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

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

**Student Information**

* **Full Name:** Mujtaba Khan
* **Registration No:** 2021-GU-804
* **Roll No:** 02

**Ghazi University, Dera Ghazi Khan**

### Overview of the Problem

The objective is to design and verify a **Transaction System** that validates transactions while ensuring the following requirements are met:

#### Requirements:

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

### System Constraints

1. **Mutual Exclusivity**: Each transaction must be processed independently.
2. **Safety**: The balance must always remain non-negative.
3. **Predictable Behavior**: Transactions must strictly adhere to defined rules.
4. **No Invalid States**: Prevent negative transaction amounts, duplicate IDs, and overdrafts.

### Formal Representation

The system utilizes **Z3**, a theorem prover, to define and enforce constraints.

#### Variables:

* balance: The current account balance.
* txn\_id: Unique identifiers for each transaction.
* txn\_amount: The amount associated with a transaction.

#### Constraints:

1. balance >= 0 — Ensures no overdraft occurs.
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 for deposits, negative for withdrawals).

#### Outputs:

* **Updated balance** after each transaction.
* **Transaction history** of all valid transactions.

### Implementation

#### Dynamic Transaction Validation using Z3

from z3 import \*

class TransactionSystemZ3:

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\_\_":

Transaction\_system = TransactionSystemZ3(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: "))

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

elif choice == "2":

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

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

elif choice == "3":

Transaction\_system.check\_constraints()

elif choice == "4":

print("Exiting program.")

break

else:

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

### Verification and Testing

#### Testing Scenarios:

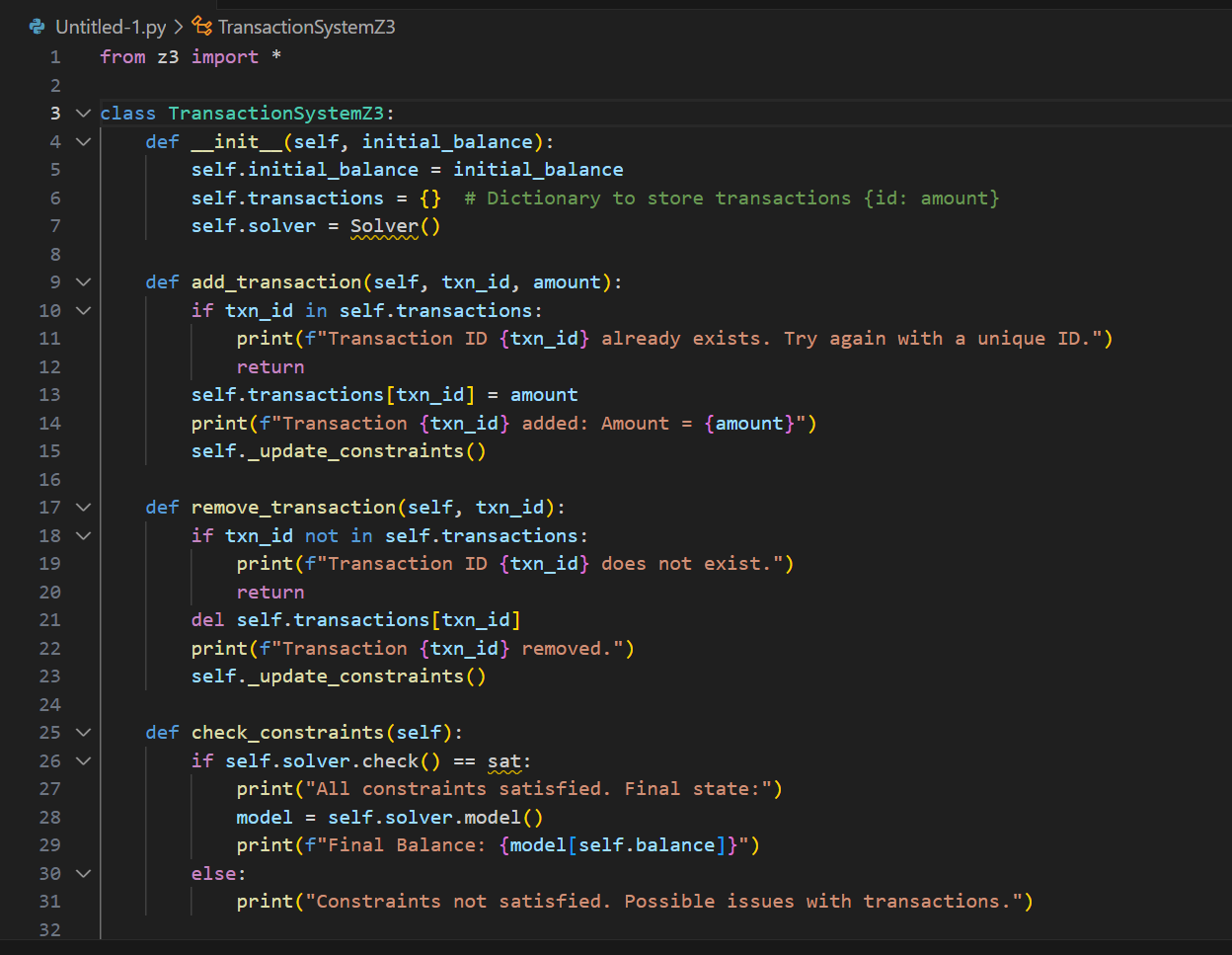
1. Add a valid transaction and verify that the balance updates correctly.
2. Attempt to add a transaction with a duplicate ID and ensure the system detects the error.
3. Add a transaction that would cause an overdraft and verify that constraints are not satisfied.
4. Remove a transaction and confirm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

