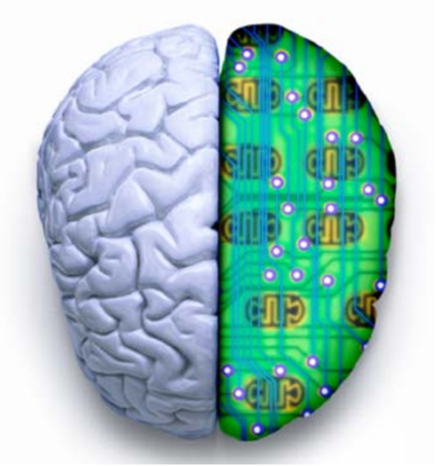
Advanced Artificial Intelligence CM4107 (Week 2)



Logic, Reasoning and Inferring

Dr. Yann Savoye
School of Computing Science and Digital Media
Robert Gordon University

Module Information

- **Assessment:**

  - Coursework (2 components / not related)
    - Component 1: literature review
    - Component 2: paper implementation

  - No mid-term or final written exam.

    All deadlines are strong:
    - *It will not be possible to upload material after the deadline.*
    - *No deadline extension will be granted. No excuse.*
    - *Only the content submitted via the Moodle will be mark.*

Coursework

- **Submission of the Coursework Part 1:**

  Deadline - Monday, October 30th, 2017 23:00:

  Activity 1 and Activity 2

  - *2-pages written reports (in PDF format)*
  - *7 slides presentation (in PDF format)*

Coursework

- **Submission of the Coursework Part 2:**

Deadline - Monday, December 11th, 2017 23:00:

Activity 3 and Activity 4

- *Prolog Programming (code in Prolog)*
- *Paper Implementation (code Java or C++)*

Activity 3 and Activity 4 are not related to the first coursework Part 2,
We will provide you the coursework 2 after the submission of the coursework 1.
The paper to implement for Activity 4 will be imposed.

Overview

- Part I – Logic and AI
- Part II – Propositional Logic
- Part III – First and Second Order Logic
- Part IV – Reasoning
- Part V – Logic Programming with Prolog
- Part IV – Fuzzy Logic

Part I – Logic and AI

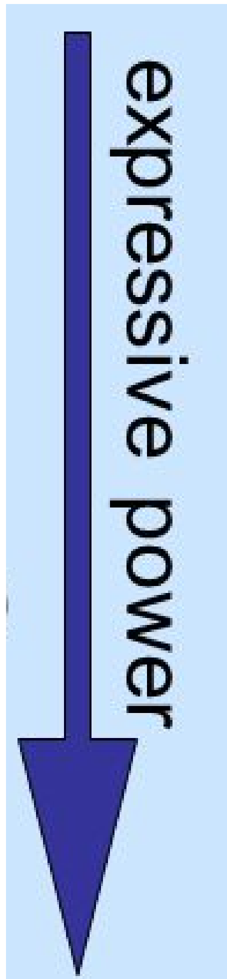
Foundation of the Logical Approach

- **Logic** originally meaning "what is spoken"
- Formalizing **natural languages** and **grammars**.
- Logic is **very important for AI** to know facts about the world.
- **Logical theory:** Logics are formal languages
- **Chomsky** = logic as **deductive techniques**
- **Representing** and **manipulating** knowledge
- **Syntax** and **semantics** for representing knowledge
- **Proof theory** for generating proofs.
- The **theory of inference**
- **An inference** is not true or false, but **valid** or **invalid**.

Language and Logic

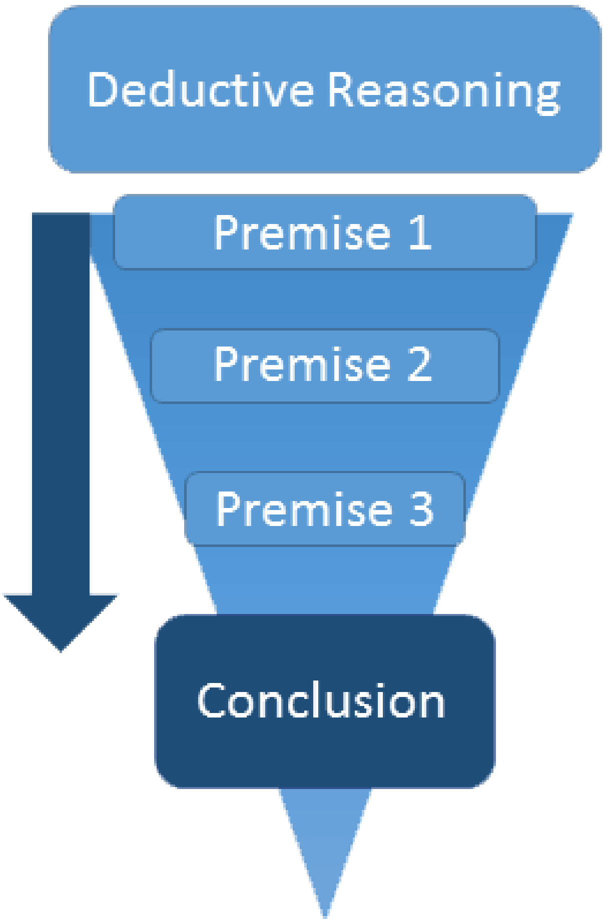
- Functions of **Language**
- **Formal patterns** of correct reasoning
- The **informative use** of language
- An **expressive use** of language,
- Literal and Emotive **Meaning**
- Informative and **partially expressive** uses of language.
- **Assessing the validity** of deductive arguments
- **Assessing the reliability** of inductive reasoning

Order for Expressivity



- **Propositional Logic**
- **Modal Logic**
- **Description Logic**
- **First-Order Logic**.
- **Second-Order Logic.**
- **Higher-Order Logic**.

