Logic Programming

Georg Moser

Institute of Computer Science @ UIBK

Summer 2015

Organisation

Time and Place

Lecture	Thursday, 11:15–13:00, HS 10
Proseminar	Thursday, 13:15–14:00, HS 10

Time and Place

Lecture	Thursday, 11:15–13:00, HS 10
Proseminar	Thursday, 13:15–14:00, HS 10

Schedule

week 1	March 5	week 8	May 7
week 2	March 12	week 9	May 21
week 3	March 19	week 10	May 28
week 4	March 26	week 11	June 11
week 5	April 16	week 12	June 18
week 6	April 23	first exam	June 25
week 7	April 30		

Time and Place

Lecture	Thursday, 11:15–13:00, HS 10
Proseminar	Thursday, 13:15–14:00, HS 10

Schedule

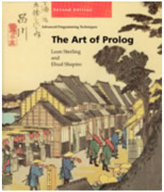
week 1	March 5	week 8	May 7
week 2	March 12	week 9	May 21
week 3	March 19	week 10	May 28
week 4	March 26	week 11	June 11
week 5	April 16	week 12	June 18
week 6	April 23	first exam	June 25
week 7	April 30		

Office Hours

Thursday, 9:00–11:00, 3M09, IfI Building

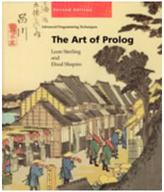
Literature

1. Leon Sterling and Ehud Shapiro
   The Art of Prolog



Literature

1. Leon Sterling and Ehud Shapiro
   The Art of Prolog

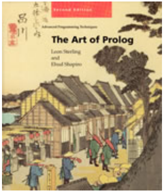


Additional Reading

- 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
- `http://groups.google.com/group/comp.lang.prolog`

Evaluations

Exam

- first exam will take place on June 25
- closed- or open-book will be decided in the lecture

Evaluations

Exam

- first exam will take place on June 25
- closed- or open-book will be decided in the lecture

Proseminar

- I'd like to combine lecture and proseminar, so that we can easily switch between lecture and practical programming
- still there will be weakly homework assignments, which will be discussed at a suitable time during the 3 hours
- your mark depends on your level of activity in the laboratory
- exercises will be easy and few, so that everybody can solve all exercises

Outline of the Lecture

Logic Programs

introduction, basic constructs, database and recursive programming, semantics

The Prolog Language

programming in pure prolog, arithmetic, structure inspection, meta-logical predicates, cuts, extra-logical predicates, efficient programs, complexity

Advanced Prolog Programming Techniques

nondeterministic programming, incomplete data structures, definite clause grammars, meta-programming, answer set programming

Outline of the Lecture

Logic Programs

introduction, basic constructs, database and recursive programming, semantics

The Prolog Language

programming in pure prolog, arithmetic, structure inspection, meta-logical predicates, cuts, extra-logical predicates, efficient programs, complexity

Advanced Prolog Programming Techniques

nondeterministic programming, incomplete data structures, definite clause grammars, meta-programming, answer set programming

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

Advertisment

> *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

Advertisment

> *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

Declarative Programming Languages

Robert Harper says[a]

[a] TYPES Mailing List, April 2013

The term "declarative" never meant a damn thing, but was often used, absurdly, to somehow lump together functional programming with logic programming, and separate it from imperative programming. It never made a lick of sense; it's just a marketing term.

Declarative Programming Languages

Robert Harper says[a]

[a] TYPES Mailing List, April 2013

The term "declarative" never meant a damn thing, but was often used, absurdly, to somehow lump together functional programming with logic programming, and separate it from imperative programming. It never made a lick of sense; it's just a marketing term.

Uday S. Reddy says

Indeed, "declarative" means a lot. But, "declarative programming language" doesn't. If somebody claims that some language is not "declarative", it just means that they never thought about its declarative interpretation, not that it doesn't exist. Ignorance is peddled as a fact of reality.

