

Logic Programming

Cezary Kaliszyk **Georg Moser**

Institute of Computer Science @ UIBK

Winter 2015



Happy New Year!

Summary of Last Lecture

Definitions

- **SLD-derivation** of logic program P and goal clause G consists of
 - 1 maximal sequence G_0, G_1, G_2, \dots of goal clauses
 - 2 sequence C_0, C_1, C_2, \dots of variants of rules in P
 - 3 sequence $\theta_0, \theta_1, \theta_2, \dots$ of substitutions

such that

- $G_0 = G$
- C_i has no variables in common with G, C_0, \dots, C_{i-1}
- **SLD refutation** is finite SLD derivation ending in \square
- **computed answer substitution** of SLD refutation of P and G with substitutions $\theta_0, \theta_1, \dots, \theta_m$ is restriction of $\theta_0\theta_1 \cdots \theta_m$ to variables in G

Summary of Last Lecture

Definitions

- **SLD-derivation** of logic program P and goal clause G consists of
 - 1 maximal sequence G_0, G_1, G_2, \dots of goal clauses
 - 2 sequence C_0, C_1, C_2, \dots of variants of rules in P
 - 3 sequence $\theta_0, \theta_1, \theta_2, \dots$ of substitutions

such that

- $G_0 = G$
- C_i has no variables in common with G, C_0, \dots, C_{i-1}
- **SLD refutation** is finite SLD derivation ending in \square
- **computed answer substitution** of SLD refutation of P and G with substitutions $\theta_0, \theta_1, \dots, \theta_m$ is restriction of $\theta_0\theta_1 \cdots \theta_m$ to variables in G

Summary of Last Lecture

Definitions

- **SLD-derivation** of logic program P and goal clause G consists of
 - 1 maximal sequence G_0, G_1, G_2, \dots of goal clauses
 - 2 sequence C_0, C_1, C_2, \dots of variants of rules in P
 - 3 sequence $\theta_0, \theta_1, \theta_2, \dots$ of substitutions

such that

- $G_0 = G$
- C_i has no variables in common with G, C_0, \dots, C_{i-1}
- **SLD refutation** is finite SLD derivation ending in \square
- **computed answer substitution** of SLD refutation of P and G with substitutions $\theta_0, \theta_1, \dots, \theta_m$ is restriction of $\theta_0\theta_1 \cdots \theta_m$ to variables in G

Summary of Last Lecture

Definitions

- **SLD-derivation** of logic program P and goal clause G consists of
 - 1 maximal sequence G_0, G_1, G_2, \dots of goal clauses
 - 2 sequence C_0, C_1, C_2, \dots of variants of rules in P
 - 3 sequence $\theta_0, \theta_1, \theta_2, \dots$ of substitutions

such that

- $G_0 = G$
- C_i has no variables in common with G, C_0, \dots, C_{i-1}
- **SLD refutation** is finite SLD derivation ending in \square
- **computed answer substitution** of SLD refutation of P and G with substitutions $\theta_0, \theta_1, \dots, \theta_m$ is restriction of $\theta_0\theta_1 \cdots \theta_m$ to variables in G

Summary of Last Lecture

Definitions

- **SLD-derivation** of logic program P and goal clause G consists of
 - 1 maximal sequence G_0, G_1, G_2, \dots of goal clauses
 - 2 sequence C_0, C_1, C_2, \dots of variants of rules in P
 - 3 sequence $\theta_0, \theta_1, \theta_2, \dots$ of substitutions

such that

- $G_0 = G$
- C_i has no variables in common with G, C_0, \dots, C_{i-1}
- **SLD refutation** is finite SLD derivation ending in \square
- **computed answer substitution** of SLD refutation of P and G with substitutions $\theta_0, \theta_1, \dots, \theta_m$ is restriction of $\theta_0\theta_1 \cdots \theta_m$ to variables in G

