Artificial Intelligence

A course about foundations

Horn Clauses and SLD Resolution

Marco Piastra

Back to Propositional Logic

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: $(D \wedge A \wedge C) \rightarrow B, \ B \rightarrow A$)

  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 $\rightarrow$)

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

$A \rightarrow (B \land C)$

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

  $(\neg A \lor B) \land (\neg A \lor C)$ (distributing $\lor$)

  $(\neg A \lor B), (\neg A \lor C)$ (CF – <u>two</u> rules)

$(A \lor B) \rightarrow C$

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

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

  $(\neg A \lor C) \land (\neg B \lor C)$ (distributing $\lor$)

  $(\neg A \lor C), (\neg B \lor C)$ (CF – <u>two</u> rules)

But not all of them:

$(A \land \neg B) \rightarrow C$

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

  $\neg A \lor B \lor C$ (De Morgan)

$A \rightarrow (B \lor C)$

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

SLD Resolution

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* are selected, in the given order
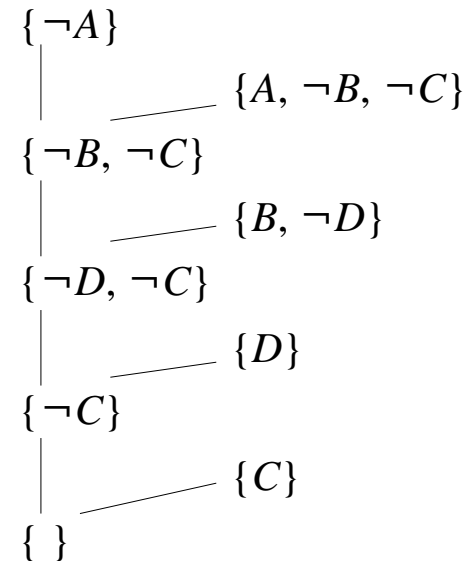 3) The resolution rule is applied generating the *resolvent*

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

  Example:

  Selection function: *leftmost subgoal first*
  Definite clauses: $\{C\}, \{D\}, \{B, \neg D\}, \{A, \neg B, \neg C\}$
  Goal: $\{\neg A\}$

  $$\{\neg A\}$$
  $$\quad \underline{\quad\quad} \ \{A, \neg B, \neg C\}$$
  $$\{\neg B, \neg C\}$$
  $$\quad \underline{\quad\quad} \ \{B, \neg D\}$$
  $$\{\neg D, \neg C\}$$
  $$\quad \underline{\quad\quad} \ \{D\}$$
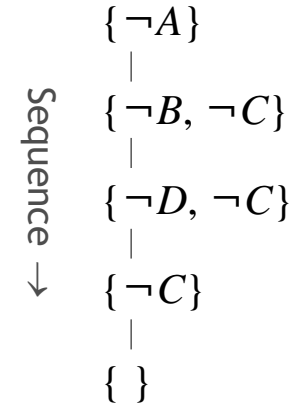 $$\{\neg C\}$$
  $$\quad \underline{\quad\quad} \ \{C\}$$
  $$\{ \}$$

SLD trees

SLD derivations

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

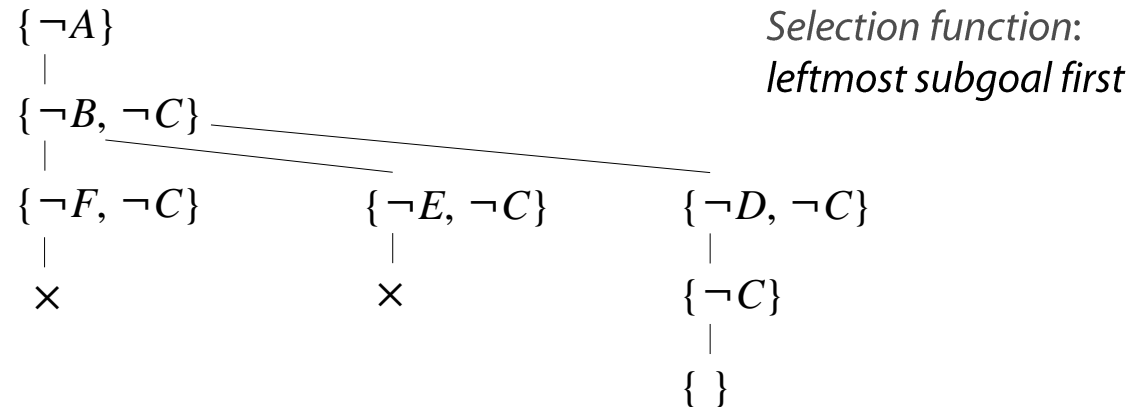*In this example each subgoal can be resolved in one mode only*

This is not true in general

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

Sequence →

- 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

$\{\neg A\}$
|
$\{\neg B, \neg C\}$
|
$\{\neg F, \neg C\}$ $\{\neg E, \neg C\}$ $\{\neg D, \neg C\}$
| | |
×   × $\{\neg C\}$
|
$\{\ \}$