Declarative Programming Languages (cont'd)

Robert Harper says

> *I am referring to the term "declarative programming language", and should have been more precise in saying that. It's died down now, mostly, but for a while there was an attempt to equate logic programming languages with functional programming languages under this term.*[a]
> *If one wishes to use "declarative" as description of a denotational semantics, that's fine, but I would point out that even Prolog can only be understood fully in operational terms, e.g. the "cut" operator !, which controls the proof search procedure.*

[a] Remark: not true for wikipedia

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	*Programmation en Logique*	A. Colmerauer & P. Roussel
1983	Warren abstract machine	D. Warren

Timeline

196?	procedural view of (Horn) logic	R. Kowalski
1972	*Programmation en Logique*	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	*Programmation en Logique*	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	*Programmation en Logique*	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	*Programmation en Logique*	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	*Programmation en Logique*	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

- speach 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 (Goals)

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

Example (Goals)

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

Definitions (Clause)

- a clause or rule is an universally quantified logical formula of the form
$$A \leftarrow B_1, B_2, \ldots, B_n \ .$$
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 (Goals)

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

Definitions (Clause)

- a clause or rule is an universally quantified logical formula of the form
  $$A \leftarrow B_1, B_2, \ldots, B_n \ .$$
  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(anna,eva).  mother(doris,franz).
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(anna,eva).  mother(doris,franz).
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 (Query)

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

$$\leftarrow A_1, A_2, \ldots, A_n$$

Definition (Query)

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

$$\leftarrow A_1, A_2, \ldots, A_n$$

Example (Queries)

$\leftarrow$ `father(andreas,boris)` $\leftarrow$ `father(andreas,X)`

Definition (Query)

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

$$\leftarrow A_1, A_2, \ldots, A_n$$

Example (Queries)

$\leftarrow$ `father(andreas,boris)` $\leftarrow$ `father(andreas,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

Definitions

- substitution is finite set of pairs

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

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

Definitions

- substitution is finite set of pairs

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

  with terms $t_1, \ldots, t_n$ and pairwise different variables $X_1, \ldots, X_n$
- application of substitution θ to term t is denoted by $t\theta$

Definitions

- substitution is finite set of pairs

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

  with terms $t_1, \ldots, t_n$ and pairwise different variables $X_1, \ldots, X_n$
- 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

$$\theta_1 = \{\mathtt{X} \mapsto \mathtt{boris}\}$$
$$\theta_2 = \{\mathtt{X} \mapsto \mathtt{boris}, \mathtt{Y} \mapsto \mathtt{eva}\}$$
$$\theta_3 = \{\mathtt{X} \mapsto \mathtt{s}(\mathtt{Y}), \mathtt{Y} \mapsto \mathtt{0}\}$$

$$\mathtt{father}(\mathtt{andreas}, \mathtt{X})\theta_1 = \mathtt{father}(\mathtt{andreas}, \mathtt{boris})$$
$$\mathtt{father}(\mathtt{X}, \mathtt{Y})\theta_2 = \mathtt{father}(\mathtt{boris}, \mathtt{eva})$$
$$\mathtt{list}(\mathtt{X}, \mathtt{list}(\mathtt{X}, \mathtt{Y}))\theta_3 = \mathtt{list}(\mathtt{s}(\mathtt{Y}), \mathtt{list}(\mathtt{s}(\mathtt{Y}), \mathtt{0}))$$

Definitions

- substitution is finite set of pairs

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

  with terms $t_1, \ldots, t_n$ and pairwise different variables $X_1, \ldots, X_n$
- application of substitution θ to term t is denoted by $t\theta$
- $t\theta$ is instance of t

Examples

$$\theta_1 = \{\mathtt{X} \mapsto \mathtt{boris}\}$$
$$\theta_2 = \{\mathtt{X} \mapsto \mathtt{boris}, \mathtt{Y} \mapsto \mathtt{eva}\}$$
$$\theta_3 = \{\mathtt{X} \mapsto \mathtt{s(Y)}, \mathtt{Y} \mapsto \mathtt{0}\}$$

$$\mathtt{father(andreas, X)}\theta_1 = \mathtt{father(andreas, boris)}$$
$$\mathtt{father(X, Y)}\theta_2 = \mathtt{father(boris, eva)}$$
$$\mathtt{list(X, list(X, Y))}\theta_3 = \mathtt{list(s(Y), list(s(Y), 0))}$$

