**Project Assignment: Formal Methods in Software Engineering**

**Title: Banking System (Transaction Validation)**

**Student Information**

* **Full Name:** Danial Abid
* **Registration No:** 2021-GU-825
* **Roll No:** 23

**Ghazi University, Dera Ghazi Khan**

**Overview of the Problem**

The goal is to model and verify a banking system that validates transactions. The system must enforce the following requirements:

**Requirements:**

1. **No Overdraft**: The balance must never go negative.
2. **Unique Transaction IDs**: Every transaction must have a unique identifier.
3. **Positive Amounts**: Transaction amounts must be greater than zero.
4. **Transaction History**: The system must maintain a history of all transactions.

**System Constraints**

**Mutual Exclusivity:**

* Each transaction must be handled independently.

**Safety:**

* The balance must always remain non-negative.

**Predictable Behavior:**

* Transactions must follow the specified rules without deviation.

**No Invalid States:**

* Negative transactions, duplicate IDs, or negative balances must be prevented.

**Formal Representation**

We utilize **Z3**, a theorem prover, to define the system.

**Variables:**

1. balance: The current balance in the account.
2. txn\_id: Unique identifiers for transactions.
3. txn\_amount: The amount associated with a transaction.

**Constraints:**

1. balance >= 0 – Ensures no overdraft.
2. txn\_amount > 0 – Validates transaction amounts.
3. txn\_id is unique for every transaction.

**System Inputs and Outputs**

**Inputs:**

* Transaction IDs.
* Transaction amounts (positive values for deposits, negative values for withdrawals).

**Outputs:**

* Updated balance after each transaction.
* A record of all valid transactions.

**Implementation**

**Dynamic Transaction Validation using Z3**

from z3 import \*

class BankingSystemZ3:

def \_\_init\_\_(self, initial\_balance):

self.initial\_balance = initial\_balance

self.transactions = {} # Dictionary to store transactions {id: amount}

self.solver = Solver()

def add\_transaction(self, txn\_id, amount):

if txn\_id in self.transactions:

print(f"Transaction ID {txn\_id} already exists. Try again with a unique ID.")

return

self.transactions[txn\_id] = amount

print(f"Transaction {txn\_id} added: Amount = {amount}")

self.\_update\_constraints()

def remove\_transaction(self, txn\_id):

if txn\_id not in self.transactions:

print(f"Transaction ID {txn\_id} does not exist.")

return

del self.transactions[txn\_id]

print(f"Transaction {txn\_id} removed.")

self.\_update\_constraints()

def check\_constraints(self):

if self.solver.check() == sat:

print("All constraints satisfied. Final state:")

model = self.solver.model()

print(f"Final Balance: {model[self.balance]}")

else:

print("Constraints not satisfied. Possible issues with transactions.")

def \_update\_constraints(self):

self.solver.reset()

self.balance = Int('balance')

self.solver.add(self.balance >= 0)

balance\_expr = self.initial\_balance

for txn\_id, amount in self.transactions.items():

txn\_var = Int(f'txn\_{txn\_id}')

self.solver.add(txn\_var == amount)

self.solver.add(txn\_var > 0)

balance\_expr += amount

self.solver.add(self.balance == balance\_expr)

if \_\_name\_\_ == "\_\_main\_\_":

banking\_system = BankingSystemZ3(initial\_balance=1000)

while True:

print("\nMenu:")

print("1. Add Transaction")

print("2. Remove Transaction")

print("3. Check Constraints")

print("4. Exit")

choice = input("Enter your choice: ")

if choice == "1":

txn\_id = input("Enter transaction ID: ")

amount = int(input("Enter transaction amount: "))

banking\_system.add\_transaction(txn\_id, amount)

elif choice == "2":

txn\_id = input("Enter transaction ID to remove: ")

banking\_system.remove\_transaction(txn\_id)

elif choice == "3":

banking\_system.check\_constraints()

elif choice == "4":

print("Exiting program.")

break

else:

print("Invalid choice. Please try again.")

**Verification and Testing**

**Testing Scenarios:**

1. Add a valid transaction and verify the balance updates correctly.
2. Attempt to add a transaction with a duplicate ID and check for errors.
3. Add a transaction that would cause an overdraft and ensure constraints are not satisfied.
4. Remove a transaction and verify the updated balance and constraints.

