Logic Programming with Prolog

Mathematical Logic – First Term 2023-2024

MZI

School of Computing
Telkom University

SoC Tel-U

October-November 2023

Acknowledgements I

This slide is compiled using the materials in the following sources:

1. *Logic Programming with Prolog*, 2nd Edition, 2013, by Max Bramer.
2. *Discrete Mathematics and Its Applications* (Chapter 1), 7th Edition, 2012, by K. H. Rosen.
3. *Discrete Mathematics with Applications* (Chapter 3), 4th Edition, 2010, by S. S. Epp.
4. *Mathematical Logic for Computer Science* (Chapter 5, 6), 2nd Edition, 2000, by M. Ben-Ari.
5. Computational logic lecture slides at Fasilkom UI by L. Y. Stefanus.
6. Logic programming lecture slides at ILLC, University of Amsterdam by U. Endriss.
7. Functional programming lecture slides at Fasilkom UI by A. Azurat.
8. Mathematical Logic slides in Telkom University by A. Rakhmatsyah and B. Purnama.

Acknowledgements II

Some figures are excerpted from those sources. This slide is intended for internal academic purpose in SoC Telkom University. No slides are ever free from error nor incapable of being improved. Please convey your comments and corrections (if any) to <pleasedontspam>@telkomuniversity.ac.id.

Contents

1. What is Prolog?
2. Prolog Installation
3. Using the Interactive Interpreter
4. Elementary Prolog Programming: Basic Facts and Queries
5. Simple Rules in Prolog
6. Representation of Quantifiers in Prolog (Supplementary)
7. Term Equality and Inequality in Prolog (Supplementary)

Contents

1 What is Prolog?

2 Prolog Installation

3 Using the Interactive Interpreter

4 Elementary Prolog Programming: Basic Facts and Queries

5 Simple Rules in Prolog

6 Representation of Quantifiers in Prolog (Supplementary)

7 Term Equality and Inequality in Prolog (Supplementary)

Epigram

A language that doesn't affect the way you think about programming, is not worth knowing
(*Alan Jay Perlis, first recipient of Turing Award*)

(Excerpted from `Alan Perlis Quotes`)

Programming Language Paradigm

Imperative language: language that emphasizes on **how to perform** computation as an action

- *Structural language*: C, Pascal, Ada, Fortran, Python, Matlab/ Octave
- *Object oriented language*: C++, Java

Programming Language Paradigm

Imperative language: language that emphasizes on **how to perform** computation as an action

- *Structural language*: C, Pascal, Ada, Fortran, Python, Matlab/ Octave
- *Object oriented language*: C++, Java

Descriptive language: language that emphasizes on **what to perform** in a computation

- *Logic programming*: Prolog, Flora, Logtalk
- *Functional programming*: Haskell, ML

Imperative versus Descriptive Language

In imperative language:

- a group of commands is assembled together using ";" or *indentation*
- a command is controlled using selection (with `if-then-else` or `case-of` *statements*) or repetition (with `for` *loop*, `while` *loop*, or `repeat-until`)
- a program is a set of commands for changing the value of one or more variables (*the state of the variables*)

Imperative versus Descriptive Language

In imperative language:

- a group of commands is assembled together using "`;`" or *indentation*
- a command is controlled using selection (with `if-then-else` or `case-of` *statements*) or repetition (with `for` *loop*, `while` *loop*, or `repeat-until`)
- a program is a set of commands for changing the value of one or more variables (*the state of the variables*)

In descriptive language – logic programming:

- a program is a set of expression
- a program consists of *facts* and *rules*
- a computation in the program is a *deduction* or *inference*
- programs do not use *assignment* as a basic operator such as in C or Pascal

Why do we need to learn logic programming?

Logic programming with Prolog is useful for following reasons:

- Prolog is suitably applied for *symbolical computation* (i.e., a *non-numerical computation*)
- Prolog is suitably applied for solving problems which relate to *object conditions* or the *relations of several objects*
- Prolog is suitably applied for illustrating the *reasoning process in artificial intelligence*

Because of these reasons, Prolog is suitably applied in *artificial intelligence*, *natural language processing*, and *database*.

Why do we need to learn logic programming?

Logic programming with Prolog is useful for following reasons:

- Prolog is suitably applied for *symbolical computation* (i.e., a *non-numerical computation*)
- Prolog is suitably applied for solving problems which relate to *object conditions* or the *relations of several objects*
- Prolog is suitably applied for illustrating the *reasoning process in artificial intelligence*

Because of these reasons, Prolog is suitably applied in *artificial intelligence*, *natural language processing*, and *database*.

Prolog is less suitable for performing:

- computations which require *high numerical precision* (such as graphical computation)
- computations which require *high portability* (such as real-time messaging service)

Brief History of Prolog

- Prolog is an acronym of **Pro***gramming in* **Log***ic*.
- Prolog was developed between 1972-1973 by Alain Colmerauer and Phillipe Roussel at Aix-Marseille Université, France.
- The purpose of development was initially for processing natural language (human language).
- Currently, Prolog is the most widely taught logical programming language in the world.
- Prolog is also utilized in industrial world, such as by IBM and Apache.

Meet Watson: The Supercomputer

Watson is a (super) computer capable of answering questions in natural language (English) and it is developed in IBM's DeepQA project. The artificial intelligence in Watson is constructed using Prolog.

Watson in *Jeopardy!* exhibition match. Source:.WIKIPEDIA: Watson (Computer)

Contents

1. What is Prolog?
2. Prolog Installation
3. Using the Interactive Interpreter
4. Elementary Prolog Programming: Basic Facts and Queries
5. Simple Rules in Prolog
6. Representation of Quantifiers in Prolog (Supplementary)
7. Term Equality and Inequality in Prolog (Supplementary)

Supporting Softwares

The *source file* for the installation can be downloaded from following links:

- Amzi! Prolog (`http://www.amzi.com/products/prolog_products.htm`).
- Logic Programming Associates Prolog (`http://www.lpa.co.uk`).
- SWI-Prolog (`http://www.swi-prolog.org/`).
- Visual Prolog (`http://www.visual-prolog.com/`).
- W-Prolog (`http://waitaki.otago.ac.nz/~michael/wp/`).

In this lecture slides, Prolog programs are written in SWI-Prolog syntax. All programs in this slide are executed in SWI-Prolog 64-bit version 7.2.2 which have been tested on Windows 7 64-bit and Linux Ubuntu 14.04 64-bit operating system. SWI-Prolog is the simplest and easiest version of Prolog for learning logic programming. SWI-Prolog can be run on Windows, Linux, and Macintosh operating systems.

SWI-Prolog and *Prolog Programming Contest*

In 2013, SWI-Prolog is the only permitted Prolog variant to be used in *Prolog Programming Contest*. This event is held annually in conjunction with the ICLP (*International Conference on Logic Programming*).

SWI-Prolog Installation: Windows and Macintosh

For Windows and Macintosh operating system, the installation file can be downloaded at `http://www.swi-prolog.org/download/stable`. For Windows XP/ Vista/ 7/ 8 (32 or 64-bit), the installation can be performed like typical program installation. Installation directory can be customized, for example in `C:\Program Files\swipl`. Installation of SWI-Prolog has been successfully tested in Windows 7 64-bit operating system.

SWI-Prolog Installation: Linux (Ubuntu)

For Linux Ubuntu operating system, SWI-Prolog installation can be performed using PPA (*Personal Package Archive*). Open terminal, and then type following command:

```
% sudo apt-add-repository ppa:swi-prolog/stable
% sudo apt-get update
% sudo apt-get install swi-prolog
```

After the installation is completed, we can run SWI-Prolog via terminal by typing following command:

```
% swipl
?- emacs.
```

The command `?- emacs.` is used to run the GUI in Linux which is similar to that in Windows.

Illustration for running the SWI-Prolog GUI in Linux Ubuntu.

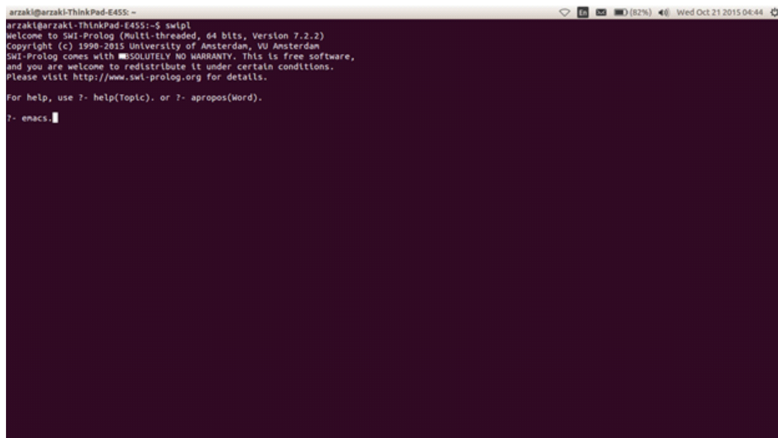
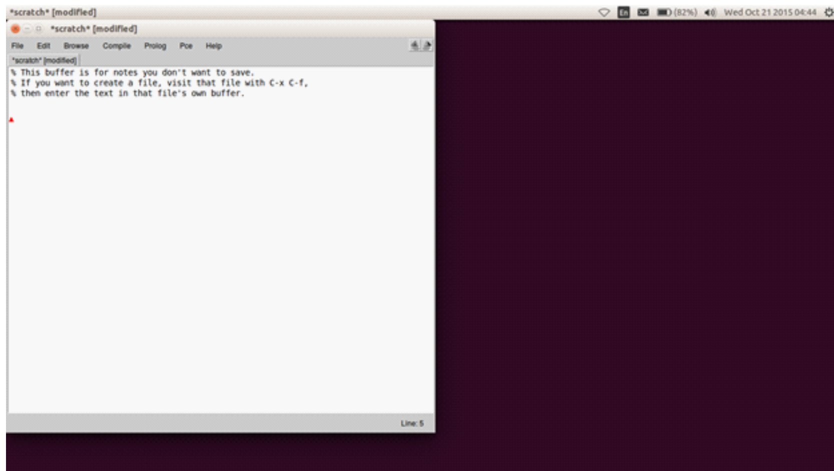


Illustration of the SWI-Prolog GUI in Linux Ubuntu.



SWI-Prolog Installation: Linux (Other Distros)

For Linux-based operating system other than Ubuntu, installation can be performed by consulting following link:
`http://www.swi-prolog.org/build/LinuxDistro.txt`.

About SWI-Prolog

Excerpted from `http://www.swi-prolog.org/`:

> *SWI-Prolog offers a comprehensive free Prolog environment. Since its start in 1987, SWI-Prolog development has been driven by the needs of real world applications. SWI-Prolog is widely used in research and education as well as commercial applications. Join over a million users who have downloaded SWI-Prolog.*

SWI is an acronym of *Sociaal-Wetenschappelijke Informatica* ("*Social Science Informatics*"), which is a Dutch name of *Human-Computer Studies* research group at the *University of Amsterdam*, The Netherlands. SWI-Prolog is written by Jan Wielemaker.

