LLMs for Prolog Coding

In this report, we'll investigate the use of an LLM (Large Language Model) for generating Prolog code. Specifically, we'll examine how Claude.ai handles Question 2 from Assignment 1.

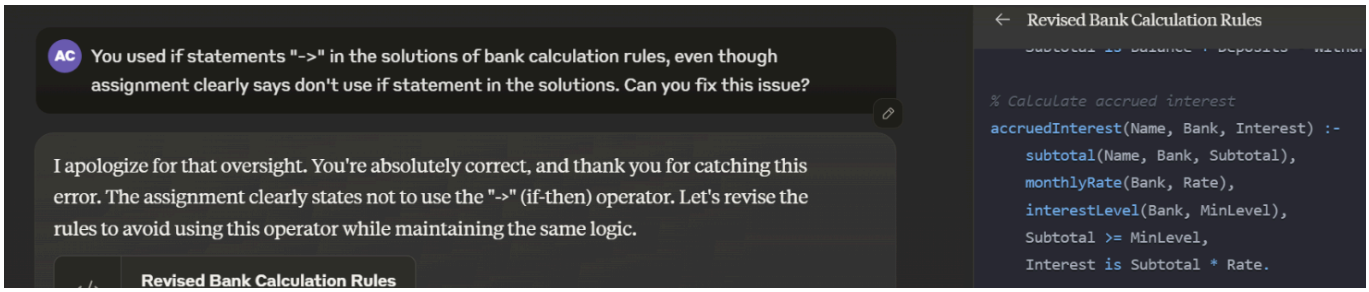
We approached the problem in three parts:

1. Creating a Knowledge Base: Initially, Claude.ai created a knowledge base, but the procedure clauses weren't consecutive. After pointing out this error, the LLM quickly fixed the issue and produced a correct knowledge base without any further mistakes.

```
hasAccount(alice, bmo, 1500).
totalDeposits(alice, cibc, 500).
totalDeposits(alice, bmo, 300).
totalWithdrawals(alice, cibc, 200).
totalWithdrawals(alice, bmo, 100).

% Person 2: Bob
hasAccount(bob, cibc, 1000).
hasAccount(bob, bmo, 3000).
```

2. Adding Rules: Claude.ai successfully created q2_rules, which largely solved the problem. However, it used if-then statements ("->") in two out of five rules, despite the assignment clearly stating that their use is not allowed. This oversight might be due to the LLM's training on a wide variety of Prolog code, where such constructs are common. After stating this mistake, Claude acknowledged the mistake and provided alternative solutions without using if-then statements.



3.Creating Queries: The LLM successfully generated all required queries without any issues. However, it also provided the expected results for these queries, which contained some calculation errors. For instance, for the query "endOfMonthBalance(bob, TotalBalance).", Claude stated the answer would be TotalBalance = 4523.2, while the correct answer is actually TotalBalance = 4519.2.

Overall, Claude.ai completed the assignment successfully, but it wasn't perfect. some guidance for organizing the knowledge base, following specific assignment constraints, and it made minor calculation errors. This experience showed that an LLM is not a solution creator but is a tool to create a solution.