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:

1. To express disjunctions like [Rain, Sleet, Snow] to represent incomplete knowledge.
2. To express a conditional - disjunctions like [¬Child, ¬Male, Boy] - although it can be read as "someone is not a child, or is not 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".

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 $p_1 \wedge \cdots \wedge p_n \Rightarrow q$ 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 in the derivation (i.e. clauses not in S) are negative.

<u>Proof</u>. [$c_1, \dots, c_n = c$ is a derivation iff $c_i \in S$ or c_i is a resolvent of two previous clauses in the sequence]

Suppose we have a derivation with new positive clauses. Let c' be the last one:

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

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

$q \in C_i$ positive clause

$p \in C_j$ positive clause, $p \ni \bar{q}$ ... $\xrightarrow{Res}$ $C' \ni p$ positive literal

C', d negative clause, $d \ni \bar{p}$ $\xrightarrow{Res}$ C_{neg}

$$C_i \text{ Res } C_j \longrightarrow C' \qquad C' \text{ Res } d \longrightarrow C_{neg}$$

Replaced by $(C_j \text{ Res } d) \text{ Res } C_i \longrightarrow C_{neg}$

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.

<u>Prop</u>. 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 one in the derivation and a clause from S.

<u>Proof</u>. Using the previous Prop., we can assume that all new clauses in the derivation are negative. So, all the positive clauses are from S.

$$c_1, \dots, \underset{\text{new negative clause}}{c'}, \dots c_n = c$$

c_i positive $\in S$, c_j negative $\xrightarrow{Res}$ c'

$S \ni C_K$ negative, C_K' positive $\in S$ $\xrightarrow{Res}$ ... C_2 negative, c_2' pos $\in S$ $\xrightarrow{Res}$ c_1 neg, c_1' pos $\in S$ $\xrightarrow{Res}$ C

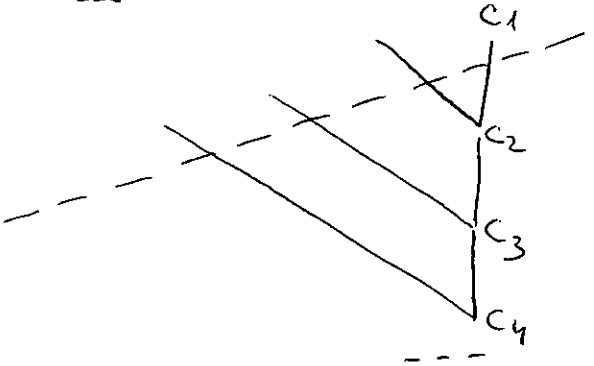
and we discard all the clauses that are not in this chain.

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 clause in the derivation and a clause from S.

<u>SLD Resolution</u> (<u>S</u>elected literals, <u>L</u>inear pattern, over <u>D</u>efinite 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.

<u>Def</u>. 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.

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 $[\bar{\rho}]$ for some literal ρ. But S does not contain any unit clauses, so there is no element from S in the last step of Resolution. That means that $S \not\vdash_{SLD} [\,]$.