Contents

1. What is Prolog?
2. Prolog Installation
3. Using the Interactive Interpreter
4. Elementary Prolog Programming: Basic Facts and Queries
5. Simple Rules in Prolog
6. Representation of Quantifiers in Prolog (Supplementary)
7. Term Equality and Inequality in Prolog (Supplementary)

Using the Interactive Interpreter

When you open SWI-Prolog, you'll see following prompt:

```
1 ?-
```

Now try to type following command in SWI-Prolog:

```
write('Hello World'),nl,write('Welcome to Prolog'),nl.
```

```
1 ?-write('Hello World'),nl,write('Welcome to Prolog'),nl.
```

SWI-Prolog subsequently produces:

Using the Interactive Interpreter

When you open SWI-Prolog, you'll see following prompt:

```
1 ?-
```

Now try to type following command in SWI-Prolog:

`write('Hello World'),nl,write('Welcome to Prolog'),nl.`

```
1 ?-write('Hello World'),nl,write('Welcome to Prolog'),nl.
```

SWI-Prolog subsequently produces:

```
Hello World
Welcome to Prolog
true.
```

`write` and `nl` are two examples of *built-in-predicate* (BIP) in SWI-Prolog, `write('x')` serves to show `x` into the screen and `nl` serves to move the next output into the new line. **All Prolog commands must be ended by a period sign (.)**.

Arithmetic in Interactive Interpreter

Simple arithmetic operations can be performed in SWI-Prolog interpreter.

```
1 ?- X is 1+2.
```

Arithmetic in Interactive Interpreter

Simple arithmetic operations can be performed in SWI-Prolog interpreter.

```
1 ?- X is 1+2.
X = 3.
2 ?- X is 3-7.
```

Arithmetic in Interactive Interpreter

Simple arithmetic operations can be performed in SWI-Prolog interpreter.

```
1 ?- X is 1+2.
X = 3.
2 ?- X is 3-7.
X = -4.
3 ?- X is 2*3.
```

Arithmetic in Interactive Interpreter

Simple arithmetic operations can be performed in SWI-Prolog interpreter.

```
1 ?- X is 1+2.
X = 3.
2 ?- X is 3-7.
X = -4.
3 ?- X is 2*3.
X = 6.
4 ?- X is 7/3.
```

Arithmetic in Interactive Interpreter

Simple arithmetic operations can be performed in SWI-Prolog interpreter.

```
1 ?- X is 1+2.
X = 3.
2 ?- X is 3-7.
X = -4.
3 ?- X is 2*3.
X = 6.
4 ?- X is 7/3.
X = 2.3333333333333335.
5 ?- X is 3^2.
```

[Powering operation x^y can also be expressed as `x**y`]

Arithmetic in Interactive Interpreter

Simple arithmetic operations can be performed in SWI-Prolog interpreter.

```
1 ?- X is 1+2.
X = 3.
2 ?- X is 3-7.
X = -4.
3 ?- X is 2*3.
X = 6.
4 ?- X is 7/3.
X = 2.3333333333333335.
5 ?- X is 3^2.
X = 9.
6 ?- X is 3+5*2.
```

[Powering operation x^y can also be expressed as x ∗ ∗y]

Arithmetic in Interactive Interpreter

Simple arithmetic operations can be performed in SWI-Prolog interpreter.

```
1 ?- X is 1+2.
X = 3.
2 ?- X is 3-7.
X = -4.
3 ?- X is 2*3.
X = 6.
4 ?- X is 7/3.
X = 2.3333333333333335.
5 ?- X is 3^2.
X = 9.
6 ?- X is 3+5*2.
X = 13.
7 ?- X = 3*3.
```

[Powering operation x^y can also be expressed as x ∗ ∗y]

Arithmetic in Interactive Interpreter

Simple arithmetic operations can be performed in SWI-Prolog interpreter.

```
1 ?- X is 1+2.
X = 3.
2 ?- X is 3-7.
X = -4.
3 ?- X is 2*3.
X = 6.
4 ?- X is 7/3.
X = 2.3333333333333335.
5 ?- X is 3^2.
X = 9.
6 ?- X is 3+5*2.
X = 13.
7 ?- X = 3*3.
X = 3*3.
```

[Powering operation x^y can also be expressed as x ∗ ∗y]

Arithmetic calculations in SWI-Prolog are performed using the predicate `is`. The symbol = in Prolog is reserved for term equality.

More About Arithmetic Operations in Prolog

Prolog uses following arithmetical operators: $+$, $-$, $*$, $/$, $//$, and mod. Operator $//$ serves as an integer division operator (div operator).

```
1 ?- X is 9/4.
```

More About Arithmetic Operations in Prolog

Prolog uses following arithmetical operators: $+$, $-$, $*$, $/$, $//$, and mod. Operator $//$ serves as an integer division operator (div operator).

```
1 ?- X is 9/4.
X = 2.25.
2 ?- X is 9//4.
```

More About Arithmetic Operations in Prolog

Prolog uses following arithmetical operators: $+$, $-$, $*$, $/$, $//$, and mod. Operator $//$ serves as an integer division operator (div operator).

```
1 ?- X is 9/4.
X = 2.25.
2 ?- X is 9//4.
X = 2.
3 ?- X is 9 mod 4.
```

More About Arithmetic Operations in Prolog

Prolog uses following arithmetical operators: $+$, $-$, $*$, $/$, $//$, and mod. Operator $//$ serves as an integer division operator (div operator).

```
1 ?- X is 9/4.
X = 2.25.
2 ?- X is 9//4.
X = 2.
3 ?- X is 9 mod 4.
X = 1.
```

Comparison of Numerical Values in Prolog

We can use SWI-Prolog to compare numerical values.

```
1 ?- 1 < 2.
```

Comparison of Numerical Values in Prolog

We can use SWI-Prolog to compare numerical values.

```
1 ?- 1 < 2.
true.
2 ?- 3-2 > 3+2.
```

Comparison of Numerical Values in Prolog

We can use SWI-Prolog to compare numerical values.

```
1 ?- 1 < 2.
true.
2 ?- 3-2 > 3+2.
false.
3 ?- 3+2 =:= 6-1.
```

[symbol =:= means numerical equality]

Comparison of Numerical Values in Prolog

We can use SWI-Prolog to compare numerical values.

```
1 ?- 1 < 2.
true.
2 ?- 3-2 > 3+2.
false.
3 ?- 3+2 =:= 6-1.
true.
4 ?- 3+2 =\= 3-2.
```

[symbol =:= means numerical equality]

[symbol =\= means numerical inequality]

Comparison of Numerical Values in Prolog

We can use SWI-Prolog to compare numerical values.

```
1 ?- 1 < 2.
true.
2 ?- 3-2 > 3+2.
false.
3 ?- 3+2 =:= 6-1.
true.
4 ?- 3+2 =\= 3-2.
true.
5 ?- 3+2 >= 3-2.
```

[symbol =:= means numerical equality]

[symbol =\= means numerical inequality]

Comparison of Numerical Values in Prolog

We can use SWI-Prolog to compare numerical values.

```
1 ?- 1 < 2.
true.
2 ?- 3-2 > 3+2.
false.
3 ?- 3+2 =:= 6-1.
true.
4 ?- 3+2 =\= 3-2.
true.
5 ?- 3+2 >= 3-2.
true.
6 ?- 3+2 =< 3-2.
```

[symbol =:= means numerical equality]

[symbol =\= means numerical inequality]

Comparison of Numerical Values in Prolog

We can use SWI-Prolog to compare numerical values.

```
1 ?- 1 < 2.
true.
2 ?- 3-2 > 3+2.
false.
3 ?- 3+2 =:= 6-1.
true.
4 ?- 3+2 =\= 3-2.
true.
5 ?- 3+2 >= 3-2.
true.
6 ?- 3+2 =< 3-2.
false.
7 ?- 3+2 => 3-2.
```

[symbol =:= means numerical equality]

[symbol =\= means numerical inequality]

Comparison of Numerical Values in Prolog

We can use SWI-Prolog to compare numerical values.

```
1 ?- 1 < 2.
true.
2 ?- 3-2 > 3+2.
false.
3 ?- 3+2 =:= 6-1.                    [symbol =:= means numerical equality]
true.
4 ?- 3+2 =\= 3-2.                    [symbol =\= means numerical inequality]
true.
5 ?- 3+2 >= 3-2.
true.
6 ?- 3+2 =< 3-2.
false.
7 ?- 3+2 => 3-2.
ERROR: Syntax error:  Operator expected
ERROR: 3+2
ERROR: ** here **
ERROR: => 3-2.
```

Numerical Value Comparison Operators in Prolog

The numerical value comparison operators in Prolog are:

Mathematical symbol	Prolog symbol
$=$	`=:=`
$\neq$	`=\=`
$<$	`<`
$>$	`>`
$\leq$	`=<`
$\geq$	`>=`

The symbols `<=` and `=>` **are not the symbol for numerical value comparison** in Prolog.

Contents

1. What is Prolog?
2. Prolog Installation
3. Using the Interactive Interpreter
4. Elementary Prolog Programming: Basic Facts and Queries
5. Simple Rules in Prolog
6. Representation of Quantifiers in Prolog (Supplementary)
7. Term Equality and Inequality in Prolog (Supplementary)

Writing your first SWI-Prolog program

Writing a program in SWI-Prolog can be performed using any text editor (e.g., notepad or notepad++), however SWI-Prolog is also equipped with an editor that can check the syntax of Prolog program built.

Creating a script in Windows

Open SWI-Prolog, then choose `File` → `New` and put an appropriate file name for the program (make sure that the file type is *Prolog Source* with `.pl` extension). Afterward, we can write the program in the text editor provided.

Creating a script in Linux Ubuntu

Open terminal and then type `swipl`, after entering SWI-Prolog type `emacs.` (with `.` sign). Next choose `File` → `New` and put an appropriate file name for the program (make sure that the file has `.pl` extension). Afterward, we can write the program in the text editor provided.

Translation of Simple Predicate Formulas to Prolog

Suppose D is a domain consisting of 12 people, where
$D = \{Alice, Bob, Charlie, David, Emma, Fiona, Grace, Hans, Irene, Jim, Kelly, Lily\}$.

Suppose $\mathrm{Male}\,(x)$ is a predicate which states that "x is a male" and $\mathrm{Female}\,(x)$ is a predicate which states that "x is a female".

Suppose we have following facts:

- Bob, Charlie, David, Hans, and Jim are male.
- Alice, Emma, Fiona, Grace, Irene, Kelly, and Lily are female.

These facts can be expressed in predicate formulas as:

Translation of Simple Predicate Formulas to Prolog

Suppose D is a domain consisting of 12 people, where
$D = \{Alice, Bob, Charlie, David, Emma, Fiona, Grace, Hans, Irene, Jim, Kelly, Lily\}$.

