CS2510
MODERN PROGRAMMING LANGUAGES

Logic Programming 1

Prof. Peter Edwards

p.edwards@abdn.ac.uk

Logic

- Aristotle described *syllogisms* 2300 years ago.
- Sample syllogism:

  Socrates is a man.
  All men are mortal.
  Therefore, Socrates is mortal.

- A syllogism gives two premises, then asks:
  "What can we conclude?"
  - This is forward reasoning - from premises to conclusions.
  - Inefficient when lots of premises.
- Alternative:
  - Use backward reasoning - from (potential) conclusions to facts.
  - **Is Socrates mortal?**

Logic

- Symbolic logic can be used for the basic needs of formal logic:
  - Used to express ***propositions***
    - A logical statement that may or may not be true;
    - Consist of objects and relationships of objects to each other.
  - Describe how new propositions can be inferred from other propositions.

Example Propositions
All cows are brown.
The Earth is further from the sun than Venus.
There is life on Mars.

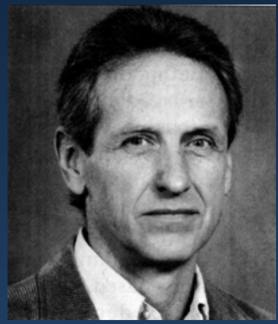
- Particular form of symbolic logic used for logic programming called ***predicate calculus***.

Predicate Calculus Examples
dog(fido)
friends(owen, john)
$\forall(x): man(X) \Rightarrow mortal(X)$

Prolog

- **Prolog**
  - Programming in logic
  - Represents facts, rules and queries.
- Prolog is a ***declarative programming*** language unlike most common programming languages.
- In a declarative language, the programmer specifies a goal to be achieved and the Prolog system works out how to achieve it.

1974 - R.A. Kowalski: 'Predicate logic as a programming language'



- First-order predicate logic for the specification of data and relations among data
- Computation = logical deduction

1972 - A. Colmerauer & P. Roussel: first Prolog-like interpreter

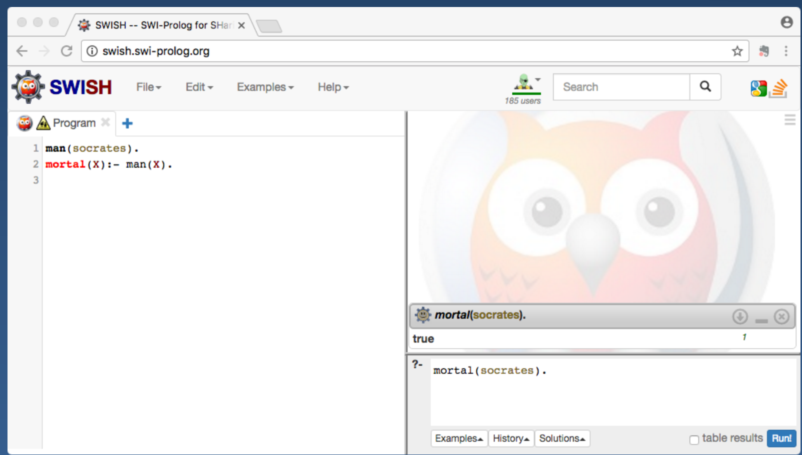
1980s - Adopted as language for Japanese 5th Generation project.

Logic Programming in *Prolog*

Prolog

Socrates is a man.	man(socrates).	Fact
All men are mortal.	mortal(X) :- man(X).	Rule
Is Socrates mortal?	?- mortal(socrates).	Query

SWI Prolog



Logic Programming: Logic + Control

- Logic: ***What***
  - Logical <u>Formulae</u>: $\forall x\,(p(x) \rightarrow q(x))$

- Control: ***How***
  - Logical <u>Deduction</u>: if $A \rightarrow B$ and A, then B

UNIVERSITY OF ABERDEEN

What: *Problem Description*

- *Horn clause*: $\boldsymbol{A \leftarrow B_1,\ldots,B_n}$
  - $A,B_1,\ldots,B_n$ are *predicates*, i.e., $p(a_1,\ldots,a_m)$
- Meaning:
  - A is **true** if B_1, B_2,…., and B_n are **true**
- Horn clauses allow us to specify
  - ***Facts***
  - ***Rules***
  - ***Queries***

  about *objects* and *relations* between them.

What: *Problem Description*

- Facts are represented as $A \leftarrow$

  has(owen, jaguar) $\leftarrow$ "owen has a jaguar"

- Rules are represented as $A \leftarrow B_1,\ldots,B_n$

  rich(X) $\leftarrow$ has(X, jaguar) "someone is rich if they have a jaguar"

- Queries are represented as $\leftarrow B_1,\ldots,B_n$

  $\leftarrow$ rich(Y) "who is rich?"

- **Facts + Rules: *Knowledge Base (KB)***

SWI Prolog

UNIVERSITY OF ABERDEEN

How: *Computing with Logic*

- Computation as ***resolution***, a deduction mechanism.
- A query starts up a computation which uses rules and facts (KB) to answer the query.
- A query is answered as ***true*** or ***false***, and its variables are (possibly) assigned values.
- Queries may loop!
  - as in any programming language programs may loop…

How: *Computing with Logic*

resolution(KB, Query): **boolean**
    let Query be $\leftarrow C_1,\ldots,C_n$
    for i = 1 to n **do**
        if there is a fact $A \leftarrow$ in KB such that $A = C_i$
        then true
        else if there is a rule $A \leftarrow B_1,\ldots,B_n$ in KB
            such that $A = C_i$
        then resolution(KB, $\leftarrow B_1,\ldots,B_n$)
        else false
    if all C_i are **true then return true**
        else return false

- Recursive formulation of resolution.
- Exhaustive:
  - in order to say "no" resolution must try ***all*** possibilities!!
- Simple & general definition of computation.

