

Logic Programming

Cezary Kaliszyk **Georg Moser**

Institute of Computer Science @ UIBK

Winter 2015



Organisation

Time and Place

Lecture	Monday, 10:15–11:45, HS 11	Georg Moser
Proseminar	Wednesday, 14:15–15:00, SR 1 (ICT)	Cezary Kaliszyk

Time and Place

Lecture	Monday, 10:15–11:45, HS 11	Georg Moser
Proseminar	Wednesday, 14:15–15:00, SR 1 (ICT)	Cezary Kaliszzyk

Schedule

week 1	October 5	week 8	December 7
week 2	October 12	week 9	December 14
week 3	October 19	week 10	January 11
week 4	November 9	week 11	January 18
week 5	November 16	week 12	January 25
week 6	November 23	first exam	February 1
week 7	November 30		

Time and Place

Lecture	Monday, 10:15–11:45, HS 11	Georg Moser
Proseminar	Wednesday, 14:15–15:00, SR 1 (ICT)	Cezary Kaliszzyk

Schedule

week 1	October 5	week 8	December 7
week 2	October 12	week 9	December 14
week 3	October 19	week 10	January 11
week 4	November 9	week 11	January 18
week 5	November 16	week 12	January 25
week 6	November 23	first exam	February 1
week 7	November 30		

Office Hours

- Thursday, 9:00–11:00, 3M09, Ifl Building (GM)
- Thursday, 14:15–15:45, 3M12, Ifl Building (CK)

Literature

- 1 Leon Sterling and Ehud Shapiro
The Art of Prolog



Literature

- 1 Leon Sterling and Ehud Shapiro
The Art of Prolog



Additional Reading

- Patrick Blackburn, Johan Bos and Kristina Striegnitz
Learn Prolog Now!
Texts in Computing 7, College Publications, 2006, ISBN
1-904987-17-6
- William F. Clocksin and Christopher S. Mellish
Programming in Prolog (fifth edition)
Springer Verlag, 2003, ISBN 978-3-540-00678-7

Literature

- 1 Leon Sterling and Ehud Shapiro
The Art of Prolog



Additional Reading

- Patrick Blackburn, Johan Bos and Kristina Striegnitz
Learn Prolog Now!
Texts in Computing 7, College Publications, 2006, ISBN 1-904987-17-6
- William F. Clocksin and Christopher S. Mellish
Programming in Prolog (fifth edition)
Springer Verlag, 2003, ISBN 978-3-540-00678-7
- Ulrich Neumerkel, Logikorientierte Programmierung

Declarative program development in Prolog with GUPU

- GUPU stands for “Gesprächsunterstützte Programmierübungsumgebung”
- GUPU has been maintained since 1991 by Ulrich Neumerkel et al.
- provides IDE for (a monotone subset) of Prolog (plus constraints)
- features (silent) termination checker and reference implementation
- online tutorial
- online homework submission and assessment
- runs on servers of the Vienna University of Technology

Declarative program development in Prolog with GUPU

- **GUPU** stands for “**G**esprächs**u**nterstützte **P**rogrammierübungs**u**mgebung”
- GUPU has been maintained since 1991 by Ulrich Neumerkel et al.
- provides IDE for (a monotone subset) of Prolog (plus constraints)
- features (silent) termination checker and reference implementation
- online tutorial
- online homework submission and assessment
- runs on servers of the Vienna University of Technology

Proseminar

- homework assignment and solution (only) in GUPU
- fully automated homework assessment
- “Abgabegespräch” at the end of the term if needed
- selected examples will be presented in the lecture or exercises

Evaluations

Exercise Schedule

Examples 1–2	October 9	Examples 38–46	November 20
Examples 3–7	October 16	Examples 47–57	November 27
Examples 8–19	October 23	Examples 58–66	December 4
Examples 20–29	November 6	Examples 67–71	December 11
Examples 30–37	November 13	late submission	January 29
		assessment	February 3