Definitions

- substitution is finite set of pairs

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

  with terms $t_1, \ldots, t_n$ and pairwise different variables $X_1, \ldots, X_n$
- application of substitution θ to term t is denoted by $t\theta$
- $t\theta$ is instance of t

Examples

$$\theta_1 = \{\mathtt{X} \mapsto \mathtt{boris}\}$$
$$\theta_2 = \{\mathtt{X} \mapsto \mathtt{boris}, \mathtt{Y} \mapsto \mathtt{eva}\}$$
$$\theta_3 = \{\mathtt{X} \mapsto \mathtt{s(Y)}, \mathtt{Y} \mapsto \mathtt{0}\}$$

$$\mathtt{father(andreas, X)}\theta_1 = \mathtt{father(andreas, boris)}$$
$$\mathtt{father(X, Y)}\theta_2 = \mathtt{father(boris, eva)}$$
$$\mathtt{list(X, list(X, Y))}\theta_3 = \mathtt{list(s(Y), list(s(Y), 0))}$$

Example (Natural Numbers)

```
natural_number(0).
natural_number(s(X)) ← natural_number(X).
```

Example (Addition on Natural Numbers)

```
natural_number(0).
natural_number(s(X)) ← natural_number(X).
plus(0,X,X).
```

$0 + X = X$

Example (Addition on Natural Numbers)

```
natural_number(0).
natural_number(s(X)) ← natural_number(X).
plus(0,X,X) ← natural_number(X).
```

$0 + X = X$

Example (Addition on Natural Numbers)

```
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).
```

$$0 + X = X$$
$$s(X) + Y = s(X + Y)$$

Example (Addition on Natural Numbers)

```
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).

 ← plus(s(0),s(0),X)
```

$$0 + X = X$$
$$s(X) + Y = s(X + Y)$$

Example (Addition on Natural Numbers)

```
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).
```

$$0 + X = X$$
$$s(X) + Y = s(X + Y)$$

```
← plus(s(0),s(0),X)    ← plus(s(0),X,s(s(s(0))))
```

Example (Addition on Natural Numbers)

```
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).
```

$0 + X = X$

$s(X) + Y = s(X + Y)$

```
← plus(s(0),s(0),X)    ← plus(s(0),X,s(s(s(0))))    ← plus(X,X,s(0))
```

Example (Addition on Natural Numbers)

```
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).
```

$$0 + X = X$$
$$s(X) + Y = s(X + Y)$$

```
← plus(s(0),s(0),X)    ← plus(s(0),X,s(s(s(0))))    ← plus(X,X,s(0))
```

```
[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).

 ?-
```

Notation

- A :- $A_1, \ldots, A_m$. instead of $A \leftarrow A_1, \ldots, A_m$. for rules

Notation

- A `:-` $A_1, \ldots, A_m$`.` instead of $A \leftarrow A_1, \ldots, A_m$. for rules
- `?-` $A_1, \ldots, A_m$`.` instead of $\leftarrow A_1, \ldots, A_m$ for queries

Notation

- A `:-` $A_1, \ldots, A_m$. instead of $A \leftarrow A_1, \ldots, A_m$. for rules
- `?-` $A_1, \ldots, A_m$. instead of $\leftarrow A_1, \ldots, A_m$ for queries

Example

```
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).
```

Notation

- A `:-` $A_1, \ldots, A_m$`.` instead of $A \leftarrow A_1, \ldots, A_m$. for rules
- `?-` $A_1, \ldots, A_m$`.` instead of $\leftarrow A_1, \ldots, A_m$ for queries

Example

```
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).
```

Notation

- A `:-` $A_1, \ldots, A_m$`.` instead of $A \leftarrow A_1, \ldots, A_m$. for rules
- `?-` $A_1, \ldots, A_m$`.` instead of $\leftarrow A_1, \ldots, A_m$ for queries

Example

```
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).
```

Query

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

Notation

- A `:-` $A_1, \ldots, A_m$`.` instead of $A \leftarrow A_1, \ldots, A_m$. for rules
- `?-` $A_1, \ldots, A_m$`.` instead of $\leftarrow A_1, \ldots, A_m$ for queries

