Building a Family Tree in Prolog built with Prolong

Samrat Baral

Date: April 25, 2025

- **Term:** Spring 2025 Second Bi-term
- **Class:** Advanced Programming Languages -MSCS-632-B01
- **Instructor:** Dr. Dax Bradley
- **University of the Cumberlands**

The GitHub repository, https://github.com/baralsamrat/MSCS632_Assignment_8

REPORT : - 📒 PDF

Screenshots

Capture-1.PNG 1

Objective

The assignment required encoding a small genealogy in Prolog, defining basic facts (*parent*, *male*, *female*) and writing rules for *grandparent*, *sibling*, *cousin*, plus generic *ancestor* / *descendant* relations that exploit recursion. A `run.sh` helper script was included to streamline execution on SWI-Prolog or GNU Prolog.

Approach

- **Knowledge representation** – Each person-to-person fact is a ground term `parent/2`, keeping the database minimal and index-friendly. Gender facts are optional but useful for later queries (e.g., *uncle*/*aunt* extensions).
- **Derived rules** – `grandparent/2` chains two `parent/2` predicates; `sibling/2` succeeds when two individuals share at least one parent and are different (`\=/2`). `cousin/2` is defined declaratively by elevating to the parents and applying `sibling/2`.
- **Recursion** – `ancestor/2` is the canonical example. The base clause covers a direct edge; the recursive clause walks up the tree one level at a time until the base triggers. Using this, `descendant/2` becomes a single inverse rule, showing how Prolog's backward chaining provides bidirectional reasoning almost "for free."
- **Query design** – Sample queries were selected to exercise each rule and to illustrate Prolog's nondeterminism (`;` to back-track). They double as unit tests: if future edits break a rule, expected answers will change.

queries (with expected results)

#	Query	Expected answers (`;` = press `;` for more)
1	`parent(john, X).`	`X = mary ; X = james.`

#	Query	Expected answers (`;` = press `;` for more)
2	`grandparent(john, X).`	`X = alice ; X = bob ; X = charlie.`
3	`sibling(mary, X).`	`X = james.`
4	`cousin(alice, X).`	`X = charlie.`
5	`ancestor(john, X).`	`X = mary ; james ; alice ; bob ; charlie.`
6	`descendant(X, jane).`	`X = mary ; james ; alice ; bob ; charlie.`

Challenges & Resolutions

1. **Infinite recursion risk** – Without care, a symmetric rule such as `sibling/2` defined both directions and reused in `cousin/2` can loop. I addressed this with the disequality guard `A \= B`, preventing self-unification, and by **not** making `sibling/2` explicitly symmetrical (Prolog's back-tracking will still yield both orders).
2. **Output ordering** – Prolog returns results in database order, which may confuse end users expecting alphabetical output. For teaching purposes I left the default order; in production one could wrap queries in `setof/3` for ordered, duplicate-free sets.
3. **Interpreter portability** – SWI-Prolog and GNU Prolog invoke files differently. The Bash script detects the interpreter and passes the right flags, ensuring grading works on any common environment.

Conclusion

The exercise highlights how a few declarative rules can model complex kinship networks and how Prolog's inference engine naturally handles recursion and pattern-matching. The main learning curve is thinking in relations rather than procedures; once that shift occurs, adding new rules (e.g., "uncle", "great-grandparent") becomes almost mechanical.