Evaluations

Exercise Schedule

Examples 1–2	October 9	Examples 38–46	November 20
Examples 3–7	October 16	Examples 47–57	November 27
Examples 8–19	October 23	Examples 58–66	December 4
Examples 20–29	November 6	Examples 67–71	December 11
Examples 30–37	November 13	late submission	January 29
		assessment	February 3

Deadlines and Grading

- deadline is always noon (CET)
- grading is automatic
- grading will be verified in the (voluntary) assessment

Evaluations (cont'd)

Remarks

- you can always pre-evaluate your homework
- the system not only verifies your solutions, but also weights the solution according to its timeliness and documentation
- every student gets unique extra questions, so cheating is detected

Evaluations (cont'd)

Remarks

- you can always pre-evaluate your homework
- the system not only verifies your solutions, but also weights the solution according to its **timeliness** and **documentation**
- every student gets unique extra questions, so cheating is detected

Evaluations (cont'd)

Remarks

- you can always pre-evaluate your homework
- the system not only verifies your solutions, but also weights the solution according to its timeliness and documentation
- every student gets unique extra questions, so cheating is detected

Exam

- first exam (closed-book) will take place on February 1
- your homework will be taken into account in the grading of the exam

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 nondeterministic programming, efficient programs, complexity

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 nondeterministic programming, efficient programs, complexity

Logic Programs

Attempt at a Definition

logic programming is a **declarative** programming paradigm, that is, the **specification** of a problem is made a first-class citizen; the idea can be summarised as follows:

program	set of judgements
computation	proof of a goal statement from the program

Attempt at a Definition

logic programming is a **declarative** programming paradigm, that is, the **specification** of a problem is made a first-class citizen; the idea can be summarised as follows:

program	set of judgements
computation	proof of a goal statement from the program

Advertisement

In its ultimate and purest form, logic programming suggests that even explicit instructions for operations not be given, but, rather, the knowledge about the problem and assumptions that are sufficient to solve it be stated explicitly, as logical axioms.

Attempt at a Definition

logic programming is a **declarative** programming paradigm, that is, the **specification** of a problem is made a first-class citizen; the idea can be summarised as follows:

program	set of judgements
computation	proof of a goal statement from the program

Advertisement

In its ultimate and purest form, logic programming suggests that even explicit instructions for operations not be given, but, rather, the knowledge about the problem and assumptions that are sufficient to solve it be stated explicitly, as logical axioms.

this is very abstract, over-simplified, and becomes false, when subject to scrutiny ... still logic programming is a pearl

Timeline

196? procedural view of (Horn) logic R. Kowalski

Timeline

- 196? procedural view of (Horn) logic R. Kowalski
- 1972 *Programmation en Logique* A. Colmerauer & P. Roussel

Timeline

- | | | |
|------|---------------------------------|----------------------------|
| 196? | procedural view of (Horn) logic | R. Kowalski |
| 1972 | <i>Programmation en Logique</i> | A. Colmerauer & P. Roussel |
| 1983 | Warren abstract machine | D. Warren |

Timeline

196?	procedural view of (Horn) logic	R. Kowalski
1972	<i>Programmation en Logique</i>	A. Colmerauer & P. Roussel
1983	Warren abstract machine	D. Warren
1987	constraint logic programming	Jaffar & Maher

Timeline

196?	procedural view of (Horn) logic	R. Kowalski
1972	<i>Programmation en Logique</i>	A. Colmerauer & P. Roussel
1983	Warren abstract machine	D. Warren
1987	constraint logic programming	Jaffar & Maher
1994	answer set programming	Dimopoulos, Nebel & Köhler

Timeline

196?	procedural view of (Horn) logic	R. Kowalski
1972	<i>Programmation en Logique</i>	A. Colmerauer & P. Roussel
1983	Warren abstract machine	D. Warren
1987	constraint logic programming	Jaffar & Maher
1994	answer set programming	Dimopoulos, Nebel & Köhler
	⋮	
2015	SWI-Prolog, Version 6.4.1	free