Deductive Reasoning



- **Deductive reasoning**
- To **understand the world** around you.
- In math, "If **A = B** and **B = C**, then **A = C**".
- A=B and B=C are **the premises**.
- A=C is **the conclusion**.
- **Other Patterns exist**.

Part II – Propositional Logic

Symbolisation

- A **capital letter** to symbolize a simple sentence / statement.
- **Simple sentences** are relatively short
- "**T**" and "**F**" are two capital letters, but T (**true**) and F (**false**).
- **Without quotation** marks

S : Snow is white.
B: The sky is blue.
Q : Nancy will go to the party.

- Simple sentences do **not contain any other sentence**.

For instance , "Nancy will ***not*** *go to the party" is not a simple statement.*

Propositional Logic

```
if (a < b || (a >= b && c == d)) ...
```

- **Propositional logic** = propositions and statements
- Represents **facts** as being either **true or false**
- **Natural language** framed by **the truth-functionals.**
- **Truth-functional logic** = **logical operators** and **connectives**
- **Artificial languages** :logical properties of natural language
- Use the **truth table** to determine the validity.
- **Propositional formulas** are built using **atoms** and **connectives.**

Syntax of Propositional Logic

- An atomic proposition ϕ is a well-formed formula.
- If ϕ is a well-formed formula, then so is $\neg\phi$.
- If ϕ and ψ are well-formed formulas, then so are $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \rightarrow \psi$, and $\phi \Leftrightarrow \psi$.
- If ϕ is a well-formed formula, then so is (ϕ).

Alternatively, can use *Backus-Naur Form (BNF)*:

Countable alphabet Σ of **atomic propositions**: $a, b, c, \ldots$.

Propositional formulas:

$$
\begin{array}{rclll}
\phi, \psi & \longrightarrow & a & & \textit{atomic propositions} \\
& | & \bot & & \textit{false} \\
& | & \top & & \textit{true} \\
& | & \neg\phi & & \textit{negation} \\
& | & \phi \wedge \psi & & \textit{conjunction} \\
& | & \phi \vee \psi & & \textit{disjunction} \\
& | & \phi \rightarrow \psi & & \textit{implication} \\
& | & \phi \leftrightarrow \psi & & \textit{equivalence}
\end{array}
$$

Operator and Connectives

- **Boolean Algebra / Logical connectives**
- **Unary Operators** : "not” (negation)
- **Binary Operators** : are "and" (conjunction), "or" (disjunction).

Name	**Represented**	**Meaning**
Negation	$\neg p$	“not p”
Conjunction	$p \wedge q$	“p and q”
Disjunction	$p \vee q$	“p or q (or both)”
Exclusive Or	$p \oplus q$	“either p or q, but not both”
Implication	$p \rightarrow q$	“if p then q”
Biconditional	$p \leftrightarrow q$	“p if and only if q”

Statements and Truth Tables

- **Implication** / Conditional statement: “if p then q”

$$p \longrightarrow q.$$

Where p is the **hypothesis**

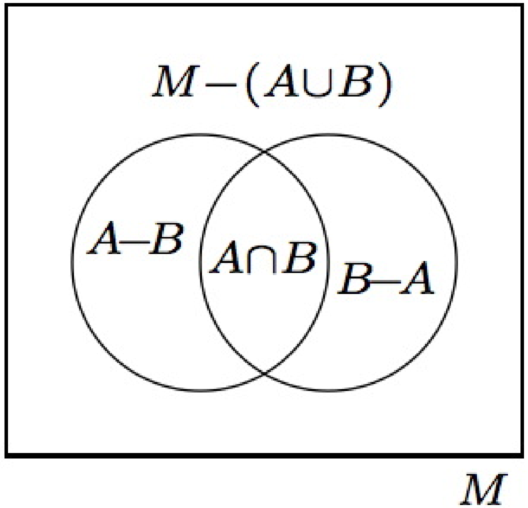
q is the **conclusion**

- A **truth table** is a handy **little logical device**
- Truth tables are used to show **logically validity.**
- A truth table has **one column for each input variable**, and
- One **final column** showing **all possible results**
- **Each row** contains **one possible configuration.**

Truth Tables

T: true

F: False



ϕ	$\neg\phi$
T	F
F	T

negation (not)

ϕ	ψ	$\phi \wedge \psi$
T	T	T
T	F	F
F	T	F
F	F	F

conjunction (and)

ϕ	ψ	$\phi \vee \psi$
T	T	T
T	F	T
F	T	T
F	F	F

disjunction (inclusive or)

ϕ	ψ	$\phi \rightarrow \psi$
T	T	T
T	F	F
F	T	T
F	F	T

implication (if-then)

Logical Equivalence

- **Logical equivalence** is established using **truth tables**
- Every model of the axiom set is a model of the formula.
- **"if and only if"** and is symbolized by a double-lined

$$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$$

p	q	$p \rightarrow q$	$q \rightarrow p$	$(p \rightarrow q) \wedge (q \rightarrow p)$	$p \leftrightarrow q$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	T	F	F	F
F	F	T	T	T	T

identical

Tautology and Contradiction

Contradiction = A statement that is always false

Tautology = A statement that is always true.

p	$\neg p$	$p \vee \neg p$	$p \wedge \neg p$
T	F	T	F
T	F	T	F
F	T	T	F
F	T	T	F

tautology contradiction

Laws of Logic

The ***complement of the union of two sets*** *is equal to* ***the intersection of their complements***

the ***complement of the intersection of two sets*** *is equal to* ***the union of their complements.***

