CE 474 - LOGIC OF COMPUTER SCIENCE

Lecture Note 8 — Logic Programming in Prolog

Warm-up / Prediction

Warm-up (Prediction)

Before reading further, predict the answers Prolog will return for the query: `?- parent(bob, X).`

Which bindings for `X` do you expect? Try to list them in the order Prolog would produce them.

1. Learning Objectives

By the end of this handout you should be able to:

- Explain the Prolog execution model (SLD-resolution, unification, backtracking).
- Write simple Prolog programs (facts, rules) and run basic queries.
- Trace Prolog execution (a timeline-style trace) on example queries.
- Understand common pitfalls (cuts, negation-as-failure) and best practices.

2. What is Logic Programming?

Logic programming declares *what* is true, not *how* to compute it. Prolog (**PRO**gramming in **LOG**ic) is the canonical language of this paradigm.

Key idea

A Prolog program is a set of *clauses*: facts and rules. The Prolog engine answers queries by searching for proofs (goal-satisfaction) using unification and backtracking.

3. Prolog Syntax and Execution Model

Atoms, Variables, Terms

- **Atoms:** lowercase identifiers or quoted strings: `alice`, `'hello world'`.
- **Variables:** begin with uppercase letters: `X`, `Person`.
- **Compound terms:** `likes(alice,chocolate)`.

Clauses

- **Fact:** `parent(alice,bob).`
- **Rule:** `grandparent(X,Z) :- parent(X,Y), parent(Y,Z).`
- **Query:** `?- grandparent(alice,Who).`

Execution model (high level)

Prolog uses a leftmost selection rule and SLD-resolution:

1. Select leftmost goal.
2. Find a clause whose head unifies with the goal.
3. Apply the most-general unifier (MGU), replace the goal by the clause body (if any).
4. Continue. On failure, backtrack to previous choice points and try alternatives.

Checkpoint

Explain in one sentence what *unification* does and why backtracking is necessary in Prolog.

4. Unification

Definition. Unification finds a substitution (variable bindings) that makes two terms identical. The most general such substitution is the *most general unifier* (MGU).

Examples

- `X = 5` $\Rightarrow$ {`X` $\mapsto$ `5`}
- `parent(alice, Y) = parent(X, bob)` $\Rightarrow$ {`X`$\mapsto$`alice`, `Y`$\mapsto$`bob`}
- `likes(X,chocolate) = likes(vincent,Z)` $\Rightarrow$ {`X`$\mapsto$`vincent`, `Z`$\mapsto$`chocolate`}

Occurs-check note Many Prolog systems omit the occurs-check for efficiency; this can allow infinite structures (deliberate or accidental).

Practice

Give the MGU for: `f(X,g(Y))` and `f(a,g(b))`.
(Answer: {`X`$\mapsto$`a`, `Y`$\mapsto$`b`})

5. Backtracking

When a branch of the search fails, Prolog *undoes* variable bindings up to the latest choice point and tries the next alternative clause or binding. This depth-first, left-to-right search is simple and predictable — but it can loop if rules are not structured well.

Tip

Goal ordering and placing base/termination cases early in recursive rules helps avoid infinite search. Use the cut operator `!` sparingly.

6. Worked Example: Family Tree Program

Program

```
% Facts
parent(alice,bob).
parent(bob,carol).
parent(bob,david).

% Rule: grandparent
grandparent(X,Z) :-
    parent(X,Y),
    parent(Y,Z).

% Rule: sibling (sharing a parent)
sibling(A,B) :-
    parent(P,A),
    parent(P,B),
    A \= B.

% Rule: ancestor
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
```

Sample queries and expected results

- `?- grandparent(alice, Who).`
  Prolog will return `Who = carol ; Who = david`.
- `?- sibling(carol, S).`
  Prolog will return `S = david`.