Suppose $\mathrm{Male}\,(x)$ is a predicate which states that "x is a male" and $\mathrm{Female}\,(x)$ is a predicate which states that "x is a female".

Suppose we have following facts:

- Bob, Charlie, David, Hans, and Jim are male.
- Alice, Emma, Fiona, Grace, Irene, Kelly, and Lily are female.

These facts can be expressed in predicate formulas as:

- $\mathrm{Male}\,(Bob)$, $\mathrm{Male}\,(Charlie)$, $\mathrm{Male}\,(David)$, $\mathrm{Male}\,(Hans)$, $\mathrm{Male}\,(Jim)$

Translation of Simple Predicate Formulas to Prolog

Suppose D is a domain consisting of 12 people, where $D = \{Alice, Bob, Charlie, David, Emma, Fiona, Grace, Hans, Irene, Jim, Kelly, Lily\}$.

Suppose $\mathrm{Male}\,(x)$ is a predicate which states that "x is a male" and $\mathrm{Female}\,(x)$ is a predicate which states that "x is a female".

Suppose we have following facts:

- Bob, Charlie, David, Hans, and Jim are male.
- Alice, Emma, Fiona, Grace, Irene, Kelly, and Lily are female.

These facts can be expressed in predicate formulas as:

- $\mathrm{Male}\,(Bob)$, $\mathrm{Male}\,(Charlie)$, $\mathrm{Male}\,(David)$, $\mathrm{Male}\,(Hans)$, $\mathrm{Male}\,(Jim)$
- $\mathrm{Female}\,(Alice)$, $\mathrm{Female}\,(Emma)$, $\mathrm{Female}\,(Fiona)$, $\mathrm{Female}\,(Grace)$, $\mathrm{Female}\,(Irene)$, $\mathrm{Female}\,(Kelly)$, $\mathrm{Female}\,(Lily)$

Prolog Script

Our previous facts can be expressed in Prolog script as follows:

```
% male(x) states that x is a male.
/* Bob, Charlie, David, Hans, & Jim are male */
male(bob).
male(charlie).
male(david).
male(hans).
male(jim).
% female(x) states that x is a female.
/* Alice, Emma, Fiona, Grace, Irene, Kelly, & Lily are female */
female(alice).
female(emma).
female(fiona).
female(grace).
female(irene).
female(kelly).
female(lily).
```

- Predicate is started with lowercase letter, the same thing goes for constant term as well. **Constant term cannot be started with uppercase letter**.
- A constant term can be written using single quotation mark, such as `male('Bob')`.
- Comments in Prolog script are preceded by % symbol or enclosed with /* and */ symbol.

Running Prolog program: Windows and Linux

In the text editor choose `Compile` → `Make` and then `Compile` → `Compile Buffer`. Interpreter (or terminal in Linux) will then produce the warning stated that the compilation process has been carried out successfully.

Example of a successful compilation process in Linux Ubuntu 14.04.

```
arzaki@arzaki-ThinkPad-E455:~$ swipl
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 7.2.2)
Copyright (c) 1990-2015 University of Amsterdam, VU 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.
true.

?- % /home/arzaki/Documents/#.MyPrologProject/prolog001 compiled 0.00 sec, 8 clauses
```

Queries in Prolog

After a successful compilation, we can enter several queries in interactive interpreter (or terminal in Linux). Move to the next line is performed using enter or tab.

```
?- male(bob).
```

[Is Bob a male?]

Queries in Prolog

After a successful compilation, we can enter several queries in interactive interpreter (or terminal in Linux). Move to the next line is performed using enter or tab.

```
?- male(bob).                    [Is Bob a male?]
true.
?- not(male(charlie)).           [Is Charlie not a male?]
```

Queries in Prolog

After a successful compilation, we can enter several queries in interactive interpreter (or terminal in Linux). Move to the next line is performed using enter or tab.

```
?- male(bob).                              [Is Bob a male?]
true.
?- not(male(charlie)).                     [Is Charlie not a male?]
false.
?- male(bob),female(alice).                [, is the ∧ operator]
```

Queries in Prolog

After a successful compilation, we can enter several queries in interactive interpreter (or terminal in Linux). Move to the next line is performed using enter or tab.

```
?- male(bob).                    [Is Bob a male?]
true.
?- not(male(charlie)).           [Is Charlie not a male?]
false.
?- male(bob),female(alice).      [, is the ∧ operator]
true.
?- male(david),male(emma).
```

Queries in Prolog

After a successful compilation, we can enter several queries in interactive interpreter (or terminal in Linux). Move to the next line is performed using enter or tab.

```
?- male(bob).                        [Is Bob a male?]
true.
?- not(male(charlie)).               [Is Charlie not a male?]
false.
?- male(bob),female(alice).          [, is the ∧ operator]
true.
?- male(david),male(emma).
false.
?- male(david);male(fiona).          [; is the ∨ operator]
```

Queries in Prolog

After a successful compilation, we can enter several queries in interactive interpreter (or terminal in Linux). Move to the next line is performed using enter or tab.

```
?- male(bob).                                   [Is Bob a male?]
true.
?- not(male(charlie)).                          [Is Charlie not a male?]
false.
?- male(bob),female(alice).                     [, is the ∧ operator]
true.
?- male(david),male(emma).
false.
?- male(david);male(fiona).                     [; is the ∨ operator]
true.
?- female(charlie);male(grace).
```

Queries in Prolog

After a successful compilation, we can enter several queries in interactive interpreter (or terminal in Linux). Move to the next line is performed using enter or tab.

```
?- male(bob).                          [Is Bob a male?]
true.
?- not(male(charlie)).                 [Is Charlie not a male?]
false.
?- male(bob),female(alice).            [, is the ∧ operator]
true.
?- male(david),male(emma).
false.
?- male(david);male(fiona).            [; is the ∨ operator]
true.
?- female(charlie);male(grace).
false.
```

Closed World Assumption

In our previous program, we can inquire following query.

```
?- male(anakin).
```

Closed World Assumption

In our previous program, we can inquire following query.

```
?- male(anakin).
false.
?- female(anakin).
```

Closed World Assumption

In our previous program, we can inquire following query.

```
?- male(anakin).
false.
?- female(anakin).
false.
?- not(male(anakin)).
```

Closed World Assumption

In our previous program, we can inquire following query.

```
?- male(anakin).
false.
?- female(anakin).
false.
?- not(male(anakin)).
true.
?- not(female(anakin)).
```

Closed World Assumption

In our previous program, we can inquire following query.

```
?- male(anakin).
false.
?- female(anakin).
false.
?- not(male(anakin)).
true.
?- not(female(anakin)).
true.
```

Remark

When Prolog process `not(...)` query, Prolog checked whether the fact in the bracket, i.e., `(...)`, is true. If the fact `(...)` is not true, then Prolog returns the value `false`. The reasoning carried out in Prolog is based on the *closed world assumption*. According to this assumption, anything is true whenever it is a fact in the program or it can be derived from the facts in the program. In addition, any conclusion that cannot be proved to follow from the facts and rules in the program is false.

Prolog Variables and Their Queries

Variables in Prolog **always started with uppercase letters**. Usually variables are only written with one capital letter (e.g., `X`, `Y`, or `Z`). Variables are used in the interpreter to show all objects which satisfy the true condition of particular predicates.

```
?- male(X).
```

Prolog Variables and Their Queries

Variables in Prolog **always started with uppercase letters**. Usually variables are only written with one capital letter (e.g., `X`, `Y`, or `Z`). Variables are used in the interpreter to show all objects which satisfy the true condition of particular predicates.

```
?- male(X).
X = bob;
```

[press tab to see the next result]

Prolog Variables and Their Queries

Variables in Prolog **always started with uppercase letters**. Usually variables are only written with one capital letter (e.g., `X`, `Y`, or `Z`). Variables are used in the interpreter to show all objects which satisfy the true condition of particular predicates.

```
?- male(X).
X = bob;
X = charlie;
```

[press tab to see the next result]

Prolog Variables and Their Queries

Variables in Prolog **always started with uppercase letters**. Usually variables are only written with one capital letter (e.g., `X`, `Y`, or `Z`). Variables are used in the interpreter to show all objects which satisfy the true condition of particular predicates.

```
?- male(X).
X = bob;
X = charlie;
X = david;
```

[press tab to see the next result]

Prolog Variables and Their Queries

Variables in Prolog **always started with uppercase letters**. Usually variables are only written with one capital letter (e.g., `X`, `Y`, or `Z`). Variables are used in the interpreter to show all objects which satisfy the true condition of particular predicates.

```
?- male(X).
X = bob;
X = charlie;
X = david;
X = hans;
```

[press tab to see the next result]

Prolog Variables and Their Queries

Variables in Prolog **always started with uppercase letters**. Usually variables are only written with one capital letter (e.g., `X`, `Y`, or `Z`). Variables are used in the interpreter to show all objects which satisfy the true condition of particular predicates.

```
?- male(X).
X = bob;                    [press tab to see the next result]
X = charlie;
X = david;
X = hans;
X = jim.
```

```
?- female(Person).
```

```
?- female(Person).
Person = alice;
```

```
?- female(Person).
Person = alice;
Person = emma;
```

```
?- female(Person).
Person = alice;
Person = emma;
Person = fiona;
```

```
?- female(Person).
Person = alice;
Person = emma;
Person = fiona;
Person = grace;
```

```
?- female(Person).
Person = alice;
Person = emma;
Person = fiona;
Person = grace;
Person = irene;
```

```
?- female(Person).
Person = alice;
Person = emma;
Person = fiona;
Person = grace;
Person = irene;
Person = kelly;
```

```
?- female(Person).
Person = alice;
Person = emma;
Person = fiona;
Person = grace;
Person = irene;
Person = kelly;
Person = lily.
```

Binary Predicate in Prolog

Suppose D is the universe of discourse in our previous example and $\mathrm{Parent}\,(x, y)$ is a binary predicate which states that "x is the parent of y". Suppose we have following facts in predicate formulas:

- $\mathrm{Parent}\,(Alice, Charlie)$, $\mathrm{Parent}\,(Bob, Charlie)$, $\mathrm{Parent}\,(Bob, Emma)$.
- $\mathrm{Parent}\,(Charlie, Fiona)$, $\mathrm{Parent}\,(Charlie, Grace)$, $\mathrm{Parent}\,(Emma, Irene)$.
- $\mathrm{Parent}\,(Fiona, David)$, $\mathrm{Parent}\,(Fiona, Lily)$, $\mathrm{Parent}\,(Grace, Jim)$, $\mathrm{Parent}\,(Grace, Kelly)$, $\mathrm{Parent}\,(Hans, Jim)$, $\mathrm{Parent}\,(Hans, Kelly)$.