Selection function:
leftmost subgoal first

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

SLD Resolution

- 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)

SLD resolution
in First-Order Logic

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*

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

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

  $\{Human(x), \neg Greek(x)\}$,
  $\forall x \, (Greek(x) \rightarrow Human(x))$

  $\{\neg Female(x), \neg Parent(k(x),x), \neg Parent(k(y),y), \; Sister(x,y)\}$
  $\forall x \forall y \, ((Female(x) \wedge \exists z \, (Parent(z,x) \wedge Parent(z,y))) \rightarrow Sister(x,y))$

  $\{\neg Above(x,y), On(x,k(x))\}$, $\{\neg Above(x,y), On(j(y),y)\}$
  $\forall x \forall y \, (Above(x,y) \rightarrow (\exists z \, On(x,z) \wedge \exists v \, On(v,y)))$

  Goal: a clause containing negative literals only

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

  $\{\neg Sister(sally,x), \neg Sister(x,paul)\}$
  Negation of $\quad \exists x \, (Sister(sally,x) \wedge Sister(x,paul))$

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:

  $$\gamma_1, \gamma_2, \dots, \gamma_n \qquad \text{(each } \gamma_i \text{ is a } \textit{definite clause}\text{)}$$

  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 $std(\gamma_i)$ (i.e. apply the standardization of variables to α')
  4) If there is a *unifier σ* of $\neg\alpha$ and $std(\gamma_i)$, 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):

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

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

"Is there anyone who is mortal?"

1: $\{\neg Mortal(x)\}$ []
|
$\{\neg Mortal(x)\}, \{Mortal(y_1), \neg Human(y_1),\}$ []
|
2: $\{\neg Human(y_1)\}$ $[x/y_1]$
|
$\{\neg Human(y_1)\}, \{Human(x_1), \neg Greek(x_1)\}$ $[x/y_1]$
|
3: $\{\neg Greek(x_1)\}$ $[x/y_1][y_1/x_1]$

$\{\neg Greek(x_1)\}$ $\{Greek(socrates)\}$ $[x/y_1][y_1/x_1]$
|
|
|
4: $\{\}$ $[x/y_1][y_1/x_1][x_1/socrates]$
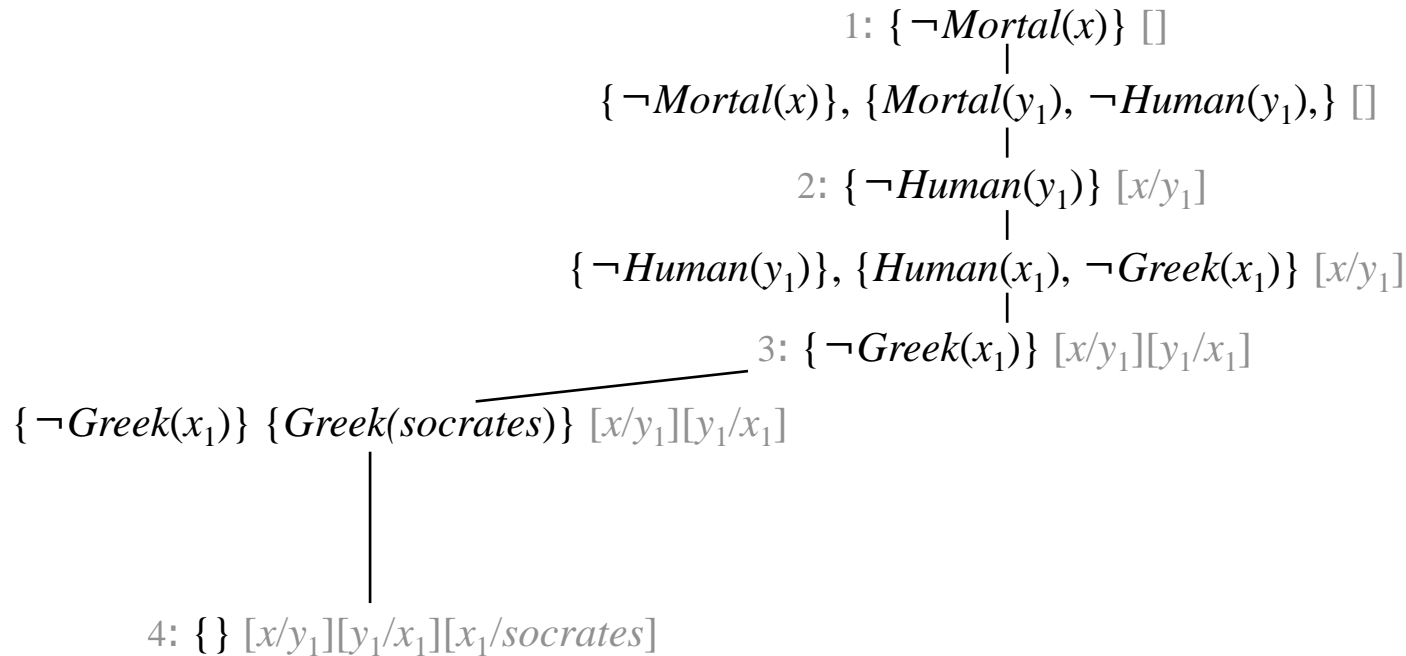
SLD Trees

