Horn Clauses and SLD Resolution

Marco Piastra

Horn Clauses (in L_P)

- Definition

  A ***Horn Clause*** is a wff in CF
  that contains at most <u>one</u> literal in positive form

- Three types of *Horn Clauses*:

  Rule: two or more literals, one positive

  Examples: $\{B, \neg D, \neg A, \neg C\}, \{A, \neg B\}$ (equivalent to: $(D \land A \land C) \to B,\ B \to A$)

  Facts: just one positive literal

  Examples: $\{B\}, \{A\}$

  Goal: one or more literals, all negative

  Examples: $\{\neg B\}, \{\neg A, \neg B\}$

  More terminology:

  Rules and facts are also called *definite clauses*

  Goals are allo called *negative clauses*

Lost in Translation…

Many wffs can be translated into Horn clauses:

$(A \land B) \rightarrow C$
$\neg(A \land B) \lor C$ (rewriting →)
$\neg A \lor \neg B \lor C$ (De Morgan - CF – it is a rule)

$A \rightarrow (B \land C)$
$\neg A \lor (B \land C)$ (rewriting →)
$(\neg A \lor B) \land (\neg A \lor C)$ (distributing ∨)
$(\neg A \lor B), (\neg A \lor C)$ (CF – <u>two</u> rules)

$(A \lor B) \rightarrow C$
$\neg(A \lor B) \lor C$ (rewriting →)
$(\neg A \land \neg B) \lor C$ (De Morgan)
$(\neg A \lor C) \land (\neg B \lor C)$ (distributing ∨)
$(\neg A \lor C), (\neg B \lor C)$ (CF – <u>two</u> rules)

But not all of them:

$(A \land \neg B) \rightarrow C$
$\neg(A \land \neg B) \lor C$ (rewriting →)
$\neg A \lor B \lor C$ (De Morgan)

$A \rightarrow (B \lor C)$
$\neg A \lor B \lor C$ (rewriting →)

SLD Resolution (in L_P)

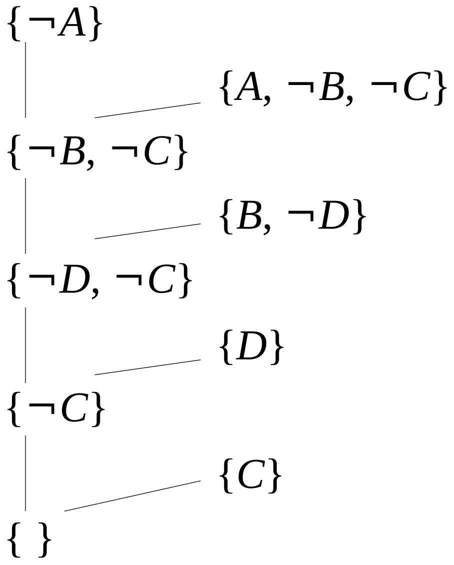
Linear resolution with Selection function for Definite clauses

- Algorithm

  Starts from a set of *definite clauses* (also the *program*) + a *goal*

  1) At each step, the *selection function* identifies a *literal* in the *goal* (i.e. *subgoal*)
  2) All *definite clause* applicable to the *subgoal* is selected
  3) The resolution rule is applied generating the *resolvent*

  Termination: either the empty clause { } is obtained or step 2) fails.



$\{\neg A\}$

$\{A, \neg B, \neg C\}$

$\{\neg B, \neg C\}$

$\{B, \neg D\}$

$\{\neg D, \neg C\}$

$\{D\}$

$\{\neg C\}$

$\{C\}$

$\{\ \}$

Example:

Selection function: *leftmost subgoal first*

Definite clauses: $\{C\}, \{D\}, \{B, \neg D\}, \{A, \neg B, \neg C\}$

Goal: $\{\neg A\}$

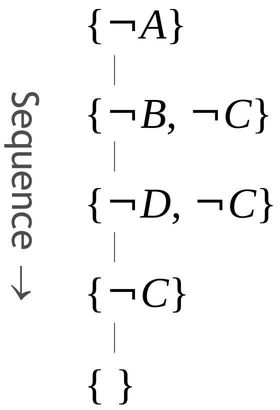
SLD trees (in L_P)

