

Artificial Intelligence

An Advanced Course About Foundations



SLD Resolution in First-Order Logic

Marco Piastra

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 $\exists x (Sister(sally, x) \wedge Sister(x, paul))$

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:

$\gamma_1, \gamma_2, \dots, \gamma_n$ (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 $std(\gamma_i)$ (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. apply σ to both ψ and $std(\gamma_i)$ then generate the resolvent)
- 5) Then, if the *resolvent* is the empty clause, terminate with success, 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 failure

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 least 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)

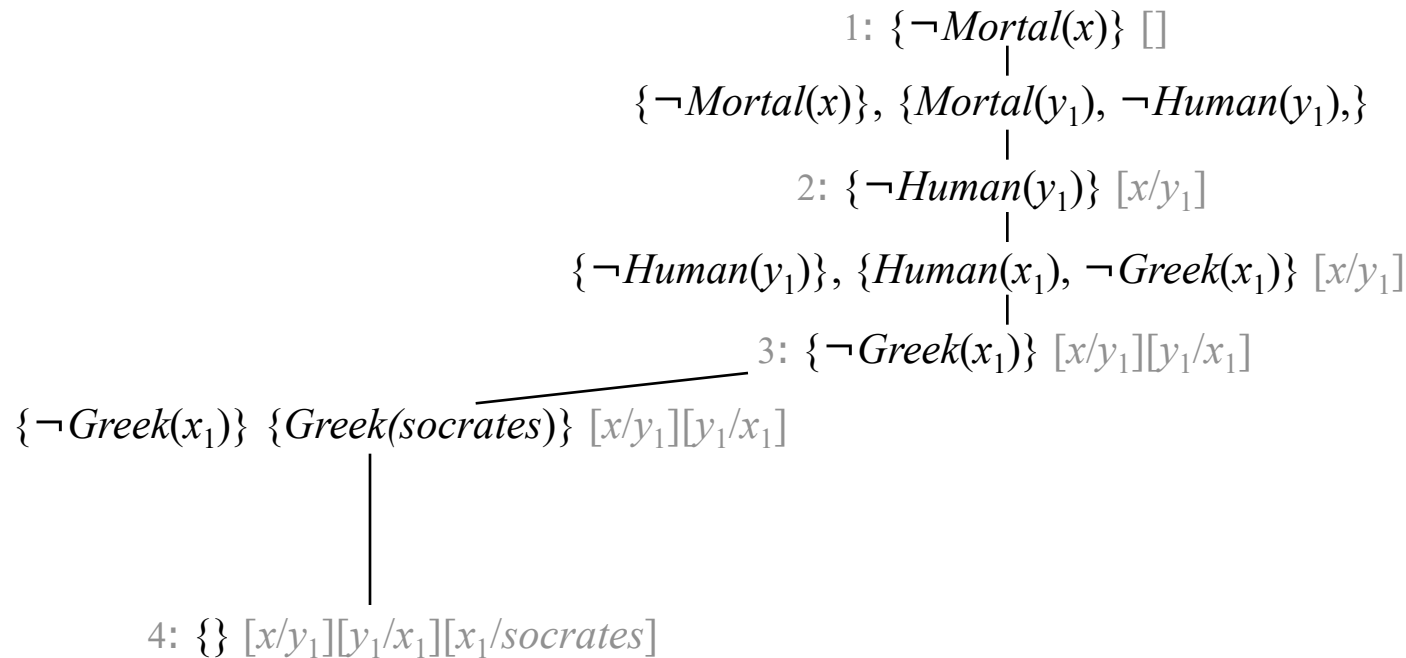
SLD Trees

■ Example (depth-first selection function):

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

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

"Is there anyone who is mortal?"