- **Example** (depth-first selection function, forcing full exploration of SLD tree):
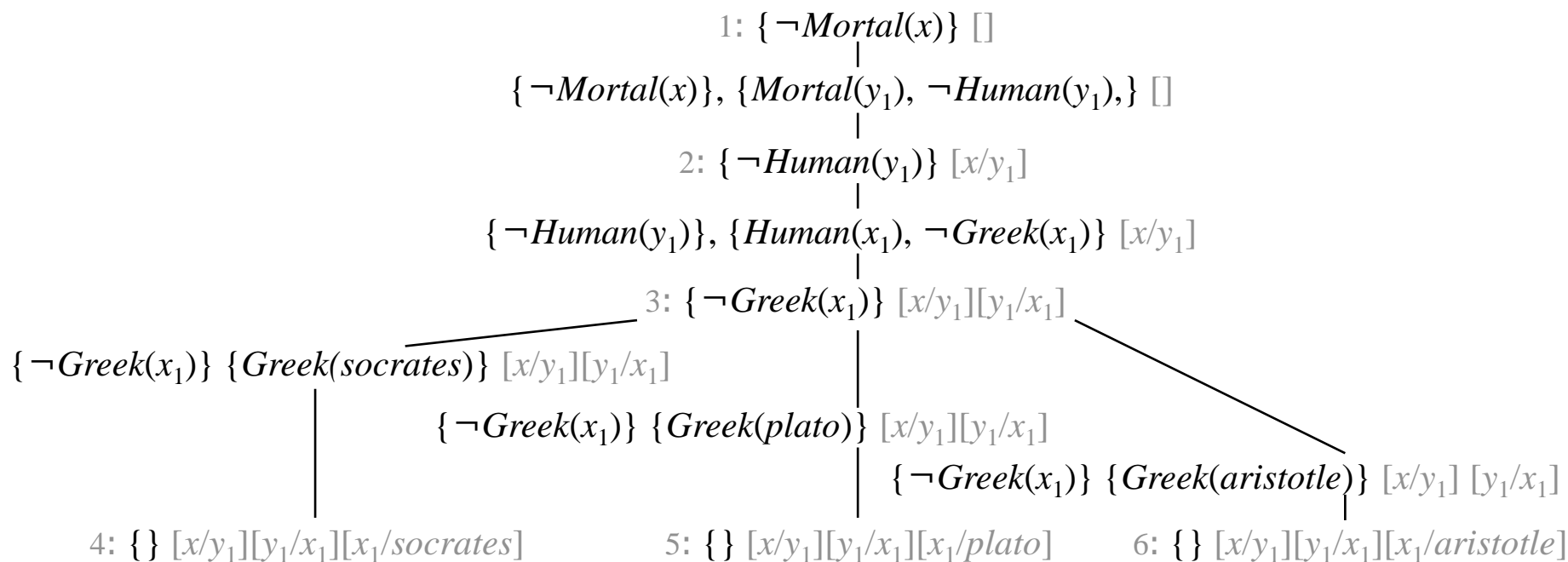
    $\Pi \equiv \{\{Human(x), \neg Greek(x)\}, \{Mortal(y), \neg Human(y)\},$
    $\qquad\{Greek(socrates)\}, \{Greek(plato)\}, \{Greek(aristotle)\}\}$

    $goal \equiv \{\neg Mortal(x)\}$
    "Is there anyone who is mortal?"

1: $\{\neg Mortal(x)\}$ []
|
$\{\neg Mortal(x)\}, \{Mortal(y_1), \neg Human(y_1),\}$ []
|
2: $\{\neg Human(y_1)\}$ $[x/y_1]$
|
$\{\neg Human(y_1)\}, \{Human(x_1), \neg Greek(x_1)\}$ $[x/y_1]$
|
3: $\{\neg Greek(x_1)\}$ $[x/y_1][y_1/x_1]$

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

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

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

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), ¬Cat(felix)*$\}, \{$*Human(x), ¬Greek(x)*$\}, \{$*Mortal(y), ¬Human(y)*$\},$
$\{$*Greek(socrates)*$\}, \{$*Greek(plato)*$\}, \{$*Greek(aristotle)*$\}\}$

