Introduction to Prolog

• Useful references:

Negation as Failure

Using not will not help you. Do not try to remedy this by defining:

\[ \text{guilty(X)} \leftarrow \text{not(innocent(X))}. \]

This is useless, and makes matters even worse:

\[ ?- \text{guilty(st.francis)}. \]

yes

It is one thing to show that st.francis cannot be demonstrated to be innocent. But it is quite another thing to incorrectly show that he is guilty.

Negation-by-failure can be non-logical

Some disturbing behaviour even more subtle than the innocent/guilty problem, and can lead to some extremely obscure programming errors. Here is a restaurant database:

\[ \text{good.standard(goedels)}. \]
\[ \text{good.standard(hilberts)}. \]
\[ \text{expensive(goedels)}. \]
\[ \text{reasonable(R)} \leftarrow \text{not(expensive(R))}. \]

Consider the following dialogue:

\[ ?- \text{good.standard(X)}, \text{reasonable(X)}. \]

X = hilberts

But if we ask the logically equivalent question:

\[ ?- \text{reasonable(X)}, \text{good.standard(X)}. \]

no

Question

Why do we get different answers for what seem to be logically equivalent queries?

The difference between the questions is as follows. In the first question, the variable X is always instantiated when reasonable(X) is executed. In the second question, X is not instantiated when reasonable(X) is executed. The semantics of reasonable(X) differ depending on whether its argument is instantiated.
Not a Good Idea!

It is bad practice to write programs that destroy the correspondence between the logical and procedural meaning of a program without any good reason for doing so.

Negation-by-failure does not correspond to logical negation, and so requires special care.

How to fix it?

One way is to specify that negation is undefined whenever an attempt is made to negate a non-ground formula.

A formula is ‘ground’ if is has no unbound variables.

Some Prolog systems issue a run-time exception if you try to negate a non-ground goal.

Clauses and Databases

In a relational database, relations are regarded as tables, in which each element of an n-ary relation is stored as a row of the table having n columns.

supplier
  jones chair red 10
  smith desk black 50

Using clauses, a table can be represented by a set of unit clauses. An n-ary relation is named by an n-ary predicate symbol.

supplier(jones, chair, red, 10).
supplier(smith, desk, black, 50).

Clauses and Databases

Advantages of using clauses:
1. Rules as well as facts can coexist in the description of a relation.
2. Recursive definitions are allowed.
3. Multiple answers to the same query are allowed.
4. There is no role distinction between input and output.
5. Inference takes place automatically.
Negation and Representation

Like databases, clauses cannot represent negative information. Only true instances are represented.

The battle of Waterloo occurred in 1815.

How can we show that the battle of Waterloo did not take place in 1923? The database cannot tell us when something is not the case, unless we do one of the following:

1. ‘Complete’ the database by adding clauses to specify the battle didn’t occur in 1814, 1813, 1812, ..., 1816, 1817, 1818,...
2. Add another clause saying the battle did not take place in another year (the battle occurred in and only in 1815).
3. Make the ‘closed world assumption’, implemented by ‘negation by failure’.

Constraint Satisfaction Problem

• Finite set of variables: X1,...,Xn
• Variable Xi has values in domain Di.
• Constraints C1…Cm. A constraint specifies legal combinations of the values.
• Assignment: selection of values for all variables.
• Consistent: assignment satisfies all constraints.

Example: Map-coloring

Domain: colors = {red, blue, green}
Variables: X1, X2, X3, X4; domain colors
Xi is color of country Ci.

Constraints:
C1 touches C2 ... X1 =/= X2
C1 touches C3 ... X1 =/= X3
C2 touches C3 ... X2 =/= X3
C2 touches C4 ... X2 =/= X4
C3 touches C4 ... X3 =/= X4

MapColoring in Prolog

• color(r).
• color(g).
• color(b).
• colormap(C1,C2,C3,C4):-
  color(C1),color(C2),color(C3), C1=\=C2,
  C1=\=C3, C2=\=C3,C2=\=C4,C3=\=C4.
• ?- colormap(X,Y,Z,U).
  X = r, Y = g, Z = b, U = r.
• Is that it. Yes! Turn on trace.
Satisfiability as CSP