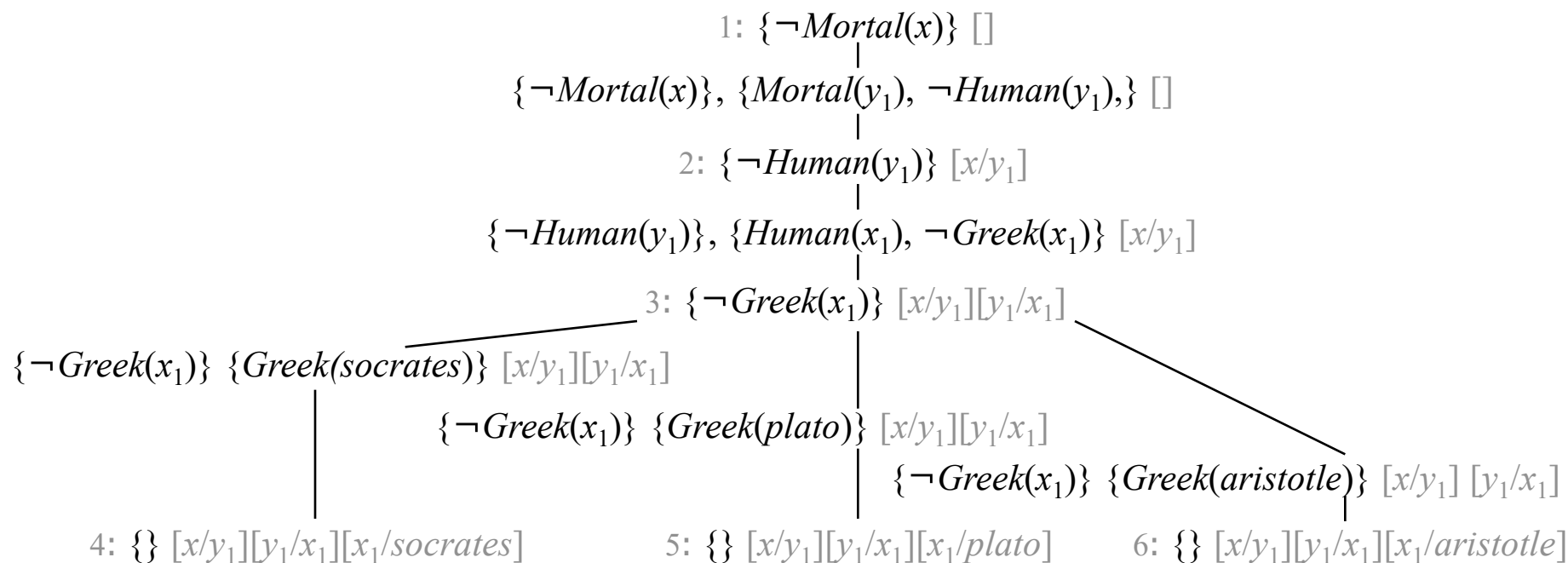
SLD Trees

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

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

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

"Is there anyone who is mortal?"



