PLP-Experiment-10

JHANVI PAREKH

60009210033

CSE(DS)

EXP-10 Case study on prolog and Haskell

Case study-Prolog

Title: Using Prolog for Employee Management System

Abstract: This case study explores the use of Prolog, a logic programming language, to develop an employee management system for a fictional company. The system handles various tasks such as employee record management, salary calculations, and leave tracking. By utilizing Prolog's logic-based querying and rule-based programming capabilities, the system provides efficient and accurate solutions for managing employee data.

Introduction: The employee management system aims to streamline administrative tasks related to employee records, payroll, and leave management. Prolog's unique features, such as pattern matching and rule-based programming, make it an ideal choice for building such a system. The case study discusses the design, implementation, and evaluation of the Prolog-based employee management system.

System Design: The system design involves identifying the key entities and relationships in the domain, which include employees, departments, salaries, leaves, and attendance. The design also incorporates the required functionalities, such as adding and updating employee records, calculating salaries, tracking leaves, and generating reports.
Implementation: The implementation phase covers the translation of the system design into Prolog code. Prolog's declarative nature allows expressing facts and rules to represent the knowledge base and the system's behavior. The case study elaborates on the specific Prolog predicates and rules used to implement various features, such as querying employee information, calculating salaries based on predefined rules, and managing leave requests.

System Evaluation: The case study evaluates the performance and effectiveness of the Prolog-based employee management system. It discusses the system's ability to handle large datasets efficiently, its responsiveness in generating reports and calculating salaries, and its accuracy in providing reliable information. The evaluation also compares the Prolog-based implementation with other traditional programming approaches to highlight the advantages and drawbacks of using Prolog in this context.

Results and Discussion: The results section presents the outcomes of the evaluation, including metrics such as system response time, accuracy of calculations, and user feedback. The discussion analyzes the strengths and limitations of using Prolog for the employee management system, highlighting its benefits in terms of concise code, expressiveness, and

ease of maintenance. It also addresses any challenges faced during the development process and suggests possible improvements or future directions.

Conclusion: The case study concludes by summarizing the key findings and emphasizing the suitability of Prolog for building an employee management system. It highlights the advantages of Prolog's logic programming paradigm in handling complex relationships and rules, making it a powerful tool for similar applications. The case study also encourages further exploration of Prolog and its potential applications in other domains.
References: The case study provides a list of references used during the research and development process, including relevant Prolog resources, literature on employee management systems, and related studies.
Note: This case study is fictional and serves as an example to illustrate the application of Prolog in building an employee management system.

Case study-Haskell

Title: Applying Haskell for Financial Portfolio Optimization

Abstract: This case study explores the use of Haskell, a functional programming language, for financial portfolio optimization. The study focuses on developing a portfolio management system that utilizes Haskell's strong static typing, immutability, and higher-order functions to efficiently analyze and optimize investment portfolios. By leveraging Haskell's functional programming principles, the system aims to provide robust and reliable portfolio recommendations based on various financial metrics and constraints.

Introduction: The financial portfolio optimization system aims to assist investors in constructing and managing investment portfolios that maximize returns while considering risk factors and investment constraints. Haskell's functional programming paradigm aligns well with this domain due to its emphasis on immutability, pure functions, and strong type system. This case study discusses the design, implementation, and evaluation of the Haskell-based portfolio optimization system.

System Design: The system design phase involves identifying the key components and functionalities required for portfolio optimization. This includes defining the types and structures to represent financial assets, portfolios, risk measures, constraints, and optimization algorithms. The design also considers the integration of external data sources and libraries for financial analysis.

Implementation: The implementation phase covers the translation of the system design into Haskell code. Haskell's static typing and purity enable the creation of robust and concise code that ensures type safety and minimizes runtime errors. The case study explains the usage of higher-order functions, algebraic data types, and type classes in implementing various aspects of the portfolio optimization system, such as portfolio construction, risk assessment, and optimization algorithms.

System Evaluation: The case study evaluates the performance and effectiveness of the Haskell-based portfolio optimization system. It assesses the system's ability to handle large

datasets, its runtime efficiency, and the accuracy of the optimization results. The evaluation also compares the Haskell implementation with other programming languages and approaches, highlighting the advantages and drawbacks of using Haskell in the context of financial portfolio optimization.

Results and Discussion: The results section presents the outcomes of the evaluation, including performance metrics, optimization results, and user feedback. The discussion analyzes the strengths of Haskell in terms of code expressiveness, type safety, and concise yet maintainable implementations. It also addresses any challenges faced during the development process and suggests potential improvements or future directions.

Conclusion: The case study concludes by summarizing the key findings and highlighting the suitability of Haskell for financial portfolio optimization. It emphasizes the benefits of using a functional programming language like Haskell, including the ability to express complex financial models with clarity, leverage type safety for robustness, and leverage higher-order functions for modularity and composability. The case study also encourages further exploration of Haskell and its application in other domains within finance and beyond.

References: The case study provides a list of references used during the research and development process, including relevant Haskell resources, literature on financial portfolio optimization, and related studies.

Note: This case study is fictional and serves as an example to illustrate the application of Haskell in financial portfolio optimization.