Example

How: *Computing with Logic*

KB={ has(owen, jaguar) ←, rich(X) ← has(X, jaguar) }

← rich(Y)

resolution(**KB**, ← **rich(Y)**)

let Query be ← **rich(Y)**

for i = 1 to 1 **do**

if there is a fact ***A*** ← in KB such that ***A*** = **rich(Y)**

then true

else if there is a rule $A \leftarrow B_1,\ldots,B_n$ in KB

such that $A = C_i$

then resolution(KB, $\leftarrow B_1,\ldots,B_n$)

else false

if all C_i are **true then return true**

else return false

UNIVERSITY OF ABERDEEN

How: *Computing with Logic*

KB={ has(owen, jaguar) ←, rich(X) ← has(X, jaguar) }

← rich(Y)

resolution(KB, ← **rich(Y)**)
    let Query be ← **rich(Y)**
    for i = 1 to 1 **do**
        if there is a fact A ← in KB such that A = **rich(Y)**
        then true
        else if there is a rule $A \leftarrow B_1,\ldots,B_n$ in KB
            such that A = *rich(Y)*
        then resolution(KB, $\leftarrow B_1,\ldots,B_n$)
        else false
    if all C_i are **true then return true**
        else return false

Yes, If **X** = **Y** then
rich(X) ← has(X, jaguar)

How: *Computing with Logic*

KB={ has(owen, jaguar) ←, rich(X) ← has(X, jaguar) }

← rich(Y)

resolution(KB, ← **rich(Y)**)
    let Query be ← **rich(Y)**
    for i = 1 to 1 **do**
        if there is a fact *A* ← in KB such that *A* = **rich(Y)**
        then true
        else if there is a rule ***A*** ← $B_1,\ldots,B_n$ in KB
            such that ***A*** = ***rich(Y)***
        then resolution(**KB**, ← $B_1,\ldots,B_n$)
        else false
    if all C_i are **true then return true**
        else return false

UNIVERSITY OF ABERDEEN

How: *Computing with Logic*

KB={ has(owen, jaguar) ←, rich(X) ← has(X, jaguar) }

← rich(Y)

resolution(**KB**, ← **rich(Y)**)

let Query be ← **rich(Y)**

for *i* = 1 to 1 **do**

if there is a fact *A* ← in KB such that *A* = **rich(Y)**

then true

else if there is a rule *A* ← $B_1,\ldots,B_n$ in KB

such that *A* = ***rich(Y)***

then resolution(**KB**, ← ***has(Y, jaguar)***)

else false

if all C_i are **true then return true**

else return false

How: *Computing with Logic*

KB={ has(owen, jaguar) $\leftarrow$, rich(X) $\leftarrow$ has(X, jaguar) }

$\leftarrow$ rich(Y)

resolution(**KB**, $\leftarrow$ **has(Y, jaguar)**)

Yes, If Y = **owen**

    let Query be $\leftarrow$ **has(Y, jaguar)**
    for i = 1 to 1 **do**
        if there is a fact $\boldsymbol{A} \leftarrow$ in KB such that $\boldsymbol{A}$ = **has(Y, jaguar)**
        then true
        else if there is a rule $\boldsymbol{A} \leftarrow \boldsymbol{B_1,\ldots,B_n}$ in KB
            such that $\boldsymbol{A} = \boldsymbol{C_i}$
        then resolution(KB, $\leftarrow \boldsymbol{B_1,\ldots,B_n}$)
        else false
    if all C_i are **true then return true**
        else return false

UNIVERSITY OF ABERDEEN

How: *Computing with Logic*

KB={ has(owen, jaguar) ←, rich(X) ← has(X, jaguar) }

← rich(Y)

resolution(KB, ← **rich(Y)**)
  let Query be ← **rich(Y)**
  for i = 1 to 1 **do**
    if there is a fact ***A*** ← in KB such that ***A*** = **rich(Y)**
    then true
    else if there is a rule ***A*** ← $B_1,\ldots,B_n$ in KB
      such that ***A*** = ***rich(Y)***
    then resolution(KB, ← ***has(owen, jaguar)***)
    else false
  if all C_i are **true then return true**
    else return false

That is, Y = owen

UNIVERSITY OF ABERDEEN

How: *Computing with Logic*

KB={ has(owen, jaguar) ←, rich(X) ← has(X, jaguar) }

← rich(Y)

resolution(KB, ← **rich(Y)**)
  let Query be ← **rich(Y)**
  for i = 1 to 1 **do**
    if there is a fact ***A*** ← in KB such that ***A*** = **rich(Y)**
    then true
    else if there is a rule ***A*** ← ***B*$_1$,…,*B*$_n$** in KB
      such that ***A*** = ***rich(Y)***
    then resolution(**KB**, ← ***has(owen, jaguar)***)
    else false
  if all ***rich(Y)*** is **true then return true**
    else return false

Yes and Y = owen

How: *Computing with Logic*

resolution(KB, Query): **boolean**
  let Query be $\leftarrow C_1,\ldots,C_n$
  for i = 1 to n **do**
    $\boxed{\textbf{if} \text{ there is a fact } A \leftarrow \text{ in KB}}$ such that $\boxed{A = C_i}$
    then true
    else $\boxed{\textbf{if} \text{ there is a rule } A \leftarrow B_1,\ldots,B_n \text{ in KB}}$
      such that $\boxed{A = C_i}$
    then resolution(KB, $\leftarrow B_1,\ldots,B_n$)
    else false
  if all C_i are **true then return true**
    else return false

- **Non-determinism**:
  - If there is more than one, which one to choose?
- ***Unification***:
  - Is it possible to find values of variables that would make A and C_i equal?