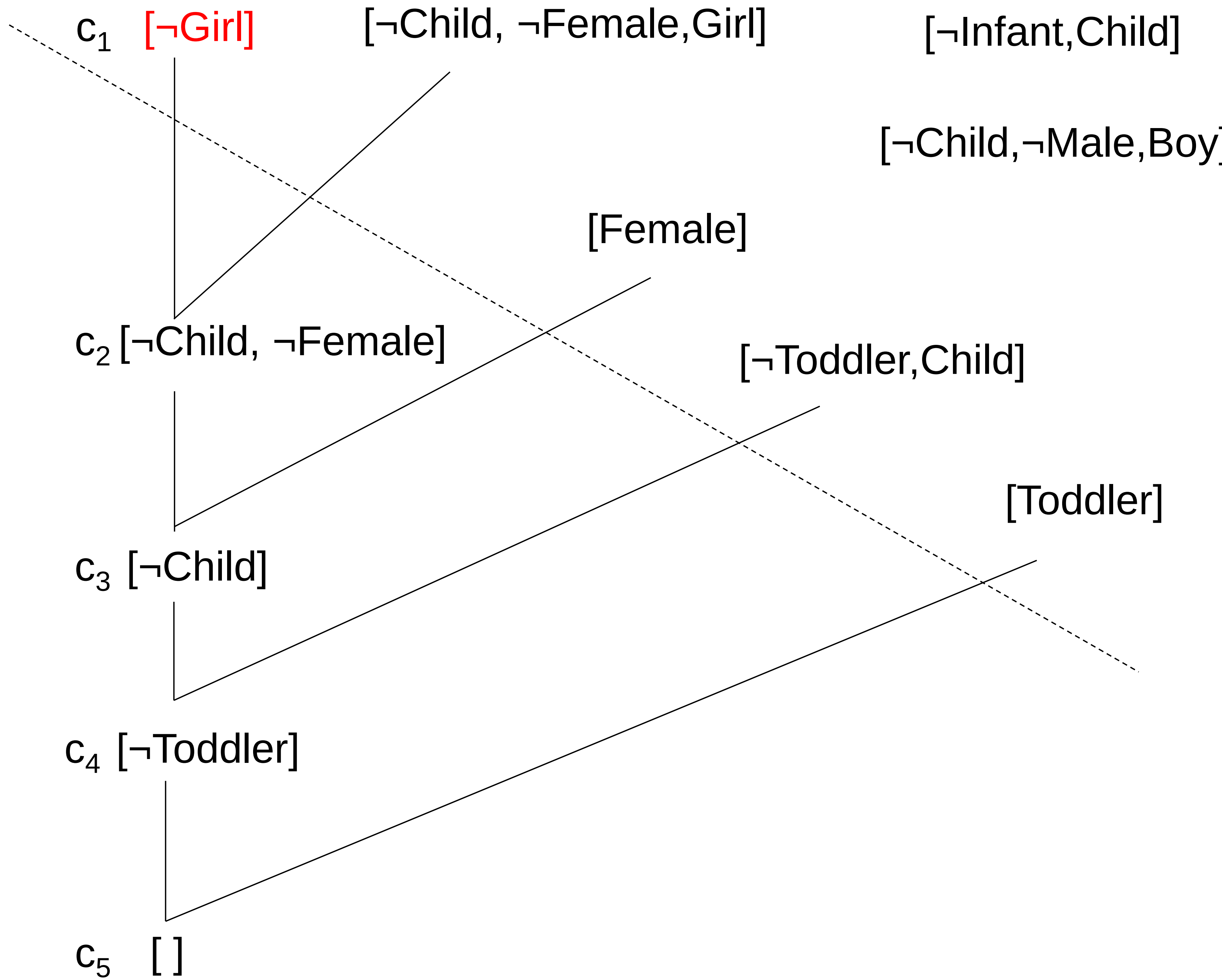
Child∧Female⊃Girl

Child∧Male⊃Boy

Female

Question: Girl

Obs. [¬Girl] is the only negative clause in S, so c_1=[¬Girl]



SLD derivation: c_1, c_2, c_3, c_4, c_5

Computing SLD derivations - Backward chaining

Given KB, a set of positive Horn clauses, we want to determine whether a set of atoms can be entailed from KB.

The case considered here consists of determining the satisfiability of a set of Horn clauses containing exactly one negative clause.