SLD derivations

Example: $\{C\}, \{D\}, \{B, \neg D\}, \{A, \neg B, \neg C\}$ goal $\{\neg A\}$

In this example each subgoal can be resolved in one mode only

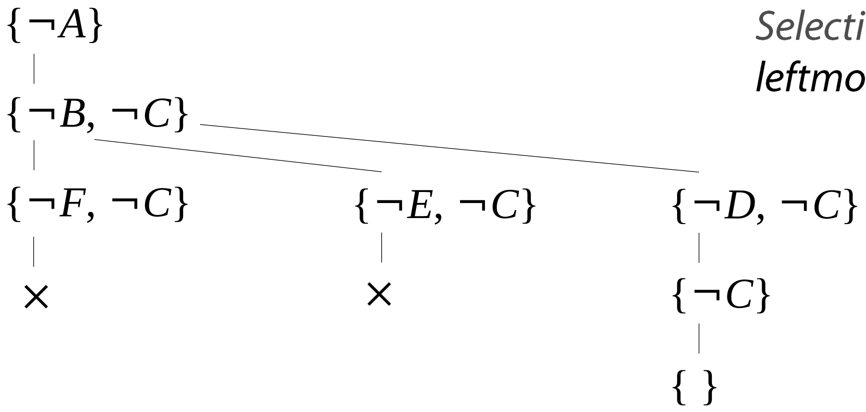
This is not true in general



- SLD trees (= trace of all SLD derivations from a goal)

Example: $\{C\}, \{D\}, \{B, \neg F\}, \{B, \neg E\}, \{B, \neg D\}, \{A, \neg B, \neg C\}$ goal $\{\neg A\}$

A few new rules have been added: there are now different possibilities



Selection function:
leftmost subgoal first

Each branch correspond to a possible resolution for a *subgoal*

SLD Resolution (in L_P)

- A resolution method <u>for Horn clauses</u> in L_P

    It always terminates

    It is *correct*: $\Gamma \vdash \varphi \Rightarrow \Gamma \models \varphi$

    It is *complete*: $\Gamma \models \varphi \Rightarrow \Gamma \vdash \varphi$

- Computationally efficient

  It has polynomial time complexity (w.r.t the # of propositional symbols occurring in Γ and φ)

- Limitations

  Not all problems can be translated into Horn clauses

      The “Harry is happy” problem does not translate

    Γ : only a set of *rules* and *facts*

    φ : only a conjunction of *facts*

Horn Clauses in L_{FO}

The definition is very similar to the propositional case

- Horn Clauses (of the skolemization of a set *sentences*)

  Each clause contains at most one literal in positive form

  Facts, rules and goals

  Fact: a clause with just an individual *atom*

  $\{Human(socrates)\}$, $\{Pyramid(x)\}$, $\{Sister(sally, motherOf(paul))\}$

  Rule: a clause with at least two literals, exactly one in positive form

  $\{Human(x), \neg Philosopher(x)\}$,

  $\forall x\ (Philospher(x) \rightarrow Human(x))$

  $\{\neg Female(x), \neg Parent(k(x),x), \neg Parent(k(y),y), Sister(x,y)\}$

  $\forall x \forall y\ ((Female(x) \wedge \exists z\ (Parent(z,x) \wedge Parent(z,y))) \rightarrow Sister(x,y))$

  $\{\neg Above(x,y), On(x,k(x))\}$, $\{\neg Above(x,y), On(j(y),y)\}$

  $\forall x \forall y\ (Above(x,y) \rightarrow (\exists z\ On(x,z) \wedge \exists v\ On(v,y)))$

  Goal: a clause containing negative literals only

  $\{\neg Human(socrates)\}$

  $\{\neg Sister(sally,x), \neg Sister(x,paul)\}$

  Negation of $\exists x\ (Sorella(sally,x) \wedge Sorella(x,paul))$

SLD Resolution in L_{FO}

- Input: a program Π and a goal ϕ

Program Π (i.e. a set of *definite clauses:* rules + facts) in some predefined linear order:

$\gamma_1, \gamma_2, \ldots, \gamma_n$ (each γ_i is a *definite clause*)

Goal ϕ (i.e. a conjunction of facts in negated form), which becomes the *current goal* ψ

Note: the *selection function* for the *current goal* and *subgoal* will be discussed in the next slide

