Université d'Ottawa | University of Ottawa

CSI2120 Programming Paradigms

Jochen Lang

jlang@uottawa.ca

Logic Programming in Prolog

- Predicate calculu
  - Predicates
  - Horn clauses
  - Proof by Contradiction: Resolution
- Search Trees
  - Backtracking

Prolog Predicates

- A rule is a claus[illegible]e body is non-empty while a fact is a clause w[illegible]pty body. Most rules contain variables.
- Prolog Definition with an anonymous variable, written as "_"

```
salary(X) :- employed(Y,X). % Ok but with
                            % a warning
```

  - Or with an anonymous variable

```
salary(X) :- employed(_,X).
```

- Facts and rules are predicates.

Predicate Calculus

- First Order Logic [illegible]
  - predicate sy [illegible] ,z (constants and variables)
    - and compound terms
  - equality: ≡
  - negation: ¬
  - logic binary connections: ∨,∧,→
  - quantifiers 'for all …' and 'there exists … such that'
    - universal quantifier ∀
    - existential quantifier ∃

Predicates in Prolog

- $b \leftarrow a_1 \wedge a_2 \wedge \cdots \wedge$
  - All terms $a_1,$ the body of the predicate have to be true for the head to be true. Or, $a_1, a_2, \ldots, a_3$ being true, implies b is true.
- $b \leftarrow$
  - This is a fact because truth is always implied.
- $\leftarrow a$
  - Without a head, it is goal for which correctness still needs to be proven. This may be considered a question in logic programming in Prolog. Proofing correctness requires deductive reasoning.

Horn Clauses

- We can express [illegible] logic with Horn* clauses and solve predicate [illegible]echanically
- Horn clauses are the foundation of logic programming
- Horn formulas are the only logic formulas in Prolog
  - Atomic (i.e., unique) formulas and their negation. They are also called literals
  - Disjunction of literals to form clauses
  - A Horn clause has exactly one non-negated literal
  - Conjunctive normal form (CNF) is a conjunction of Horn clauses

* Alfred Horn, 1918-2001 American Mathematician

Converting to a Horn Formula

- Implication (a in $\longrightarrow b$ is the same as $\neg a \vee b$
  - Use truth tab rm

a	b	¬a ∨ b	a → b
0	0	true	true
0	1	true	true
1	0	false	false
1	1	true	true

This may be suprising at first! Because a is false, nothing can be "implied", b can be true or false, the implication cannot be false. In logic if something is not false, it must be true.

- Equivalence (a equivalent to b) $a \equiv b$ is the same as $(a \wedge b) \vee (\neg a \wedge \neg b)$

Resolution

- Rule of inferenc [illegible] em proofing in propositional logic.
- Resolution rule
  - For a one-literal clause (modus ponens) $\frac{p \rightarrow q, p}{q}$ which reads (p implies q and p) entails q
    - In other words, as p implies q and we are asserted that p is true, q must be true
  - Can use for multi-literal clauses $\frac{((p_0 \wedge p_1) \rightarrow q, p_0, p_1)}{q}$ which reads ($p_0 \wedge p_1$ implies q and p_0 and p_1) entails q
    - In other words, as $p_0 \wedge p_1$ implies q and we are asserted that p_0 and p_1 are true, q must be true

Prolog Example using Resolution

- Our program for want to proof f to be true.
  We have the rul b.
  And the true fact `a.`
  And the true fact `b.`
- Our program expressed in predicate logic
  $(a \wedge b) \longrightarrow f$ *and a and b*
- Turn Horn formula $(a \wedge b) \longrightarrow f$ into CNF
  $(\neg(a \wedge b) \vee f) \equiv (\neg a \vee \neg b \vee f)$

Proof by Contradiction with Repeated Application of Resolution

- Consider

  $(\neg a \vee \neg b \vee f)$ *and* a *and* b

- Proof f by contradiction, i.e., assume $\neg f$

  $(\neg a \vee \neg b \vee f)$ *and* a *and* b *and* $\neg f$

- **Simplify by resolution**

  $((\neg a \text{ but } a \text{ is true}) \vee (\neg b \vee f))$ *and* b *and* $\neg f$