DeMorgan's Laws

$$\neg(p \vee q) \equiv \neg p \wedge \neg q$$

$$\neg(p \wedge q) \equiv \neg p \vee \neg q$$

$\sim(\sim p) \equiv p$ Law of Double Negation

$p \rightarrow q \equiv \sim p \vee q$ Law of Implication

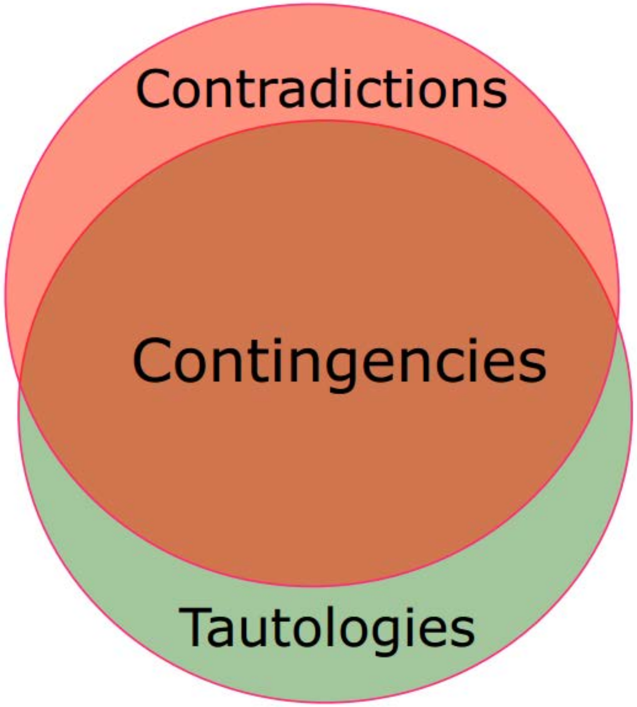
$p \rightarrow q \equiv \sim q \rightarrow \sim p$ Law of Contraposition

For every statement p, either p is true or p is false. Law of Excluded Middle

Satisfiability

A propositional formula is:

- a **tautology** if its truth value is 1 under any valuation.
- **satisfiable** if its truth value is 1 under some valuation.
- **contingent** if its truth value is 1 under some valuation and 0 under another valuation.
- **refutable** if its truth value is 0 under some valuation.



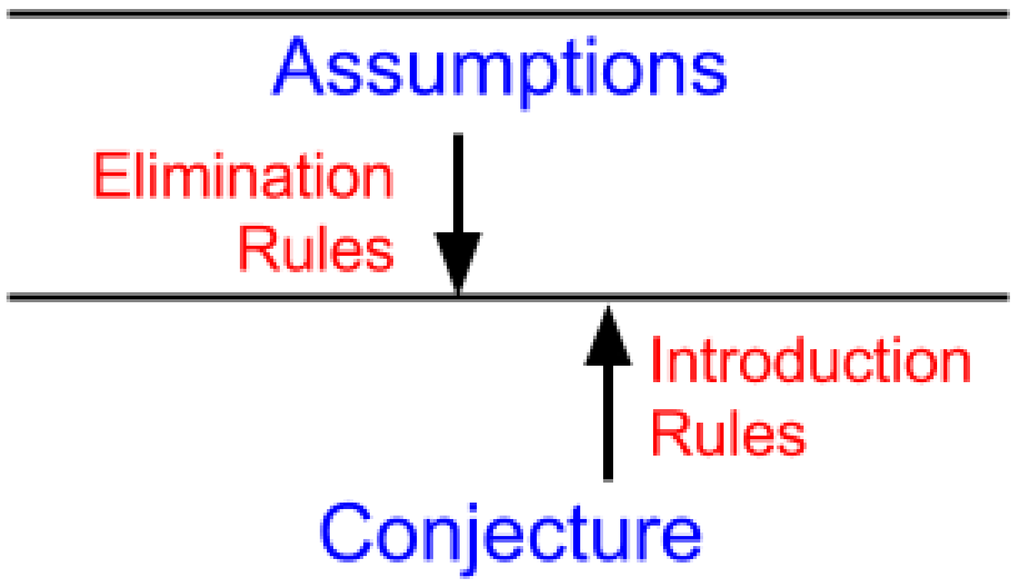
…… This can be checked with a truth table.

Horn Clauses

- Logical formula in **a rule-like form**.
- A **disjunction of literals**.
- Tailored for **resolution**.

$$A_1 \wedge A_2 \wedge \cdots \wedge A_n \rightarrow B$$

Proof Theory: Natural Deduction



- **Natural deduction**: logical reasoning is expressed by **inference rules**.
- **Deduction from premises** by applying **inference rules repeatedly.**
- The **rules** come from **“introduction”** and **“elimination”**
- **Introduction of a conjunction** into a proof
- **Elimination** in favor of simpler formulas

Arguments and Laws

*An **argument** is a collection of **ordered Statements** from **premises** to **conclusion***

Premise #1: $p \rightarrow q$
Premise #2: p
Conclusion: q

Law of Detachment or *Modus Ponens*	Premise #1: $p \rightarrow q$ Premise #2: p Conclusion: q
Law of Transitivity or Syllogism	Premise #1: $p \rightarrow q$ Premise #2: $q \rightarrow r$ Conclusion: $p \rightarrow r$
Law of Disjunctive Syllogism	Premise #1: $p \vee q$ Premise #2: $\sim p$ Conclusion: q
Law of Contraposition or *Modus Tollens*	Premise #1: $p \rightarrow q$ Premise #2: $\sim q$ Conclusion: $\sim p$

Inference Rules

- Represent the environment in a **knowledge base**
- **Deductive inference**. 'If A then B. A is true'
- Rules of inference: **introduction** and **elimination**