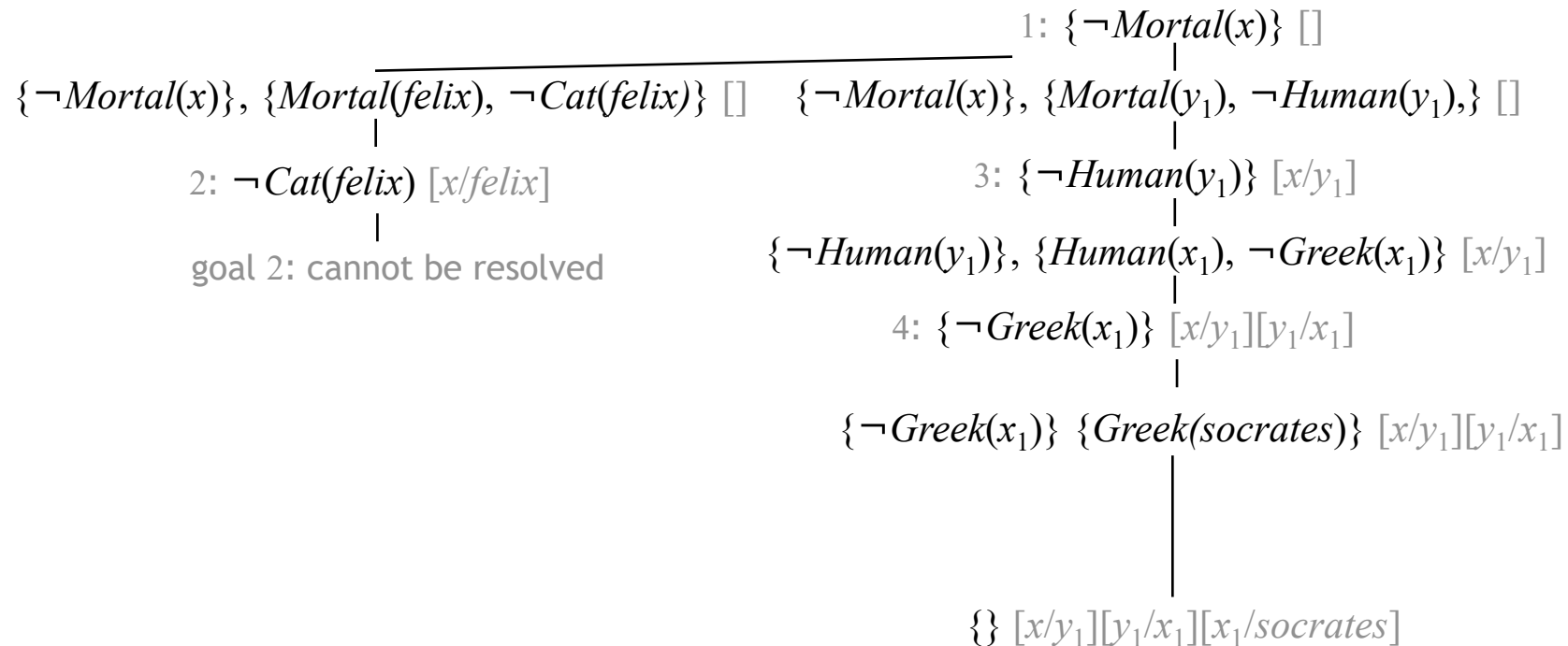
SLD Trees

- Another example (depth-first selection function):

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

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

"Is there anyone who is mortal?"



Infinite SLD Trees (*fairness of SLD*)

- An example:

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

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

$$\begin{array}{c} \text{goal: } \neg S(a, x) \\ | \\ \{\neg S(a, x)\}, \{S(a, b)\} \\ | \\ \{\} [x/b] \end{array}$$

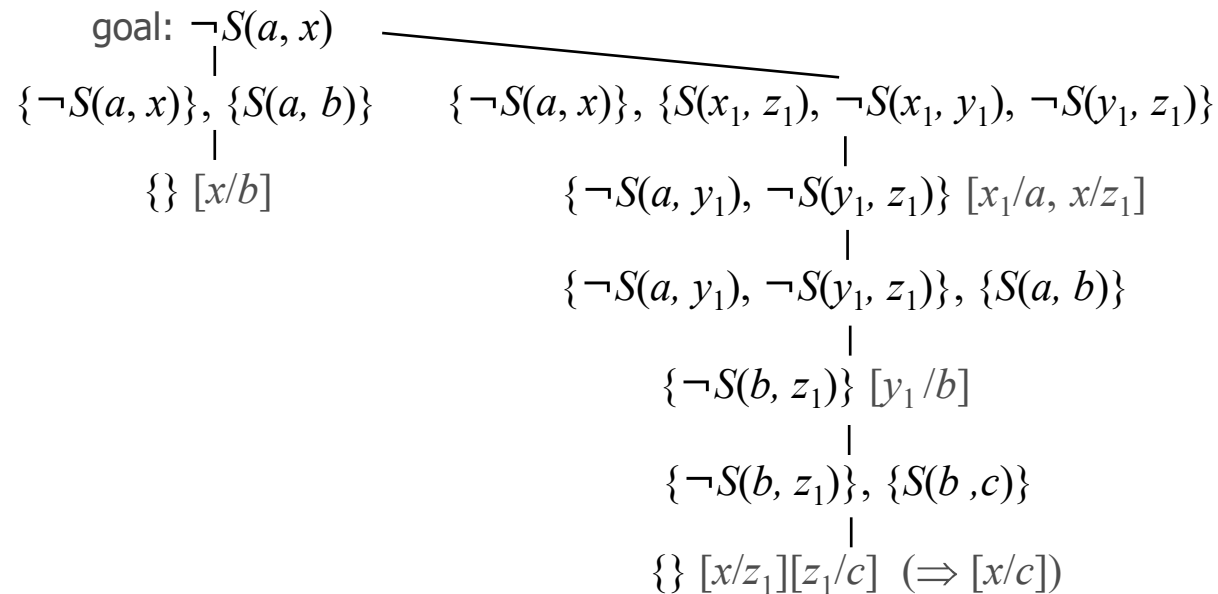
Easy...

