An Attempt to Formalize ACID in Transaction

This paper presents a formal framework to guarantee atomicity and isolation in transactional systems using minimal, well-defined rules and preconditions. By focusing on sequential state transitions and synchronized operation sets, together with durability as pre-condition, we demonstrate how atomicity and isolation naturally lead to consistent transactional systems. The framework integrates rollback handling as an inherent part of state transitions and addresses challenges in practical implementation.

1. Introduction

In transactional systems, ensuring correctness requires atomicity and isolation as core properties. Atomicity guarantees that operations either complete fully or leave no effect, while isolation ensures that concurrently running transactions do not interfere. Together, these properties naturally lead to consistent and dependable outcomes. This work presents a minimal but general framework to formalize atomicity and isolation using local rules, preconditions, and state evolution principles. The framework also seamlessly integrates rollback handling by treating it as a natural part of state transitions.

2. Core Notations and Definitions

- **Operations:** Actions or transformations denoted as O_1, O_2, and O_3 that update the system's state.
- **State Variables:** Denoted as S_0, S_1, S_2, and S_3, representing the condition of the system at different stages of the operation sequence.
- **Addition (+), Subtraction (–):** Denotes the application of an operation to a state. For example, $S_1 = S_0 + O_1$ means applying operation O_1 to base state S_0 to derive state S_1. $S_1 + O_1^{-1}$ undoes the effect of operation O_1.
- **Equality (=):** Expresses consistency relationships, not an operand, between states and operations.
- **Operation Set:** A collection of sub-operations that must complete simultaneously. If their completion times differ, they are treated as separate operations. With this, Operation is defined as $Operation \in \{Operation | OperationSet\}$.

---

3. Preconditions

To guarantee correctness, the following preconditions must hold:

1. **Durability:**
   State does not change unless some operations are made.

2. **Ordering of Operations:**

$$O_1 < O_2 < O_3$$

Operations are applied sequentially, respecting this order.

3. **State Birth Times:**
   States are initialized when their respective operations are completed.

$$birthof(O_1) = S_1, \quad birthof(O_2) = S_2, \quad birthof(O_3) = S_3$$

4. **Previous State:**
   S_0 is the previous state before occurrence of O_1

These preconditions ensure that state transitions and dependencies between operations are well-defined.

---

4. Core Rules Governing State Evolution

The system's state evolves according to the following rules:

1. **Initialization of S_1:**

$$S_1 = S_0 + O_1$$

State S_1 is derived from the base state S_2 through the application of operation O_1.

3. **Update of Y:**

$$S_2 = S_1 + O_2$$

State $S_2 $ depends on the successful completion of state S_1 and operation O_2.

5. **Update of S_3:**

$$S_3 = S_2 + O_3$$

The final state S_3 is derived from state S_2 after applying operation O_3.

These rules ensure that partial or intermediate updates do not propagate, maintaining atomicity at each step.

---

5. Generalized Invariant for Multiple States and Operations

The following invariant ensures correct state evolution across any number of operations and states:

$$\forall i, j \quad (i < j) \implies state_j = state_i + operation_j$$

This relationship guarantees that the most recent valid state is always used when applying the next operation.

Alternatively, the following equation expresses more holistically with use of functions.

$$\forall o \in operations, \quad positionAt(operations, o) = positionAt(states, stateAfter(o))$$

where

- **Ordered sets** $operations$ and $states$:
  - $operations = \{o_1, o_2, \ldots, o_n\}$ is the ordered set of events.
  - $states = \{stateAfter(o_1), stateAfter(o_2), \ldots, stateAfter(o_n)\}$ is the ordered set of resulting states.

- **Functions**: $stateAfter(e)$ is the resulting state immediately after processing the event e. $positionAt(oset, e)$ is the position of e within the ordered set $oset$.

---

6. Rollback Handling as Inverse Operations

Rollback is treated as the inverse of forward operations:

1. **Inverse State Transitions:**

$$S_2 = S_3 + O_3^{-1}, \quad S_1 = S_2 + O_2^{-1}, \quad S_0 = S_1 + O_1^{-1}$$

Rollback restores the system to its previous state by applying the inverse of the corresponding forward operations.

3. **Simultaneous Completion Requirement:**
   Rollback operations within an operation set must complete together to maintain atomicity. If simultaneous rollback is not possible, the rollback operations must be treated as separate and independently atomic.

---

7. Atomicity and Isolation

- **Atomicity:** Ensures that each operation set completes fully or not at all. Partial updates are not allowed to propagate.
- **Isolation:** Guarantees that concurrently running transactions or operations do not interfere. Isolation builds on atomicity by considering interactions between multiple actors or processes.

---

8. Practical Considerations and Optimizations

While the framework ensures correctness, implementing it in real-world systems may present challenges. Potential optimizations include:

- **Breaking large operations into smaller atomic sets.**
- **Using commit protocols to synchronize operation completion across distributed systems.**
- **Applying optimistic concurrency control where strict locking is impractical.**

---

9. Conclusion

This framework provides a structured and minimal approach to formalizing atomicity and isolation. By enforcing well-defined local rules and sequential state transitions, it naturally integrates rollback handling and ensures consistent state evolution. Future work could extend this framework to address performance optimizations and failure handling in distributed environments.