Rule of Inference	Description
Modus Ponens or Implication Elimination	If we know P, and $P \Rightarrow Q$, then we can infer Q.
And Elimination	If we know $P \wedge Q$, then we can infer both P, and Q.
And Introduction	If we know P, and we know Q, then we can infer $P \wedge Q$.
Or Introduction	If we know P, then for any Q, we can infer $P \vee Q$.
Double Negation Elimination	We can substitute P for $\neg\neg P$ and vice versa.
Universal Elimination	e.g. If we know $\forall x\, R(x) \Rightarrow S(x)$, we can infer $S(T)$ if $R(T)$ is true.
Existential Elimination	e.g. If we know $\exists x\, R(x) \wedge S(x)$, we can infer both $R(T1)$ and $S(T1)$ as long as $T1$ is not already in the knowledge base.
Existential Introduction	e.g. If we know $R(S,T)$, we can infer $\exists x\, R(x,T)$

Part III – First and Second Order Logic

First-Order Logic

- **First-Order Logic = object, properties, function.**
- **Predicate logic** = may contain **quantifiers**
- **Quantifies** only variables that **range over individuals**

- $\exists z$: **Existential Quantification** ("it exists at least one")
- $\forall x.$: **Universal Quatification** ("for all ")

- *"Every elephant is gray"*: $\forall$ x (elephant(x) $\rightarrow$ gray(x))
- *"There is a white alligator"*: $\exists$ x (alligator(X) ^ white(X))

Second-Order Logic

- **Second-order logic** is an extension of first-order logic (which itself is an extension of propositional logic).
- **Second-order logic** also includes **quantification over functions**, and other variables
- **Second-order logic** is **more expressive** than first-order logic.

$$\forall P \forall x (x \in P \vee x \notin P).$$

Part V – Reasoning

Proof

- A **statement** is not accepted as **valid** or correct unless it is **accompanied by a proof**.
- A **proof** is an **argument from hypotheses** (assumptions) to a conclusion.
- **Proof by Contradiction**: find at least one example to show it is false.
- **Proof by Induction**: base case, step case, recursion.
- **Proof by resolution**: by applying inference rules.
- **Construct proofs** using various **rules of inference** (from p and q I can validly infer r)
- Replacement with **logically equivalent propositions**.

Resolution

- **Resolution:** The derivation of new statements.
- The logic programming system **combines existing statements** to find new statements. For instance,

$$\begin{array}{l} C \text{ :- } A, B \\ D \text{ :- } C \\ \hline D \text{ :- } A, B \end{array}$$

- Resolution**: Enumerating all possible values** is not very elegant
- A **better inference** search **involves using resolution**
- **Inference with Resolution:** Done using proof by contradiction
- Look for **resolutions in every pair of disjunctions**

Part VI – Logic Programming with Prolog

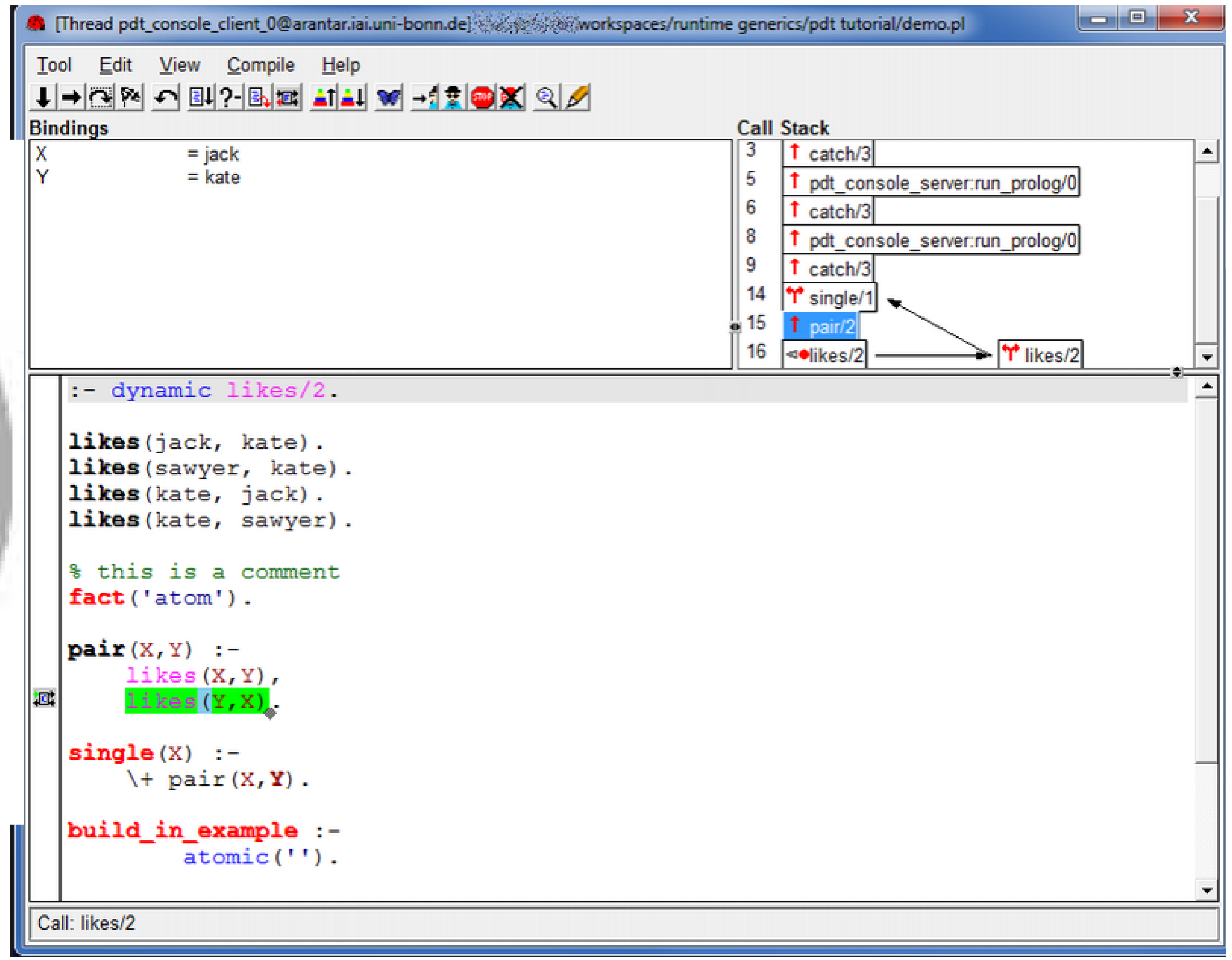
Logic Programming

- **Logic programming** is a form of **declarative programming**
- Any program written in a **logic programming language** is a **set of sentences** in **logical form**, expressing facts and rules.
- Rules are written as logical clauses.