This fact can be expressed in Prolog script as::

```
% parent(x,y) states that x is a parent of y
parent(alice,charlie).
parent(bob,charlie).
parent(bob,emma).
parent(charlie,fiona).
parent(charlie,grace).
parent(emma,irene).
parent(fiona,david).
parent(fiona,lily).
parent(grace,jim).
parent(grace,kelly).
parent(hans,jim).
parent(hans,kelly).
```

These facts can be added to our previous Prolog script.

Remark

Editing Prolog file which was created earlier can be done by:
In Windows:

- In Prolog interpreter, choose `File` $\longrightarrow$ `Edit` and then select the file you want to edit, or
- In a text editor, choose `File` $\longrightarrow$ `Open` then select the file you want to edit.

In Linux Ubuntu: on emacs choose `File` $\longrightarrow$ `Open` and then select the file you want to edit.

After the program is compiled, we can execute following queries:

```
?- parent(TheParent,TheKid).
```

After the program is compiled, we can execute following queries:

```
?- parent(TheParent,TheKid).
TheParent = alice,
TheKid = charlie;
```

After the program is compiled, we can execute following queries:

```
?- parent(TheParent,TheKid).
TheParent = alice,
TheKid = charlie;
TheParent = bob,
TheKid = charlie;
```

⋮ [several other outputs]

After the program is compiled, we can execute following queries:

```
?- parent(TheParent,TheKid).
TheParent = alice,
TheKid = charlie;
TheParent = bob,
TheKid = charlie;
```

⋮ [several other outputs]

```
TheParent = hans,
TheKid = jim;
```

After the program is compiled, we can execute following queries:

```
?- parent(TheParent,TheKid).
TheParent = alice,
TheKid = charlie;
TheParent = bob,
TheKid = charlie;
```

⋮ [several other outputs]

```
TheParent = hans,
TheKid = jim;
TheParent = hans,
TheKid = kelly.
```

In order to know all of Fiona's children, we can perform the query `parent(fiona,X)`. Similarly, in order to know all of Jim's parents, we can perform the query `parent(X,jim)`.

```
?- parent(fiona,X).
```

In order to know all of Fiona's children, we can perform the query `parent(fiona,X)`. Similarly, in order to know all of Jim's parents, we can perform the query `parent(X,jim)`.

```
?- parent(fiona,X).
X = david;
X = lily.

?- parent(X,jim).
```

In order to know all of Fiona's children, we can perform the query `parent(fiona,X)`. Similarly, in order to know all of Jim's parents, we can perform the query `parent(X,jim)`.

```
?- parent(fiona,X).
X = david;
X = lily.

?- parent(X,jim).
X = grace;
X = hans.
```

Contents

1. What is Prolog?
2. Prolog Installation
3. Using the Interactive Interpreter
4. Elementary Prolog Programming: Basic Facts and Queries
5. Simple Rules in Prolog
6. Representation of Quantifiers in Prolog (Supplementary)
7. Term Equality and Inequality in Prolog (Supplementary)

We can add following facts to the previous Prolog script:

```
% adult(x) states that x is an adult.
% Alice, Bob, Charlie, Emma, Fiona, Grace, & Hans are adults.
adult(alice).
adult(bob).
adult(charlie).
adult(emma).
adult(fiona).
adult(grace).
adult(hans).
% teen(x) states that x is a teenager.
% Irene, David, & Lily are teenagers.
teen(irene).
teen(david).
teen(lily).
```

```
% kid(x) states that x is a little child.
% Jim & Kelly are little children.
kid(jim).
kid(kelly).
```

Translating Predicate Formulas to Prolog Script

Suppose we have following derivative predicates:

1. $\mathrm{Gentleman}\,(x)$: "x is an adult male". The predicate $\mathrm{Gentleman}\,(x)$ is defined as $\mathrm{Gentleman}\,(x) :=$

Translating Predicate Formulas to Prolog Script

Suppose we have following derivative predicates:

1. $\mathrm{Gentleman}\,(x)$: "x is an adult male". The predicate $\mathrm{Gentleman}\,(x)$ is defined as $\mathrm{Gentleman}\,(x) := \mathrm{Male}\,(x) \wedge \mathrm{Adult}\,(x)$.
2. $\mathrm{Lady}\,(x)$: "x is an adult female". The predicate $\mathrm{Lady}\,(x)$ is defined as $\mathrm{Lady}\,(x) :=$

Translating Predicate Formulas to Prolog Script

Suppose we have following derivative predicates:

1. $\mathrm{Gentleman}\,(x)$: "x is an adult male". The predicate $\mathrm{Gentleman}\,(x)$ is defined as $\mathrm{Gentleman}\,(x) := \mathrm{Male}\,(x) \wedge \mathrm{Adult}\,(x)$.
2. $\mathrm{Lady}\,(x)$: "x is an adult female". The predicate $\mathrm{Lady}\,(x)$ is defined as $\mathrm{Lady}\,(x) := \mathrm{Female}\,(x) \wedge \mathrm{Adult}\,(x)$.
3. $\mathrm{TeenBoy}\,(x)$: "x is a male teenager". The predicate $\mathrm{TeenBoy}\,(x)$ is defined as $\mathrm{TeenBoy}\,(x) :=$

Translating Predicate Formulas to Prolog Script

Suppose we have following derivative predicates:

1. $\mathrm{Gentleman}\,(x)$: "x is an adult male". The predicate $\mathrm{Gentleman}\,(x)$ is defined as $\mathrm{Gentleman}\,(x) := \mathrm{Male}\,(x) \wedge \mathrm{Adult}\,(x)$.
2. $\mathrm{Lady}\,(x)$: "x is an adult female". The predicate $\mathrm{Lady}\,(x)$ is defined as $\mathrm{Lady}\,(x) := \mathrm{Female}\,(x) \wedge \mathrm{Adult}\,(x)$.
3. $\mathrm{TeenBoy}\,(x)$: "x is a male teenager". The predicate $\mathrm{TeenBoy}\,(x)$ is defined as $\mathrm{TeenBoy}\,(x) := \mathrm{Male}\,(x) \wedge \mathrm{Teen}\,(x)$.
4. $\mathrm{TeenGirl}\,(x)$: "x is a female teenager". The predicate $\mathrm{TeenGirl}\,(x)$ is defined as $\mathrm{TeenGirl}\,(x) :=$

Translating Predicate Formulas to Prolog Script

Suppose we have following derivative predicates:

1. $\mathrm{Gentleman}\,(x)$: "x is an adult male". The predicate $\mathrm{Gentleman}\,(x)$ is defined as $\mathrm{Gentleman}\,(x) := \mathrm{Male}\,(x) \land \mathrm{Adult}\,(x)$.
2. $\mathrm{Lady}\,(x)$: "x is an adult female". The predicate $\mathrm{Lady}\,(x)$ is defined as $\mathrm{Lady}\,(x) := \mathrm{Female}\,(x) \land \mathrm{Adult}\,(x)$.
3. $\mathrm{TeenBoy}\,(x)$: "x is a male teenager". The predicate $\mathrm{TeenBoy}\,(x)$ is defined as $\mathrm{TeenBoy}\,(x) := \mathrm{Male}\,(x) \land \mathrm{Teen}\,(x)$.
4. $\mathrm{TeenGirl}\,(x)$: "x is a female teenager". The predicate $\mathrm{TeenGirl}\,(x)$ is defined as $\mathrm{TeenGirl}\,(x) := \mathrm{Female}\,(x) \land \mathrm{Teen}\,(x)$.
5. $\mathrm{LittleBoy}\,(x)$: "x is a male little child". The predicate $\mathrm{LittleBoy}\,(x)$ is defined as $\mathrm{LittleBoy}\,(x) :=$

Translating Predicate Formulas to Prolog Script

Suppose we have following derivative predicates:

1. $\text{Gentleman}(x)$: "x is an adult male". The predicate $\text{Gentleman}(x)$ is defined as $\text{Gentleman}(x) := \text{Male}(x) \wedge \text{Adult}(x)$.
2. $\text{Lady}(x)$: "x is an adult female". The predicate $\text{Lady}(x)$ is defined as $\text{Lady}(x) := \text{Female}(x) \wedge \text{Adult}(x)$.
3. $\text{TeenBoy}(x)$: "x is a male teenager". The predicate $\text{TeenBoy}(x)$ is defined as $\text{TeenBoy}(x) := \text{Male}(x) \wedge \text{Teen}(x)$.
4. $\text{TeenGirl}(x)$: "x is a female teenager". The predicate $\text{TeenGirl}(x)$ is defined as $\text{TeenGirl}(x) := \text{Female}(x) \wedge \text{Teen}(x)$.
5. $\text{LittleBoy}(x)$: "x is a male little child". The predicate $\text{LittleBoy}(x)$ is defined as $\text{LittleBoy}(x) := \text{Male}(x) \wedge \text{Kid}(x)$.
6. $\text{LittleGirl}(x)$: "x is a female little child". The predicate $\text{LittleGirl}(x)$ is defined as $\text{LittleGirl}(x) :=$

Translating Predicate Formulas to Prolog Script

Suppose we have following derivative predicates:

1. $\mathrm{Gentleman}\left(x\right)$: "x is an adult male". The predicate $\mathrm{Gentleman}\left(x\right)$ is defined as $\mathrm{Gentleman}\left(x\right) := \mathrm{Male}\left(x\right) \wedge \mathrm{Adult}\left(x\right)$.
2. $\mathrm{Lady}\left(x\right)$: "x is an adult female". The predicate $\mathrm{Lady}\left(x\right)$ is defined as $\mathrm{Lady}\left(x\right) := \mathrm{Female}\left(x\right) \wedge \mathrm{Adult}\left(x\right)$.
3. $\mathrm{TeenBoy}\left(x\right)$: "x is a male teenager". The predicate $\mathrm{TeenBoy}\left(x\right)$ is defined as $\mathrm{TeenBoy}\left(x\right) := \mathrm{Male}\left(x\right) \wedge \mathrm{Teen}\left(x\right)$.
4. $\mathrm{TeenGirl}\left(x\right)$: "x is a female teenager". The predicate $\mathrm{TeenGirl}\left(x\right)$ is defined as $\mathrm{TeenGirl}\left(x\right) := \mathrm{Female}\left(x\right) \wedge \mathrm{Teen}\left(x\right)$.
5. $\mathrm{LittleBoy}\left(x\right)$: "x is a male little child". The predicate $\mathrm{LittleBoy}\left(x\right)$ is defined as $\mathrm{LittleBoy}\left(x\right) := \mathrm{Male}\left(x\right) \wedge \mathrm{Kid}\left(x\right)$.
6. $\mathrm{LittleGirl}\left(x\right)$: "x is a female little child". The predicate $\mathrm{LittleGirl}\left(x\right)$ is defined as $\mathrm{LittleGirl}\left(x\right) := \mathrm{Female}\left(x\right) \wedge \mathrm{Kid}\left(x\right)$.