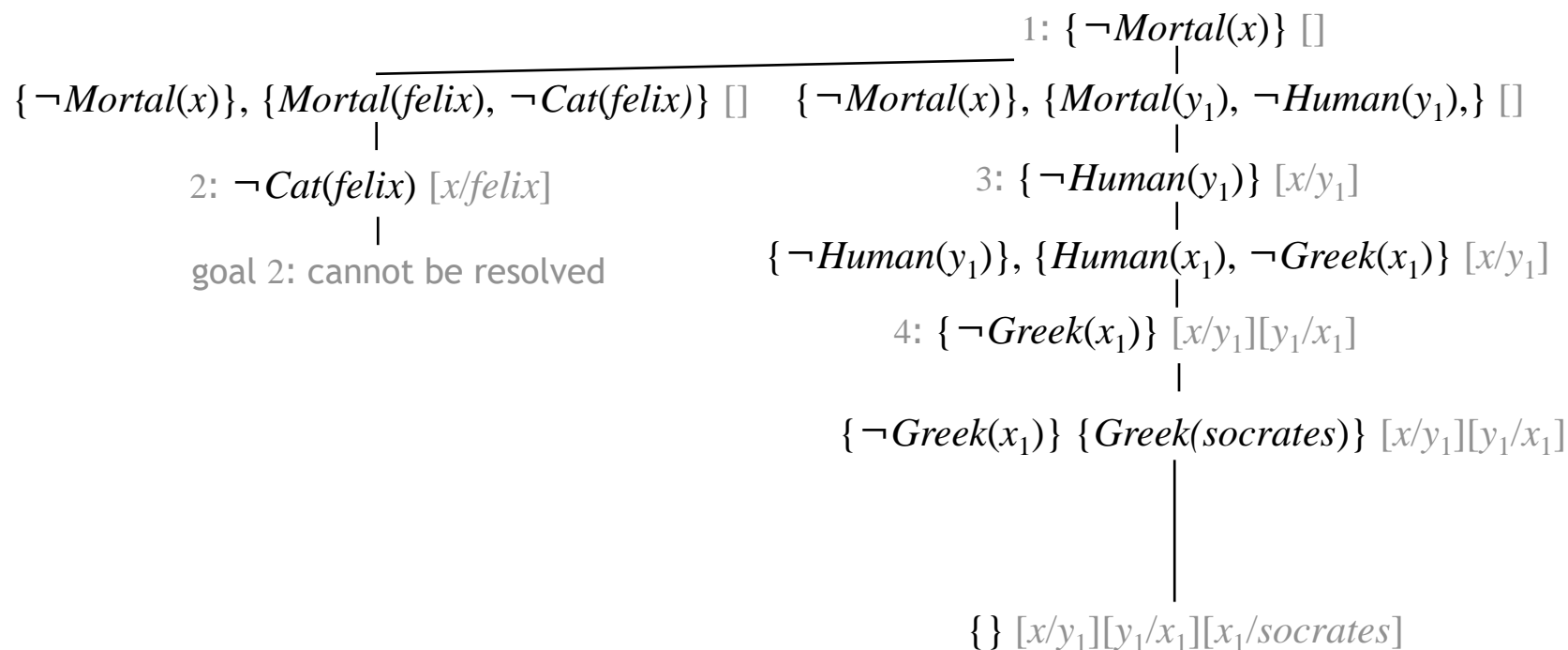$goal \equiv \{$*¬Mortal(x)*$\}$

"Is there anyone who is mortal?"

1: $\{$*¬Mortal(x)*$\}$ $[\,]$

$\{$*¬Mortal(x)*$\}, \{$*Mortal(felix), ¬Cat(felix)*$\}$ $[\,]$ $\{$*¬Mortal(x)*$\}, \{$*Mortal(y_1), ¬Human(y_1),*$\}$ $[\,]$

2: *¬Cat(felix)* $[x/felix]$ 3: $\{$*¬Human(y_1)*$\}$ $[x/y_1]$

goal 2: cannot be resolved $\{$*¬Human(y_1)*$\}, \{$*Human(x_1), ¬Greek(x_1)*$\}$ $[x/y_1]$

4: $\{$*¬Greek(x_1)*$\}$ $[x/y_1][y_1/x_1]$

$\{$*¬Greek(x_1)*$\}$ $\{$*Greek(socrates)*$\}$ $[x/y_1][y_1/x_1]$

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

*The discreet charme of functions

- Representing data structures: *lists of items* [*a*, *b*, *c*, …]

  ## Symbols in Σ

  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*))) represents 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, represents the *empty list*.

