Introduction

- **Evolving Digital Landscape**: Increased focus on security and reliability to protect sensitive data
- **Challenges in Software Testing**
  - Ensuring the **correctness** of **complex business logic** is critical.
  - Traditional **manual testing** approaches struggle with coverage and scalability
  - **Incomplete tests** can lead to **undetected defects** in critical applications.
- **Cause-Effect Graphs in Testing**
  - Provide a structured way to model business logic and dependencies.
  - Visualize relationships between causes (inputs) and effects (outputs) effectively.
  - Current methods for generating test cases are manual, inefficient, and lack scalability.

Objectives

- Develop an automated framework for transforming **textual** cause-effect graphs into:
  - Logical formulas.
  - Decision table.
  - Optimized test sets.
- **Improve test coverage and minimize manual effort** in validating business-critical applications.

Problem Statement

- Challenges in Testing Complex Systems
  - Ensuring **business logic accuracy** is difficult in modern, interconnected systems.
  - Traditional manual testing struggles with **scalability** and **completeness**.
- Cause-Effect Graphs
  - Strength
    - Provide a **structured** and **visual** representation of relationships between causes (inputs) and effects (outputs).
    - Aid in **understanding** and modeling complex dependencies.
  - Challenge
    - Transitioning from **visual** cause-effect graphs to **actionable** test cases is cumbersome.
    - Current methods rely on **manual** processes or hard-to-scale specialized solutions.

Methodology

- Key Steps

    - **Text-Based Input**: Users define cause-effect relationships in a structured textual format.
    - **Graph Model Creation**: Parse input into a cause-effect graph representing logical relationships.
    - **Logic Transformation**: Convert graphs into logical formulas (e.g., DNF).
    - **Optimization**: Simplify and refine formulas to reduce redundancy and complexity.
    - **Decision Table Generation**: Translate optimized logic into decision tables for test case derivation.
    - **Automated Test Case Creation**: Produce test sets for validating business logic.
- Highlights
    - Focus on minimizing manual intervention through automation.
    - Supports large and complex rule systems with performance optimizations
- Outcome
    - Seamless transition from textual inputs to actionable test cases.

Application Architecture

- Front-End
    - Built with React for **dynamic user interaction**.
    - Intuitive **graph visualization** and editing tools.
    - Real-time **syntax highlighting** and **error feedback** with Monaco Editor.
- Back-End
    - Implements core **business logic** processing.
    - Parses **textual inputs** into graph models.
    - Converts graph models into **logical formulas** and **decision tables**.
    - Manages communication with **Kotlin Language Server**.
- Deployment Infrastructure
    - **Dockerized** environment for easy scaling and distribution.
    - Automated **pipelines** via Azure DevOps for build, test, and deployment.
- Communication Flow
    - Two-Way Communication
    - Front-End <-> Back-End via HTTP/HTTPS and WebSockets.
    - Back-End manages multiple processes for real-time processing.

Core Features

- Textual Cause-Effect Graph Editing
    - Allows users to define complex **business logic** through a simple textual **representation**.
    - Supports **intuitive syntax** for defining causes, effects, and logical relationships.
- Real-Time **Syntax Checking** and **IntelliSense**
    - Integrated Monaco Editor with real-time feedback and error checking.
- Visual Graph Representation

    - Interactive and dynamic **graph visualization** for better understanding of business logic.
- Decision Table Generation
    - Convert **DNF** rules into testable **decision table** columns for optimized test case generation.
- GPT Export
    - Convert the decision table output into a **GPT** structure format to generate Boundary Value Analysis (BVA)-based test cases.

Results

- Performance Efficiency
    - Efficient performance even with **large datasets**, though slow client-side rendering for large results.
- Transformation Accuracy
    - 100\% **accuracy** for simple rules with a single cause.
    - **Reliable transformations** for nested rules and logical operators (AND, OR, NOT).
    - Effective error handling with **clear notifications** for syntax and logical errors.
- Scalability and Responsiveness
    - Concurrent Usage: Handles moderate **user concurrency** with minimal delay.
    - Performance drops with high concurrency (50+ simultaneous sessions), indicating need for further **scalability** solutions.
- Usability
    - Intuitive user interface with real-time error notifications.
    - Easy transformation from **cause-effect** graphs to **decision tables** and **test cases.**

Contributions

- Custom Graph-Based Language
    - Developed using **Kotlin's DSL** capabilities for flexible modeling of cause-effect relationships and logical rules.
- Modular React Front-End
    - Created an **interactive**, **user-friendly** interface for defining, visualizing, and transforming cause-effect graphs.
- Back-End Logic Processing
    - Built a **robust** server-side system to handle complex rule transformations into logical formulas and decision tables.
- Automated Development Process
    - **Azure Pipelines** for build processes, and deployment workflows.
    - Usage of **Docker** to ensure scalability and streamlined updates.
- Optimized Test Generation
    - Seamless conversion of logical rules into decision tables and optimized outputs.
- Strong Foundation for Further Development
    - Solid base for ongoing testing, scalability **improvements**, and **future features**.

Limitations

- Lack of Persistent Data Storage
    - Currently unable to save user progress and historical data, limiting reuse and continuity across sessions.
- Language Server Integration
    - While functional, further improvements are needed for enhanced syntax support, error handling, and helpful hints in the editor.
- Scalability Challenges
    - High concurrent usage and extensive client-server interactions require further enhancements for better scalability.

Conclusion

- Successfully developed a tool for cause-effect graph creation and testing.
- Validated functionality and performance with large datasets.

Future Work

- Persistent data storage, user and session management
- Improved processing capacity
- Enhanced editor features
- Integration with external test generation tools