Suppose in Prolog all of those six predicates are translated respectively as `gentleman`, `lady`, `teen_boy`, `teen_girl`, `little_boy`, and `little_girl`. These predicates can be defined as *rules* which are derived from the previously existed predicate (i.e., `male`, `female`, `adult`, `teen`, and `kid`). The definition are as follows:

Suppose in Prolog all of those six predicates are translated respectively as `gentleman`, `lady`, `teen_boy`, `teen_girl`, `little_boy`, and `little_girl`. These predicates can be defined as *rules* which are derived from the previously existed predicate (i.e., `male`, `female`, `adult`, `teen`, and `kid`). The definition are as follows:

```
gentleman(X):- male(X),adult(X).
lady(X):- female(X),adult(X).

teen_boy(X):- male(X),teen(X).
teen_girl(X):- female(X),teen(X).

little_boy(X):- male(X),kid(X).
little_girl(X):- female(X),kid(X).
```

The expression `gentleman(X):- male(X),adult(X)` is an example of a *clause* in Prolog.

Clause, *Fact*, and *Rules*

A Prolog script is a collection of one or more *clauses*. A *clause* may be a *fact* – something that is defined directly in the script (such as `male(bob)`) or *rule* – a formula which is derived from one or more facts. Every clause must be ended by a period (`.`).

Rules are of the form
$\langle \texttt{head} \rangle$`:-` $\langle \texttt{t}_1 \rangle, \langle \texttt{t}_2 \rangle, \ldots, \langle \texttt{t}_\texttt{n} \rangle$ where $\texttt{n} \geq 1$. The symbol "," means conjunction ($\wedge$). The symbol "," can be replaced with ";" which means disjunction ($\vee$).

In a clause:

Clause, *Fact*, and *Rules*

A Prolog script is a collection of one or more *clauses*. A *clause* may be a *fact* – something that is defined directly in the script (such as `male(bob)`) or *rule* – a formula which is derived from one or more facts. Every clause must be ended by a period (`.`).

Rules are of the form
$\langle\texttt{head}\rangle$`:-` $\langle\texttt{t}_1\rangle$,$\langle\texttt{t}_2\rangle$,...,$\langle\texttt{t}_n\rangle$ where $\texttt{n} \geq 1$. The symbol "," means conjunction ($\wedge$). The symbol "," can be replaced with ";" which means disjunction ($\vee$).

In a clause:

- $\langle\texttt{head}\rangle$ is called the *head of the clause* (or the *head of the rule*), $\langle\texttt{head}\rangle$ usually defines a new predicate which is derived from the predicate in the facts.

Clause, *Fact*, and *Rules*

A Prolog script is a collection of one or more *clauses*. A *clause* may be a *fact* – something that is defined directly in the script (such as `male(bob)`) or *rule* – a formula which is derived from one or more facts. Every clause must be ended by a period (`.`).

Rules are of the form
$\langle\texttt{head}\rangle\texttt{:-}\ \langle\texttt{t}_1\rangle\texttt{,}\langle\texttt{t}_2\rangle\texttt{,}\ldots\texttt{,}\langle\texttt{t}_\texttt{n}\rangle$ where $\texttt{n} \geq 1$. The symbol "," means conjunction ($\wedge$). The symbol "," can be replaced with ";" which means disjunction ($\vee$).

In a clause:

- $\langle\texttt{head}\rangle$ is called the *head of the clause* (or the *head of the rule*), $\langle\texttt{head}\rangle$ usually defines a new predicate which is derived from the predicate in the facts.
- $\langle\texttt{t}_1\rangle\texttt{,}\langle\texttt{t}_2\rangle\texttt{,}\ldots\texttt{,}\langle\texttt{t}_\texttt{n}\rangle$ is called the *body of the clause* (or the *body of the rule*), body usually specifies the condition that must be met in order for the conclusion, represented by the head, to be satisfied.

Clause, *Fact*, and *Rules*

A Prolog script is a collection of one or more *clauses*. A *clause* may be a *fact* – something that is defined directly in the script (such as `male(bob)`) or *rule* – a formula which is derived from one or more facts. Every clause must be ended by a period (`.`).

Rules are of the form
$\langle\texttt{head}\rangle$`:-` $\langle t_1\rangle, \langle t_2\rangle, \ldots, \langle t_n\rangle$ where $n \geq 1$. The symbol "," means conjunction ($\wedge$). The symbol "," can be replaced with ";" which means disjunction ($\vee$).

In a clause:

- $\langle\texttt{head}\rangle$ is called the *head of the clause* (or the *head of the rule*), $\langle\texttt{head}\rangle$ usually defines a new predicate which is derived from the predicate in the facts.
- $\langle t_1\rangle, \langle t_2\rangle, \ldots, \langle t_n\rangle$ is called the *body of the clause* (or the *body of the rule*), body usually specifies the condition that must be met in order for the conclusion, represented by the head, to be satisfied.
- `:-` is called *the neck of the clause*, this symbol is similar to the *assignment* symbol in an imperative program. Semantically this symbol is read as "*if*".

In every clause usually: every variable is started with capital letter and constant term (object) is started with lowercase letters.

Declarative Semantics in Prolog

Suppose we have a clause:
`⟨P⟩:- ⟨Q⟩,⟨R⟩.`

This clause has following declarative meaning:
"⟨P⟩ is true if ⟨Q⟩ **and** ⟨R⟩ are true", or
"if ⟨Q⟩ **and** ⟨R⟩ is satisfied, then ⟨P⟩ is also satisfied".

Similarly, the clause `⟨P⟩:- ⟨Q⟩;⟨R⟩.` means "if ⟨Q⟩ **or** ⟨R⟩ is satisfied, then ⟨P⟩ is also satisfied".

Example

Declarative Semantics in Prolog

Suppose we have a clause:
`⟨P⟩:- ⟨Q⟩,⟨R⟩.`

This clause has following declarative meaning:
"⟨P⟩ is true if ⟨Q⟩ **and** ⟨R⟩ are true", or
"if ⟨Q⟩ **and** ⟨R⟩ is satisfied, then ⟨P⟩ is also satisfied".

Similarly, the clause `⟨P⟩:- ⟨Q⟩;⟨R⟩.` means "if ⟨Q⟩ **or** ⟨R⟩ is satisfied, then ⟨P⟩ is also satisfied".

Example

We have `gentlemen(X):- male(X),adult(X)`, which equivalent to the predicate formula $\mathrm{Male}\,(x) \wedge \mathrm{Adult}\,(x) \rightarrow \mathrm{Gentleman}\,(x)$. In other words, if x is a male and x is an adult, then x is a *gentleman*.

Example: Simple Translation with Disjunction

Suppose we want to express these things in our previous Prolog script:

1. every male teenager and male child loves FIFA21
2. every female teenager and female child loves *Candy Crush*

We can define following predicate formulas:

1. $\mathrm{LovesFIFA21}\,(x) :=$

Example: Simple Translation with Disjunction

Suppose we want to express these things in our previous Prolog script:

1. every male teenager and male child loves FIFA21
2. every female teenager and female child loves *Candy Crush*

We can define following predicate formulas:

1. $\mathrm{LovesFIFA21}\,(x) := \mathrm{TeenBoy}\,(x) \vee \mathrm{LittleBoy}\,(x)$
2. $\mathrm{LovesCandyCrush}\,(x) :=$

Example: Simple Translation with Disjunction

Suppose we want to express these things in our previous Prolog script:

1. every male teenager and male child loves FIFA21
2. every female teenager and female child loves *Candy Crush*

We can define following predicate formulas:

1. $\mathrm{LovesFIFA21}\,(x) := \mathrm{TeenBoy}\,(x) \vee \mathrm{LittleBoy}\,(x)$
2. $\mathrm{LovesCandyCrush}\,(x) := \mathrm{TeenGirl}\,(x) \vee \mathrm{LittleGirl}\,(x)$

These formulas are translated to Prolog script as:

Example: Simple Translation with Disjunction

Suppose we want to express these things in our previous Prolog script:

1. every male teenager and male child loves FIFA21
2. every female teenager and female child loves *Candy Crush*

We can define following predicate formulas:

1. $\text{LovesFIFA21}(x) := \text{TeenBoy}(x) \vee \text{LittleBoy}(x)$
2. $\text{LovesCandyCrush}(x) := \text{TeenGirl}(x) \vee \text{LittleGirl}(x)$

These formulas are translated to Prolog script as:

```
loves_FIFA21(X):- teen_boy(X); little_boy(X).
loves_CandyCrush(X):- teen_girl(X); little_girl(X).
```

Defining a New Predicate with Term Swapping

The predicate $\text{Parent}(x, y)$ states that "x is a parent of y". Suppose we want to construct a predicate $\text{Child}(x, y)$ which states that "x is a child of y". This predicate can be expressed using the previously defined $\text{Parent}(x, y)$ predicate. Observe that

$$\text{Child}(x, y) \quad \text{means}$$

Defining a New Predicate with Term Swapping

The predicate $\text{Parent}(x, y)$ states that "x is a parent of y". Suppose we want to construct a predicate $\text{Child}(x, y)$ which states that "x is a child of y". This predicate can be expressed using the previously defined $\text{Parent}(x, y)$ predicate. Observe that

$\text{Child}(x, y)$	means	"x is a child of y"
	means	

Defining a New Predicate with Term Swapping

The predicate $\mathrm{Parent}\,(x, y)$ states that "x is a parent of y". Suppose we want to construct a predicate $\mathrm{Child}\,(x, y)$ which states that "x is a child of y". This predicate can be expressed using the previously defined $\mathrm{Parent}\,(x, y)$ predicate. Observe that

$\mathrm{Child}\,(x, y)$	means	"x is a child of y"
	means	"y is a parent of x"
	means	

Defining a New Predicate with Term Swapping

The predicate $\mathrm{Parent}\,(x, y)$ states that "x is a parent of y". Suppose we want to construct a predicate $\mathrm{Child}\,(x, y)$ which states that "x is a child of y". This predicate can be expressed using the previously defined $\mathrm{Parent}\,(x, y)$ predicate. Observe that

$$\begin{aligned} \mathrm{Child}\,(x, y) \quad & \text{means} \quad \text{“}x \text{ is a child of } y\text{”} \\ & \text{means} \quad \text{“}y \text{ is a parent of } x\text{”} \\ & \text{means} \quad \mathrm{Parent}\,(y, x)\,. \end{aligned}$$

Therefore, we can define $\mathrm{Child}\,(x, y) :=$

Defining a New Predicate with Term Swapping

The predicate $\text{Parent}\,(x, y)$ states that "x is a parent of y". Suppose we want to construct a predicate $\text{Child}\,(x, y)$ which states that "x is a child of y". This predicate can be expressed using the previously defined $\text{Parent}\,(x, y)$ predicate. Observe that

$$\begin{array}{lll} \text{Child}\,(x, y) & \text{means} & \text{``}x\text{ is a child of }y\text{''} \\ & \text{means} & \text{``}y\text{ is a parent of }x\text{''} \\ & \text{means} & \text{Parent}\,(y, x)\,. \end{array}$$