Procedure:

1) Select a negative literal $\neg\alpha$ (i.e. the *subgoal*) in the *current goal* ψ
2) Scan the program (in the predefined order) to identify a clause candidate literal γ_i
3) Try unifying $\neg\alpha$ and $std(\alpha')$ (i.e. apply the standardization of variables to α')
4) If there is a *unifier* σ of $\neg\alpha$ and $std(\alpha')$, replace the current goal with the *resolvent* of $std(\gamma_i)[\sigma]$ and $\psi[\sigma]$ (i.e. first apply σ to both $std(\gamma_i)$ and ψ and then generate the resolvent)
5) Then, if the *resolvent* is the empty clause, terminate with <u>success</u>, otherwise select a new *current goal* and resume from step 1)
6) Else, if the unification fails , scan the program and select a new candidate literal γ_i and resume from step 3)
7) Else, if there are no further clauses in the program, select a new *current goal* and resume from step 1)
8) If all the goals in the tree have been fully explored, terminate with <u>failure</u>

SLD Resolution in L_{FO}

- *Two alternative selection functions:*

  Depth-first (which is the most common…)

  - Always select the *most recent goal*, i.e. the *resolvent* which has been generated last, as the *current goal* ϕ
  - Then, in the current goal ϕ, select the leftmost *subgoal* $\neg\alpha$ not selected yet
  - When the current goal ϕ is fully explored and no new *resolvent* has been generated, select the next *most recent* goal in the tree (*backtracking*)

  Breadth-first

  - Always select the <u>*least*</u> *recent* goal as the *current goal* ϕ
  - Then, in the current goal ϕ, select the leftmost *subgoal* $\neg\alpha$ not selected yet
  - When the current goal ϕ is fully explored select the next *least recent* goal in the tree

  Comparison

  Breadth-first is a *fair* selection function, in the sense that every possible resolution will be eventually attempted (i.e. 'it leaves nothing behind').

  Depth-first tends to save memory and be more efficient, but it is NOT *fair* (more to follow)

