Outline

• Predicate Calculus
• Theorem Proving
• Logic Programming
• Case Study: Prolog
• Examples
  • Sudoku
  • N-Queens
Logic Programming

• Expressed in a form of symbolic logic

• Applies logical inferencing to produce results

• **Key insight:** *Declarative* (instead of *Procedural*)
  • Specification of results are stated
  • Rather than the procedures which can produce them
Predicate Calculus

Preposition

A logical statement that may or may not be true

• Consists of objects and relationships

Predicate Calculus

Logic founded upon prepositions, variables, constants, and quantifiers

• **Variable** – a symbol that can represent different objects at different times

• **Constant** – a symbol that represents one object

• **Quantifier** – a countable amount (for all, there exists)
Propositions == Compound Terms

• Atomic propositions consist of compound terms
• Compound term describes a relation, but is often expressed as a function (can be written as a table)
• Two parts to a compound term
  • Functor – function symbol that names the relationship
  • Parameters – ordered list (akin to a tuple)
• Examples:
  student(jon)
  like(beth, macOS)
  like(chris, windows)
  like(will, linux)
Proposition Forms

• Propositions can be stated in two forms:
  • Fact – proposition is assumed to be true
  • Query – truth of proposition is to be determined

• Compound Proposition
  • Have two or more atomic propositions
  • Propositions are connected by operators
# Logical Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>negation</td>
<td>¬</td>
<td>¬ a</td>
<td>not a</td>
</tr>
<tr>
<td>conjunction</td>
<td>∧</td>
<td>a ∧ b</td>
<td>a and b</td>
</tr>
<tr>
<td>disjunction</td>
<td>∨</td>
<td>a ∨ b</td>
<td>a or b</td>
</tr>
<tr>
<td>equivalence</td>
<td>≡</td>
<td>a ≡ b</td>
<td>a is equivalent to b</td>
</tr>
<tr>
<td>implication</td>
<td>⊃</td>
<td>a ⊃ b</td>
<td>a implies b</td>
</tr>
<tr>
<td></td>
<td>⊂</td>
<td>a ⊂ b</td>
<td>b implies a</td>
</tr>
</tbody>
</table>
## Quantifiers

<table>
<thead>
<tr>
<th>Name</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>universal</td>
<td>$\forall X. P$</td>
<td>For all $X$, $P$ is true</td>
</tr>
<tr>
<td>existential</td>
<td>$\exists X. P$</td>
<td>There exists a value of $X$ such that $P$ is true</td>
</tr>
</tbody>
</table>
Clausal Form

• We will use a standard form for all propositions

• **Antecedent**
  • Right side
  • What must be true

• **Consequent**
  • Left side
  • What could be true

• **Example**
  • $B_1 \cup B_2 \cup \ldots \cup B_n \subseteq A_1 \cap A_2 \cap \ldots \cap A_m$
  • means if all the As are true, then at least one B is true
Theorem Proving

• Given known axioms an theorems...
  
  *We should be able to discover new theorems!*

• Resolution
  
  • A principle of inference that allows inferred propositions to be computed from given propositions

• Unification
  
  • Finding values for variables in propositions

• Instantiation
  
  • Assigning temporary values to variables to allow unification

• After instantiation: if matching fails, we may backtrack
Logic Programming

- Declarative
- Non-Procedural

*Programs do not state how to do something!*

*... Programs state what the result will be.*
Logic Programming: Sorting

• Describe the characteristics of a sorted list, rather than the process of rearranging a list

\[
\text{sort}(\text{old\_list}, \text{new\_list}) \subseteq \\
\text{permute } (\text{old\_list}, \text{new\_list}) \cap \text{sorted } (\text{new\_list})
\]

\[
\text{sorted } (\text{list}) \subseteq \\
\forall j \text{ such that } 1 \leq j < n, \text{ list}(j) \leq \text{ list } (j+1)
\]
Prolog
Prolog