Therefore, we can define $\text{Child}\,(x, y) := \text{Parent}\,(y, x)$. In Prolog we express this condition as follows:

Defining a New Predicate with Term Swapping

The predicate $\text{Parent}\,(x,y)$ states that "x is a parent of y". Suppose we want to construct a predicate $\text{Child}\,(x,y)$ which states that "x is a child of y". This predicate can be expressed using the previously defined $\text{Parent}\,(x,y)$ predicate. Observe that

$$\begin{aligned} \text{Child}\,(x,y) \quad & \text{means} \quad \text{“}x \text{ is a child of } y\text{”} \\ & \text{means} \quad \text{“}y \text{ is a parent of } x\text{”} \\ & \text{means} \quad \text{Parent}\,(y,x)\,. \end{aligned}$$

Therefore, we can define $\text{Child}\,(x,y) := \text{Parent}\,(y,x)$. In Prolog we express this condition as follows:

```
child(X,Y):- parent(Y,X).
```

Exercise 1

Exercise

1. Suppose there is a domain D and predicates $\mathrm{Parent}\,(x, y)$: "x is a parent of y", $\mathrm{Male}\,(x)$: "x is a male", and $\mathrm{Female}\,(x)$: "x is a female". Using these predicates alone, write the definition for the predicates $\mathrm{Father}\,(x, y)$ and $\mathrm{Mother}\,(x, y)$. The predicate $\mathrm{Father}\,(x, y)$ means "x is a father of y" and the predicate $\mathrm{Mother}\,(x, y)$ means "x is a mother of y".
2. Use the result in no. 1 to define `father(X,Y)` and `mother(X,Y)` in Prolog.

Solution:

Exercise 1

Exercise

1. Suppose there is a domain D and predicates $\mathrm{Parent}\,(x,y)$: "x is a parent of y", $\mathrm{Male}\,(x)$: "x is a male", and $\mathrm{Female}\,(x)$: "x is a female". Using these predicates alone, write the definition for the predicates $\mathrm{Father}\,(x,y)$ and $\mathrm{Mother}\,(x,y)$. The predicate $\mathrm{Father}\,(x,y)$ means "x is a father of y" and the predicate $\mathrm{Mother}\,(x,y)$ means "x is a mother of y".
2. Use the result in no. 1 to define `father(X,Y)` and `mother(X,Y)` in Prolog.

Solution:

1. $\mathrm{Father}\,(x,y) := \mathrm{Parent}\,(x,y) \wedge \mathrm{Male}\,(x)$ and $\mathrm{Mother}\,(x,y) := \mathrm{Parent}\,(x,y) \wedge \mathrm{Female}\,(x)$. This is because a father is a male parent and a mother is a female parent.

Exercise 1

Exercise

1. Suppose there is a domain D and predicates $\text{Parent}(x,y)$: "x is a parent of y", $\text{Male}(x)$: "x is a male", and $\text{Female}(x)$: "x is a female". Using these predicates alone, write the definition for the predicates $\text{Father}(x,y)$ and $\text{Mother}(x,y)$. The predicate $\text{Father}(x,y)$ means "x is a father of y" and the predicate $\text{Mother}(x,y)$ means "x is a mother of y".
2. Use the result in no. 1 to define `father(X,Y)` and `mother(X,Y)` in Prolog.

Solution:

1. $\text{Father}(x,y) := \text{Parent}(x,y) \wedge \text{Male}(x)$ and $\text{Mother}(x,y) := \text{Parent}(x,y) \wedge \text{Female}(x)$. This is because a father is a male parent and a mother is a female parent.
2. We have `father(X,Y):- parent(X,Y),male(X)` and `mother(X,Y):- parent(X,Y),female(X).`

Contents

1. What is Prolog?
2. Prolog Installation
3. Using the Interactive Interpreter
4. Elementary Prolog Programming: Basic Facts and Queries
5. Simple Rules in Prolog
6. Representation of Quantifiers in Prolog (Supplementary)
7. Term Equality and Inequality in Prolog (Supplementary)

Universal Quantifiers in Prolog

Suppose we have a clause of the form:
$\langle\texttt{head}\rangle$`:-` $\langle\texttt{t}_1\rangle$,$\langle\texttt{t}_2\rangle$,...,$\langle\texttt{t}_\texttt{n}\rangle$ where $\texttt{n} \geq 1$ (operator , which means conjunction can be replaced with ; which means disjunction).
The declarative semantics of the above clause is:
If $\langle\texttt{t}_1\rangle$, $\langle\texttt{t}_2\rangle$, ..., $\langle\texttt{t}_\texttt{n}\rangle$ is true, then $\langle\texttt{head}\rangle$ is also true, or
$\langle\texttt{t}_1\rangle \wedge \langle\texttt{t}_2\rangle \wedge \cdots \wedge \langle\texttt{t}_\texttt{n}\rangle \rightarrow \langle\texttt{head}\rangle$ is T.

Universal Quantifier in Prolog

If a variable appears in the $\langle\texttt{head}\rangle$ of a clause, then this variable is bounded by a universal quantifier ($\forall$).

Example

Universal Quantifiers in Prolog

Suppose we have a clause of the form:
$\langle\texttt{head}\rangle$`:-` $\langle\texttt{t}_1\rangle$,$\langle\texttt{t}_2\rangle$,...,$\langle\texttt{t}_\texttt{n}\rangle$ where $\texttt{n} \geq 1$ (operator , which means conjunction can be replaced with ; which means disjunction).
The declarative semantics of the above clause is:
If $\langle\texttt{t}_1\rangle$, $\langle\texttt{t}_2\rangle$, ..., $\langle\texttt{t}_\texttt{n}\rangle$ is true, then $\langle\texttt{head}\rangle$ is also true, or
$\langle\texttt{t}_1\rangle \wedge \langle\texttt{t}_2\rangle \wedge \cdots \wedge \langle\texttt{t}_\texttt{n}\rangle \rightarrow \langle\texttt{head}\rangle$ is T.

Universal Quantifier in Prolog

If a variable appears in the $\langle\texttt{head}\rangle$ of a clause, then this variable is bounded by a universal quantifier ($\forall$).

Example

We've seen the rule `gentleman(X):- male(X),adult(X)`. In predicate logic, this formula can be expressed as $\mathrm{Male}(x) \wedge \mathrm{Adult}(x) \rightarrow \mathrm{Gentleman}(x)$. Because `X` appears in the $\langle\texttt{head}\rangle$ of a clause, then `X` is bounded by a universal quantifier. This makes the rule `gentleman(X):- male(X),adult(X)` is more accurately expressed as the formula $\forall x\,(\mathrm{Male}(x) \wedge \mathrm{Adult}(x) \rightarrow \mathrm{Gentleman}(x))$.

From the previous derived rule, we have:

- `lady(X):- female(X),adult(X)` is accurately expressed as

From the previous derived rule, we have:

- `lady(X):- female(X),adult(X)` is accurately expressed as
  $\forall x\,(\mathrm{Female}\,(x) \wedge \mathrm{Adult}\,(x) \rightarrow \mathrm{Lady}\,(x))$
- `teen_boy(X):- male(X),teen(X)` is accurately expressed as

From the previous derived rule, we have:

- `lady(X):- female(X),adult(X)` is accurately expressed as
  $\forall x\,(\mathrm{Female}\,(x) \wedge \mathrm{Adult}\,(x) \rightarrow \mathrm{Lady}\,(x))$
- `teen_boy(X):- male(X),teen(X)` is accurately expressed as
  $\forall x\,(\mathrm{Male}\,(x) \wedge \mathrm{Teen}\,(x) \rightarrow \mathrm{TeenBoy}\,(x))$
- `teen_girl(X):- female(X),teen(X)` is accurately expressed as

From the previous derived rule, we have:

- `lady(X):- female(X),adult(X)` is accurately expressed as $\forall x\, (\text{Female}\,(x) \wedge \text{Adult}\,(x) \rightarrow \text{Lady}\,(x))$
- `teen_boy(X):- male(X),teen(X)` is accurately expressed as $\forall x\, (\text{Male}\,(x) \wedge \text{Teen}\,(x) \rightarrow \text{TeenBoy}\,(x))$
- `teen_girl(X):- female(X),teen(X)` is accurately expressed as $\forall x\, (\text{Female}\,(x) \wedge \text{Teen}\,(x) \rightarrow \text{TeenGirl}\,(x))$
- `little_boy(X):- male(X),kid(X)` is accurately expressed as

From the previous derived rule, we have:

- `lady(X):- female(X),adult(X)` is accurately expressed as $\forall x\,(\mathrm{Female}\,(x) \wedge \mathrm{Adult}\,(x) \rightarrow \mathrm{Lady}\,(x))$
- `teen_boy(X):- male(X),teen(X)` is accurately expressed as $\forall x\,(\mathrm{Male}\,(x) \wedge \mathrm{Teen}\,(x) \rightarrow \mathrm{TeenBoy}\,(x))$
- `teen_girl(X):- female(X),teen(X)` is accurately expressed as $\forall x\,(\mathrm{Female}\,(x) \wedge \mathrm{Teen}\,(x) \rightarrow \mathrm{TeenGirl}\,(x))$
- `little_boy(X):- male(X),kid(X)` is accurately expressed as $\forall x\,(\mathrm{Male}\,(x) \wedge \mathrm{Kid}\,(x) \rightarrow \mathrm{LittleBoy}\,(x))$
- `little_girl(X):- female(X),kid(X)` is accurately expressed as

From the previous derived rule, we have:

- `lady(X):- female(X),adult(X)` is accurately expressed as
  $\forall x\,(\text{Female}\,(x) \wedge \text{Adult}\,(x) \rightarrow \text{Lady}\,(x))$
- `teen_boy(X):- male(X),teen(X)` is accurately expressed as
  $\forall x\,(\text{Male}\,(x) \wedge \text{Teen}\,(x) \rightarrow \text{TeenBoy}\,(x))$
- `teen_girl(X):- female(X),teen(X)` is accurately expressed as
  $\forall x\,(\text{Female}\,(x) \wedge \text{Teen}\,(x) \rightarrow \text{TeenGirl}\,(x))$
- `little_boy(X):- male(X),kid(X)` is accurately expressed as
  $\forall x\,(\text{Male}\,(x) \wedge \text{Kid}\,(x) \rightarrow \text{LittleBoy}\,(x))$
- `little_girl(X):- female(X),kid(X)` is accurately expressed as
  $\forall x\,(\text{Female}\,(x) \wedge \text{Kid}\,(x) \rightarrow \text{LittleGirl}\,(x))$
- `loves_FIFA21(X):- teen_boy(X);little_boy(X)` is accurately expressed as

From the previous derived rule, we have:

- `lady(X):- female(X),adult(X)` is accurately expressed as $\forall x\,(\mathrm{Female}\,(x) \wedge \mathrm{Adult}\,(x) \rightarrow \mathrm{Lady}\,(x))$
- `teen_boy(X):- male(X),teen(X)` is accurately expressed as $\forall x\,(\mathrm{Male}\,(x) \wedge \mathrm{Teen}\,(x) \rightarrow \mathrm{TeenBoy}\,(x))$
- `teen_girl(X):- female(X),teen(X)` is accurately expressed as $\forall x\,(\mathrm{Female}\,(x) \wedge \mathrm{Teen}\,(x) \rightarrow \mathrm{TeenGirl}\,(x))$
- `little_boy(X):- male(X),kid(X)` is accurately expressed as $\forall x\,(\mathrm{Male}\,(x) \wedge \mathrm{Kid}\,(x) \rightarrow \mathrm{LittleBoy}\,(x))$
- `little_girl(X):- female(X),kid(X)` is accurately expressed as $\forall x\,(\mathrm{Female}\,(x) \wedge \mathrm{Kid}\,(x) \rightarrow \mathrm{LittleGirl}\,(x))$
- `loves_FIFA21(X):- teen_boy(X);little_boy(X)` is accurately expressed as $\forall x\,(\mathrm{TeenBoy}\,(x) \vee \mathrm{LittleBoy}\,(x) \rightarrow \mathrm{LovesFIFA21}\,(x))$
- `loves_CandyCrush(X):- teen_girl(X);little_girl(X)` is accurately expressed as

From the previous derived rule, we have:

- `lady(X):- female(X),adult(X)` is accurately expressed as $\forall x\,(\mathrm{Female}\,(x) \wedge \mathrm{Adult}\,(x) \rightarrow \mathrm{Lady}\,(x))$
- `teen_boy(X):- male(X),teen(X)` is accurately expressed as $\forall x\,(\mathrm{Male}\,(x) \wedge \mathrm{Teen}\,(x) \rightarrow \mathrm{TeenBoy}\,(x))$
- `teen_girl(X):- female(X),teen(X)` is accurately expressed as $\forall x\,(\mathrm{Female}\,(x) \wedge \mathrm{Teen}\,(x) \rightarrow \mathrm{TeenGirl}\,(x))$
- `little_boy(X):- male(X),kid(X)` is accurately expressed as $\forall x\,(\mathrm{Male}\,(x) \wedge \mathrm{Kid}\,(x) \rightarrow \mathrm{LittleBoy}\,(x))$
- `little_girl(X):- female(X),kid(X)` is accurately expressed as $\forall x\,(\mathrm{Female}\,(x) \wedge \mathrm{Kid}\,(x) \rightarrow \mathrm{LittleGirl}\,(x))$
- `loves_FIFA21(X):- teen_boy(X);little_boy(X)` is accurately expressed as $\forall x\,(\mathrm{TeenBoy}\,(x) \vee \mathrm{LittleBoy}\,(x) \rightarrow \mathrm{LovesFIFA21}\,(x))$
- `loves_CandyCrush(X):- teen_girl(X);little_girl(X)` is accurately expressed as $\forall x\,(\mathrm{TeenGirl}\,(x) \vee \mathrm{LittleGirl}\,(x) \rightarrow \mathrm{LovesCandyCrush}\,(x))$

In addition

- `child(X,Y):- parent(Y,X)` is accurately expressed as

In addition

- `child(X,Y):- parent(Y,X)` is accurately expressed as $\forall x \forall y \left(\text{Parent}\left(y, x\right) \rightarrow \text{Child}\left(x, y\right)\right)$
- `father(X,Y):- parent(X,Y),male(X)` is accurately expressed as

In addition

- `child(X,Y):- parent(Y,X)` is accurately expressed as $\forall x \forall y\, (\text{Parent}\,(y,x) \rightarrow \text{Child}\,(x,y))$
- `father(X,Y):- parent(X,Y),male(X)` is accurately expressed as $\forall x \forall y\, (\text{Parent}\,(x,y) \wedge \text{Male}\,(x) \rightarrow \text{Father}\,(x,y))$
- `mother(X,Y):- parent(X,Y),female(X)` is accurately expressed as

In addition

- `child(X,Y):- parent(Y,X)` is accurately expressed as
  $\forall x \forall y \left(\text{Parent}\left(y,x\right) \rightarrow \text{Child}\left(x,y\right)\right)$
- `father(X,Y):- parent(X,Y),male(X)` is accurately expressed as
  $\forall x \forall y \left(\text{Parent}\left(x,y\right) \land \text{Male}\left(x\right) \rightarrow \text{Father}\left(x,y\right)\right)$
- `mother(X,Y):- parent(X,Y),female(X)` is accurately expressed as
  $\forall x \forall y \left(\text{Parent}\left(x,y\right) \land \text{Female}\left(x\right) \rightarrow \text{Mother}\left(x,y\right)\right)$

Existential Quantifiers in Prolog

Suppose we want to construct predicate $\mathrm{Grandparent}(x, y)$: "x is a grandparent of y". We can define this predicate using the previously defined $\mathrm{Parent}(x, y)$ predicate. Observe that:

$\mathrm{Grandparent}(x, y)$ means

Existential Quantifiers in Prolog

Suppose we want to construct predicate $\mathrm{Grandparent}(x, y)$: "x is a grandparent of y". We can define this predicate using the previously defined $\mathrm{Parent}(x, y)$ predicate. Observe that:

$\mathrm{Grandparent}(x, y)$	means	"x is a *grandparent* of y"
	means	

Existential Quantifiers in Prolog

Suppose we want to construct predicate $\mathrm{Grandparent}(x, y)$: "x is a grandparent of y". We can define this predicate using the previously defined $\mathrm{Parent}(x, y)$ predicate. Observe that:

$\mathrm{Grandparent}(x, y)$	means	"x is a *grandparent* of y"
	means	"**there exists** z such that x is a parent of z **and** z is a parent of y"
	means	

Existential Quantifiers in Prolog

Suppose we want to construct predicate $\mathrm{Grandparent}(x, y)$: "x is a grandparent of y". We can define this predicate using the previously defined $\mathrm{Parent}(x, y)$ predicate. Observe that:

$\mathrm{Grandparent}(x, y)$	means	"x is a *grandparent* of y"
	means	"**there exists** z such that x is a parent of z **and** z is a parent of y"
	means	there is z such that $\mathrm{Parent}(x, z) \wedge \mathrm{Parent}(z, y)$
	means	

Existential Quantifiers in Prolog

Suppose we want to construct predicate $\text{Grandparent}(x, y)$: "x is a grandparent of y". We can define this predicate using the previously defined $\text{Parent}(x, y)$ predicate. Observe that:

$\text{Grandparent}(x, y)$	means	"x is a *grandparent* of y"
	means	"**there exists** z such that x is a parent of z **and** z is a parent of y"
	means	there is z such that $\text{Parent}(x, z) \wedge \text{Parent}(z, y)$
	means	$\exists z\,(\text{Parent}(x, z) \wedge \text{Parent}(z, y))$.

We can translate this expression in Prolog as:

```
grandparent(X,Y):-
```

Existential Quantifiers in Prolog

Suppose we want to construct predicate $\mathrm{Grandparent}(x, y)$: "x is a grandparent of y". We can define this predicate using the previously defined $\mathrm{Parent}(x, y)$ predicate. Observe that:

$\mathrm{Grandparent}(x, y)$	means	"x is a *grandparent* of y"
	means	"**there exists** z such that x is a parent of z **and** z is a parent of y"
	means	there is z such that $\mathrm{Parent}(x, z) \wedge \mathrm{Parent}(z, y)$
	means	$\exists z\,(\mathrm{Parent}(x, z) \wedge \mathrm{Parent}(z, y))$.

We can translate this expression in Prolog as:

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

Existential Quantifier in Prolog

If a variable appears in the *body* of a clause but does not appear in the ⟨`head`⟩ of that clause, then this variable is bounded by an existential quantifier ($\exists$).

As a result, the corresponding predicate formula for the script:

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

is

Existential Quantifier in Prolog

If a variable appears in the *body* of a clause but does not appear in the ⟨`head`⟩ of that clause, then this variable is bounded by an existential quantifier ($\exists$).

As a result, the corresponding predicate formula for the script:

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

is

$$\forall x \forall y \left(\exists z \left(\text{Parent}\left(x,z\right) \wedge \text{Parent}\left(z,y\right)\right) \rightarrow \text{Grandparent}\left(x,y\right)\right) \text{ or}$$
$$\forall x \forall y \exists z \left(\text{Parent}\left(x,z\right) \wedge \text{Parent}\left(z,y\right) \rightarrow \text{Grandparent}\left(x,y\right)\right).$$

Using the facts from our previous program, the query result for `grandparent(Grandparent,Grandkid)` is:

```
?- grandparent(Grandparent,Grandkid).
```

Using the facts from our previous program, the query result for `grandparent(Grandparent,Grandkid)` is:

```
?- grandparent(Grandparent,Grandkid).
Grandparent = alice,
Grandkid = fiona ;
```

Using the facts from our previous program, the query result for `grandparent(Grandparent,Grandkid)` is:

```
?- grandparent(Grandparent,Grandkid).
Grandparent = alice,
Grandkid = fiona ;
Grandparent = alice,
Grandkid = grace ;
```

⋮ [several other outputs]

Using the facts from our previous program, the query result for `grandparent(Grandparent,Grandkid)` is:

```
?- grandparent(Grandparent,Grandkid).
Grandparent = alice,
Grandkid = fiona ;
Grandparent = alice,
Grandkid = grace ;

⋮ [several other outputs]

Grandparent = charlie,
Grandkid = kelly;
```

Singleton Variable

Suppose we want to construct a predicate $\text{Has-a-Child}(x)$: "x has a child". This predicate is true if there is a y such that $\text{Parent}\,(x, y)$ is true. In other words, x has a child if there is a y such that x is the parent of y. Therefore, $\text{Has-a-Child}(x)$ can be expressed as

$$\text{Has-a-Child}\,(x) :=$$

Singleton Variable

Suppose we want to construct a predicate $\text{Has-a-Child}(x)$: "x has a child". This predicate is true if there is a y such that $\text{Parent}\left(x,y\right)$ is true. In other words, x has a child if there is a y such that x is the parent of y. Therefore, $\text{Has-a-Child}(x)$ can be expressed as

$$\text{Has-a-Child}\left(x\right) := \exists y \text{Parent}\left(x,y\right).$$

Or in an implication from

Singleton Variable

Suppose we want to construct a predicate $\text{Has-a-Child}(x)$: "x has a child". This predicate is true if there is a y such that $\text{Parent}\,(x, y)$ is true. In other words, x has a child if there is a y such that x is the parent of y. Therefore, $\text{Has-a-Child}(x)$ can be expressed as

$$\text{Has-a-Child}\,(x) := \exists y \text{Parent}\,(x, y)\,.$$

Or in an implication from $\forall x\,(\exists y\,(\text{Parent}\,(x, y) \rightarrow \text{Has-a-Child}\,(x)))$. If this expression is translated directly to Prolog, then we have following script:

Singleton Variable

Suppose we want to construct a predicate $\text{Has-a-Child}(x)$: "x has a child". This predicate is true if there is a y such that $\text{Parent}\,(x, y)$ is true. In other words, x has a child if there is a y such that x is the parent of y. Therefore, $\text{Has-a-Child}(x)$ can be expressed as

$$\text{Has-a-Child}\,(x) := \exists y \text{Parent}\,(x, y)\,.$$

Or in an implication from $\forall x\,(\exists y\,(\text{Parent}\,(x, y) \rightarrow \text{Has-a-Child}\,(x)))$. If this expression is translated directly to Prolog, then we have following script:

```
has_a_child(X):- parent(X,Y).
```

The above script cannot be compiled by SWI-Prolog, SWI-Prolog produce a warning: `Singleton variables: [Y]`.