  ## Axioms (**AL**)

  $\forall x$ *Append*(*nil*, *x*, *x*)
  $\forall x \, \forall y \, \forall z$ (*Append*(*x*, *y*, *z*) $\rightarrow \forall s$ *Append*(*cons*(*s*, *x*), *y*, *cons*(*s*, *z*)))

  ## Examples of entailment

  {**AL** + $\exists z$ *Append*(*cons*(*a*, *nil*), *cons*(*b*, *cons*(*c*, *nil*), *z*) }

  $\models$ *Append*(*cons*(*a*, *nil*), *cons*(*b*, *cons*(*c*, *nil*)), *cons*(*a*, *cons*(*b*, *cons*(c, nil))))

  {**AL** + $\exists x \, \exists y$ *Append*(*x*, *y*, *cons*(*a*, *cons*(*b*, *nil*)))}

  $\models$ *Append*(*cons*(*a*, *nil*), *cons*(*b*, *nil*), *cons*(*a*, *cons*(*b*, *nil*)))
  $\models$ *Append*(*nil*, *cons*(*a*, *cons*(*b*, *nil*)), *cons*(*a*, *cons*(*b*, *nil*)))
  $\models$ *Append*(*cons*(*a*, *cons*(*b*, *nil*)),*nil*, *cons*(*a*, *cons*(*b*, *nil*)))