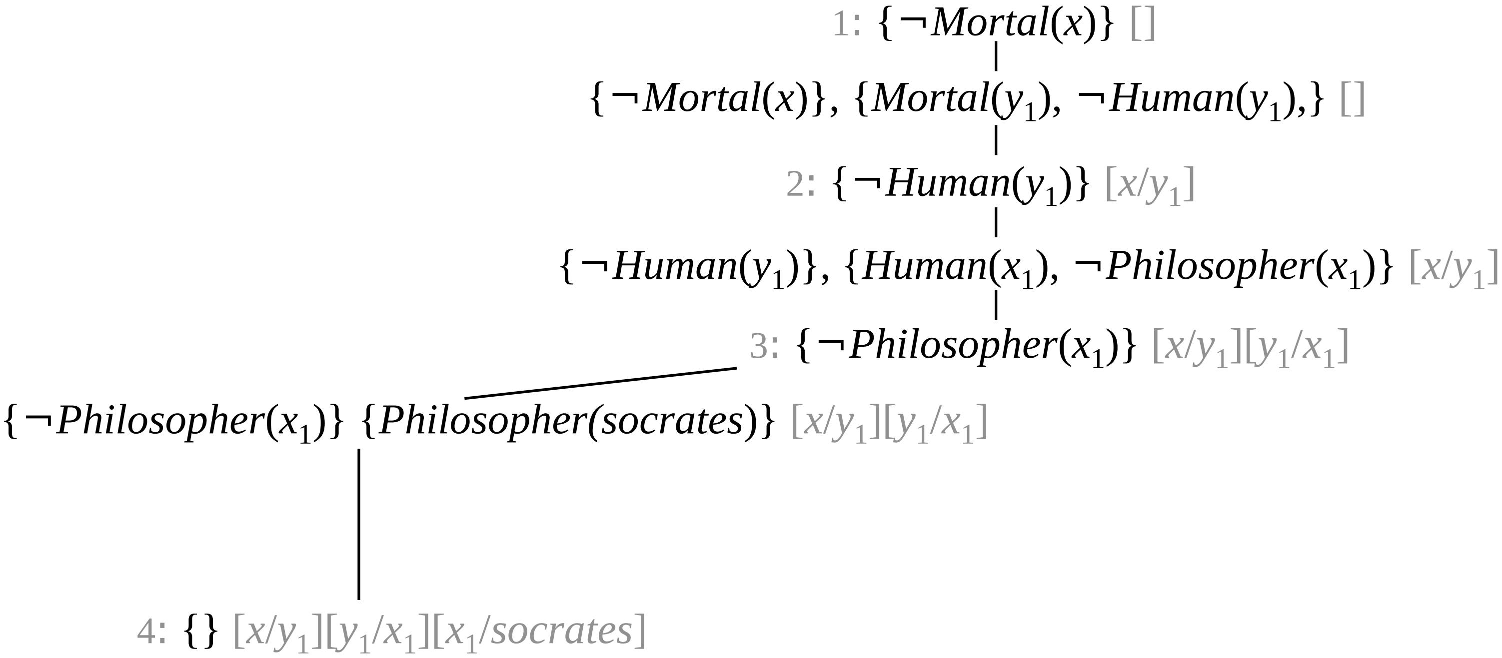
SLD Trees

- Example (depth-first selection function):

$$\Pi \equiv \{\{Human(x), \neg Philosopher(x)\}, \{Mortal(y), \neg Human(y)\},$$
$$\{Philosopher(socrates)\}, \{Philosopher(plato)\}, \{Philosopher(aristotle)\}\}$$

$$goal \equiv \{\neg Mortal(x), \neg Human(x)\}$$

"Is there anyone who is both human and mortal?"