About Singleton Variable

A singleton variable can be a variable which appears in the *body* of a clause of a binary predicate (or any n-ary predicate with $n \geq 2$) , but does not appear in the $\langle$`head`$\rangle$ of the clause.

We can fix this singleton variable problem by performing one of these following procedure:

1. Add an "_" (*underscore*) in front of the singleton variable. For example, the script of `has_a_child(X)` becomes:

About Singleton Variable

A singleton variable can be a variable which appears in the *body* of a clause of a binary predicate (or any n-ary predicate with $n \geq 2$) , but does not appear in the $\langle$`head`$\rangle$ of the clause.

We can fix this singleton variable problem by performing one of these following procedure:

1. Add an "_" (*underscore*) in front of the singleton variable. For example, the script of `has_a_child(X)` becomes:

```
has_a_child(X):- parent(X,_Y).
```

2. Add an additional expression concerning the singleton variable which does not change the declarative semantics of the clause formed. Because the domain of `Y` is the set of all humans and `Y` is either `male(Y)` or `female(Y)`, then the script for `has_a_child(X)` becomes:

About Singleton Variable

A singleton variable can be a variable which appears in the *body* of a clause of a binary predicate (or any n-ary predicate with $n \geq 2$) , but does not appear in the ⟨head⟩ of the clause.

We can fix this singleton variable problem by performing one of these following procedure:

1. Add an "_" (*underscore*) in front of the singleton variable. For example, the script of `has_a_child(X)` becomes:

```
has_a_child(X):- parent(X,_Y).
```

2. Add an additional expression concerning the singleton variable which does not change the declarative semantics of the clause formed. Because the domain of `Y` is the set of all humans and `Y` is either `male(Y)` or `female(Y)`, then the script for `has_a_child(X)` becomes:

```
has_a_child(X):- parent(X,Y),(male(Y);female(Y)).
```

This script is represented in predicate formula as:

About Singleton Variable

A singleton variable can be a variable which appears in the *body* of a clause of a binary predicate (or any n-ary predicate with $n \geq 2$) , but does not appear in the ⟨head⟩ of the clause.

We can fix this singleton variable problem by performing one of these following procedure:

1. Add an "_" (*underscore*) in front of the singleton variable. For example, the script of `has_a_child(X)` becomes:

```
has_a_child(X):- parent(X,_Y).
```

2. Add an additional expression concerning the singleton variable which does not change the declarative semantics of the clause formed. Because the domain of `Y` is the set of all humans and `Y` is either `male(Y)` or `female(Y)`, then the script for `has_a_child(X)` becomes:

```
has_a_child(X):- parent(X,Y),(male(Y);female(Y)).
```

This script is represented in predicate formula as:
$\forall x\, (\exists y \mathrm{Parent}\,(x, y) \wedge (\mathrm{Male}\,(y) \vee \mathrm{Female}\,(y)) \rightarrow \text{Has-a-Child}\,(x))$.

Exercise 2

Exercise

1. Suppose there is a domain D and predicates $\mathrm{Father}\,(x,y)$: "x is a father of y" and $\mathrm{Mother}\,(x,y)$: "x is a mother of y". Using these predicates alone, write the definition for the predicates $\text{Is-a-Daddy}(x)$ and $\text{Is-a-Mommy}(x)$. The predicate $\text{Is-a-Daddy}(x)$ means "x is a daddy (a father)" and the predicate $\text{Is-a-Mommy}(x)$ means "x is a mommy (a mother)".
2. Use the result in no. 1 to define `is_a_daddy(X)` and `is_a_mommy(X)` in Prolog.

Solution:

Solution:

1. $\text{Is-a-Daddy}(x) := \exists y \text{Father}\,(x, y)$ and $\text{Is-a-Mommy}(x) := \exists y \text{Mother}\,(x, y)$.

Solution:

1. $\text{Is-a-Daddy}(x) := \exists y \text{Father}(x, y)$ and $\text{Is-a-Mommy}(x) := \exists y \text{Mother}(x, y)$.
2. We have
   `is_a_daddy(X):- father(X,_Y)` and
   `is_a_mommy(X):- mother(X,_Y)`.
   The script can also be expressed as:

Solution:

1. $\text{Is-a-Daddy}(x) := \exists y \text{Father}\,(x, y)$ and $\text{Is-a-Mommy}(x) := \exists y \text{Mother}\,(x, y)$.
2. We have
   `is_a_daddy(X):- father(X,_Y)` and
   `is_a_mommy(X):- mother(X,_Y).`
   The script can also be expressed as:
   `is_a_daddy(X):- father(X,Y),(male(Y);female(Y))` and
   `is_a_mommy(X):- mother(X,Y),(male(Y);female(Y)).`

Contents

1 What is Prolog?

2 Prolog Installation

3 Using the Interactive Interpreter

4 Elementary Prolog Programming: Basic Facts and Queries

5 Simple Rules in Prolog

6 Representation of Quantifiers in Prolog (Supplementary)

7 Term Equality and Inequality in Prolog (Supplementary)

Term Equality and Inequality in Prolog

Suppose t_1 and t_2 are two terms in Prolog, we can inspect the equality and inequality for t_1 and t_2 using the operators == or = (for equality) and \== or \= (for inequality).

Suppose the predicate $\text{Sibling}\,(x, y)$ means "x is a sibling (brother/ sister) of y". We can define this predicate using the previously defined $\text{Parent}(z, y)$ predicate as follows:

$\text{Sibling}(x, y)$ means

Term Equality and Inequality in Prolog

Suppose t_1 and t_2 are two terms in Prolog, we can inspect the equality and inequality for t_1 and t_2 using the operators == or = (for equality) and \== or \= (for inequality).

Suppose the predicate $\mathrm{Sibling}\,(x, y)$ means "x is a sibling (brother/ sister) of y". We can define this predicate using the previously defined $\mathrm{Parent}(z, y)$ predicate as follows:

$\mathrm{Sibling}(x, y)$ means "x is a sibling of y"
means

Term Equality and Inequality in Prolog

Suppose $\texttt{t}_1$ and $\texttt{t}_2$ are two terms in Prolog, we can inspect the equality and inequality for $\texttt{t}_1$ and $\texttt{t}_2$ using the operators == or = (for equality) and \== or \= (for inequality).

Suppose the predicate $\mathrm{Sibling}\,(x, y)$ means "x is a sibling (brother/ sister) of y". We can define this predicate using the previously defined $\mathrm{Parent}(z, y)$ predicate as follows:

$\mathrm{Sibling}(x, y)$	means	"x is a sibling of y"
	means	"x and y are different people who have common parents"
	means	

Term Equality and Inequality in Prolog

Suppose $\texttt{t}_1$ and $\texttt{t}_2$ are two terms in Prolog, we can inspect the equality and inequality for $\texttt{t}_1$ and $\texttt{t}_2$ using the operators == or = (for equality) and \== or \= (for inequality).

Suppose the predicate $\text{Sibling}\,(x, y)$ means "x is a sibling (brother/ sister) of y". We can define this predicate using the previously defined $\text{Parent}(z, y)$ predicate as follows:

$\text{Sibling}(x, y)$	means	"x is a sibling of y"
	means	"x and y are different people who have common parents"
	means	**there exists** z such that $\text{Parent}(z, x)$ **and** $\text{Parent}(z, y)$ **and** $x \neq y$
	means	

Term Equality and Inequality in Prolog

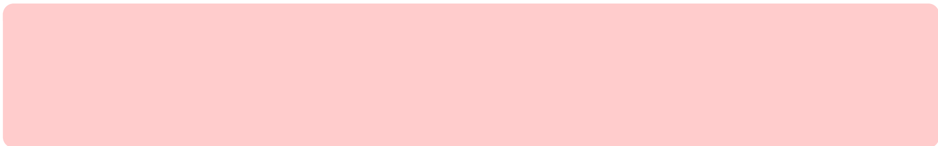
Suppose `t1` and `t2` are two terms in Prolog, we can inspect the equality and inequality for `t1` and `t2` using the operators == or = (for equality) and \== or \= (for inequality).

Suppose the predicate $\text{Sibling}\,(x, y)$ means "x is a sibling (brother/ sister) of y". We can define this predicate using the previously defined $\text{Parent}(z, y)$ predicate as follows:

$\text{Sibling}(x, y)$	means	"x is a sibling of y"
	means	"x and y are different people who have common parents"
	means	**there exists** z such that $\text{Parent}(z, x)$ **and** $\text{Parent}(z, y)$ **and** $x \neq y$
	means	$\exists z\,(\text{Parent}\,(z, x) \wedge \text{Parent}\,(z, y)) \wedge (x \neq y)$.

In predicate logic we have
$\text{Sibling}(x, y) := \exists z\,(\text{Parent}\,(z, x) \wedge \text{Parent}\,(z, y)) \wedge (x \neq y)$, which gives us the formula
$\forall x \forall y\,(\exists z\,(\text{Parent}\,(z, x) \wedge \text{Parent}\,(z, y)) \wedge (x \neq y) \rightarrow \text{Sibling}\,(x, y))$.

In Prolog, the predicate formula $\forall x \forall y \left(\exists z \left(\text{Parent}\left(z,x\right) \wedge \text{Parent}\left(z,y\right)\right) \wedge \left(x \neq y\right) \to \text{Sibling}\left(x,y\right)\right)$ can be expressed using following rule:



In Prolog, the predicate formula $\forall x \forall y\, (\exists z\, (\mathrm{Parent}\,(z,x) \wedge \mathrm{Parent}\,(z,y)) \wedge (x \neq y) \rightarrow \mathrm{Sibling}\,(x,y))$ can be expressed using following rule:

```
sibling(X,Y):-
            parent(Z,X), % Z is a parent of X
            parent(Z,Y), % Z is a parent of Y
            X \== Y. % X and Y are different people
```

The above script is an another writing of the following expression:

```
sibling(X,Y):- parent(Z,X),parent(Z,Y),X \== Y.
```