Summary of Last Lecture

Definitions

- **SLD-derivation** of logic program P and goal clause G consists of
 - 1 maximal sequence G_0, G_1, G_2, \dots of goal clauses
 - 2 sequence C_0, C_1, C_2, \dots of variants of rules in P
 - 3 sequence $\theta_0, \theta_1, \theta_2, \dots$ of substitutions

such that

- $G_0 = G$
- G_{i+1} is resolvent of G_i and C_i with mgu θ_i
- C_i has no variables in common with G, C_0, \dots, C_{i-1}
- **SLD refutation** is finite SLD derivation ending in \square
- **computed answer substitution** of SLD refutation of P and G with substitutions $\theta_0, \theta_1, \dots, \theta_m$ is restriction of $\theta_0\theta_1 \cdots \theta_m$ to variables in G

Definition

a **search tree** (aka **SLD tree**) of a goal G is a tree T such that

- the root of T is labelled with G
- the nodes of T are labelled with conjunctions of goals, where one goal is selected (wrt a selection function)
- \exists edge from node N for each clause, whose head unifies with the selected goal
- edges are labelled with (partial) answer substitutions
- leaves are **success nodes**, if the empty goal (denoted by \square) has been reached or **failure nodes** otherwise

Definition

a program P is called

- **correct** with respect to the intended meaning M , if the meaning of P is a subset of M
- **complete** if the intended meaning M is a subset of the meaning of P

Outline of the Lecture

Monotone Logic Programs

introduction, basic constructs, unification, database and recursive programming, termination

Incomplete Data Structures and Constraints

incomplete data structures, definite clause grammars, constraint logic programming, answer set programming

Full Prolog

semantics, correctness proofs, meta-logical predicates, cuts, complexity, efficient programs

Outline of the Lecture

Monotone Logic Programs

introduction, basic constructs, unification, database and recursive programming, termination

Incomplete Data Structures and Constraints

incomplete data structures, definite clause grammars, constraint logic programming, answer set programming

Full Prolog

semantics, **correctness proofs**, meta-logical predicates, cuts, complexity, efficient programs

Example

```
natural_number(0).  
natural_number(s(X)) :- natural_number(X).
```

Example

```
natural_number(0).  
natural_number(s(X)) :- natural_number(X).
```

Lemma

the program is complete wrt the set of facts

$$M := \{\text{natural_number}(s^i(0)) \mid i \geq 0\}$$

Example

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).
```

Lemma

the program is complete wrt the set of facts

$$M := \{\text{natural_number}(s^i(0)) \mid i \geq 0\}$$

Proof of Completeness.

1 let N be a natural number

Example

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).
```

Lemma

the program is complete wrt the set of facts

$$M := \{\text{natural_number}(s^i(0)) \mid i \geq 0\}$$

Proof of Completeness.

- 1 let N be a natural number
- 2 we show that $\text{natural_number}(s^N(0))$ is deducible by given a explicit SLD tree