Timeline

196?	procedural view of (Horn) logic	R. Kowalski
1972	<i>Programmation en Logique</i>	A. Colmerauer & P. Roussel
1983	Warren abstract machine	D. Warren
1987	constraint logic programming	Jaffar & Maher
1994	answer set programming	Dimopoulos, Nebel & Köhler
	⋮	
2015	SWI-Prolog, Version 6.4.1	free
	SICStus Prolog, Version 4.3.1	SICS

Timeline

196?	procedural view of (Horn) logic	R. Kowalski
1972	<i>Programmation en Logique</i>	A. Colmerauer & P. Roussel
1983	Warren abstract machine	D. Warren
1987	constraint logic programming	Jaffar & Maher
1994	answer set programming	Dimopoulos, Nebel & Köhler
	⋮	
2015	SWI-Prolog, Version 6.4.1	free
	SICStus Prolog, Version 4.3.1	SICS

A Few Applications

- speech recognition: Clarissa
- networks: Ericsson Network Resource Manager
- program analysis: Julia

Basic Constructs

Definitions

- **terms** are built from **logical variables**, **constants** and **functors**
- **ground** term contains no variables; **nonground** term contains variables

Basic Constructs

Definitions

- **terms** are built from **logical variables**, **constants** and **functors**
- **ground** term contains no variables; **nonground** term contains variables

Definition

- **goals** (aka formulas) are constants or compound terms
- goals are typically non-ground

Basic Constructs

Definitions

- **terms** are built from **logical variables**, **constants** and **functors**
- **ground** term contains no variables; **nonground** term contains variables

Definition

- **goals** (aka formulas) are constants or compound terms
- goals are typically non-ground

Notation

we confuse function symbols and predicate symbols (= functors) in the definition of a term; this makes meta-level predicates more natural

Example (Goal)

```
father( andreas , boris )
```

Example (Goal)

```
father( andreas , boris )
```

Definitions (Clause)

- a **clause** or **rule** is an universally quantified logical formula of the form

$$A :- B_1, B_2, \dots, B_n.$$

where A and the B_i 's are goals

- A is called the **head** of the clause; the B_i 's are called the **body**
- a rule of the form $A \leftarrow$ is called a **fact**; we write facts simply A .

Example (Goal)

```
father( andreas , boris )
```

Definitions (Clause)

- a **clause** or **rule** is an universally quantified logical formula of the form

$$A :- B_1, B_2, \dots, B_n.$$

where A and the B_i 's are goals

- A is called the **head** of the clause; the B_i 's are called the **body**
- a rule of the form $A \leftarrow$ is called a **fact**; we write facts simply A .

Definition

a **logic program** is a finite set of clauses

Example (Facts)

```
father( andreas , boris ).      female( doris ).   male( andreas ).
father( andreas , christian ).  female( eva ).     male( boris ).
father( andreas , doris ).     male( christian ).
father( boris , eva ).         male( franz ).
father( franz , georg ).       male( georg ).
mother( helga , doris ).
mother( doris , franz ).
mother( anna , eva ).
mother( eva , georg ).
```

Example (Facts)

```

father( andreas , boris ).      female( doris ).  male( andreas ).
father( andreas , christian ).  female( eva ).    male( boris ).
father( andreas , doris ).     male( christian ).
father( boris , eva ).         male( franz ).
father( franz , georg ).      male( georg ).
mother( helga , doris ).
mother( doris , franz ).
mother( anna , eva ).
mother( eva , georg ).

```

Example (Rules)

```

daughter(X,Y) :- father(Y,X), female(X).
daughter(X,Y) :- mother(Y,X), female(X).
grandfather(X,Y) :- father(X,Z), father(Z,Y).
grandfather(X,Y) :- father(X,Z), mother(Z,Y).

