2. FORMAL SYSTEM SPECIFICATION

Formal system specification in software engineering refers to the process of precisely describing the behavior, properties, and requirements of a software system using formal methods and mathematical notations. It aims to eliminate ambiguity, clarify system requirements, and enable formal analysis and verification of system properties.

Here are some key aspects of formal system specification:

Formal Methods:

Formal methods are mathematical techniques used to specify, model, and reason about software systems. They provide a rigorous and precise approach to system specification, ensuring clarity and unambiguous representation. Common formal methods used for system specification include Z notation, B method, and Specification and Description Language (SDL).

Mathematical Notations:

Formal system specification often involves using mathematical notations to express system requirements, constraints, and behaviors. These notations can include predicate logic, set theory, algebraic expressions, temporal logic, and process calculi, dependingon the chosen formal method.

Language Constructs:

Formal specification languages provide constructs and syntax for expressing system properties. For example, Z notation includes constructs for defining data types, predicates, schema definitions, and invariants. These language constructs allow for a systematic and structured representation of the system.

Requirements Capture:

Formal system specification helps in capturing and refining system requirements. It enables the identification of inconsistencies, ambiguities, and gaps in

Requirements early in the development process. Formal methods facilitate the translation of informal requirements into precise and unambiguous specifications.

Verification and Validation:

Formal system specification enables formal analysis, verification, and validation of system properties. Formal methods support the use of automated tools and theorem provers to check the consistency, correctness, and completeness of system specifications. This helps in identifying design flaws, detecting logical inconsistencies, and verifying that the system meets its intended requirements.

Refinement:

Formal specification allows for stepwise refinement of system specifications. System requirements can be progressively refined into more detailed specifications, ensuring that the design and implementation faithfully adhere to the original requirements.

Documentation:

Formal system specifications serve as a valuable source of documentation. They provide a clear and unambiguous representation of system requirements, constraints, and behavior, which aids in system understanding, maintenance, and future enhancements.

Formal system specification plays a crucial role in software engineering by enhancing the quality, reliability, and maintainability of software systems. It supports rigorous analysis, verification, and validation, leading to more dependable and trustworthy software designs.

3. FINITE STATE MACHINE

- A state machine is a software computation model. It's just a model to solve a complex application, and it comprises a finite number of states. Hence it is also called a Finite State Machine. States are nothing but situations of your application (different situations).
- Since states are finite, there is a finite number of transitions among the states.

Transitions are triggered by the incidents or input events fed to the state machine. An FSMis an event-driven reactive system.

- A state machine also produces an output. The output produced depends on the current state of the state machine sometimes, and sometimes it also depends on the input events fedto the state machine.

Benefits of using state machines:

- It is used to describe situations or scenarios of your application (Modelling the lifecycle of a reactive object through interconnections of states.

- FSMs are helpful to model complex applications that involve lots of decision-making,producing different outputs, and processing various events.
- State machines are visualized through state machine diagrams in the form of statecharts, which helps to communicate between non-developers and developers.
- FSM makes it easier to visualize and implement the changes to the behavior of theproject.
- Complex application can be visualized as a collection of different states processing afixed set of events and producing a fixed set of outputs.
- Loose coupling: An application can be divided into multiple behaviors or state machines, and each unit can be tested separately or could be reused in other applications.
- Easy debugging and easy code maintenance.
- Scalable
- Narrow down the whole application completely to state-level complexity, analyze andimplement.

Different types of state machines:

1. Mealy machine
2. Moore machine
3. Harel state charts
4. UML state machines

Some of these state machines are used for software engineering, and some state machines arestill being used in digital electronics, VLSI design, etc.

Types of Finite State Machines

Mealy machines are a type of simple state machine where the next state is determined by thecurrent state and the given input.

The next state is determined by checking which input generates the next state with the current state. Imagine a button that only works when you're logged in.

That button is a state machine with login as the current state.

The new state is determined by logging in. Once you're logged in, the next state is determined by the current state which is logged in.

Moore Machines

A Moore State Machine is a type of state machine for modeling systems with a high degree of uncertainty. In these systems, predicting the exact sequence of future events is difficult. As a result, it is difficult to determine the next event.

A Moore model captures this uncertainty by allowing the system to move from one state to another based on the outcome of a random event.

A Moore model has many applications in both industry and academia. For example, it can be used to predict when a system will fail or when certain events will occur with a high probability (e.g., when there will be an earthquake).

It can also be used as part of an optimization algorithm when dealing with uncertain inputs (e.g., produce only 1% more product than standard).

In addition, Moore models are often used as rules for automatic control systems (e.g., medical equipment) that need to respond quickly and accurately without human intervention.

Turing Machine

The Turing Machine consists of an input tape (with symbols on it), an internal tape (which corresponds to memory), and an output tape (which contains the result).

A Turing Machine operates through a series of steps: it scans its input tape, reads out one symbol at a time from its internal tape, and then applies this symbol as a command (or decision) to its output tape. For example: "If you see 'X' on the input tape, then print 'Y' on the output tape."

The input tape can be considered a finite set of symbols, while the internal and

output tapes areinfinite. The Turing Machine must read an entire symbol from its internal tape before it can move its head to the next symbol on the input tape.

Once it has moved its head to the next symbol, it can read that symbol out of its internal tape and then move to the next symbol on its input tape.

This process continues until no more input or output symbols are left in the Turing Machine's internal or external tapes (at which point, it stops).

State Machine Use Cases

State machines are helpful for a variety of purposes. They can be used to model the flow of logic within a program, represent the states of a system, or for modeling the flow of events in a business process.

There are many different types of state machines, ranging from simple to highly complex. A few common use cases include:

Modeling Business Workflow Processes

State machines are ideal for modeling business workflows. This includes account setup flows, order completion, or a hiring process.

These things have a beginning and an end and follow some sort of sequential order. State machines are also great for modeling tasks that involve conditional logic.

Business Decision-Making

Companies pair FSMs with their data strategy to explore the cause and effect of business scenarios to make informed business decisions.

Business scenarios are often complex and unpredictable. There are many possible outcomes, and each one impacts the business differently.

A simulation allows you to try different business scenarios and see how each plays out. You can then assess the risk and determine the best course of action.

ROHINI COLLEGE OF ENGINEERING AND TECHNOLOGY
PALKULAM, KANYAKUMARI
OBSERVE OPTIMIZE OUTSPREAD