CS 4402-01 Comparative Programming Languages

Instructor: Professor Rajesh Soni

Name: Ryohei Hayashi

Learning Journal 8

**Overview of the Week**

This week focused on understanding the foundational concepts of logic programming and applying them through Prolog. The primary topics included unification, resolution, and recursion, alongside studying Prolog’s syntax and unique concepts. Specifically, we explored key components such as facts, rules, queries, lists, and functions.

In addition, we compared logic programming with imperative programming, examining the distinctions and unique characteristics of logic-based approaches. A discussion activity involved researching how logic programming excels at solving certain computational problems, particularly in fields like artificial intelligence and inference systems. Practical exercises included creating a “Hello World” program, defining facts and rules, implementing a recursive counting function, and solving the "Towers of Hanoi" problem in Prolog.

**Personal Reflections**

Learning Prolog this week introduced me to an entirely different approach to programming compared to imperative languages. In imperative programming, we define step-by-step procedures, whereas Prolog focuses on defining the "what" rather than the "how" by structuring programs around facts and rules and querying them for logical evaluation. Although this paradigm shift was initially challenging, it ultimately revealed Prolog’s potential for efficient problem-solving.

One aspect that impressed me was Prolog's ability to function like an inference engine. For example, querying `mortal(X)` can automatically deduce and list all matching entities based on the defined facts and rules. This feature highlights the power of logical evaluation without requiring the developer to specify detailed conditions manually.

Working on the “Towers of Hanoi” exercise was particularly enlightening. The recursive approach to moving disks using auxiliary pegs showcased the elegance of Prolog’s logical reasoning. While the process was conceptually challenging at first, observing Prolog’s efficient resolution and execution deepened my understanding of recursive programming.

That said, adapting to Prolog’s syntax and unique error messages, such as unification errors, was a steep learning curve. Debugging initially felt tedious, but through repeated practice, I became more adept at interpreting these messages and refining my queries.

**Topics Studied in Depth**

**Logic Programming and Unification**

Unification, a core concept in logic programming, is the process by which Prolog matches queries against facts and rules. For instance, running the query `mortal(X)` systematically explores all matches for `X` and returns the results sequentially. This automated process is strikingly different from imperative programming, where developers manually implement such logic. Observing how Prolog binds variables during unification, even in complex queries involving lists or functions, reinforced my appreciation for its efficiency.

**Recursion and Logical Resolution**

Recursion plays a central role in Prolog, compensating for the absence of iterative constructs like loops in imperative languages. The counting program demonstrated how recursion can achieve iterative behavior by repeatedly calling a function with modified parameters. Similarly, the "Towers of Hanoi" exercise illustrated how dividing problems into a base case and recursive cases enables efficient problem-solving.

Through these exercises, I learned that Prolog not only uses recursion to simplify complex problems but also employs resolution—matching queries with rules and facts—to derive logical conclusions. The recursive calls in the “Towers of Hanoi” program were particularly fascinating as they dynamically solved the puzzle step by step, showing Prolog’s power in handling intricate logic.

**Practical Problem-Solving with Prolog**

Practical applications of Prolog helped me understand its utility in real-world scenarios. Research revealed that Prolog is commonly used in fields like artificial intelligence, forensic evidence systems, and chatbot development. For example, its ability to evaluate logical rules and infer conclusions makes it suitable for tasks like decision-making in AI. This research provided a clearer picture of Prolog’s practical potential and how its strengths align with specific problem domains.

Word Count: 584

References

1. Ben-Ari, M. (2006). *Understanding programming languages.* Weizman Institute of Science.