```

Definition (Queries and Use Cases)

a complex query is a conjunction of goals of the following form:

$:- A1, A2, \dots, An$

Definition (Queries and Use Cases)

a complex query is a conjunction of goals of the following form:

```
:- A1, A2, ..., An
```

Example (Queries)

```
:- father( andreas , boris ).  
:- father( andreas , X ).  
:- father( X, Y ), female( X ).
```

Definition (Queries and Use Cases)

a complex query is a conjunction of goals of the following form:

```
:- A1, A2, ..., An
```

Example (Queries)

```
:- father( andreas , boris ).  
:- father( andreas , X ).  
:-/ father( X, Y ), female( X ).
```

Observations

- 1** **existential** query contains logical variable(s)

Definition (Queries and Use Cases)

a complex query is a conjunction of goals of the following form:

```
:- A1, A2, ..., An
```

Example (Queries)

```
:- father( andreas , boris ).  
:- father( andreas , X ).  
:- father( X, Y ), female( X ).
```

Observations

- 1 **existential** query contains logical variable(s)
- 2 **universal** fact contains logical variable(s)

Definition (Queries and Use Cases)

a complex query is a conjunction of goals of the following form:

```
:- A1, A2, ..., An
```

Example (Queries)

```
:- father( andreas , boris ).  
:- father( andreas , X ).  
:- father( X, Y ), female( X ).
```

Observations

- 1 **existential** query contains logical variable(s)
- 2 **universal** fact contains logical variable(s)
- 3 **conjunctive** query is conjunction of goals posed as query

Definition (Queries and Use Cases)

a complex query is a conjunction of goals of the following form:

```
:- A1, A2, ..., An
```

Example (Queries)

```
:- father( andreas , boris ).  
:- father( andreas , X ).  
:- father( X, Y ), female( X ).
```

Observations

- 1 **existential** query contains logical variable(s)
- 2 **universal** fact contains logical variable(s)
- 3 **conjunctive** query is conjunction of goals posed as query
- 4 it is good style to write use case **before** the actual program

Definitions

- **substitution** is finite set of pairs

$$\{X_1 \mapsto t_1, \dots, X_n \mapsto t_n\}$$

with terms t_1, \dots, t_n and pairwise different variables X_1, \dots, X_n

Definitions

- substitution is finite set of pairs

$$\{X_1 \mapsto t_1, \dots, X_n \mapsto t_n\}$$

with terms t_1, \dots, t_n and pairwise different variables X_1, \dots, X_n

- **application** of substitution θ to term t is denoted by $t\theta$

Definitions

- substitution is finite set of pairs

$$\{X_1 \mapsto t_1, \dots, X_n \mapsto t_n\}$$

with terms t_1, \dots, t_n and pairwise different variables X_1, \dots, X_n

- application of substitution θ to term t is denoted by $t\theta$
- $t\theta$ is **instance** of t

Definitions

- substitution is finite set of pairs

$$\{X_1 \mapsto t_1, \dots, X_n \mapsto t_n\}$$

with terms t_1, \dots, t_n and pairwise different variables X_1, \dots, X_n

- application of substitution θ to term t is denoted by $t\theta$
- $t\theta$ is instance of t

Examples

$$\theta_1 = \{X \mapsto \text{boris}\}$$

$$\theta_2 = \{X \mapsto \text{boris}, Y \mapsto \text{eva}\}$$

$$\theta_3 = \{X \mapsto s(Y), Y \mapsto 0\}$$

$$\text{father}(\text{andreas}, X)\theta_1 = \text{father}(\text{andreas}, \text{boris})$$

$$\text{father}(X, Y)\theta_2 = \text{father}(\text{boris}, \text{eva})$$

$$\text{list}(X, \text{list}(X, Y))\theta_3 = \text{list}(s(Y), \text{list}(s(Y), 0))$$