Example

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).
```

Lemma

the program is complete wrt the set of facts

$$M := \{\text{natural_number}(s^i(0)) \mid i \geq 0\}$$

Proof of Completeness.

- 1 let N be a natural number
- 2 we show that $\text{natural_number}(s^N(0))$ is deducible by given a explicit SLD tree
- 3 case distinction on $N = 0$ and $N > 0$

Lemma

the program is correct wrt the set of facts

$$M := \{\text{natural_number}(s^i(0)) \mid i \geq 0\}$$

Lemma

the program is correct wrt the set of facts

$$M := \{\text{natural_number}(s^i(0)) \mid i \geq 0\}$$

Proof of Correctness.

- 1 suppose $\text{natural_number}(s^m(0))$ is deducible in n deductions

Lemma

the program is correct wrt the set of facts

$$M := \{\text{natural_number}(s^i(0)) \mid i \geq 0\}$$

Proof of Correctness.

- 1 suppose $\text{natural_number}(s^m(0))$ is deducible in n deductions
- 2 we use induction on n

Lemma

the program is correct wrt the set of facts

$$M := \{\text{natural_number}(s^i(0)) \mid i \geq 0\}$$

Proof of Correctness.

- 1 suppose $\text{natural_number}(s^m(0))$ is deducible in n deductions
- 2 we use induction on n
- 3 $n = 1$: then $\text{natural_number}(s^m(0))$ implies $m = 0$

Lemma

the program is correct wrt the set of facts

$$M := \{\text{natural_number}(s^i(0)) \mid i \geq 0\}$$

Proof of Correctness.

- 1 suppose $\text{natural_number}(s^m(0))$ is deducible in n deductions
- 2 we use induction on n
- 3 $n = 1$: then $\text{natural_number}(s^m(0))$ implies $m = 0$
- 4 $n > 0$: the goal must be of form $\text{natural_number}(s(t))$

Lemma

the program is correct wrt the set of facts

$$M := \{\text{natural_number}(s^i(0)) \mid i \geq 0\}$$

Proof of Correctness.

- 1 suppose `natural_number(sm(0))` is deducible in n deductions
- 2 we use induction on n
- 3 $n = 1$: then `natural_number(sm(0))` implies $m = 0$
- 4 $n > 0$: the goal must be of form `natural_number(s(t))`
- 5 thus `natural_number(t)` is deducible with $n - 1$ deductions

Lemma

the program is correct wrt the set of facts

$$M := \{\text{natural_number}(s^i(0)) \mid i \geq 0\}$$

Proof of Correctness.

- 1 suppose $\text{natural_number}(s^m(0))$ is deducible in n deductions
- 2 we use induction on n
- 3 $n = 1$: then $\text{natural_number}(s^m(0))$ implies $m = 0$
- 4 $n > 0$: the goal must be of form $\text{natural_number}(s(t))$
- 5 thus $\text{natural_number}(t)$ is deducible with $n - 1$ deductions
- 6 $t = s^{m'}(0)$ for some $m' \in \mathbb{N}$

Lemma

the program is correct wrt the set of facts

$$M := \{\text{natural_number}(s^i(0)) \mid i \geq 0\}$$

Proof of Correctness.

- 1 suppose $\text{natural_number}(s^m(0))$ is deducible in n deductions
- 2 we use induction on n
- 3 $n = 1$: then $\text{natural_number}(s^m(0))$ implies $m = 0$
- 4 $n > 0$: the goal must be of form $\text{natural_number}(s(t))$
- 5 thus $\text{natural_number}(t)$ is deducible with $n - 1$ deductions
- 6 $t = s^{m'}(0)$ for some $m' \in \mathbb{N}$
- 7 $\text{natural_number}(s^{m'+1}(0)) \in M$ and $m = m' + 1$

Example

is the program is complete wrt the following set?

$$M := \{\text{natural_number}(s^i(0)) \mid 0 \leq i \leq K\}$$

Example

is the program is correct wrt the following set?

$$M := \{\text{natural_number}(s^i(0)) \mid 0 \leq i \leq K\}$$

Example

is the program is correct wrt the following set?

$$M := \{\text{natural_number}(s^i(0)) \mid 0 \leq i \leq K\}$$

Attempted Proof

- 1 suppose `natural_number(sm(0))` is deducible in n deductions
- 2 we use induction on n

Example

is the program is correct wrt the following set?

$$M := \{\text{natural_number}(s^i(0)) \mid 0 \leq i \leq K\}$$

Attempted Proof

- 1 suppose `natural_number(sm(0))` is deducible in n deductions
- 2 we use induction on n
- 3 $n = 0$: then `natural_number(sm(0))` implies $m = 0$

Example

is the program is correct wrt the following set?

$$M := \{\text{natural_number}(s^i(0)) \mid 0 \leq i \leq K\}$$

Attempted Proof

- 1 suppose `natural_number($s^m(0)$)` is deducible in n deductions
- 2 we use induction on n
- 3 $n = 0$: then `natural_number($s^m(0)$)` implies $m = 0$
- 4 $n > 0$: the goal must be of form `natural_number($s(t)$)`

Example

is the program is correct wrt the following set?

$$M := \{\text{natural_number}(s^i(0)) \mid 0 \leq i \leq K\}$$

Attempted Proof

- 1 suppose `natural_number(sm(0))` is deducible in n deductions
- 2 we use induction on n
- 3 $n = 0$: then `natural_number(sm(0))` implies $m = 0$
- 4 $n > 0$: the goal must be of form `natural_number(s(t))`
- 5 thus `natural_number(t)` is deducible with $n - 1$ deductions

Example

is the program is correct wrt the following set?

$$M := \{\text{natural_number}(s^i(0)) \mid 0 \leq i \leq K\}$$

Attempted Proof

- 1 suppose `natural_number(sm(0))` is deducible in n deductions
- 2 we use induction on n
- 3 $n = 0$: then `natural_number(sm(0))` implies $m = 0$
- 4 $n > 0$: the goal must be of form `natural_number(s(t))`
- 5 thus `natural_number(t)` is deducible with $n - 1$ deductions
- 6 $t = s^{m'}(0)$ for some $0 \leq m' \leq K$ and $m = m' + 1$

Example

is the program is correct wrt the following set?

$$M := \{\text{natural_number}(s^i(0)) \mid 0 \leq i \leq K\}$$

Attempted Proof

- 1 suppose `natural_number(sm(0))` is deducible in n deductions
- 2 we use induction on n
- 3 $n = 0$: then `natural_number(sm(0))` implies $m = 0$
- 4 $n > 0$: the goal must be of form `natural_number(s(t))`
- 5 thus `natural_number(t)` is deducible with $n - 1$ deductions
- 6 $t = s^{m'}(0)$ for some $0 \leq m' \leq K$ and $m = m' + 1$
- 7 `natural_number(sm(0))` $\in M$ iff $m \leq K$

Example

is the program is correct wrt the following set?

$$M := \{\text{natural_number}(s^i(0)) \mid 0 \leq i \leq K\}$$

Attempted Proof

- 1 suppose $\text{natural_number}(s^m(0))$ is deducible in n deductions
- 2 we use induction on n
- 3 $n = 0$: then $\text{natural_number}(s^m(0))$ implies $m = 0$
- 4 $n > 0$: the goal must be of form $\text{natural_number}(s(t))$
- 5 thus $\text{natural_number}(t)$ is deducible with $n - 1$ deductions
- 6 $t = s^{m'}(0)$ for some $0 \leq m' \leq K$ and $m = m' + 1$
- 7 $\text{natural_number}(s^m(0)) \in M$ iff $m \leq K$
- 8 what happens for $m > K$?

Example