Infinite SLD Trees (*fairness of SLD*)

■ An example:

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

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



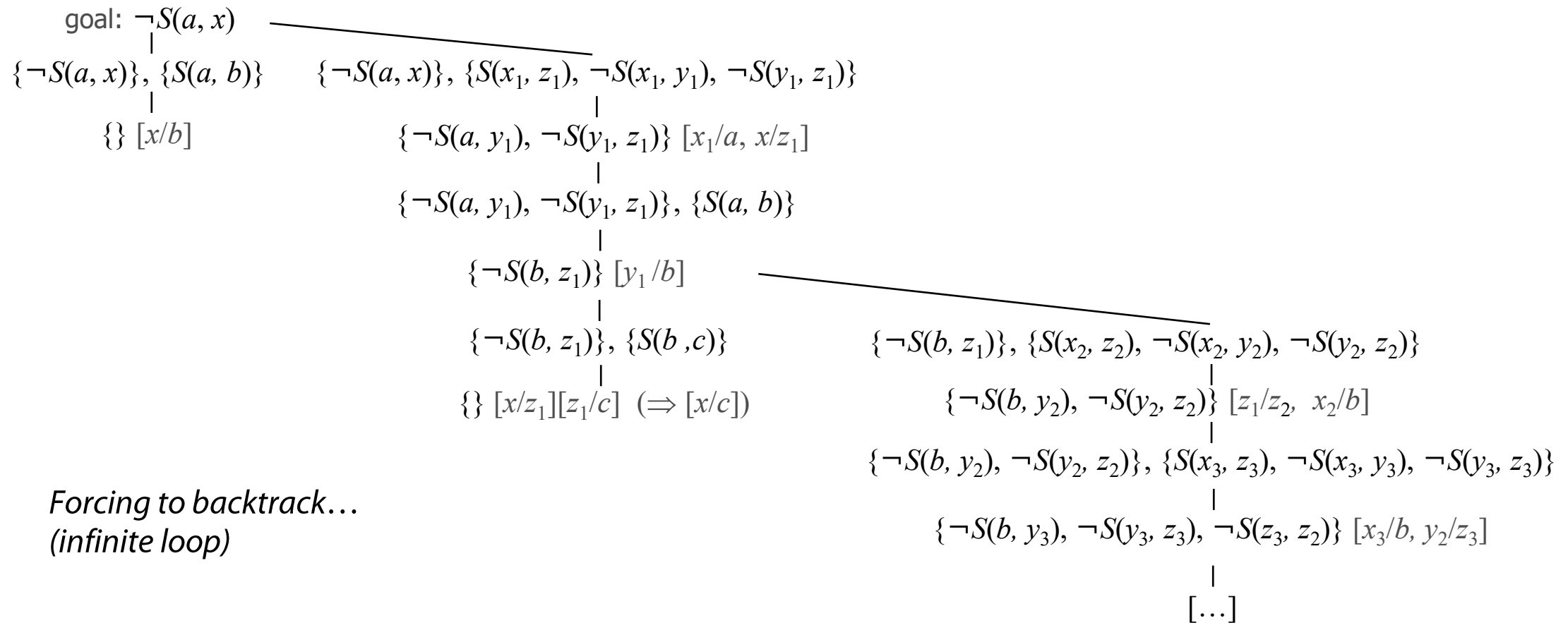
Forcing to backtrack...
(easy again)

Infinite SLD Trees (*fairness of SLD*)

■ An example:

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

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



Forcing to backtrack...
(infinite loop)

