Chapter 3 Exercises Solutions

Question	Answer	
3.1	Purpose	Petri nets are a graphical modeling language used to describe the behavior of concurrent, distributed systems. They excel at representing systems with: Concurrency: Multiple events happening simultaneously. Synchronization: Events needing to occur in a specific order. Resource sharing: Limited resources being used by different parts of the system.
	Use Cases	System designers: They use Petri nets to model and analyze system behavior during the design phase. System analysts: They use Petri nets to understand existing systems and identify potential problems. Software developers: They use Petri nets to verify the correctness of concurrent software.
	Concepts	Places: Represented by circles, they denote states or conditions within the system. They can hold tokens. Transitions: Represented by rectangles, they represent events or actions that cause the system to change from one state to another. Tokens: Represented by black dots within places, they signify resources, data items, or control signals flowing through the system. Arcs: Directed arrows connecting places and transitions. They define the flow of tokens and how events are triggered.
	Relations	Arcs connect places to transitions (input) and transitions to places (output). A transition can only fire (execute) if all its input places have sufficient tokens. Firing a transition removes tokens from input places and adds them to output places. Tokens can be simple or complex, carrying additional information about the system state. Multiple transitions can be enabled simultaneously, representing concurrent events in the system.

<table>
<tr><td></td><td><table><tr><td></td><td>Arcs connecting multiple places to a transition enforce synchronization, requiring all those places to have tokens for the transition to fire.</td></tr><tr><td>Examples</td><td>Widely used in embedded systems, cyber-physical systems.</td></tr></table></td></tr>
<tr><td>3.3</td><td>Just remove the ‘initial’ connection from FiniteStateMachine to State.

Add Boolean attribute ‘initial’ inside State.</td></tr>
<tr><td>3.11</td><td>Properties to Test:
Completeness: Does the meta-model capture all the essential concepts and relationships needed to represent a finite-state machine effectively? Are there any missing elements or functionalities?
Consistency: Are the concepts and relationships within the meta-model unambiguous and well-defined? Are there any naming conflicts or inconsistencies in how elements are represented?
Accuracy: Does the meta-model accurately reflect the intended behavior and structure of finite-state machines? Does it align with established practices or domain-specific requirements?
Usability: Is the meta-model easy to understand and use? Can developers readily create and manipulate state machines based on this meta-model?
Test Cases:
Basic State Machine: Can you create a simple state machine with states and transitions using the meta-model elements (Model, Machine, State, Transition)?
Complex State Machine: Can you model a state machine with multiple states, transitions, and potentially nested states (hierarchies) using the meta-model?
State Attributes: Can you define attributes (like an isActive flag) on states within the meta-model?
Transition Triggers/Actions: Can you associate triggers (events) and actions with transitions in the meta-model?
State Machine Validation: Can you define rules or constraints within the meta-model to ensure well-formed state machines (e.g., every state must have an outgoing transition)?</td></tr>
</table>