**Tools Used:**

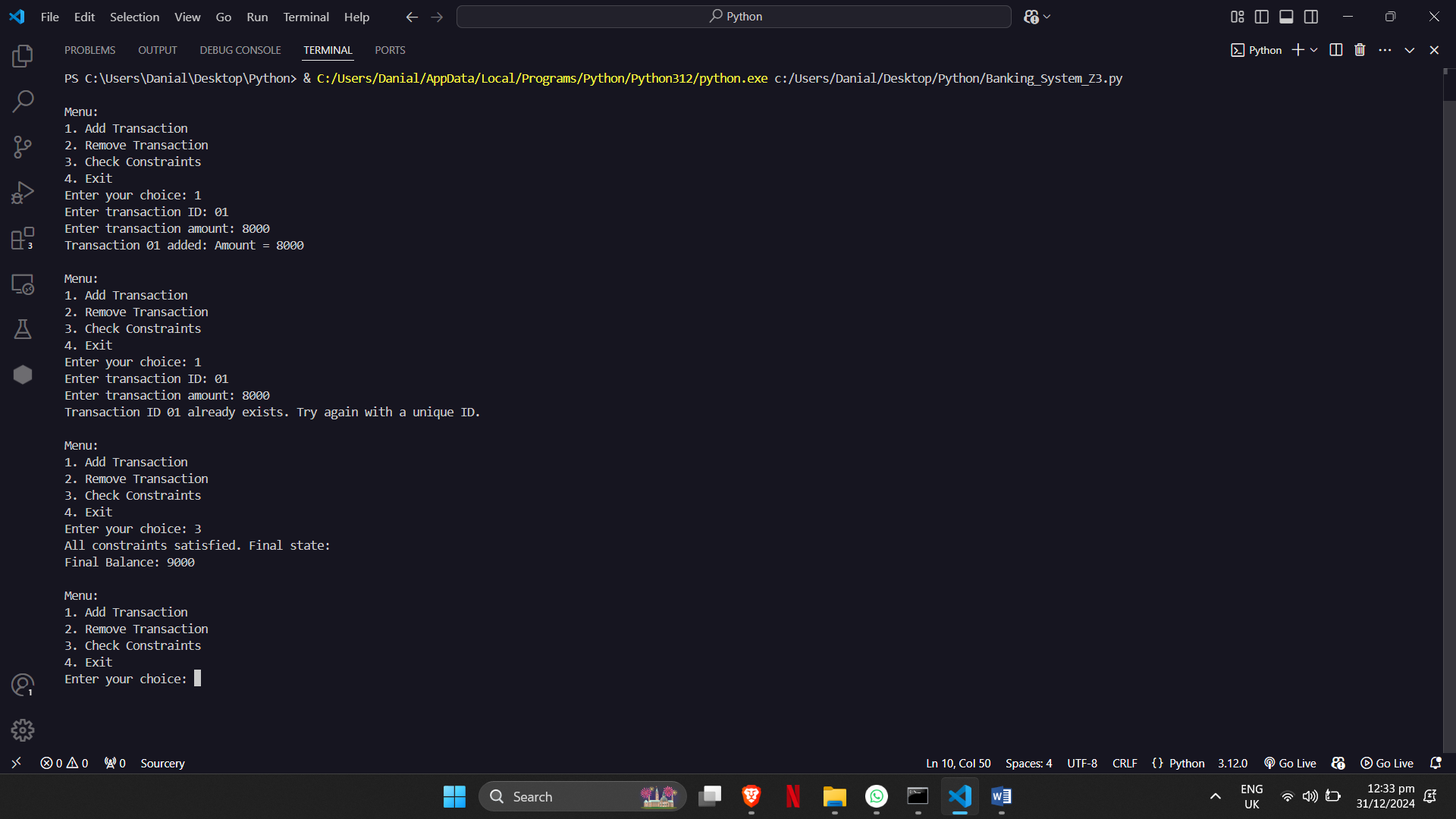
* Z3 Theorem Prover for formal verification.
* Python for dynamic interaction and simulation.

**Screenshots and Results**

**Code:**

****

**Output:**

****

**Challenges and Lessons Learned**

**Challenges:**

* Dynamically integrating Z3 constraints during runtime.
* Ensuring that invalid states are correctly detected and handled.

**Lessons Learned:**

* The importance of formal methods in preventing system errors.
* How Z3 can dynamically verify real-time system updates.

**Future Improvements**

* Add support for negative transactions (e.g., refunds).
* Implement a graphical interface for easier interaction.
* Use additional formal tools like Alloy to cross-verify the system.

**References**

* Z3 Documentation: <https://github.com/Z3Prover/z3>
* Formal Methods Tutorials

**GitHub Repository**

* Repository Link: [Banking System on GitHub](https://github.com/DanialAbid/Banking-System-Transaction-Based-by-Danial-Abid)