1: $\{\neg Mortal(x)\}$ $[]$

$\{\neg Mortal(x)\}, \{Mortal(y_1), \neg Human(y_1),\}$ $[]$

2: $\{\neg Human(y_1)\}$ $[x/y_1]$

$\{\neg Human(y_1)\}, \{Human(x_1), \neg Philosopher(x_1)\}$ $[x/y_1]$

3: $\{\neg Philosopher(x_1)\}$ $[x/y_1][y_1/x_1]$

$\{\neg Philosopher(x_1)\}$ $\{Philosopher(socrates)\}$ $[x/y_1][y_1/x_1]$

4: $\{\}$ $[x/y_1][y_1/x_1][x_1/socrates]$

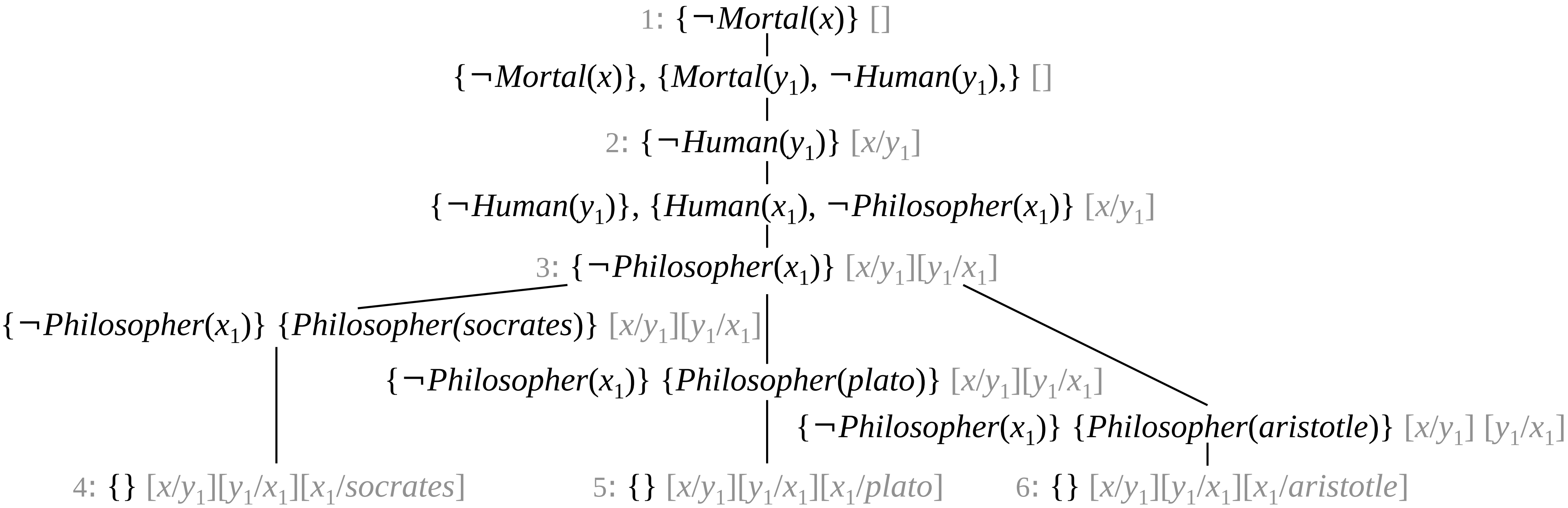
SLD Trees

- Example (depth-first selection function, forcing full exploration of SLD tree):

$$\Pi \equiv \{\{Human(x), \neg Philosopher(x)\}, \{Mortal(y), \neg Human(y)\},$$
$$\{Philosopher(socrates)\}, \{Philosopher(plato)\}, \{Philosopher(aristotle)\}\}$$

$$goal \equiv \{\neg Mortal(x), \neg Human(x)\}$$

"Is there anyone who is both human and mortal?"