Predominately used in two fields/areas (origin)

• Natural language processing
• Automated theorem proving

**Important Terms:**

- **Term** constant, variable, or structure
- **Constant** atom or integer
- **Atom** consists of either:
  - A string of letters, digits, and underscores (starts with a-z)
  - A string of printable ASCII characters delimited by ‘
Terms

**Variable**
- Any string of letters, digits, or underscores starting with a Capital letter

**Instantiation**
- Binding of a variable to a concrete value
- May be a temporary binding

**Structure**
- Represents one atomic proposition
  - `functor( parameter, list )`
Facts

Facts are used (in part) to define hypotheses

Known as a “Headless Horn” clause

female(amy).
female(stephanie).
male(will).
father(larry, amanda).
Rules

Rules are used (in part) to define hypotheses

Known as a “Headed Horn” clause

**Right side:** *antecedent* (*if* part) – can be a conjunction

**Left side:** *consequent* (*then* part) – single term

\[
\text{ancestor}(\text{mary,shelley}) :\text{ } - \text{mother}(\text{mary,shelley}).
\]

\[
\text{parent}(X,Y) :\text{ } - \text{mother}(X,Y).
\]

\[
\text{parent}(X,Y) :\text{ } - \text{father}(X,Y).
\]

\[
\text{grandparent}(X,Z) :\text{ } - \text{parent}(X,Y), \text{ parent}(Y,Z).
\]
Goals

For theorem proving, we may just want to learn or derive something *interesting* (via proving or disproving)

“Headless Horn” notation:

\[
\text{man} (\text{fred})
\]

Can also generalize with variables and propositions

\[
\text{father}(X, \text{mike})
\]

\[
\text{female}(Y)
\]
Approaches to Solving

• *Matching* is the process of proving a proposition
• Proving a subgoal is called *satisfying* the subgoal
• *Bottom-up resolution, forward chaining*
  • Begin with facts and rules of database and attempt to find sequence that leads to goal
  • Works well with a large set of possibly correct answers
• *Top-down resolution, backward chaining*
  • Begin with goal and attempt to find sequence that leads to set of facts in database
  • Works well with a small set of possibly correct answers
• Prolog implementations use backward chaining
Arithmetic

Integer variables and integer operations are supported

is operator
D is B * B - 4 * A * C.

*Illegal to do variable reassignment!*

Sum is Sum + X.
Arithmetic Example

speed(ford, 100).
speed(chevy, 105).
speed(dodge, 95).
speed(volvo, 80).
time(ford, 20).
time(chevy, 21).
time(dodge, 24).
time(volvo, 24).
distance(X, Y) :- speed(X, Speed),
                time(X, Time),
                Y is Speed * Time.

distance(chevy, Chevy_Distance).
Lists

• Lists is a sequence of any number of elements
• Elements can be atoms, atomic propositions, or other terms (even other lists!)

[ apple, orange, pear, peach ]

[ ] – empty list
[ X | Y ] – list with head X and tail Y
List Operations - Append

\( \text{append}([], \text{List}, \text{List}). \)

\( \text{append}([\text{Head} \mid \text{L1}], \text{L2}, [\text{Head} \mid \text{Out}]) :- \text{append}(	ext{L1}, \text{L2}, \text{Out}). \)
List Operations - Reverse

\[ \text{reverse}([], []). \]
\[ \text{reverse}([\text{Head} \mid \text{Tail}], \text{List}) : - \\
\quad \text{reverse}(\text{Tail}, \text{Result}), \\
\quad \text{append}(\text{Result}, [\text{Head}], \text{List}). \]
List Operations - Member

\[ \text{member}(\text{Elem}, [\text{Elem} | _]). \]
\[ \text{member}(\text{Elem}, [| \text{List}]) :- \\
\quad \text{member}(\text{Elem}, \text{List}). \]

*Underscore is an anonymous variable*
Deficiencies of Logic Programming

Resolution Order Control

• *The order of attempted matches is non-deterministic and all matches would be attempted concurrently*

The Closed-World Assumption

• *The only knowledge is what is in the database*

The Negation Problem

• *Anything not stated in the database is assumed to be false*
Examples
N-Queens

Problem:
Provided an N x N chess board, place N queens such that none of them can “take” another (for those with a chess background).