```
natural_number(0).
```

```
natural_number(s(X)) :- natural_number(X).
```

```
plus(0,X,X) :- natural_number(X).
```

```
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
```

Example

```
natural_number(0).  
natural_number(s(X)) :- natural_number(X).  
  
plus(0,X,X) :- natural_number(X).  
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
```

Lemma

the program is correct and complete wrt to the definition of addition

Example

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).

plus(0,X,X) :- natural_number(X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
```

Lemma

the program is correct and complete wrt to the definition of addition

Proof Sketch.

- 1 completeness: suppose $X + Y = Z$; then we give an SLD tree of $\text{plus}(X, Y, Z)$

Example

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).

plus(0,X,X) :- natural_number(X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
```

Lemma

the program is correct and complete wrt to the definition of addition

Proof Sketch.

- 1 completeness: suppose $X + Y = Z$; then we give an SLD tree of $\text{plus}(X, Y, Z)$
- 2 correctness: suppose $\text{plus}(X, Y, Z)$ is deducible; then we prove by induction on the length of this deduction that $X + Y = Z$



Rule Order

Observation

The rule order determines the order in which solutions are found

Rule Order

Observation

The rule order determines the order in which solutions are found

Example

```
parent(terach, abraham).      parent(abraham, isaac).
parent(isaac, jakob).        parent(jakob, benjamin).

ancestor(X, Y) :- parent(X, Y).
ancestor(X, Z) :- parent(X, Y), ancestor(Y, Z).
```

Example