1: $\{\neg Mortal(x)\}$ $[]$

$\{\neg Mortal(x)\}, \{Mortal(y_1), \neg Human(y_1),\}$ $[]$

2: $\{\neg Human(y_1)\}$ $[x/y_1]$

$\{\neg Human(y_1)\}, \{Human(x_1), \neg Philosopher(x_1)\}$ $[x/y_1]$

3: $\{\neg Philosopher(x_1)\}$ $[x/y_1][y_1/x_1]$

$\{\neg Philosopher(x_1)\}$ $\{Philosopher(socrates)\}$ $[x/y_1][y_1/x_1]$

$\{\neg Philosopher(x_1)\}$ $\{Philosopher(plato)\}$ $[x/y_1][y_1/x_1]$

$\{\neg Philosopher(x_1)\}$ $\{Philosopher(aristotle)\}$ $[x/y_1]\ [y_1/x_1]$

4: $\{\}$ $[x/y_1][y_1/x_1][x_1/socrates]$

5: $\{\}$ $[x/y_1][y_1/x_1][x_1/plato]$

6: $\{\}$ $[x/y_1][y_1/x_1][x_1/aristotle]$

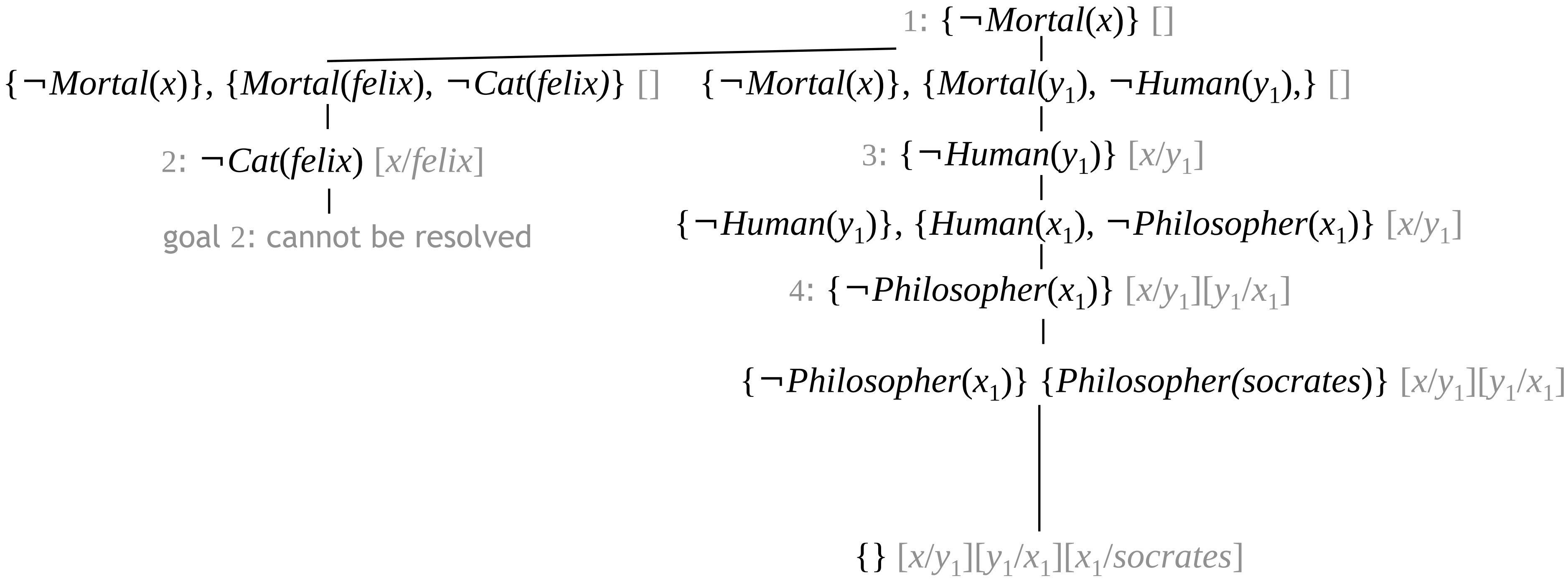
SLD Trees

- Another example (depth-first selection function):

$$\Pi \equiv \{\{Mortal(felix), \neg Cat(felix)\}, \{Human(x), \neg Philosopher(x)\}, \{Mortal(y), \neg Human(y)\}, \{Philosopher(socrates)\}, \{Philosopher(plato)\}, \{Philosopher(aristotle)\}\}$$

$$goal \equiv \{\neg Mortal(x), \neg Human(x)\}$$

"Is there anyone who is both human and mortal?"



1: $\{\neg Mortal(x)\}$ $[]$

$\{\neg Mortal(x)\}, \{Mortal(felix), \neg Cat(felix)\}$ $[]$

2: $\neg Cat(felix)$ $[x/felix]$

goal 2: cannot be resolved

$\{\neg Mortal(x)\}, \{Mortal(y_1), \neg Human(y_1),\}$ $[]$

3: $\{\neg Human(y_1)\}$ $[x/y_1]$

$\{\neg Human(y_1)\}, \{Human(x_1), \neg Philosopher(x_1)\}$ $[x/y_1]$

4: $\{\neg Philosopher(x_1)\}$ $[x/y_1][y_1/x_1]$

$\{\neg Philosopher(x_1)\}$ $\{Philosopher(socrates)\}$ $[x/y_1][y_1/x_1]$

$\{\}$ $[x/y_1][y_1/x_1][x_1/socrates]$

The world of lists

- Lists of items $[a, b, c, \ldots]$

  cons/2
  it's a function that associates items (e.g. a) to a list (e.g. $[b, c]$)
  $cons(a,cons(b,cons(c,nil)))$ is the list $[a, b, c]$

  Append/3
  it's a predicate: each pair of lists x and y is associated to their *concatenation* z

  nil
  it's a constant, the *empty list*.

Shorthand notation (Prolog):

$[] \Leftrightarrow nil$
$[a] \Leftrightarrow cons(a,nil)$
$[a,b] \Leftrightarrow cons(a,cons(b,nil))$
$[a|[b,c]] \Leftrightarrow cons(a,[b,c])$

Axioms (**AL**)

$\forall x\ Append(nil,x,x)$
$\forall x \forall y \forall z\ (Append(x,y,z) \rightarrow \forall s\ Append([s,x],y,[s,z]))$

The world of lists

Problem: $\forall x\, Append(nil, x, x) \models \exists y\, \forall x\, Append(nil, cons(y, x), cons(a, x))$

1: $\forall x\, Append(nil, x, x),\ \neg\exists y\, \forall x\, Append(nil, cons(y, x), cons(a, x))$ (refutation)

2: $\forall x\, Append(nil, x, x),\ \forall y\, \exists x\, \neg Append(nil, cons(y, x), cons(a, x))$ (prenex normal form)

3: $\{Append(nil, x, x)\}, \{\neg Append(nil, cons(y, k(y)), cons(a, k(y)))\}$

($k/1$ is a Skolem function, clausal form)

(N.B. there is no *skolemization* in Prolog : the programmer does it)

The pair of **literals**

$Append(nil, x, x),\ \neg Append(nil, cons(y, k(y)), cons(a, k(y))))$

… contains the same predicate $Append/3$ but the arguments are <u>**different**</u>

There is however an MGU $\sigma = [x/cons(a, k(a)), y/a]$ that yields

$\{Append(nil, cons(a,k(a)), cons(a,k(a)))\}, \{\neg Append(nil, cons(a, k(a)), cons(a, k(a)))\}$

From this, the resolvent is the empty clause.