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

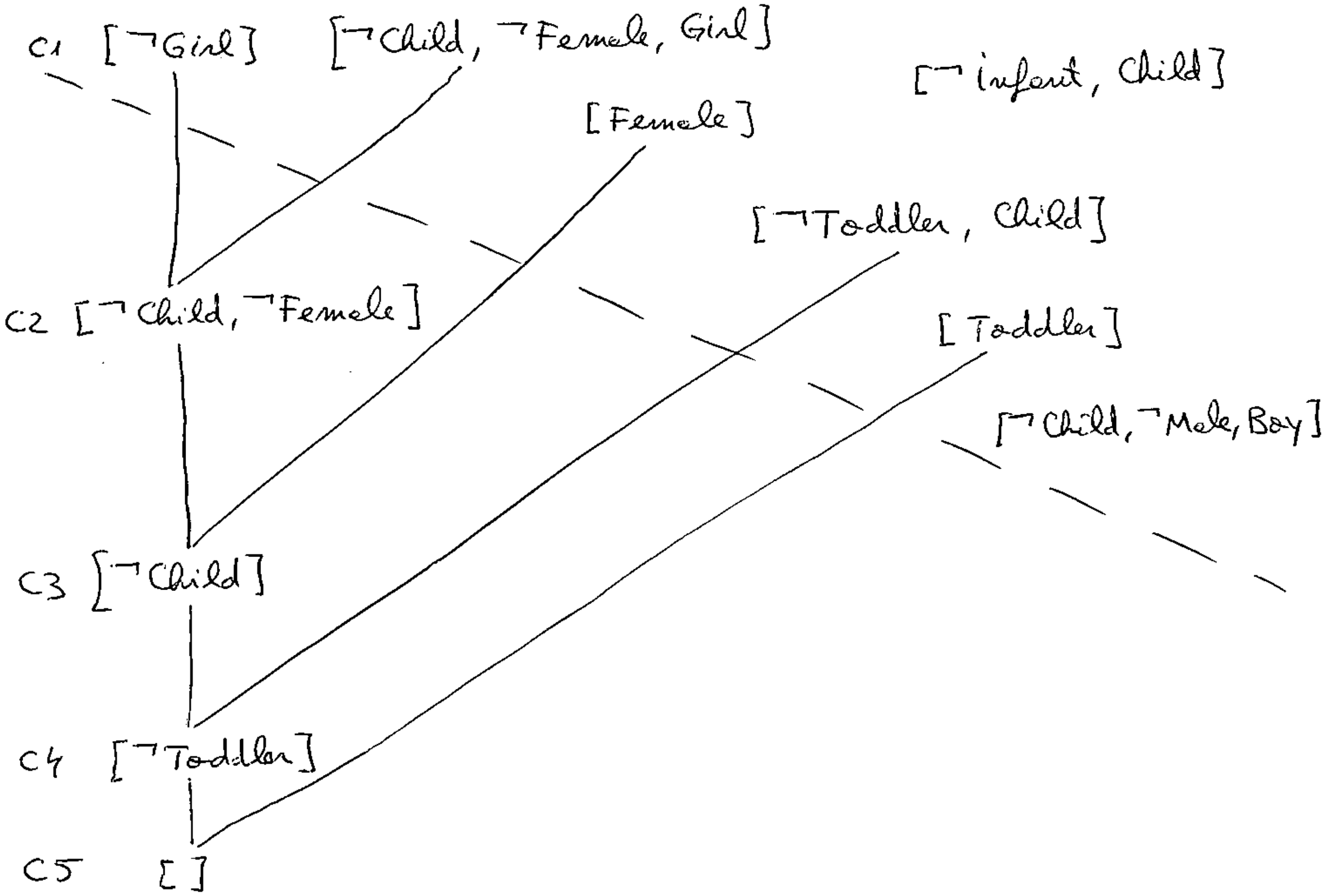
Moreover, each of the new clauses in the derivation $c_2, \dots, c_n$ can be assumed to be negative.
c_2 has a negative and a positive parent, so c_1 can be chosen to be the negative one.
Thus, for Horn clauses, SLD derivations of the empty clause begin with a negative clause in S.

Example 1

KB Horn clauses:
- Toddler
- Toddler ⊃ Child
- Child ∧ Male ⊃ Boy
- infant ⊃ Child
- Child ∧ Female ⊃ Girl
- 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

Goal trees

All the literals in all the clauses in a Horn SLD derivation of the empty clause are negative. To produce [ ], we need positive Horn clauses to eliminate the negative literals.

For example, if we have a unit positive clause in S [Toddler] and [¬Toddler] in a derivation, we say that the goal Toddler is solved.

if a positive Horn clause introduces other negative literals

[¬Toddler, Child]   [¬Child]

[¬Toddler]

we say that the goal Child reduces to the subgoal Toddler.

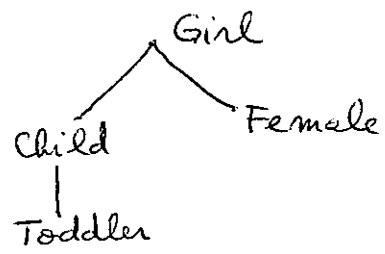
[¬Child, ¬Female, Girl]   [¬Girl]

[¬Child, ¬Female]

the goal Girl reduces to two subgoals: Child and Female.

In Example 1, the SLD derivation can be reformulated as following: we start with the goal Girl; this reduces to two subgoals Child and Female; the goal Female is solved; Child reduces to Toddler; Toddler is solved.

The associated goal tree is:



For a complete SLD derivation, the leaves of the tree are solved goals.

The Horn clauses and SLD derivations, represented as goal trees, are the basis of PROLOG.