```
append([X|Xs], Ys, [X|Zs]) :-      append([], Ys, Ys).
    append(Xs, Ys, Zs).             append([X|Xs], Ys, [X|Zs]) :-
append([], Ys, Ys).                 append(Xs, Ys, Zs).
```

Goal Order

Observation

Goal order determines the SLD tree

Goal Order

Observation

Goal order determines the SLD tree

Example

```
grandparent(X,Z) :- parent(X,Y), parent(Y,Z).  
grandparent2(X,Z) :- parent(Y,Z), parent(X,Y).
```

Goal Order

Observation

Goal order determines the SLD tree

Example

```
grandparent(X,Z) :- parent(X,Y), parent(Y,Z).  
grandparent2(X,Z) :- parent(Y,Z), parent(X,Y).
```

Example

```
reverse([X|Xs],Zs) :- reverse(Xs,Ys), append(Ys,[X],Zs).  
reverse([],[]).
```

Goal Order

Observation

Goal order determines the SLD tree

Example

```
grandparent(X,Z) :- parent(X,Y), parent(Y,Z).  
grandparent2(X,Z) :- parent(Y,Z), parent(X,Y).
```

Example

```
reverse([X|Xs],Zs) :- reverse(Xs,Ys), append(Ys,[X],Zs).  
reverse([],[]).
```

Example

```
sublist(Xs,AsXsBs) :-  
    append(AsXs,Bs,AsXsBs), append(As,Xs,AsXs).
```

Goal Order

Observation

Goal order determines the SLD tree

Example

```
grandparent(X,Z) :- parent(X,Y), parent(Y,Z).  
grandparent2(X,Z) :- parent(Y,Z), parent(X,Y).
```

Example

```
reverse([X|Xs],Zs) :- reverse(Xs,Ys), append(Ys,[X],Zs).  
reverse([],[]).
```

Example

```
sublist(Xs,AsXsBs) :-  
    append(As,Xs,AsXs), append(AsXs,Bs,AsXsBs).
```

SWI-Prolog

```
[zid-gpl.uibk.ac.at] swipl
```

```
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 5.7.11)
```

```
Copyright (c) 1990-2009 University of Amsterdam.
```

```
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,  
and you are welcome to redistribute it under certain conditions.
```

```
Please visit http://www.swi-prolog.org for details.
```

```
For help, use ?- help(Topic). or ?- apropos(Word).
```

```
?-
```

Emacs Mode

Bruda's Prolog Mode

- 1 goto http://bruda.ca/emacs/prolog_mode_for_emacs
- 2 download prolog.el, compile and put into sub-directory site-lisp
- 3 put the following into `.emacs`:

```
(autoload 'run-prolog "prolog"
          "Start a Prolog sub-process." t)
(autoload 'prolog-mode "prolog"
          "Major mode for editing Prolog programs." t)
(setq prolog-system 'swi)
(setq auto-mode-alist
      (cons (cons "\\\\.pl" 'prolog-mode) auto-mode-alist))
```

Redundant Solutions

Example

`minimum(N1,N2,N1) : - N1 ≤ N2.`

`minimum(N1,N2,N2) : - N2 ≤ N1.`

`: - minium(2,2,M)`

Redundant Solutions

Example

`minimum(N1,N2,N1)` : - $N_1 \leq N_2$.

`minimum(N1,N2,N2)` : - $N_2 \leq N_1$.

: - `minium(2,2,M)`

Example

`minimum(N1,N2,N1)` : - $N_1 \leq N_2$.

`minimum(N1,N2,N2)` : - $N_2 < N_1$.

Redundant Solutions

Example

`minimum(N1,N2,N1)` : - $N_1 \leq N_2$.

`minimum(N1,N2,N2)` : - $N_2 \leq N_1$.

: - `minium(2,2,M)`

Example

`minimum(N1,N2,N1)` : - $N_1 \leq N_2$.

`minimum(N1,N2,N2)` : - $N_2 < N_1$.

Observation

similar care is necessary with the definition of **partition**, etc.

Redundant Solutions (part II)

Example