- This category lists few programming languages that support the **logical programming paradigm**:

- *Lisp*
- *Alice (programming language)*
- *Datalog*
- *Prolog, SWI-Prolog*

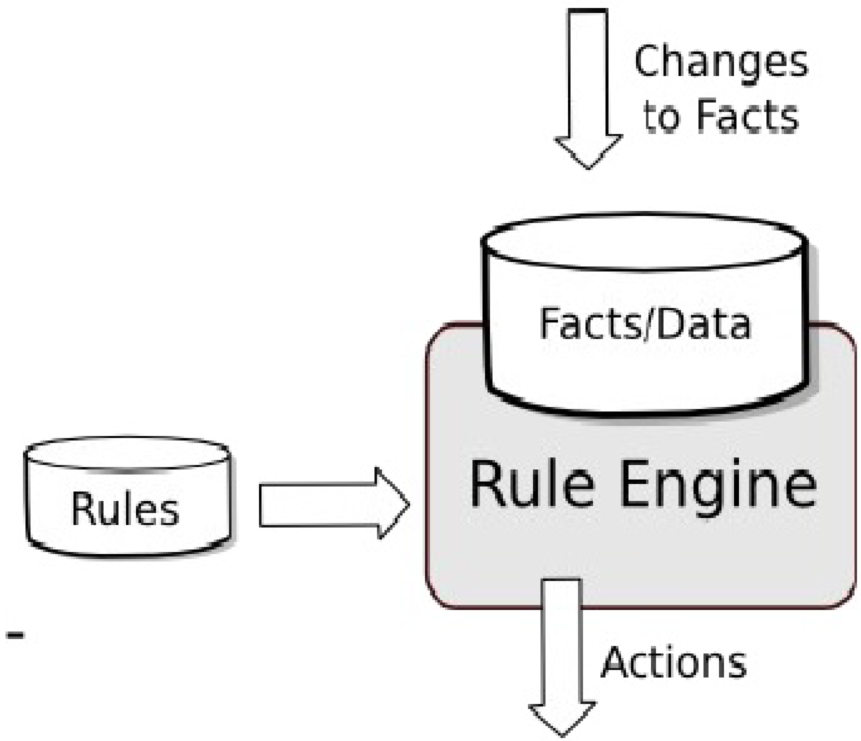
Prolog


[Thread pdt_console_client_0@arantar.iai.uni-bonn.de] workspaces/runtime generics/pdt tutorial/demo.pl
Tool Edit View Compile Help
Bindings
X = jack
Y = kate
Call Stack
3 catch/3
5 pdt_console_server:run_prolog/0
6 catch/3
8 pdt_console_server:run_prolog/0
9 catch/3
14 single/1
15 pair/2
16 likes/2
likes/2
:- dynamic likes/2.
likes(jack, kate).
likes(sawyer, kate).
likes(kate, jack).
likes(kate, sawyer).
% this is a comment
fact('atom').
pair(X,Y) :-
likes(X,Y),
likes(Y,X).
single(X) :-
\+ pair(X,Y).
build_in_example :-
atomic('').
Call: likes/2

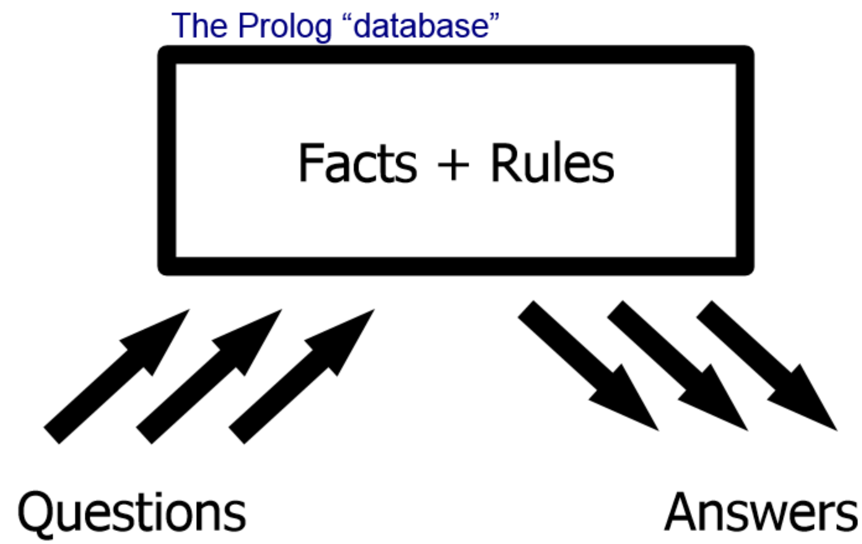
Prolog Programming

- **Prolog** = Programming in Logic.
- Prolog is a **declarative programming language**
- Originated at the University of Marseille (**Alain Colmerauer**).
- Popular **efficient implementations** from **Edinburgh University.**
- **Prolog** = computation as controlled logical inference.
- **Prolog** provides a mechanism **for symbolic reasoning.**
- **A Prolog program** consists of **a set of clauses**
- **Algorithm = logic + control**
- Use **in Artificial Intelligence** for manipulation of symbols and inference
- Prolog consists of a **series of rules and facts.**
- A **program** is run by presenting **some query.**

The basic idea of Prolog



Prolog Answers Questions



Prolog provides a knowledge base (``database'') of facts and rules fairly directly, together with one kind of inference mechanism.

