LAB EXERCISE 3 WEEK 4

Q1: Define type rules in the context of functional programming. How do type rules contribute to program correctness and safety? Provide programming examples to illustrate your explanation.

Q2: Discuss the differences between static typing and dynamic typing in functional programming. How do type rules manifest in each of these typing systems? Provide examples and analyze the advantages and disadvantages of each approach.

Q3: Explain the concept of type inference in functional programming languages. How does type inference work, and what role do type rules play in this process? Provide examples to demonstrate type inference in action.

Q4: Explore the concept of parametric polymorphism in the context of type rules. Provide a detailed explanation of parametric polymorphism, including its benefits and how it is enforced by type rules. Offer examples to illustrate your points.

Q5: Investigate the role of algebraic data types (ADTs) in enforcing type rules in functional programming languages. Discuss how ADTs contribute to type safety and provide examples demonstrating their usage.

Q6: Discuss the importance of type annotations in functional programming. How do type annotations aid in enforcing type rules and improving code readability? Provide examples to illustrate the impact of type annotations on code clarity.

Q7: Analyze the challenges associated with managing complex type systems in large-scale functional programming projects. Discuss strategies for mitigating these challenges and ensuring maintainability in such projects.

Q8: Evaluate the impact of dependent types on type rules in functional programming languages. Discuss how dependent types extend traditional type rules and analyze their practical implications for program correctness and expressiveness.

Q9: Investigate the concept of gradual typing and its relationship with type rules in functional programming. How does gradual typing blend static and dynamic typing paradigms, and what are the implications for enforcing type rules? Provide examples and discuss the trade-offs associated with gradual typing.

Q10: Create a context-free grammar (CFG) to represent the syntax of type declarations in a functional programming language. Your CFG should cover the declaration of primitive types such as integers and Booleans, as well as user-defined types. Provide production rules for defining type aliases and enumerations.

Q11: Design a context-free grammar (CFG) to describe function signatures in a functional programming language. Your CFG should include rules for specifying the return type and parameter types of functions. Ensure that your grammar supports both simple function signatures and those with multiple parameters.