  $(\neg b \vee f)$ *and* b *and* $\neg f$

  $((\neg b \text{ but } b \text{ is true}) \vee f)$ *and* $\neg f$

  f *and* $\neg f$ which is a contradiction

Prolog and Horn Clauses

- Facts and rules a Clauses as in the example.
- In general `F :-` , `Fn.`
  - meaning `F` if `F1` and `F2` and …and `Fn`
  - `F` is an atomic formula
  - `Fi` are terms or their negation
- `F` is the head of the clause
- `F1, F2,…, Fn` together are the body of the clause
- To prove `F` in Prolog, it must be true (proven) that `F1`, `F2`,…, and `Fn` are true .

Horn Clauses

- Horn clauses ca[illegible] nearly all logic expressions, all mathematical al[illegible]
- *It enables one to establish the truth of a hypothesis by establishing the truth of terms but it does not allow one to prove the falsehood of a hypothesis. False in logic programming only means that the goal can not be proven correct.*

Search Trees

- Search trees rep[illegible]eries
  - Root of the tr[illegible]question
  - Nodes (or vertices) are decisions and show which goals still need to be satisfied
  - Transitions (along edges) from one node to the next are the result of an unification between a goal and a fact or the head of a rule.
  - The edges are a step in the proof.

Nodes in Search Tree

- Goals are orc[illegible] left to right following the order in the r[illegible]ts are stated in a node.
- Leaf nodes which contain one or several goals are failure nodes. The first (left-most) goal caused the failure.
- Empty leaf nodes are success nodes. The path from the root to the leaf node contains the unifications and steps necessary for the proof. These can be found on the edges.

Solution Strategy of Prolog

- Prolog builds the [illegible] ee from the question as a root node. The tree i[illegible] depth-first fashion.
- An empty (leaf) node is a proof or a solution
  - Search can continue for other solutions by backtracking and traversing unexplored branches
- An non-empty leaf node is a failure
  - A solution may still be found by backtracking and traversing unexplored branches