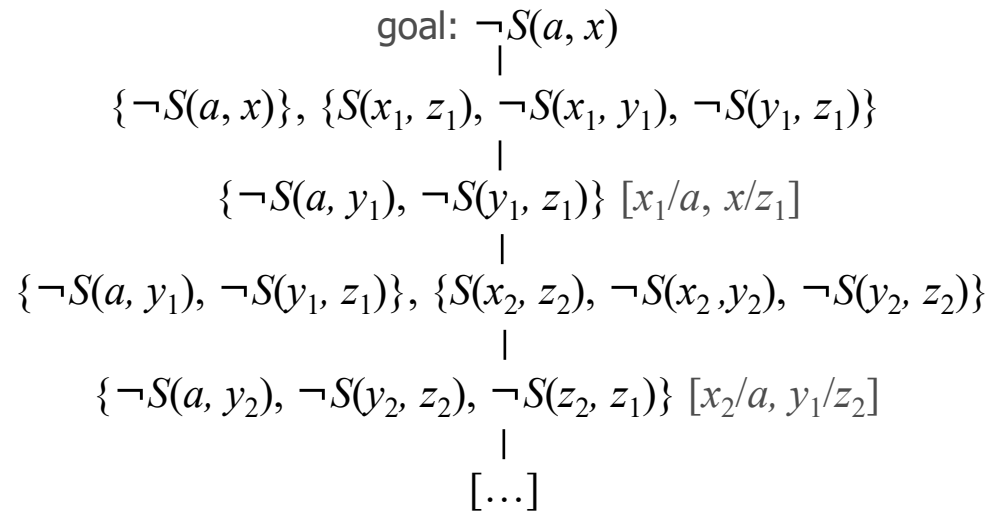
Infinite SLD Trees (*fairness of SLD*)

- 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....



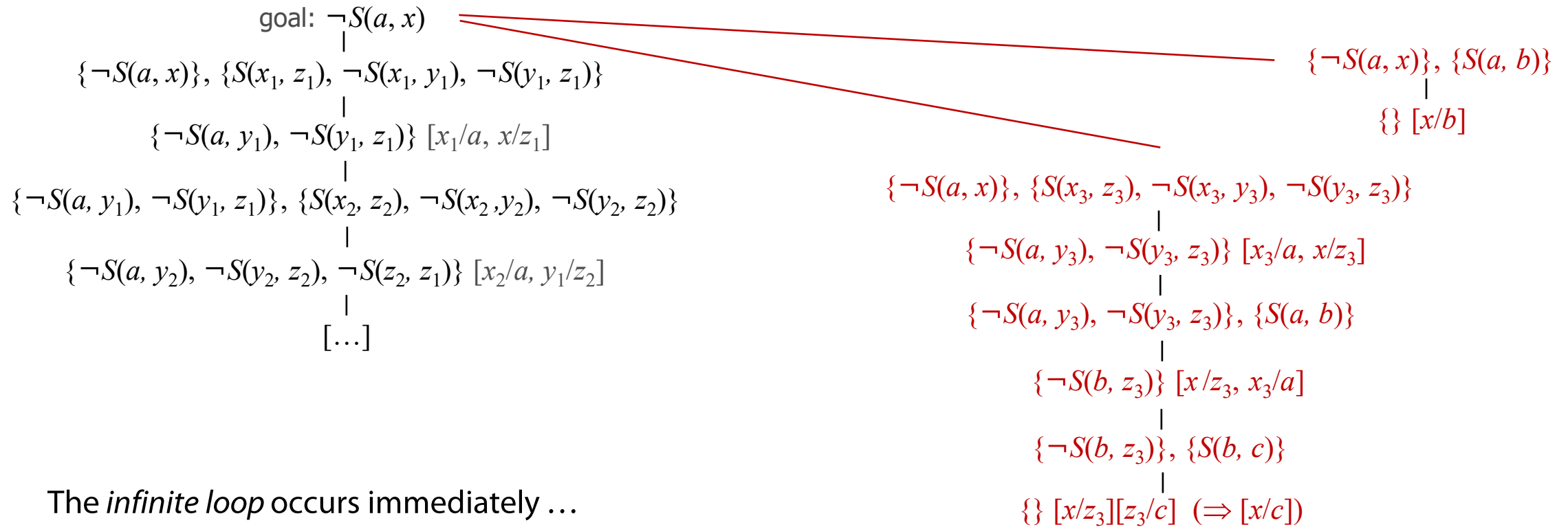
The *infinite loop* occurs immediately ...

Infinite SLD Trees (*fairness of SLD*)

■ A second example:

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

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



The *infinite loop* occurs immediately ...