The world of lists

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

    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

```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*)

- An example:

$\Pi \equiv \{\{S(a,b)\}, \{S(b,c)\}, \{S(x,z), \neg S(x,y), \neg S(y,z)\}\}$

$\neg\phi \equiv \{\neg S(a,x)\}$

$$\text{goal: } \neg S(a,x) \; []$$
$$|$$
$$\{\neg S(a,x)\}, \{S(a,b)\} \; []$$
$$|$$
$$\{\} \; [x/b]$$

Easy…

Infinite SLD Trees (*fairness of SLD*)

- An example:

    $\Pi \equiv \{\{S(a,b)\}, \{S(b,c)\}, \{S(x,z), \neg S(x,y), \neg S(y,z)\}\}$

    $\neg\phi \equiv \{\neg S(a,x)\}$

goal: $\neg S(a,x)$ []

$\{\neg S(a,x)\}, \{S(a,b)\}$ [] $\{\neg S(a,x)\}, \{S(x_3,z_3), \neg S(x_3,y_3), \neg S(y_3,z_3)\}$ []

$\{\}$ $[x/b]$ $\{\neg S(a,y_3), \neg S(y_3,z_3)\}$ $[x_3/a, x/z_3]$

$\{\neg S(a,y_3), \neg S(y_3,z_3)\}, \{S(a,b)\}$ $[x/z_3, x_3/a, y_3/b]$

$\{\neg S(b,z_3)\}$ $[x/z_3, x_3/a]$

$\{\neg S(b,z_3)\}, \{S(b,c)\}$ $[x/z_3, x_3/a]$

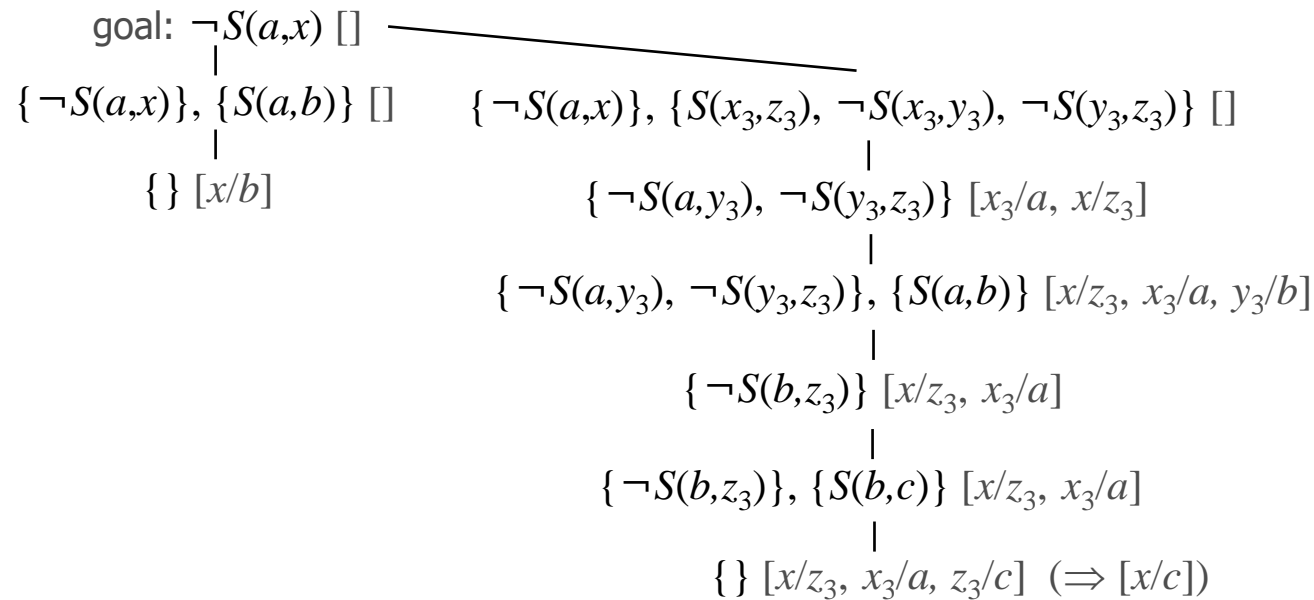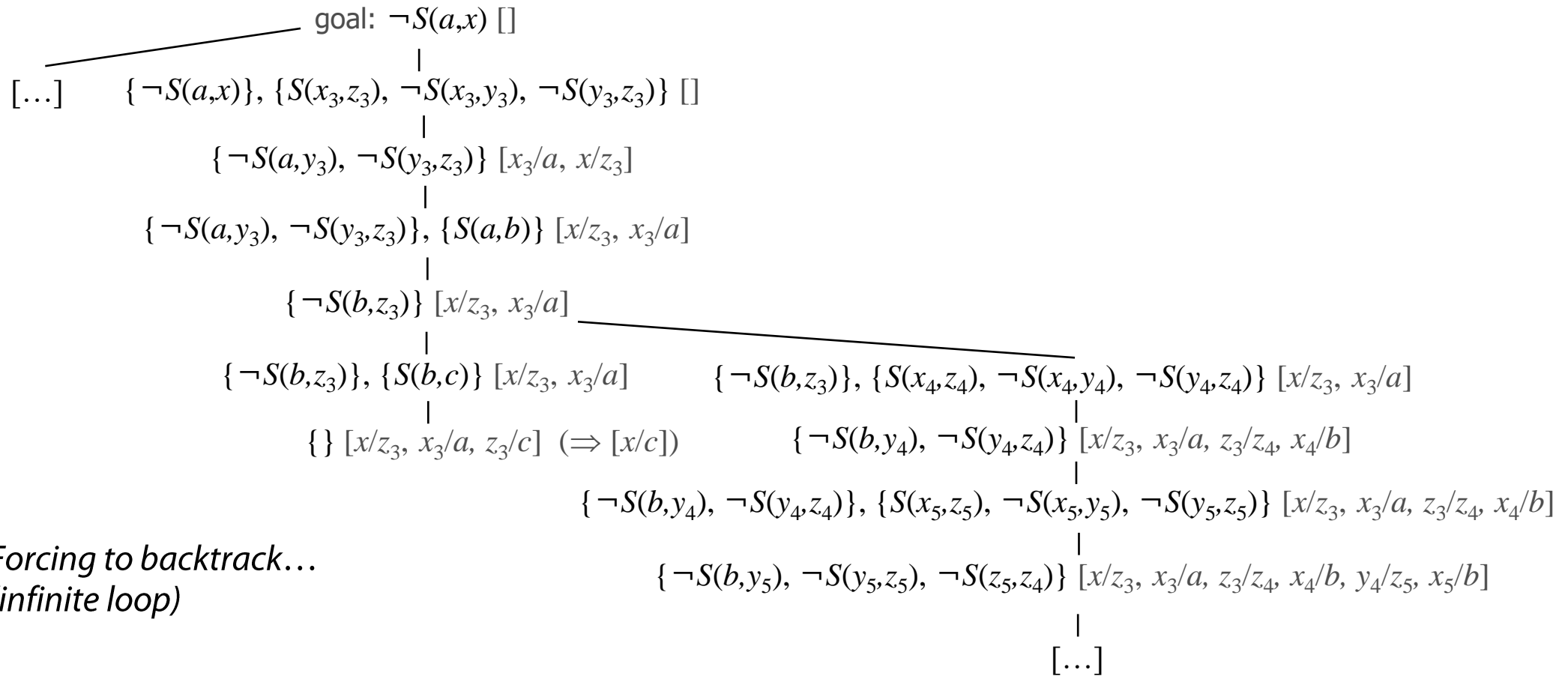$\{\}$ $[x/z_3, x_3/a, z_3/c]$ $(\Rightarrow [x/c])$

Forcing to backtrack…
(easy again)

Infinite SLD Trees (*fairness of SLD*)

- An example:

$\Pi \equiv \{\{S(a,b)\}, \{S(b,c)\}, \{S(x,z), \neg S(x,y), \neg S(y,z)\}\}$

$\neg\phi \equiv \{\neg S(a,x)\}$

goal: $\neg S(a,x)$ []

$[\ldots]$ $\{\neg S(a,x)\}, \{S(x_3,z_3), \neg S(x_3,y_3), \neg S(y_3,z_3)\}$ []

$\{\neg S(a,y_3), \neg S(y_3,z_3)\}$ $[x_3/a, x/z_3]$

$\{\neg S(a,y_3), \neg S(y_3,z_3)\}, \{S(a,b)\}$ $[x/z_3, x_3/a]$

$\{\neg S(b,z_3)\}$ $[x/z_3, x_3/a]$