Example

```
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).
```

Query

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

Notation

- A `:-` $A_1, \ldots, A_m$`.` instead of $A \leftarrow A_1, \ldots, A_m$. for rules
- `?-` $A_1, \ldots, A_m$`.` instead of $\leftarrow A_1, \ldots, A_m$ for queries

Example

```
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).
```

Query

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

Notation

- A `:-` $A_1, \ldots, A_m$ `.` instead of $A \leftarrow A_1, \ldots, A_m$. for rules
- `?-` $A_1, \ldots, A_m$ `.` instead of $\leftarrow A_1, \ldots, A_m$ for queries

Example

```
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).
```

Query

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

Notation

- A `:-` $A_1, \ldots, A_m$ `.` instead of $A \leftarrow A_1, \ldots, A_m$. for rules
- `?-` $A_1, \ldots, A_m$ `.` instead of $\leftarrow A_1, \ldots, A_m$ for queries

Example

```
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).
```

Query

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

Notation

- `A :- A1, ..., Am.` instead of $A \leftarrow A_1, \ldots, A_m$. for rules
- `?- A1, ..., Am.` instead of $\leftarrow A_1, \ldots, A_m$ for queries

Example

```
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).
```

Query

```
?- 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(s(0))))
```

Notation

- A `:-` $A_1, \ldots, A_m$`.` instead of $A \leftarrow A_1, \ldots, A_m$. for rules
- `?-` $A_1, \ldots, A_m$`.` instead of $\leftarrow A_1, \ldots, A_m$ for queries

Example

```
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).
```

Query

```
?- 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(s(0)))) ;
```

Notation

- A `:-` $A_1, \ldots, A_m$`.` instead of $A \leftarrow A_1, \ldots, A_m$. for rules
- `?-` $A_1, \ldots, A_m$`.` instead of $\leftarrow A_1, \ldots, A_m$ for queries

Example

```
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 ;
X = s(0), Y = s(0) ;
X = s(s(0)), Y = s(s(s(s(0)))) ;
```

```
?- plus(X,s(0),0).
```

Notation

- A `:-` $A_1, \ldots, A_m$`.` instead of $A \leftarrow A_1, \ldots, A_m$. for rules
- `?-` $A_1, \ldots, A_m$`.` instead of $\leftarrow A_1, \ldots, A_m$ for queries

Example

```
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 ;
X = s(0), Y = s(0) ;
X = s(s(0)), Y = s(s(s(s(0)))) ;
```

```
?- plus(X,s(0),0).
false
```

Notation

- A `:-` $A_1, \ldots, A_m$. instead of $A \leftarrow A_1, \ldots, A_m$. for rules
- `?-` $A_1, \ldots, A_m$. instead of $\leftarrow A_1, \ldots, A_m$ for queries

Example

```
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 ;
X = s(0), Y = s(0) ;
X = s(s(0)), Y = s(s(s(s(0)))) ;
```

```
?- plus(X,s(0),0).
false
?- plus(X,s(0),s(s(X))).
```

Notation

- A `:-` $A_1, \ldots, A_m$`.` instead of $A \leftarrow A_1, \ldots, A_m$. for rules
- `?-` $A_1, \ldots, A_m$`.` instead of $\leftarrow A_1, \ldots, A_m$ for queries

Example

```
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 ;
X = s(0), Y = s(0) ;
X = s(s(0)), Y = s(s(s(s(0)))) ;
```

```
?- plus(X,s(0),0).
false
?- plus(X,s(0),s(s(X))).
?- plus(s(0),X,s(s(X))).
```

Examples from LICS

Tower of Hanoi in Prolog

```
hanoi(0,_,_,_).
hanoi(N,X,Y,Z) :-
    N > 0, M is N-1,
    hanoi(M,X,Z,Y),
    move(N,X,Y),
    hanoi(M,Z,Y,X).

move(D,X,Y) :-
    write('move disk '), write(D),
    write(' from '), write(X),
    write(' to '), write(Y), nl.

?- hanoi(4,a,c,b).
```