For those without a chess background: place N queens on an N x N chess board such that there is only one queen per row, one queen per column, and no two queens’ difference in rows equals their difference in columns.
N-Queens

[https://swish.swi-prolog.org/example/queens.pl](https://swish.swi-prolog.org/example/queens.pl)

Traditional Prolog implementation
- Requires a full board definition
- Iteratively makes all constraints one queen at a time
- Relies on extensive list processing operations

[https://swish.swi-prolog.org/example/clpfd_queens.pl](https://swish.swi-prolog.org/example/clpfd_queens.pl)

Prolog Implementation relying on CLP(FD) library
- **Constraint Logic Programming** over **Finite Domain**
- Replaces lists with domains and special operations
N-Queens

n_queens(N, Qs) :-
    length(Qs, N), Qs ins 1..N, safe_queens(Qs).

safe_queens([]).

safe_queens([Q|Qs]) :-
    safe_queens(Qs, Q, 1), safe_queens(Qs).

safe_queens([], _, _).

safe_queens([Q|Qs], Q0, D0) :-
    Q0 #\= Q, abs(Q0 - Q) #\= D0, D1 #= D0 + 1,
    safe_queens(Qs, Q0, D1).
Sudoku

**Problem:**

Given a 9x9 grid subdivided into 3x3 “houses”, place the values 1 through 9 such that

- Each row contains each value exactly once
- Each column contains each value exactly once
- Each house contains each value exactly once
Sudoku

[link: https://swish.swi-prolog.org/example/clpfd_sudoku.pl]

Prolog Implementation relying on CLP(FD) library

Checking Houses:

blocks([], [], []).
blocks([A,B,C|Bs1],
       [D,E,F|Bs2],
       [G,H,I|Bs3]) :-
   all_distinct([A,B,C,D,E,F,G,H,I]),
   blocks(Bs1, Bs2, Bs3).
Sudoku

https://swish.swi-prolog.org/example/clpfd_sudoku.pl
Prolog Implementation relying on CLP(FD) library

Checking Houses:
blocks([], [], []).
blocks([A,B,C|Bs1],
   [D,E,F|Bs2],
   [G,H,I|Bs3]) :-
   all_distinct([A,B,C,D,E,F,G,H,I]),
   blocks(Bs1, Bs2, Bs3).
Sudoku Solver:
sudoku(Rows) :-
    length(Rows,9),
    maplist(same_length(Rows),Rows),
    append(Rows,Vs), Vs ins 1..9,
    maplist(all_distinct,Rows),
    transpose(Rows,Columns),
    maplist(all_distinct,Columns),
    Rows = [A,B,C,D,E,F,G,H,I],
    blocks(A,B,C), blocks(D,E,F), blocks(G,H,I).
Sudoku

Problem Definition

problem(1, [[_,_,_, _,_,_, _,_,_],
            [_,_,_, _,_,3, _,8,5],
            [_,_,1, _,2,_, _,_,_],
            [_,_,_, 5,_,7, _,_,_],
            [_,_,4, _,_,_, 1,_,_],
            [_,9,_, _,_,_, _,_,_],
            [_,_,_, 5,_,7, __,_],
            [_,_,4, _ _,_, _ ,1, _],
            [_,9,_, _,_,_, _,_,_],
            [5,_,_, _,_,_, _ ,7,3],
            [_,_,2, _ _,1,_, _ ,_],
            [_,_,_, _,4,_, _ ,_ ,9]]).
Sudoku

Problem Solution

\texttt{problem(1, Rows), sudoku(Rows).}