The world of lists in Prolog

```
% Identical to built-in predicate append/3, although it uses "cons"
% as a defined predicate, thus allowing trace-ability.

append(cons(S,X),Y,cons(S,Z)) :- append(X,Y,Z).
append(nil,X,X).

% WARNING: express your queries with cons. Examples:
% ?- append(cons(a,nil), cons(b,cons(c, nil)),cons(a,cons(b,cons(c, nil)))).
% ?- append(X,Y,cons(a,cons(b,cons(c, nil)))).
```

Infinite SLD Trees (*fairness of SLD*)

- A first example:

$\Pi \equiv \{\{P(x), \neg P(x)\}\}$

$\neg\phi \equiv \{\neg P(x)\}$

goal: $\neg P(x)$ $[]$

$\{\neg P(x)\}, \{P(x_1), \neg P(x_1),\}$ $[]$

$\{\neg P(x_1)\}$ $[x/x_1]$

$\{\neg P(x_1)\}, \{P(x_2), \neg P(x_2),\}$ $[x/x_1]$

$\{\neg P(x_2)\}$ $[x/x_1]$ $[x_1/x_2]$

…

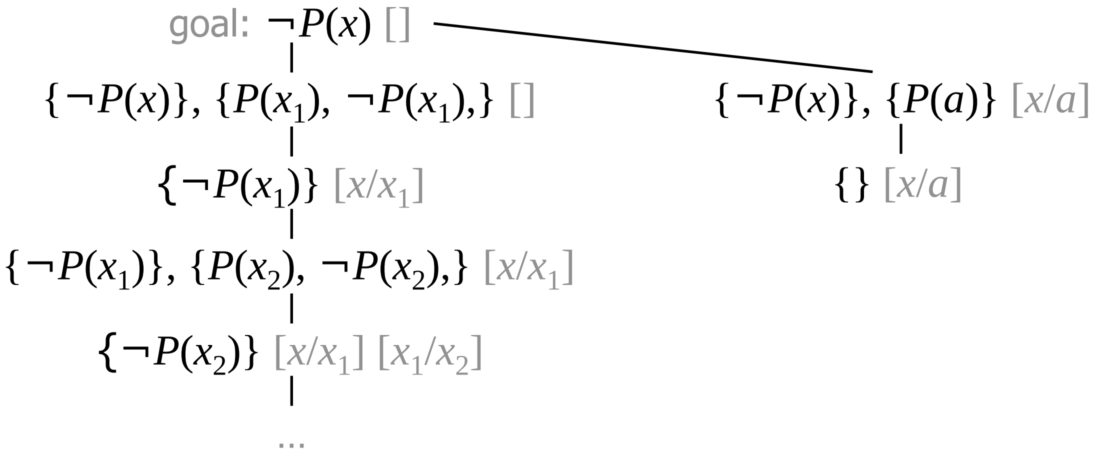
Since $\Pi \not\models \phi$, the method can *diverge* (and it does…)

Infinite SLD Trees (*fairness of SLD*)

- A second example:

$\Pi \equiv \{\{P(x), \neg P(x)\}, \{P(a)\}\}$

$\neg\phi \equiv \{\neg P(x)\}$



goal: $\neg P(x)$ $[]$

$\{\neg P(x)\}, \{P(x_1), \neg P(x_1),\}$ $[]$

$\{\neg P(x_1)\}$ $[x/x_1]$

$\{\neg P(x_1)\}, \{P(x_2), \neg P(x_2),\}$ $[x/x_1]$

$\{\neg P(x_2)\}$ $[x/x_1]$ $[x_1/x_2]$

…

$\{\neg P(x)\}, \{P(a)\}$ $[x/a]$

$\{\}$ $[x/a]$

In this case $\Pi \models \phi$, so the method should *not* diverge.

However, when a *depth-first* selection function is used, the infinite branch in the SLD-tree makes the method diverge anyway.

A ***fair*** selection function is such that no possible resolution will be postponed indefinitely: that is, any possible resolution will be performed, eventually.