$\{\neg S(b,z_3)\}, \{S(b,c)\}$ $[x/z_3, x_3/a]$ $\{\neg S(b,z_3)\}, \{S(x_4,z_4), \neg S(x_4,y_4), \neg S(y_4,z_4)\}$ $[x/z_3, x_3/a]$

$\{\}$ $[x/z_3, x_3/a, z_3/c]$ $(\Rightarrow [x/c])$ $\{\neg S(b,y_4), \neg S(y_4,z_4)\}$ $[x/z_3, x_3/a, z_3/z_4, x_4/b]$

$\{\neg S(b,y_4), \neg S(y_4,z_4)\}, \{S(x_5,z_5), \neg S(x_5,y_5), \neg S(y_5,z_5)\}$ $[x/z_3, x_3/a, z_3/z_4, x_4/b]$

Forcing to backtrack…
(infinite loop)

$\{\neg S(b,y_5), \neg S(y_5,z_5), \neg S(z_5,z_4)\}$ $[x/z_3, x_3/a, z_3/z_4, x_4/b, y_4/z_5, x_5/b]$

$[\ldots]$

Infinite SLD Trees (*fairness of SLD*)

- A second example:

  $\Pi \equiv \{\{S(x,z), \neg S(x,y), \neg S(y,z)\}, \{S(a,b)\}, \{S(b,c)\}\}$

  $\neg\phi \equiv \{\neg S(a,x)\}$

  Notice the change in clause ordering…..

goal: $\neg S(a,x)$ []
|
$\{\neg S(a,x)\}, \{S(x_1,z_1), \neg S(x_1,y_1), \neg S(y_1,z_1)\}$ []
|
$\{\neg S(a,y_1), \neg S(y_1,z_1)\}$ $[x_1/a, x/z_1]$
|
$\{\neg S(a,y_1), \neg S(y_1,z_1)\}, \{S(x_2,z_2), \neg S(x_2,y_2), \neg S(y_2,z_2)\}$ $[x_1/a, x/z_1]$
|
$\{\neg S(z_2,z_1), \neg S(x_2,y_2), \neg S(y_2,z_2)\}$ $[x_1/a, x/z_1, x_2/a, y_1/z_2]$
|
[…]

The *infinite loop* occurs immediately …
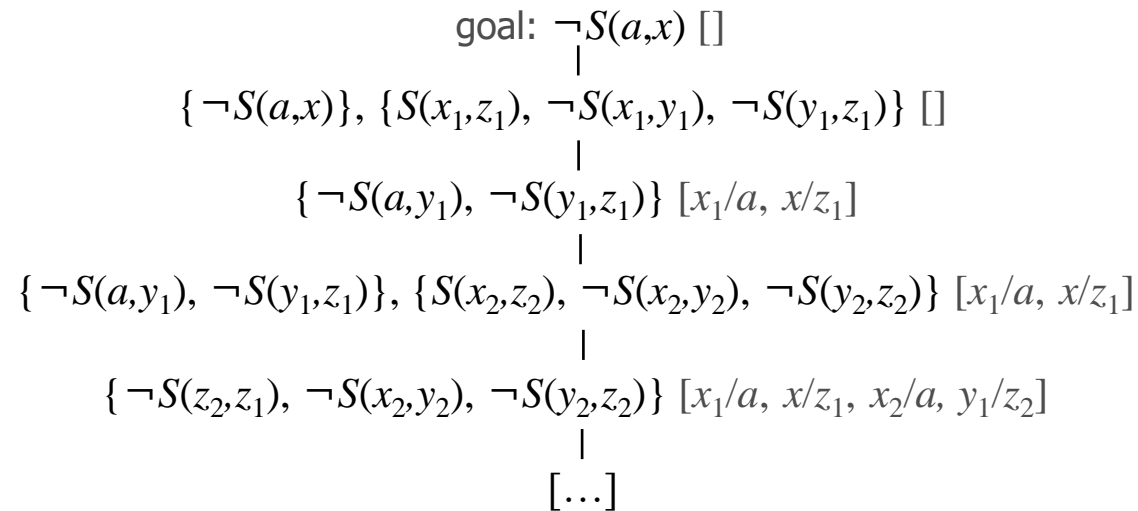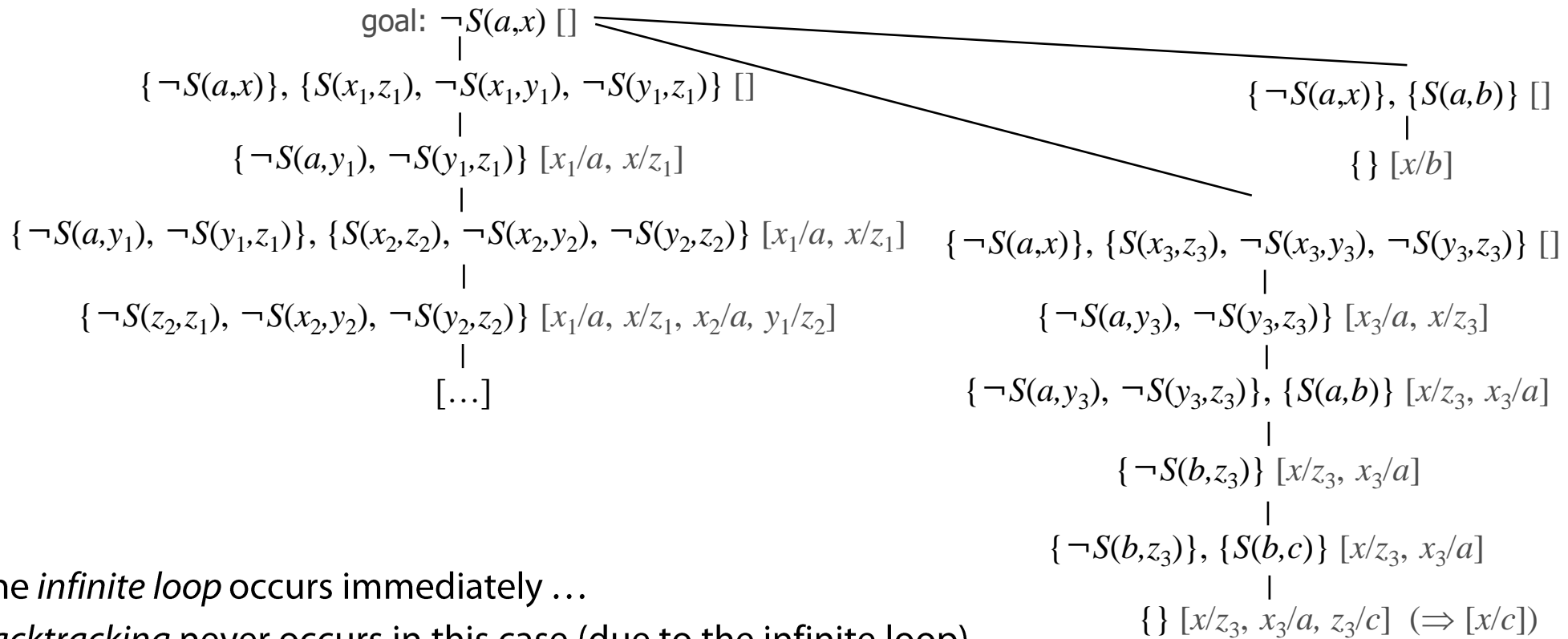
Infinite SLD Trees (*fairness of SLD*)

- A second example:

$\Pi \equiv \{\{S(x,z), \neg S(x,y), \neg S(y,z)\},\{S(a,b)\}, \{S(b,c)\}\}$

$\neg\phi \equiv \{\neg S(a,x)\}$

Notice the change in clause ordering…..

goal: $\neg S(a,x)$ []

$\{\neg S(a,x)\}, \{S(x_1,z_1), \neg S(x_1,y_1), \neg S(y_1,z_1)\}$ []

$\{\neg S(a,x)\}, \{S(a,b)\}$ []

$\{\neg S(a,y_1), \neg S(y_1,z_1)\}$ $[x_1/a,\ x/z_1]$