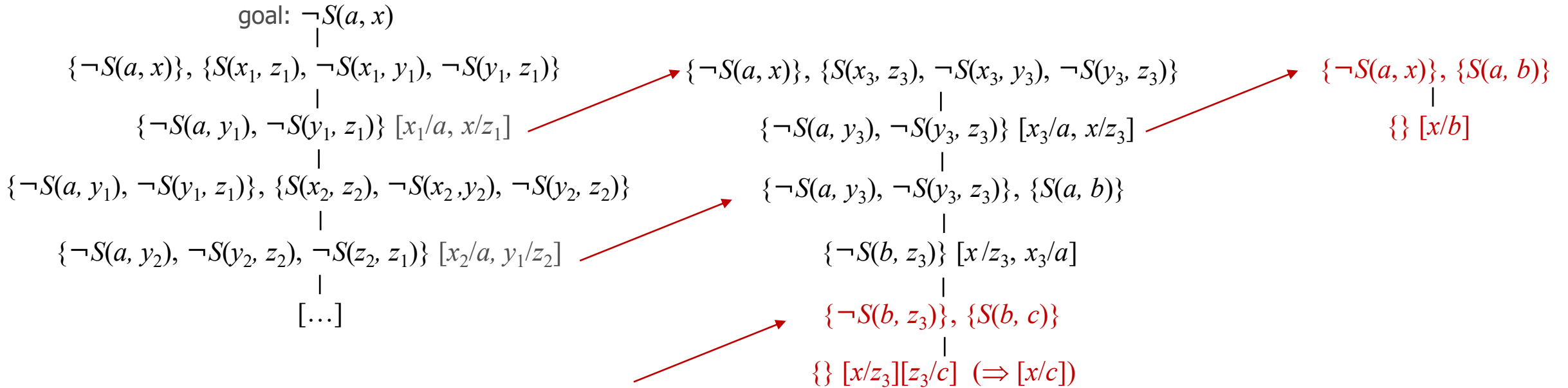
Backtracking never occurs in this case (due to the infinite loop), yet, if it occurred it would have produced the two correct results

Infinite SLD Trees (*fairness of SLD*)

- A third example: *breadth-first*

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

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



The *infinite loop* is still unavoidable

Yet, *all solutions will be found in finite time (and entailment assessed)*

Infinite SLD Trees (*fairness of SLD*)

■ Moral

- In all 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, any possible resolution will be performed, eventually.

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

Whereas, 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

*The discreet charme of functions

- Representing data structures: *lists of items* $[a, b, c, \dots]$

Symbols in Σ

cons/2

it's a function that associates items (e.g. a) to a list (e.g. $[b, c]$)

$\text{cons}(a, \text{cons}(b, \text{cons}(c, \text{nil})))$ 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)

$\forall x \text{ Append}(\text{nil}, x, x)$

$\forall x \forall y \forall z (\text{Append}(x, y, z) \rightarrow \forall s \text{ Append}(\text{cons}(s, x), y, \text{cons}(s, z)))$

Examples of entailment

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

$\models \text{Append}(\text{cons}(a, \text{nil}), \text{cons}(b, \text{cons}(c, \text{nil})), \text{cons}(a, \text{cons}(b, \text{cons}(c, \text{nil}))))$

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

$\models \text{Append}(\text{cons}(a, \text{nil}), \text{cons}(b, \text{nil}), \text{cons}(a, \text{cons}(b, \text{nil})))$

$\models \text{Append}(\text{nil}, \text{cons}(a, \text{cons}(b, \text{nil})), \text{cons}(a, \text{cons}(b, \text{nil})))$

$\models \text{Append}(\text{cons}(a, \text{cons}(b, \text{nil})), \text{nil}, \text{cons}(a, \text{cons}(b, \text{nil})))$

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. $[b, c]$)

$\text{cons}(a, \text{cons}(b, \text{cons}(c, \text{nil})))$ 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):

$$\begin{aligned} [] &\Leftrightarrow \text{nil} \\ [a] &\Leftrightarrow \text{cons}(a, \text{nil}) \\ [a, b] &\Leftrightarrow \text{cons}(a, \text{cons}(b, \text{nil})) \\ [a, [b, c]] &\Leftrightarrow \text{cons}(a, [b, c]) \end{aligned}$$

Axioms (AL)

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

The world of lists

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

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

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

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

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

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

The pair of **literals**

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

... contains the same predicate *Append*/3 but the arguments are **different**

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

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

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)))).
```