#### Tools Used:

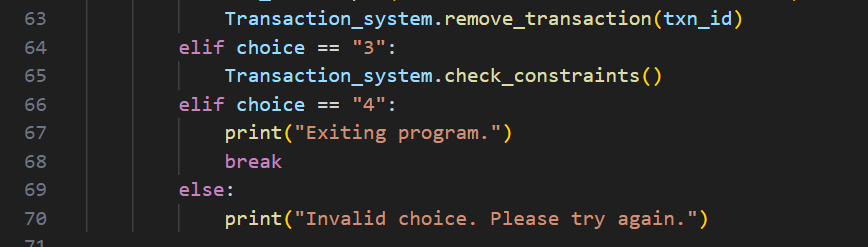
* **Z3 Theorem Prover** for formal verification.
* **Python** for dynamic interaction and simulation.

**Screenshots and Results**

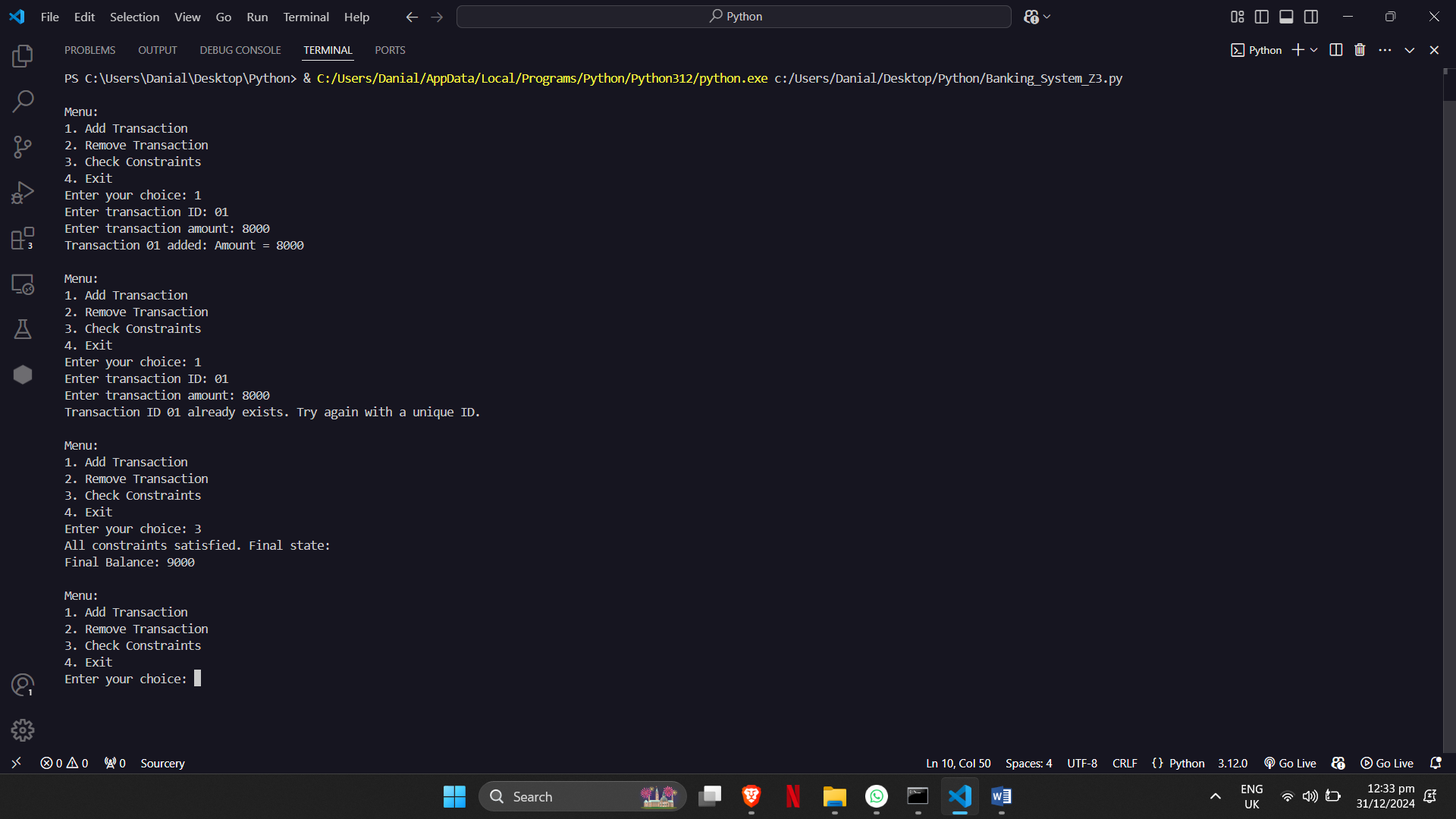
**Code:**







**Output:**

****

### Challenges and Lessons Learned

#### Challenges:

* Dynamically updating Z3 constraints during runtime.
* Ensuring invalid states are properly detected and handled.

#### Lessons Learned:

* The importance of formal methods in preventing system errors.
* Practical applications of Z3 for real-time system verification.

### Future Improvements

1. Add support for negative transactions (e.g., refunds).
2. Implement a graphical interface for user-friendly interactions.
3. Explore additional tools like **Alloy** for cross-verifying the system.

**References**

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

**GitHub Repository**

* Repository Link:[https://github.com/mujt786/Transaction-System-Based-by-Mujtaba-Khan](https://github.com/mujt786/Transaction-System-Based-by-Mujtaba-Khan" \o "https://github.com/mujt786/Transaction-System-Based-by-Mujtaba-Khan)