```
member(X, [X|Xs]).  
member(X, [Y|Xs]) : - member(X, Xs).
```

Redundant Solutions (part II)

Example

```
member(X, [X|Xs]).  
member(X, [Y|Xs]) : - member(X, Xs).
```

```
?- member(X, [a,b,a]).
```

```
X ↦ a
```

Redundant Solutions (part II)

Example

```
member(X, [X|Xs]).  
member(X, [Y|Xs]) : - member(X, Xs).
```

```
?- member(X, [a,b,a]).
```

```
X ↦ a ;
```

```
X ↦ b
```

Redundant Solutions (part II)

Example

```
member(X, [X|Xs]).  
member(X, [Y|Xs]) : - member(X, Xs).
```

```
?- member(X, [a,b,a]).
```

```
X ↦ a ;
```

```
X ↦ b ;
```

```
X ↦ a
```

Redundant Solutions (part II)

Example

```
member(X, [X|Xs]).  
member(X, [Y|Xs]) : - member(X, Xs).
```

```
?- member(X, [a,b,a]).
```

```
X ↦ a ;
```

```
X ↦ b ;
```

```
X ↦ a ;
```

```
false
```

Redundant Solutions (part II)

Example

```
member(X, [X|Xs]).  
member(X, [Y|Xs]) : - member(X, Xs).
```

```
?- member(X, [a,b,a]).
```

```
X ↦ a ;
```

```
X ↦ b ;
```

```
X ↦ a ;
```

```
false
```

Example

```
member_check(X, [X|Xs]).  
member_check(X, [Y|Ys]) : - X ≠ Y, member_check(X, Ys).
```

Fact

some care is necessary in pruning the search tree, as this may change the meaning of a program

Fact

some care is necessary in pruning the search tree, as this may change the meaning of a program

Example

```
select(X, [X|Xs], Xs).  
select(X, [Y|Ys], [Y|Zs]) :- select(X, Ys, Zs).
```

Fact

some care is necessary in pruning the search tree, as this may change the meaning of a program

Example

```
select(X, [X|Xs], Xs).
select(X, [Y|Ys], [Y|Zs]) :- select(X, Ys, Zs).
```

Example

```
select_first(X, [X|Xs], Xs).
select_first(X, [Y|Ys], [Y|Zs]) :- X  $\neq$  Y, select_first(X, Ys, Zs).
```

Fact

some care is necessary in pruning the search tree, as this may change the meaning of a program

Example

```
select(X, [X|Xs], Xs).
select(X, [Y|Ys], [Y|Zs]) :- select(X, Ys, Zs).
```

Example

```
select_first(X, [X|Xs], Xs).
select_first(X, [Y|Ys], [Y|Zs]) :- X ≠ Y, select_first(X, Ys, Zs).
```

Observation

`select(a, [a,b,a,c], [a,b,c])` is in the meaning of the 1st program;
`select_first(a, [a,b,a,c], [a,b,c])` is **not** in the meaning of the 2nd

Example

```
members([X|Xs],Ys) :- member(X,Ys), members(Xs,Ys).  
members([],Ys).
```

Example

```
members([X|Xs],Ys) :- member(X,Ys), members(Xs,Ys).  
members([],Ys).
```

Example

```
selects([X|Xs],Ys) :- select(X,Ys,Ys1), selects(Xs,Ys1).  
selects([],Ys).
```

Example

```
members([X|Xs],Ys) :- member(X,Ys), members(Xs,Ys).
members([],Ys).
```

Example

```
selects([X|Xs],Ys) :- select(X,Ys,Ys1), selects(Xs,Ys1).
selects([],Ys).
```

Observations

- 1 *members/2* ignores the multiplicity of elements
- 2 *members/2* terminates iff 1st argument is complete
- 3 the first restriction is lifted, the second altered with *selects/2*
- 4 *selects/2* terminates iff 2nd argument is complete

Example

```
%   no_doubles(Xs,Ys)  $\leftarrow$   
%       Ys is the list obtained by removing duplicate  
%       elements from the list Xs
```

Example