Definitions

- substitution is finite set of pairs

$$\{X_1 \mapsto t_1, \dots, X_n \mapsto t_n\}$$

with terms t_1, \dots, t_n and pairwise different variables X_1, \dots, X_n

- application of substitution θ to term t is denoted by $t\theta$
- $t\theta$ is instance of t

Examples

$$\theta_1 = \{X \mapsto \text{boris}\}$$

$$\theta_2 = \{X \mapsto \text{boris}, Y \mapsto \text{eva}\}$$

$$\theta_3 = \{X \mapsto s(Y), Y \mapsto 0\}$$

$$\text{father}(\text{andreas}, X)\theta_1 = \text{father}(\text{andreas}, \text{boris})$$

$$\text{father}(X, Y)\theta_2 = \text{father}(\text{boris}, \text{eva})$$

$$\text{list}(X, \text{list}(X, Y))\theta_3 = \text{list}(s(Y), \text{list}(s(Y), 0))$$

Definitions

- substitution is finite set of pairs

$$\{X_1 \mapsto t_1, \dots, X_n \mapsto t_n\}$$

with terms t_1, \dots, t_n and pairwise different variables X_1, \dots, X_n

- application of substitution θ to term t is denoted by $t\theta$
- $t\theta$ is instance of t

Examples

$$\theta_1 = \{X \mapsto \text{boris}\}$$

$$\theta_2 = \{X \mapsto \text{boris}, Y \mapsto \text{eva}\}$$

$$\theta_3 = \{X \mapsto s(Y), Y \mapsto 0\}$$

$$\text{father}(\text{andreas}, X)\theta_1 = \text{father}(\text{andreas}, \text{boris})$$

$$\text{father}(X, Y)\theta_2 = \text{father}(\text{boris}, \text{eva})$$

$$\text{list}(X, \text{list}(X, Y))\theta_3 = \text{list}(s(Y), \text{list}(s(Y), 0))$$

Example (Addition on Natural Numbers)

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

Example (Addition on Natural Numbers)

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

Queries

```
:- times(X,X,Y).
```

Example (Addition on Natural Numbers)

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

Queries

```
:- times(X,X,Y).  
X = 0, Y = 0
```

Example (Addition on Natural Numbers)

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

Queries

```
:- times(X,X,Y).  
X = 0, Y = 0  
true
```

Example (Addition on Natural Numbers)

```

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

plus(0,X,X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
times(0,X,0).
times(s(X),Y,Z) :- times(X,Y,U), plus(U,Y,Z).

```

Queries

```
:- times(X,X,Y).
```

`X = 0, Y = 0 ;` backtracking to find further solutions

Example (Addition on Natural Numbers)

```

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

plus(0,X,X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
times(0,X,0).
times(s(X),Y,Z) :- times(X,Y,U), plus(U,Y,Z).

```

Queries

```

:- times(X,X,Y).
X = 0, Y = 0 ;      backtracking to find further solutions
X = s(0), Y = s(0)

```

Example (Addition on Natural Numbers)

```

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

plus(0,X,X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
times(0,X,0).
times(s(X),Y,Z) :- times(X,Y,U), plus(U,Y,Z).

```

Queries

```

:- times(X,X,Y).
X = 0, Y = 0 ;      backtracking to find further solutions
X = s(0), Y = s(0) ;
X = s(s(0)), Y = s(s(s(0)))

```


Example (Addition on Natural Numbers)

```

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

plus(0,X,X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
times(0,X,0).
times(s(X),Y,Z) :- times(X,Y,U), plus(U,Y,Z).

```

Queries

```

:- times(X,X,Y).
X = 0, Y = 0 ;      backtracking to find further solutions
X = s(0), Y = s(0) ;
X = s(s(0)), Y = s(s(s(0))) ;

```

Example (Addition on Natural Numbers)

```

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

plus(0,X,X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
times(0,X,0).
times(s(X),Y,Z) :- times(X,Y,U), plus(U,Y,Z).