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

$\{\neg S(a,y_1), \neg S(y_1,z_1)\}, \{S(x_2,z_2), \neg S(x_2,y_2), \neg S(y_2,z_2)\}$ $[x_1/a,\ x/z_1]$

$\{\neg S(a,x)\}, \{S(x_3,z_3), \neg S(x_3,y_3), \neg S(y_3,z_3)\}$ []

$\{\neg S(z_2,z_1), \neg S(x_2,y_2), \neg S(y_2,z_2)\}$ $[x_1/a,\ x/z_1,\ x_2/a,\ y_1/z_2]$

$\{\neg S(a,y_3), \neg S(y_3,z_3)\}$ $[x_3/a,\ x/z_3]$

$[…]$

$\{\neg S(a,y_3), \neg S(y_3,z_3)\}, \{S(a,b)\}$ $[x/z_3,\ x_3/a]$

$\{\neg S(b,z_3)\}$ $[x/z_3,\ x_3/a]$

$\{\neg S(b,z_3)\}, \{S(b,c)\}$ $[x/z_3,\ x_3/a]$

The *infinite loop* occurs immediately …

Backtracking never occurs in this case (due to the infinite loop),
yet, if it occurred it would have produced the two correct results

$\{\}$ $[x/z_3,\ x_3/a,\ z_3/c]$ $(\Rightarrow [x/c])$

Infinite SLD Trees (*fairness of SLD*)

- Moral

  - In both previous examples the infinite loop (i.e. *divergence*) is unavoidable
  - Yet in the first one, the method first produces the right results and then diverges
  - So in the first case the result is *complete* (i.e. all entailed formulae are derived) while in the second case the method is not

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

  In the two previous examples, we used a *depth-first* exploration method of the SLD tree: which is <u>not</u> complete (in the above sense)

  A *breadth-first* exploration method is **fair** hence it is complete (in the above sense)

  In actual programming systems (e.g. Prolog) the depth-first is preferred for memory efficiency since the breadth-first method forces to keep (most of) the whole SLD tree in memory