Example 2 Concatenation of lists

$$\text{KB}\begin{cases}\text{Append}(nil, y, y)\\ \text{Append}(x, y, z) \Rightarrow \text{Append}(cons(w, x), y, cons(w, z))\end{cases}$$

Question: $\exists u.\ \text{Append}(cons(a, cons(b, nil)), cons(c, nil), u)$

nil - empty list

$cons(a, nil)$ — $[a]$

$cons(a, cons(b, nil))$ — $[a, b]$

[¬ Append(cons(a, cons(b, nil)), cons(c, nil), u)]

[¬ Append(x, y, z), Append(cons(w, x), y, cons(w, z))]

w/a, x/cons(b, nil)
y/cons(c, nil), z/t, u/cons(a, t)

[Append(nil, y, y)]

[¬ Append(cons(b, nil), cons(c, nil), t)]

w/b, x/nil, y/cons(c, nil)
z/q, t/cons(b, q)

[¬ Append(nil, cons(c, nil), q)]

y/cons(c, nil)
q/cons(c, nil)

[ ]

The associated goal tree is:

[Append(cons(a, cons(b, nil)), cons(c, nil), u)]

|

[Append(cons(b, nil), cons(c, nil), t)]

|

[Append(nil, cons(c, nil), q)]

The answer $u = cons(a, cons(b, cons(c, nil)))$ can be extracted from the derivation:

$q/cons(c, nil) \longrightarrow t/cons(b, q) \longrightarrow u/cons(a, t)$

Computing SLD derivations

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 $q_1, \dots, q_n$

output: YES or NOT - whether or not KB entails all q_i

```
procedure SOLVE [q1, ..., qn]
    if (n = 0) then return YES
    for each clause c ∈ KB
        if (c = [q1, ¬p1, ..., ¬pm] and SOLVE[p1, ..., pm, q2, ..., qn])
                then return YES
    return NO
```

The search goes backward, from goals to facts in KB.

$[\neg q_1, \neg q_2, \dots, \neg q_n]$

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

$[\neg p_1, \neg p_2, \dots, \neg p_m, \neg q_2, \dots, \neg q_n]$ - if it fails, it tries with another clause in KB whose positive literal is q_1. If none is found then return NO.

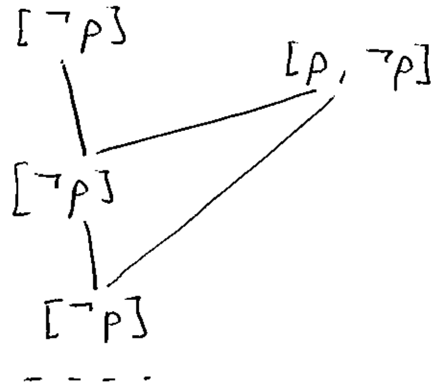
the subgoals the goal q_1 reduces to

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

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

This is how PROLOG solves goals.

<u>Obs</u>. The procedure can go into an infinite loop.



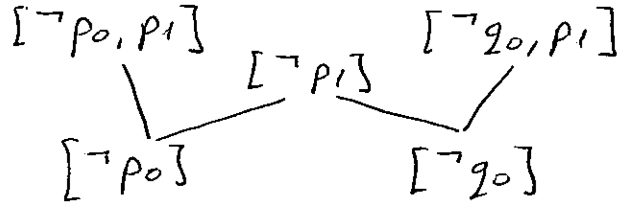
In other cases, the procedure can be exponential.

For example,

$$KB \begin{cases} p_{i-1} \Rightarrow p_i \\ p_{i-1} \Rightarrow q_i \\ q_{i-1} \Rightarrow p_i \\ q_{i-1} \Rightarrow q_i \end{cases} \qquad 0 < i < n \quad 4n-4 \text{ clauses}$$

Question: p_i (or q_i) – neither is entailed by KB

$i = 1$



2^1 steps

assume that for $k-1$, 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

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

<u>Forward chaining</u>

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

input: KB and a finite list of atomic sentences $q_1, \dots, q_n$

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

procedure

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

□

We mark atoms as solved when we determine that they are entailed by KB.

In Example 1 [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.

In the propositional case, we can determine whether or not a Horn KB entails an atom. But in FOL, the forward chaining procedure may not terminate.

[¬LessThan(0,0)] [LessThan(x, y), ¬LessThan(succ(x), y)]

$x/0,\ y/0$

[¬LessThan(1,0)]

$x/1,\ y/0$

[¬LessThan(2,0)]

– – –

The problem of determining whether a set of Horn clauses in FOL entails an atom is undecidable.