Applications of Prolog

- Expert systems
- Automated reasoning
- Problem solving
- Specification language
- Machine learning
- Intelligent data base retrieval
- Robot planning
- Natural language processing

List

- A recursive definition of the **list data structure** as found in Prolog.

List --> [ ]

List --> [Head|List]

- The list is **empty (nil)** or it may be a term that has a **head** and a tail.
- The **head** may be any term or atom.
- The **tail** is another list.
- **List Constructor** : we use the **square brackets** [ ], and the symbol **|** acts as an operator to construct a list.

?- X = [1 | [2, 3]].

X = [1, 2, 3].

Facts

- **Facts** = **Properties** or **relationships between objects**;
- **Facts** consist of a particular item or a relation between items.
- **Example of simple facts** : ``It is sunny.'' or ``It is summer.‘
- In Prolog we can make **statements by using facts.**
- **Facts** form **Prolog's database.**

- In Prolog, simple facts could be represented as follows:

sunny.

- Or facts with Arguments

likes(john,mary).

This fact may be read as either john likes mary or mary likes john.

Rules

- **Rules** extend the **capabilities of a logic program.**
- A **Prolog rule** has the form: **Head(Arg1, ..., ArgN) :- B, C , ... D.**
- **":-"** means **"if"** or **"is implied by".** Also called the **neck symbol**.
- The **left hand side** of the neck is called **the head.**
- The **right hand side** of the neck is called **the body** or **the goal**
- B, C , ... D are the **premises.**
- The comma, **","**, separating the goals, **stands for "and".**
- The **notation of the head** of a rule depends on **the number of arguments.**

- We can express 'All men are mortal' as the following Prolog rule:

mortal(X) :- human(X).

Facts and Rules

No space

argument/Objects

Facts

```
likes(john, jane).
likes(jane, john).
likes(jack, jane).
```

Dot necessary to end statements

Predicate name

variables

Rule

```
friends(X,Y) :- likes(X,Y) , likes (Y, X).
```

head

body

Predicates

- A **Prolog program** is **a set of predicates.**
- A **predicate** is defined by a **collection of clauses.**
- A **clause** is either a **rule or a fact.**
- If **any clause is true**, then the whole **predicate is true**
- **Predicates define** relations between their **arguments.**
- Each **predicate** has a **name**, and **zero or more arguments.**
- Every **predicate** has an **associated most general query,**
- Some **predicates** are already predefined / **built-in.**
- A **goal** denotes a **predicate and its arguments.**

Function and Recursion

- **Programs** consist of **procedure definitions**
- Prolog does **not provide for a function type**
- Therefore, **functions must be defined as relations**.

- Logic programming definition of the Euclidian algorithm
- Note that the definition requires **two rules**, one for the **base case** and one for the **inductive case**.

```
gcd(X,0,X) :- X > 0.
gcd(X,Y,Gcd) :- mod(X,Y,Z), gcd(Y,Z,Gcd).
```

“Cut“ and “Is” Operators

- The **“is” operator** : use is if and only if you need to evaluate something.
- This expression is evaluated and bound to the **left hand argument**.

?- X = 2, Y is X + 1.

X = 2

Y = 3

- The **Cut Operator** (!) prevents Prolog **finding all solutions.**
- The **cut operator**, is a built-in goal that stop **backtracking**

Queries

- Once we have a **database of facts and rules**, we can ask questions
- "**?-**" is **Prolog's prompt**
- **A query** : "Are there any cases for which the given predicate holds?"
- In SWI Prolog, **queries are terminated by a full stop.**
- **Prolog consults its database** to see if this is a known fact.
- If answer is **true**., the query succeeded
- If answer is **false.,** the query failed

- This means finding out if Turing lectures in a course that Fred studies.

?- lectures(turing, Course), studies(fred, Course).

Query

likes(john, jane) Dot necessary

True. Answer from Prolog interpreter

Sign on prolog query prompt

variables

?- friends(X, Y).

X= john,

Y= jane ; Type ; to get next solution

X = jane,

Y = john.

Execution

- **Running a Prolog program** = a special case of **resolution**
- Execution of a **Prolog program** is initiated by **a query.**
- Logically, **Prolog answers a query**
- The **resolution method** used by Prolog is called **SLD resolution.**
- **SLD resolution** (Selective Linear Definite clause **resolution**

- **Prolog** engine tries **to find a resolution refutation** of the negated query.
- An important step in is **syntactic unification of terms.**
- **Unification** and **Pattern Matching**
- Unification is a generalization of pattern matching.

Prolog Listing Examples

Facts	**English meanings**
food(burger).	// burger is a food
food(sandwich).	// sandwich is a food
food(pizza).	// pizza is a food
lunch(sandwich).	// sandwich is a lunch
dinner(pizza).	// pizza is a dinner
Rules	
meal(X) :- food(X).	// Every food is a meal OR Anything is a meal if it is a food
Queries / Goals	
?- food(pizza).	// Is pizza a food?
?- meal(X), lunch(X).	// Which food is meal and lunch?
?- dinner(sandwich).	// Is sandwich a dinner?