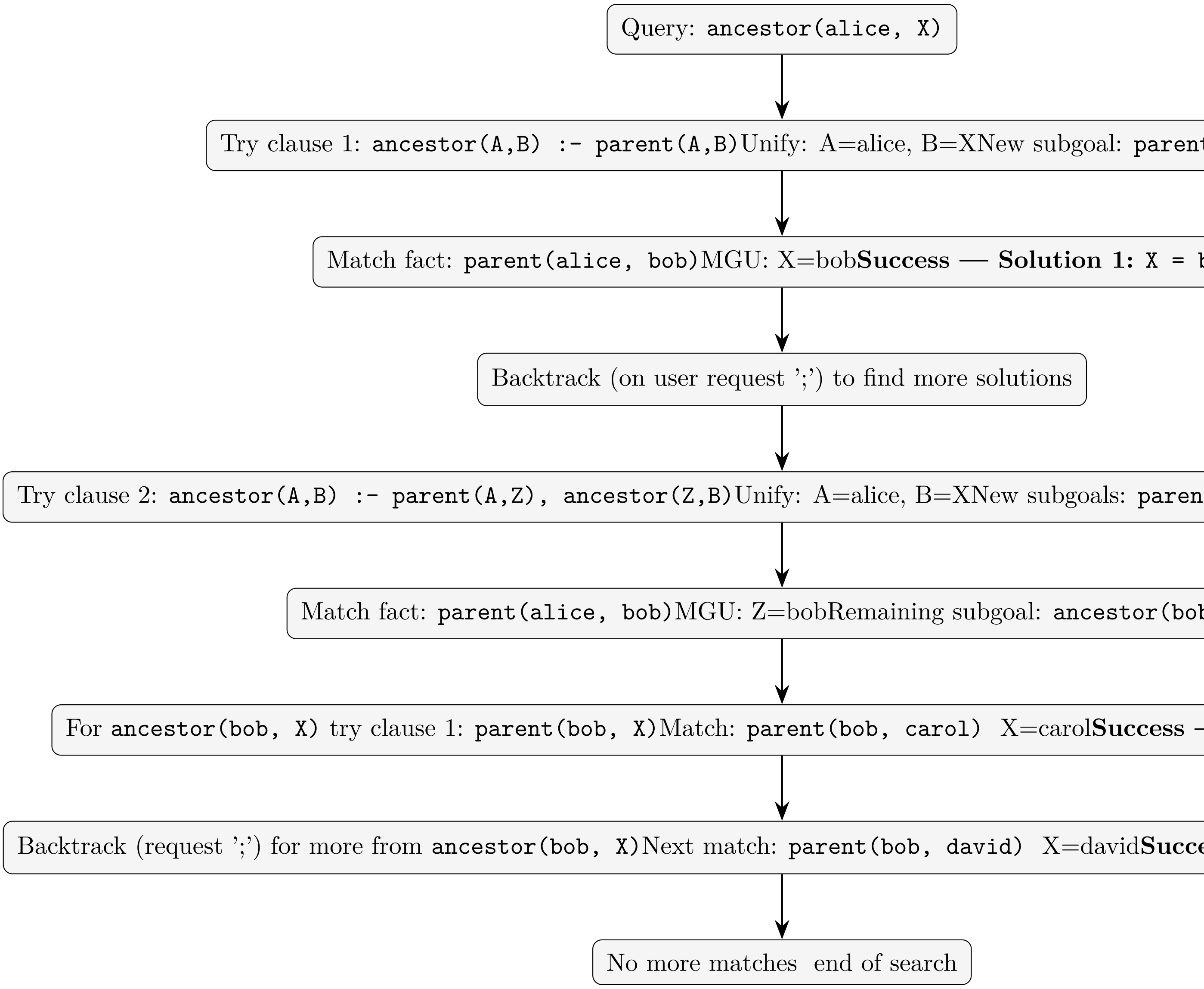
Try it in your interpreter

Load the program into SWI-Prolog and run:
`?- ancestor(alice, X).`
`?- ancestor(X, carol).`
Use `trace.` before the query to observe unification and backtracking.

7. Execution Trace (timeline) for `?- ancestor(alice, X).`

Below is a step-by-step timeline (vertical) showing how Prolog evaluates this query given the program above. The timeline follows SLD-resolution with the leftmost goal selection.

TikZ timeline (graphical)



ASCII-style timeline (simplified) — suitable for simple projector output:

```
1. Query: ancestor(alice, X)
2. Try clause1: ancestor(A,B) :- parent(A,B)
   - parent(alice, X) matches parent(alice, bob)
   => Solution 1: X = bob
3. Backtrack (user requests more)
4. Try clause2: ancestor(A,B) :- parent(A,Z), ancestor(Z,B)
   - parent(alice, Z) matches Z = bob
   - new subgoal: ancestor(bob, X)
5. For ancestor(bob, X):
   - clause1: parent(bob, X) matches carol => Solution 2: X = carol
   - backtrack: parent(bob, X) matches david => Solution 3: X = david
6. No more alternatives: end
```

Reflection

What would change in the trace if we placed the recursive ancestor clause before the base case in the program? (Hint: think about potential non-termination.)

8. In-Class Live Coding Exercise (full)

Load the family program and experiment.

Exercise: Live coding

```
% facts
parent(alice,bob).
parent(bob,carol).
parent(bob,david).

% ancestor rules
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
```

Tasks:

1. Load the file into SWI-Prolog (e.g., save as family.pl; consult via ?- [family].).
2. Run: `?- ancestor(alice, X).` Observe answers.
3. Use `trace.` then run the query to see the sequence of calls (unify, enter, exit, redo, fail).
4. Modify the program: deliberately put the recursive clause before the base clause and observe the tracer — does the query still terminate? Explain.

9. Common Pitfalls & Best Practices

- **Goal ordering matters.** Leftmost selection + DFS can loop if recursive goals are chosen prematurely.
- **Cut operator (!):** Use to prune search but beware it destroys declarative reading.
- **Negation as failure:** `Goal` succeeds if `Goal` cannot be proven; it's not classical negation.
- **Use deterministic predicates** (or cuts) when only one solution is expected.

10. Check Your Understanding

Exercises

1. Add `parent(david, elaine).` and query `?- ancestor(alice, X).` What new solutions appear and in what order?
2. Write a Prolog rule `descendant(X,Y)` (the inverse of ancestor) and test it.
3. Trace `?- ancestor(X, carol).` Describe the order Prolog tries bindings for `X`.

11. Summary & Takeaways

- Prolog programs are logical specifications — Prolog searches for proofs.
- **Unification** matches goals with clause heads; the MGU binds variables.
- **Backtracking** systematically explores alternatives; goal ordering affects performance and termination.
- Use `trace.` to learn how Prolog executes your program step-by-step.

References

- Sterling, L., & Shapiro, E. (1994). *The Art of Prolog* (2nd ed.). MIT Press.
- Clocksin, W. F., & Mellish, C. S. (2003). *Programming in Prolog* (5th ed.). Springer.
- Bratko, I. (2001). *Prolog Programming for Artificial Intelligence* (3rd ed.). Addison-Wesley.
- SWI-Prolog Development Team. (2024). https://www.swi-prolog.org/documentation