- Domain: \{ true, false \}
- Variables: Propositional variables
- Constraints: clauses

- \texttt{sat}(true). % base case
- \texttt{sat}(\texttt{not}(false)). % base case
- \texttt{sat}(\texttt{or}(X,Y))\ :- \texttt{sat}(X).
- \texttt{sat}(\texttt{or}(X,Y))\ :- \texttt{sat}(Y).
- \texttt{sat}(\texttt{and}(X,Y))\ :- \texttt{sat}(X),\texttt{sat}(Y).
- \texttt{test1}(X,Y)\ :- \texttt{sat}(\texttt{and}(\texttt{not}(X),X)).
- \texttt{test2}(X,Y)\ :- \texttt{sat}(\texttt{and}(X,\texttt{not}(Y))).

Binary Constraints

- Unary constraints: involve only one variable
  - Means that we can simply re-write the domains
    - In the problem specification to remove the constraint
- Binary constraints: involve two variables
  - Binary CSPs: all constraints are binary
  - Much researched
    - All CSPs can be written as binary CSPs (no details here)
    - Nice graphical and Matrix representations
      - Representative of CSPs in general

Different constraint systems

- Real/rational constraints: CLP(R), CLP(Q)
  - CLP(R), Sicstus Prolog, CHIP
- Finite domains constraints: CLP(FD)
  - Sicstus Prolog, CHIP
- Boolean constraints: CLP(B)
  - Sicstus Prolog, CHIP
- Interval constraints: CLP(I)
  - CLP(BNR), Numerica, Prolog IV

Binary Constraint Graph

Nodes are Variables
Edges are Constraints
Matrix Representation for Binary Constraints

The programming paradigm

Logic programming
Constraint satisfaction/solving
Optimization

Random Generation of Test Problems

- Generation of random binary CSPs
  - Choose a number of variables
  - Randomly generate a matrix for every pair of variables
- Used for benchmarking
  - e.g., efficiencies of different CSP solving techniques
- Real world problems often have more structure
  - The small world phenomena...

N-queens Example (4 in our case)

- Standard test case in CSP research
- Variables are the rows
- Values are the columns
- So, the constraints include:
  - $C_{1,2} = \{(1,3),(1,4),(2,4),(3,1),(4,1),(4,2)\}$
  - $C_{1,3} = \{(1,2),(1,4),(2,1),(2,3),(3,2),(3,4), (4,1),(4,3)\}$
  - Etc.
  - What do these constraints mean?
Example: Scheduling

A conference consists of 11 sessions of equal length. The program is to be organized as a sequence of slots, where a slot contains up to 3 parallel sessions:
1. Session 4 must take place before Session 11.
2. Session 5 must take place before Session 10.
3. Session 6 must take place before Session 11.
   ...
8. Session 6 must not be in parallel with 7 and 10.
9. Session 7 must not be in parallel with 8 and 9.
10. Session 8 must not be in parallel with 10.
Minimize the number of slots.

Example: Job shop scheduling

There are n jobs and m machines. Each job requires execution of a sequence of operations within a time interval, and each operation \( O_i \) requires exclusive use of a designated machine \( M_i \) for a specified amount of processing time \( p_i \). Determine a schedule for production that satisfies the temporal and resource capacity constraints.

Example: Manpower planning

Airport Counter Allocation problem: Allocate enough counters and staff (the number depends on the aircraft type) to each flight. The counters are grouped in islands and for each flight all assigned counters have to be in the same island. The staff has working regulations that must be satisfied (breaks etc).

Digital Circuits- full adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>Sum</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

```
adde(\_, A, B, Cin, Sum, Cout) :-
  sat(Sum =:= card([1,3], [A, B, Cin]),
  sat(Cout =:= card([2,3], [A, B, Cin])).
```
Digital Circuits - full adder
CLP(B)

?- adder(_, a, b, c, SUM, CARRY).
\[ \text{sat(SUM}=:=a\#b\#c), \]
\[ \text{sat(CARRY}=:=b\#a\#c\#a\#c\#b). \]

Black-box vs Glass-box solvers

- Most systems rely on non-extensible, black-box constraint solvers
  - Efficiency unpredictable
  - Hard to debug
- Some systems facilitate defining new constraints and solvers (glass-box approach)
  - Improved control of propagation and search
- Then again, most problems are NP-complete...