```

Queries

```

:- times(X,X,Y).                :- plus(X,s(0),0).
X = 0, Y = 0 ;
X = s(0), Y = s(0) ;
X = s(s(0)), Y = s(s(s(0))) ;

```

Example (Addition on Natural Numbers)

```

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

plus(0,X,X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
times(0,X,0).
times(s(X),Y,Z) :- times(X,Y,U), plus(U,Y,Z).

```

Queries

```

:- times(X,X,Y).                :- plus(X,s(0),0).
X = 0, Y = 0 ;                 false
X = s(0), Y = s(0) ;
X = s(s(0)), Y = s(s(s(0))) ;

```

Example (Addition on Natural Numbers)

```

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

plus(0,X,X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
times(0,X,0).
times(s(X),Y,Z) :- times(X,Y,U), plus(U,Y,Z).

```

Queries

```

:- times(X,X,Y).                :- plus(X,s(0),0).
X = 0, Y = 0 ;                  false
X = s(0), Y = s(0) ;           :- plus(X,s(0),s(s(X))).
X = s(s(0)), Y = s(s(s(0))) ;

```

Example (Addition on Natural Numbers)

```

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

plus(0,X,X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
times(0,X,0).
times(s(X),Y,Z) :- times(X,Y,U), plus(U,Y,Z).

```

Queries

<code>:- times(X,X,Y).</code>	<code>:- plus(X,s(0),0).</code>
<code>X = 0, Y = 0 ;</code>	<code>false</code>
<code>X = s(0), Y = s(0) ;</code>	<code>:- plus(X,s(0),s(s(X))).</code>
<code>X = s(s(0)), Y = s(s(s(s(0)))) ;</code>	<code>:- plus(s(0),X,s(s(X))).</code>

Example (Addition on Natural Numbers)

```

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

plus(0,X,X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
times(0,X,0).
times(s(X),Y,Z) :- times(X,Y,U), plus(U,Y,Z).

```

Queries

```

:- times(X,X,Y).                :- plus(X,s(0),0).
X = 0, Y = 0 ;                  false
X = s(0), Y = s(0) ;           :- plus(X,s(0),s(s(X))).
X = s(s(0)), Y = s(s(s(0))) ;  :- plus(s(0),X,s(s(X))).

```

Demo

compare SWI-Prolog with GPU

Comparison to Conventional Programming Languages

Fact

a programming language is characterised by its control and data manipulation mechanisms

Comparison to Conventional Programming Languages

Fact

a programming language is characterised by its control and data manipulation mechanisms

Control

	procedure <i>A</i>
	call <i>B</i> ₁
	call <i>B</i> ₂
	⋮
	call <i>B</i> _{<i>n</i>}
<i>A</i> :- <i>B</i> ₁ , <i>B</i> ₂ , ..., <i>B</i> _{<i>n</i>}	end

Comparison to Conventional Programming Languages

Fact

a programming language is characterised by its control and data manipulation mechanisms

Control

	procedure A
	call B_1
	call B_2
	⋮
	call B_n
$A :- B_1, B_2, \dots, B_n$	end

Observations

- 1 goal invocation corresponds to procedure invocation
- 2 differences show when backtracking occurs

Data Structures

- 1 data structures manipulated by logic programs (= terms) correspond to general record structures
- 2 like LISP, Prolog is a declaration free, untyped language
- 3 Prolog does not support destructive assignment where the content of the initialised variable can change

Data Structures

- 1 data structures manipulated by logic programs (= terms) correspond to general record structures
- 2 like LISP, Prolog is a declaration free, untyped language
- 3 Prolog does not support destructive assignment where the content of the initialised variable can change

Data Manipulation

- 1 data manipulation is achieved via unification
- 2 unification subsumes
 - single assignment
 - parameter passing
 - record allocation
 - read/write-once field access in records