Backward chaining

Input: KB and a finite list of atomic sentences $q_1,\ldots,q_n$

Output: YES or NOT – whether or not KB entails all q_i

procedure SOLVE($q_1,\ldots,q_n$)

  if n=0 the return YES

  for each clause c∈KB

    if ($c=[q_1,\neg p_1,\ldots,\neg p_m]$ and SOLVE($p_1,\ldots,p_m,q_2,\ldots,q_n$)) then return YES

  return NO

Computing SLD derivations - Backward chaining

The search goes backward, from goals to facts in KB

$[\neg q_1,\ldots,\neg q_n]$

$[q_1,\neg p_1,\ldots,\neg p_m]$

$[\neg p_1,\ldots,\neg p_m,\neg q_2,\ldots,\neg q_n]$

The subgoals the goal q_1 reduces to

if it fails, it tries with another clause in KB whose positive literal is q_1. If none is found then return NO

The procedure works in a depth-first manner, as it attempts to solve the new goals p_i before the old ones q_j.

It is called left-to-right because it solves the goals $q_1,\ldots,q_n$ in order $1,2,\ldots,n$.

This is how PROLOG solves goals.

Computing SLD derivations - Backward chaining

Toddler	Child∧Female⊃Girl
Toddler⊃Child	Child∧Male⊃Boy
Infant⊃Child	Female

Question: Girl

The goal Girl reduces to the subgoals Child and Female

The goal Child reduces to Toddler – Toddler is proven (it is fact in KB)

The goal Female is proven (it is fact in KB)

Return YES

Computing SLD derivations - Backward chaining

Obs. The procedure can go into a infinite loop (even in the propositional case!).

[¬p]

[p,¬p]

[¬p]

[¬p]

……

Computing SLD derivations - Backward chaining

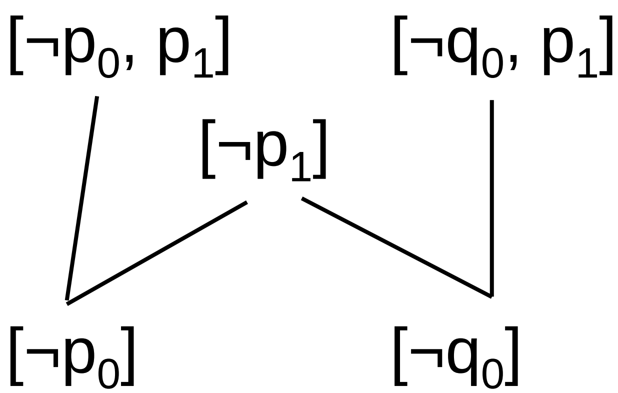
In other cases, the procedure can be exponential.

For example,

$$\left[\begin{array}{ll} p_{i-1} \Rightarrow p_i & \\ p_{i-1} \Rightarrow q_i & 0<i\leq n,\ n>0 \\ q_{i-1} \Rightarrow p_i & 4(n+1)\text{-}4 \text{ clauses} \\ q_{i-1} \Rightarrow q_i & \end{array}\right.$$

Question: p_n (or q_n) - neither is entailed by KB

n=1



2^1 steps

Assume that for n=k-1 at least 2^{k-1} steps are necessary

$p_{k-1} \Rightarrow p_k$ at least $2^{k-1}+2^{k-1}=2^k$ steps necessary to show

$q_{k-1} \Rightarrow p_k$ that p_k is not entailed by KB.

Forward chaining – a more efficient approach for the propositional case

The procedure works from the facts in a Horn KB towards the goals.

Input: KB and a finite list of atomic sentences $q_1,\ldots,q_n$

Output: YES or NOT – whether or not entails all q_i

Procedure

1. If (all the goals q_i are solved) then return YES
2. Check if there is a clause $[p,\neg p_1,\ldots,\neg p_m]$ in KB, such that all of its negative atoms $p_1,\ldots,p_m$ are marked as solved and the positive atom p is not solved.
3. If (there is such a clause) then mark p as solved and go to step 1
   else return NO

Forward chaining

Toddler	Child∧Female⊃Girl
Toddler⊃Child	Child∧Male⊃Boy
Infant⊃Child	Female

Question: Girl

[Toddler] has no negative atoms – it is marked as solved

[Child, ¬Toddler] – Child is marked

[Female] – has no negative atoms – it is marked

[Girl, ¬Child, ¬Female] – Girl is marked – return YES

The forward chaining has a much better overall behavior than the backward chaining.

At each iteration, we search for a clause in KB with an atom that has not been marked. The overall result will not be exponential.

Horn clauses in FOL

In the propositional case, we can always determine whether or not a Horn KB entails an atom (with forward chaining).

But the problem of determining whether a set of Horn clauses in FOL entails an atom remains undecidable.

Backward chaining – infinite branches

KB: $\forall x \forall y.LessThan(succ(x),y) \supset LessThan(x,y)$

Question: LessThan(zero,zero)

Forward chaining – doesn't terminate

KB: LessThan(zero,succ(zero))

$\forall x \forall y.LessThan(x,y) \supset LessThan(x,succ(y))$

Question: LessThan(5,7)