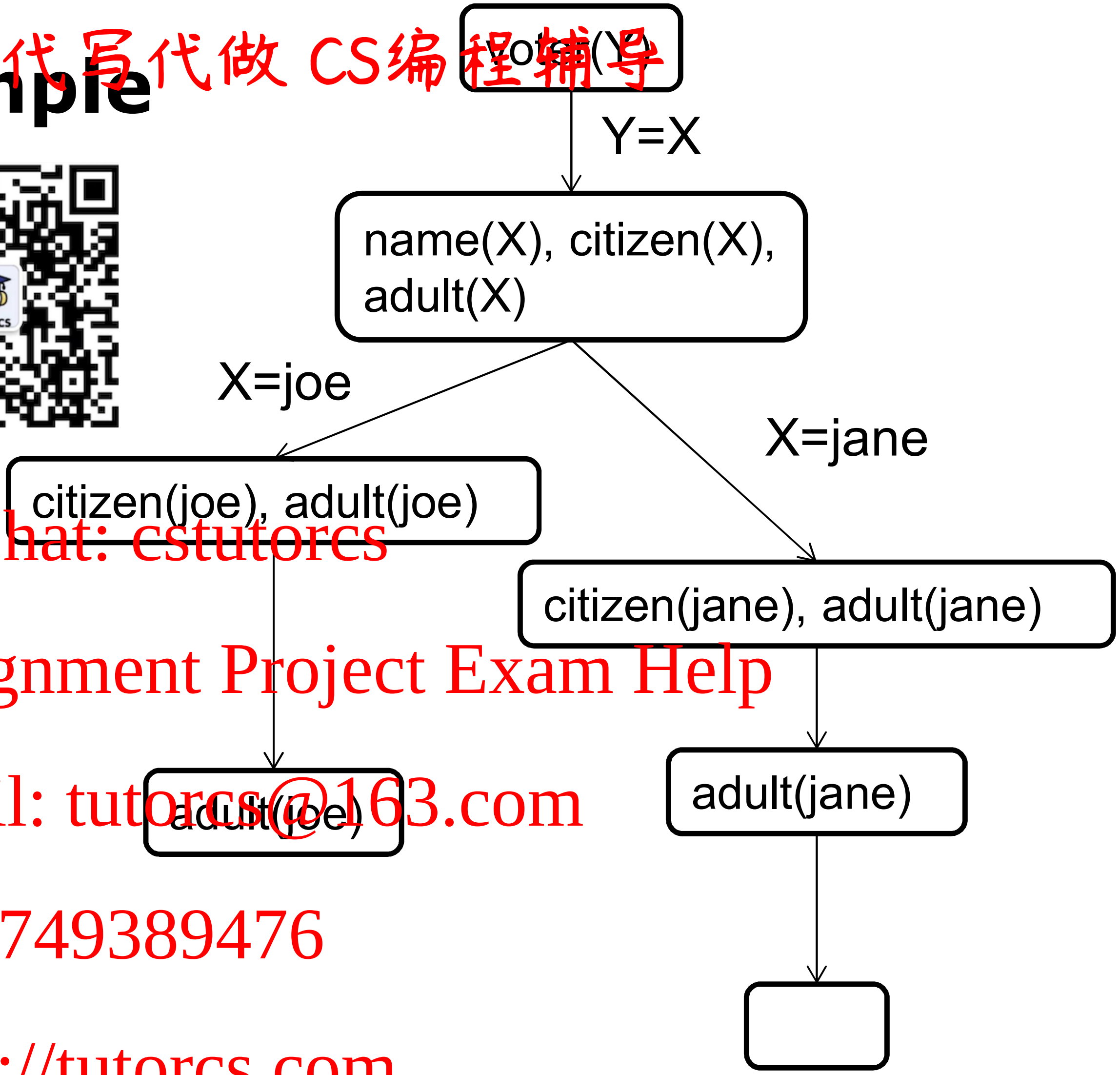
Backtracking

- If there are no m[illegible] nodes found, there are no more solutions a[illegible] answers no.
- Termination is not guaranteed. It is easy to write rules that cause an infinite recursion.
- The order in which solutions are produced depends on the order in predicates, in particular:
  - the order of the literals in the body of clause
  - the order of the predicates

A Simple Example

```
name(joe).
name(jane).
citizen(jane).
citizen(joe).
adult(jane).
voter(X):-
       name(X),
       citizen(X),
       adult(X).

?- voter(Y).
```



Another Example

Building categories

```
parent(building,farmbuilding).
parent(farmbuilding,barn).
parent(farmbuilding,silo).
parent(farmbuilding,house).
parent(barn,horsebarn).
parent(barn,cowbarn).
typeof(X,Y):- parent(Z,X),typeof(Z,Y).
typeof(X,Y):- parent(Y,X).

?- typeof(cowbarn,A).
```

Another Example: 3 Versions of French Nobleman

Version A

```
father(charles,
noble(henri).
noble(louis).
noble(charles).
noble(X):- father(Y,X),
           noble(Y).
```

Version C

```
father(charles,jean).
noble(X):- father(Y,X),
           noble(Y).
noble(henri).
noble(louis).
noble(charles).
```

Version B

```
father(charles,jean).
noble(henri).
noble(louis).
noble(charles).
noble(X):- noble(Y),
           father(Y,X).
```

?- noble(jean).

Source: R. Laganière

A Last Example

```
likes(peter,jane
likes(paul,jane)
conflict(X,Y) :- likes(X,Z),likes(Y,Z).

?- conflict(X,Y).
```

- How many solutions?

Summary

- Predicate calculu
  - Predicates
  - Horn clauses
  - Proof by Contradiction: Resolution
- Search Trees
  - Backtracking