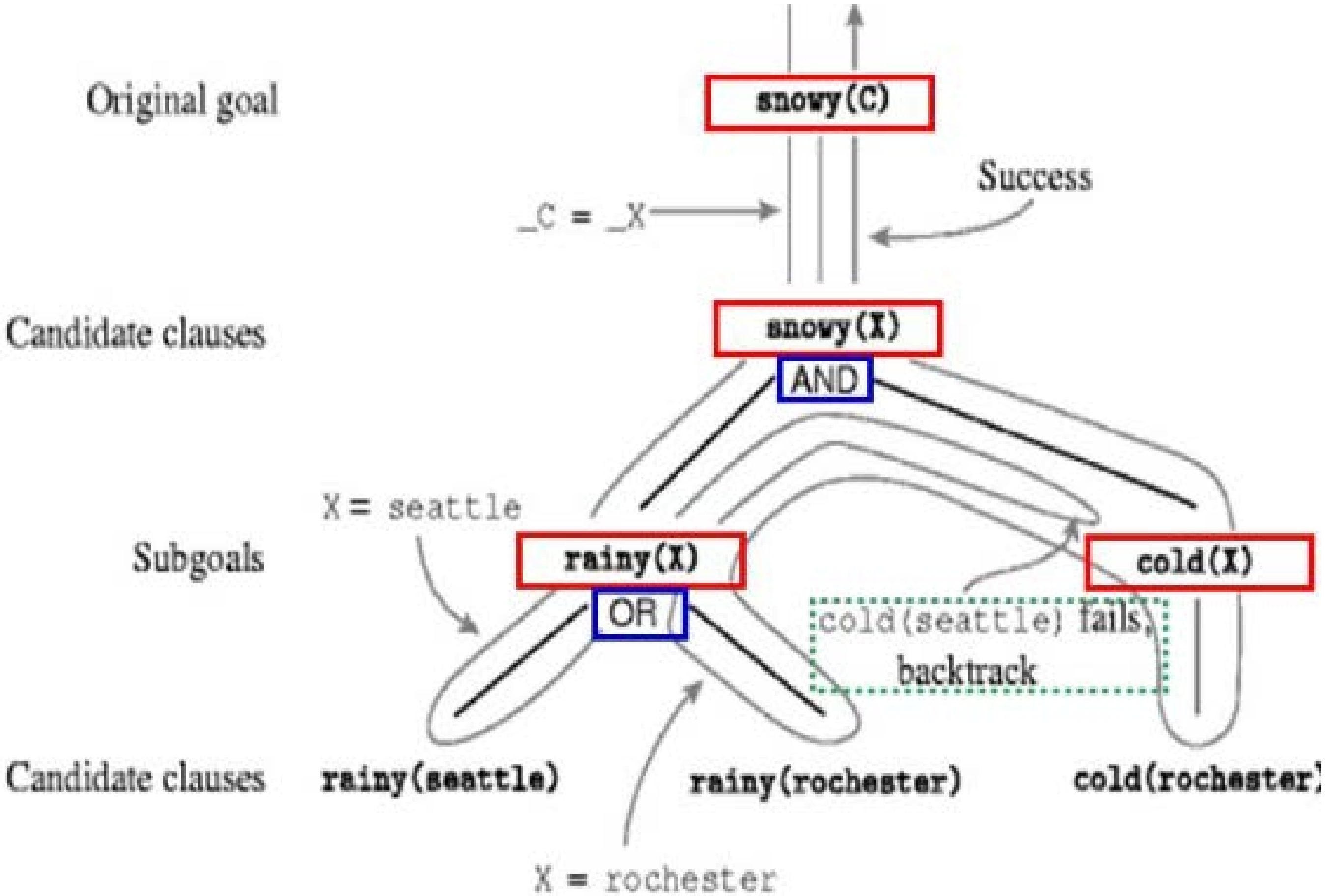
Prolog Listing Examples

Facts	**English meanings**
studies(charlie, csc135).	// charlie studies csc135
studies(olivia, csc135).	// olivia studies csc135
studies(jack, csc131).	// jack studies csc131
studies(arthur, csc134).	// arthur studies csc134
teaches(kirke, csc135).	// kirke teaches csc135
teaches(collins, csc131).	// collins teaches csc131
teaches(collins, csc171).	// collins teaches csc171
teaches(juniper, csc134).	// juniper teaches csc134
Rules	
professor(X, Y) :- teaches(X, C), studies(Y, C).	// X is a professor of Y if X teaches C and Y studies C.
Queries / Goals	
?- studies(charlie, What).	// charlie studies what? OR What does charlie study?
?- professor(kirke, Students).	// Who are the students of professor kirke.

Forward Chaining

- **Forward chaining**: methods of reasoning when using an **inference engine** as repeated application of **modus ponens.**
- **Forward chaining** uses **inference rules** to extract more data until a goal is reached.
- **Searches the inference rules** until it finds one
- **When such a rule is found**, the engine can **infer.**
- Inference **engines will iterate** through this process until a goal is reached.
- **forward-chaining** = better suited to **dynamic situations**

Backward Chaining



Backward chaining follows a classic depth-first backtracking algorithm.