```
%    no_doubles(Xs,Ys) <—  
%      Ys is the list obtained by removing duplicate  
%      elements from the list Xs
```

Example

```
non_member(X,[Y|Ys]) :- X ≠ Y, non_member(X,Ys).  
non_member(X,[]).
```


Example

```
%   no_doubles(Xs,Ys)  $\leftarrow$ 
%       Ys is the list obtained by removing duplicate
%       elements from the list Xs
```

Example

```
non_member(X,[Y|Ys]) :- X  $\neq$  Y, non_member(X,Ys).
non_member(X,[]).

no_doubles([X|Xs],Ys) :-
    member(X,Xs), no_doubles(Xs,Ys).
no_doubles([X|Xs],[X|Ys]) :-
    non_member(X,Xs), no_doubles(Xs,Ys).
no_doubles([],[]).
```

Built-in Predicates for List Manipulation

- `append/3`
- `member/2`

Built-in Predicates for List Manipulation

- `append/3`
- `member/2`
- `last/2`

```
?- last([a,b,c,d],X).  
X = d
```

Built-in Predicates for List Manipulation

- `append/3`
- `member/2`
- `last/2`

```
?- last([a,b,c,d],X).  
X = d
```

```
?- last(X,a).
```

Built-in Predicates for List Manipulation

- `append/3`
- `member/2`
- `last/2`

```
?- last([a,b,c,d],X).  
X = d
```

```
?- last(X,a).  
X = [a]
```

Built-in Predicates for List Manipulation

- `append/3`
- `member/2`
- `last/2`

```
?- last([a,b,c,d],X).  
X = d
```

```
?- last(X,a).  
X = [a] ;  
X = [_G324,a]
```

Built-in Predicates for List Manipulation

- `append/3`
- `member/2`
- `last/2`

```
?- last([a,b,c,d],X).  
X = d
```

```
?- last(X,a).  
X = [a] ;  
X = [_G324,a] ;  
X = [_G324,_G327,a]
```

Built-in Predicates for List Manipulation

- append/3
- member/2
- last/2

```
?- last([a,b,c,d],X).  
X = d
```

```
?- last(X,a).  
X = [a] ;  
X = [_G324,a] ;  
X = [_G324,_G327,a]
```

- reverse/2

```
?- reverse([a,b,c,d],X).  
X = [d,c,b,a]
```


Built-in Predicates for List Manipulation

- append/3
- member/2
- last/2

```
?- last([a,b,c,d],X).
X = d
```

```
?- last(X,a).
X = [a] ;
X = [_G324,a] ;
X = [_G324,_G327,a]
```

- reverse/2

```
?- reverse([a,b,c,d],X).
X = [d,c,b,a]
```

- select/3

```
?- select(b,[a,b,c,d],X).
X = [a,c,d]
```

Built-in Predicates for List Manipulation

- append/3
- member/2
- last/2

```
?- last([a,b,c,d],X).
X = d
```

```
?- last(X,a).
X = [a] ;
X = [_G324,a] ;
X = [_G324,_G327,a]
```

- reverse/2

```
?- reverse([a,b,c,d],X).
X = [d,c,b,a]
```

- select/3

```
?- select(b,[a,b,c,d],X).
X = [a,c,d]
```

```
?- select(b,[a,b,c,b,d],X).
X = [a,c,b,d]
```

Built-in Predicates for List Manipulation

- append/3
- member/2
- last/2

```
?- last([a,b,c,d],X).
X = d
```

```
?- last(X,a).
X = [a] ;
X = [_G324,a] ;
X = [_G324,_G327,a]
```

- reverse/2

```
?- reverse([a,b,c,d],X).
X = [d,c,b,a]
```

- select/3

```
?- select(b,[a,b,c,d],X).
X = [a,c,d]
```

```
?- select(b,[a,b,c,b,d],X).
X = [a,c,b,d]
```

- length/2

```
?- length([a,b,c,d],X).
X = 4
```