BackTracking Algorithm: Recursive formulation

- \text{recurse( assignment, csp)}
  - if complete, return assignment
  - else SELECT unassigned variable
  - for each value in domain of variable (order)
    - if consistent:
      - add variable/value call recurse
      - if recurse succeeds, return assignment
    - else remove (var/value)
      - return failure

Controlling Backtracking

- Choosing a variable to bind
- Choosing a binding for variable
  Prolog: user code does it.
  Variable ordering:
  - MRV heuristic: minimum remaining values
  - most constraining variable
  - choose variable with fewest values
  - Complete. Only affects efficiency.
Control Backtracking

- Degree Heuristic
  - choose variable involved in largest number of constraints. (tie-breaker)
- Choosing a value
  - least-constraining value
  - maximize number of future options
- Complete.
- Various heuristic change efficiency

Forward Checking

- Constraint Propagation
  - If variable X is bound, find all variables Y that are connected to it by a constraint. Eliminate values that are inconsistent.
  - Can yield dramatic improvement.
  - Avoid many unnecessary backtrackings
Arc-Consistency

- Binary CSP (can always force this)
- Arc = edge between variables that are in same constraint.
- Let X-Y be arc.
  - for each x in Domain X if there is no value y in domain Y that allows constraint to be met, then delete x from domain X.
- Can be done as preprocessing or during search.

Cost of Arc-Consistency Checking

- Let d = max number of values for any variable
- Let n = number of variables
- There are at most n^2 constraints.
- For any specific pair, at most d^2 table lookups.
- Pair revisited at most d times. So..
- O(n^2 d^3) cost.

AC-3 algorithm: Mackworth

- Binary CSP. Direct the arcs
- Queue = all arcs (constraints)
- while (queue is not empty)
  - remove first pair arc X-Y
  - check arc consistency of X-Y
  - if any values deleted, then add arcs Z-X to queue where Z is neighbor of X.

AC3 Example

- TWO+TWO = FOUR
- edge between T and F gives:
  - (T+T+Carry)/10 = F we see that only T in {5,6,7,8,9} can work for F = 1.
- Edge between F and R says R //= 1.
- Edge between (O+O)%10 = R and O=}R gives: O=}0.
AC3 in Backtrack Algorithm

1. Choose any variable as root. Order variables from root to leaves so parent precedes son. Say X1, X2, … XN.
2. For j=N to 2 apply arc consistency.
3. For j = 1 to N assign any consistent value.
   Major cost: arc consistency.
   Generalization Possible (tree decomposition)

Intelligent Backtracking

- as opposed to chronological backtracking
- Identify conflict set – variables that could change the value of constraint.
- Backtrack to most recent variable in conflict set.
- Theorem: forward checking prunes all that backjumping does (and maybe more).

Local Search for CSP

- No guarantees but…
- Hubble Scheduling 3 weeks -> 10 minutes!
- Current = random complete assignment of values (may repeat)
- For max number of steps do
  - if current = solution, return it
  - randomly choose a variable V that has conflicts
  - choose a value that minimizes number of conflicts
  - break ties randomly!

Some Hot Topics in Constraint Technology

- Formulation of CSPs
  - It’s very easy to specify CSPs (this is an attraction)
    - But some are worse than others
  - There are many different ways to specify a CSP
  - It’s a highly skilled job to work out the best
- Automated reformulation of CSPs
  - Given a simple formulation
    - Can an agent change that formulation for the better?
    - Mostly: what choice of variables are specified
    - Also: automated discovery of additional constraints
    - Can we add in extra constraints the user has missed?
More Hot Topics

- Symmetry detection
  - Can we spot whole branches of the search space
    - Which are exactly the same (symmetrical with) a branch we have already (or are going to) search
  - Humans are good at this
  - Can we get search strategies to do this automatically?
- Dynamic CSPs
  - Solving of problems which change
  - While you are trying to solve them
  - For example a packing problem
    - A new package arrives to be fitted in