Prolog Interpreter

Input: a goal G and a program P

Output: an instance of G, that is a logical consequence of P if it exists, otherwise NO

begin

$R := G$; R resolvent

finished := false;

prove the goal in the resolvent;

if $R = \{\}$

then return G

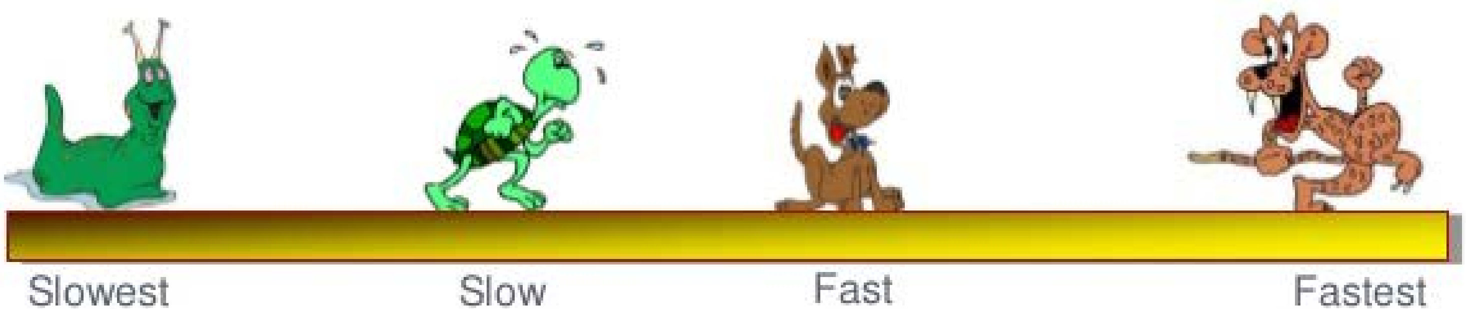
else return NO

end

Part VII – Fuzzy Logic

Fuzzy Logic

- Human thinking involves **fuzzy information**
- Originating from inherently inexact human concepts.
- **Fuzzy knowledge** that is **vague, imprecise, uncertain.**
- **Fuzzy Set Theory** / Concept of partial truth
- The term fuzzy logic was introduced in 1965 by Lotfi Zadeh.
- In set theory**, predicates** are understood to be **functions** from a set element to a truth value.



(Image courtesy of Qamar Wajid Ali

Reasoning in Fuzzy Logic

- Fuzzy logic deal with **uncertain, imprecise, vague information**
- **Fuzzy logic** allows for the inclusion of **vague human assessments**

- In **fuzzy logic**, predicates are **probability distributions.**
- The **valuation** of the predicate is a the **degree of truth.**
- **Truth values** of variables are **real number** between 0 and 1.

- **Defuzzification** is the process of producing a quantifiable result given fuzzy sets and corresponding
- **Fuzzy logic** is based on relative graded membership
- Numerous applications such **medical diagnosis** and **stock trading**.

Conclusion

"Beauty is truth, truth beauty,"—that is all
Ye know on Earth, and all ye need to know.
John Keats, Ode On A Grecian Urn, 1819

- A **model of the world** is crucial for an AI application
- A model to **represent** and **manipulate facts**
- **Propositional logic** = everything either **true** or **false**
- **Propositional Logic** is a **weak language.**
- **First-Order logic** is **expressive enough**
- **Prolog** is a programming language **to infer.**

Lab Activities

- Activity 1: Propositional Logic (25 min)
- Activity 2: Intro to Prolog (25 min)

- Break (10 min)

- Activity 3: Prolog (25min)
- Activity 4: Prolog (25min)

References

[1] - Interpreters for Propositional Logic: Implementing Negation - Universitat Dusseldorf
[2] - Propositional Logic Using truth tables - Edmund M. Clarke
[3] - A Prolog Interpreter for First-Order Intuitionistic Logic - McCarty and Shklar
[4] - Exercises for Artificial Intelligence - Dr Sean B Holden
[5] - Artificial Intelligence 1 - Matthew Huntbach
[6] - Logic, Programming, and Prolog - Ulf Nilsson and Jan Maluszyinski
[7] - First-Order Logic - Klaus Sutner
[8] - Natural Deduction - Frank Pfenning
[9] - Second-Order Logic and Fagins Theorem - Neil Immerman
[10] - First Order Logic- Stanley N. Burris
[11] - Logic, Programming and Prolog - Ulf Nilsson
[12] - Second order logic or set theory - Jouko Vaananen
[13] - Lecture Notes Prolog - Natalie Guin
[14] - Logic, Proof - Miguel A. Lerma
[15] - Introduction to natural deduction - Daniel Clemente Laboreo
[16] - First-Order Logic (First-Order Predicate Calculus) - Raymond J. Mooney
[17] - Introduction to fuzzy logic - Franck Dernoncourt
[18] - The Syntax of Propositional Logic - Derek Bridge
[19] - Expert Systems - Vladimir Vacic
[20] - Logic Programming - Introduction - Temur Kutsia

References

[21] - Some Equivalence Laws of Propositional Logic - Gordon J. Pace
[22] - Artificial Intelligence Programming in Prolog - Tim Smith
[23] - Fuzzy Logic - Krisana Chinnasarn
[24] - Logic and artificial intelligence - Nils J. Nilsson
[25] - Logical Operations and Truth Tables - Kenneth R. Koehler
[26] - Artificial Intelligence - Automated Reasoning - Andrea Torsello
[27] - Modal Logic for Artificial Intelligence - Rosja Mastop
[28] - Logic: Propositional Logic Truth Tables - Raffaella Bernardi
[29] - Introduction to Prolog read, write, assert, retract - Vacic and Koufogiannakis
[30] - Prolog - David Eyers
[31] - Prolog : A Tutorial Introduction - Lu and Mead
[32] - Natural Deduction : Cornell University
[33] - Propositional Logic - Al Aho and Jeff Ullman
[34] - Propositional Logic - Lydia Sinapova
[35] - Propositional Logic - Greg Baker
[36] - Propositional Logic - Edmund M. Clarke
[37] - Logical Forms and Equivalencies - Frederick T. Sheldon
[38] - Simply Logical - Intelligent Reasoning by Example - Peter Flach
[39] - Introduction to Second-Order Logic - K. Subramani
[40] - Statements, truth values and truth tables - Peter Wiliams

References

[31] - Logic Programming with Prolog - Max Bramer

[32] - The Early Years of Logic Programming , Robert A. Kowalsk

[33] - The Art of Prolog - Sterling and Shapiro

[34] - Problem Solving with Prolog - Ulle Endriss