

## Logic Programming

Georg Moser

Department of Computer Science @ UIBK

Winter 2016



### Organisation

#### Time and Place

Lecture	Monday, 10:15–11:45, HS 11	Georg Moser
Proseminar	Friday, 15:15–17:00, HS 11	(every other week)

#### Schedule

week 1	October 3	week 8	November 21
week 2	October 10	week 9	November 28
week 3	October 17	week 10	December 5
week 4	October 24	week 11	December 12
week 5	October 31	week 12	January 9
week 6	November 7	week 13	January 16
week 7	November 14	week 14	January 23
		first exam	January 30

#### Office Hours

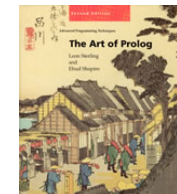
- Thursday, 9:00–11:00, 1N05, IfI Building

## Organisation

### Organisation

#### Literature

- Leon Sterling and Ehud Shapiro  
The Art of Prolog



#### Additional Reading

- Patrick Blackburn, Johan Bos and Kristina Striegnitz  
Learn Prolog Now!
- William F. Clocksin and Christopher S. Mellish  
Programming in Prolog
- Thom Frühwirth et al.  
Essentials of Constraint Programming
- Martin Gebser et al.  
Answer Set Solving in Practice

## Evaluations

### Exam

- first exam will take place on January 30
- closed-book (no materials, easier questions)

### Proseminar

- lecture and proseminar are on Monday and Friday, respectively
- each week I'll assign 3 exercises
- selection of exercises will be discussed every other week, starting October 10
- your mark depends on your level of activity in the laboratory
- exercises will be easy and few, so that everybody can solve all exercises

## 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](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))
```

## Outline of the Lecture

### Monotone Logic Programs

introduction, basic constructs, logic foundations, unification, semantics, database and recursive programming, termination, complexity

### Incomplete Data Structures and Constraints

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

### Full Prolog

semantics (revisted), cuts, correctness proofs, meta-logical predicates, efficient programs, meta programming

# Logic Programs

## History and Applications

### 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, CoFloCo

### 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

## Basic Constructs

### 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 )
```

## 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  $A :-$  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 (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:

$$:- A_1, A_2, \dots, A_n$$

## Example (Queries)

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

## Observations

- existential** query contains logical variable(s)
- universal** fact contains logical variable(s)
- conjunctive** query is conjunction of goals posed as query
- 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  $X_1, \dots, X_n$

- application** of substitution  $\theta$  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).
```

## 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

SWI-Prolog

## Comparison to Conventional Programming Languages

### Fact

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

### Control

```

                                procedure A
                                call B1
                                call B2
                                ⋮
                                call Bn
                                end

A :- B1, B2, ..., Bn
```

### 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 Manipulation

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