Part 1: Research Component  
  
1. Definition and Structure

Behavior tree:

A Behavior Tree (BT) is a hierarchical, modular, and reusable decision-making structure used in artificial intelligence (AI), robotics, and game development to model the behavior of agents (e.g., NPCs, robots, or game characters). It is composed of nodes that represent tasks, conditions, or control logic, organized in a tree-like structure. Behavior Trees are widely used because of their readability, flexibility, and scalability.

Behavior tree answers the question: “What to do next?”

Foundational elements of behavioral trees:  
Nodes:

Are the building blocks of behavioral trees, categorized into 3 types:  
Control nodes: Decides which child node to execute.

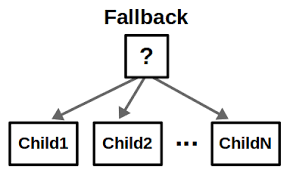
Execution nodes: Perform actions.

Decorator nodes: modify the behavior of a single child node.

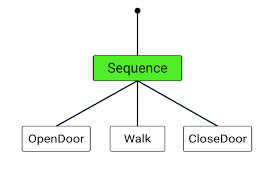
Note that tick is the synonym used to say that a parent node started an action : it ticked the action.

What really happens is:  
the parent node sends a signal for the other action to take place, the action replies with success, failure or running. And after that, the ancestor decides what to do next.

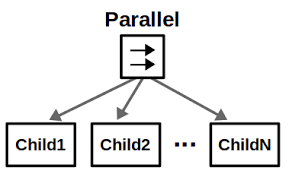
Control Nodes:  
Fallback: if there are two options, like moving forward or moving backward, it tries to move forward first, if it fails, it moves backward.



Sequence: Obviously, as the name says, if there are 2 steps, like moving forward and turning, it first moves forward, then it turns. If moving forward failed, then it won’t turn.

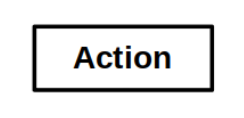


Parallel: executes multiple child nodes at the same time.



Execution nodes:

Action: Performs an action (move, rotate …)



Condition: checks a specific condition (is there a wall?)



Decorator nodes:

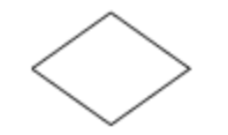
Inverter: if the result is failure, it returns success and vice versa. If running, it returns running.

Repeat: repeats the action N times, as long as it returns success, if it returns failure, it breaks out of the loop, if running, it returns running.

Force-Success: always return success unless it is running it returns running.

Force-Failure: always return failure unless it is running it returns running.

Retry: repeats the action N times, as long as it returns failure, if it returns success, it breaks out of the loop, if running, it returns running.



Blackboard: is a shared memory space used to store and exchange data between nodes.

Allows nodes to communicate and make decisions based on shared information.

Tick System: is the process of traversing and updating the tree. The tree is "ticked" from the root node, and the tick propagates through the tree based on the logic of the nodes. Each node returns a status (e.g., Success, Failure, Running) to its parent node.

Status The result of a node's execution. Success: The node completed its task successfully. Failure: The node failed to complete its task. Running: The node is still executing and will continue in the next tick.

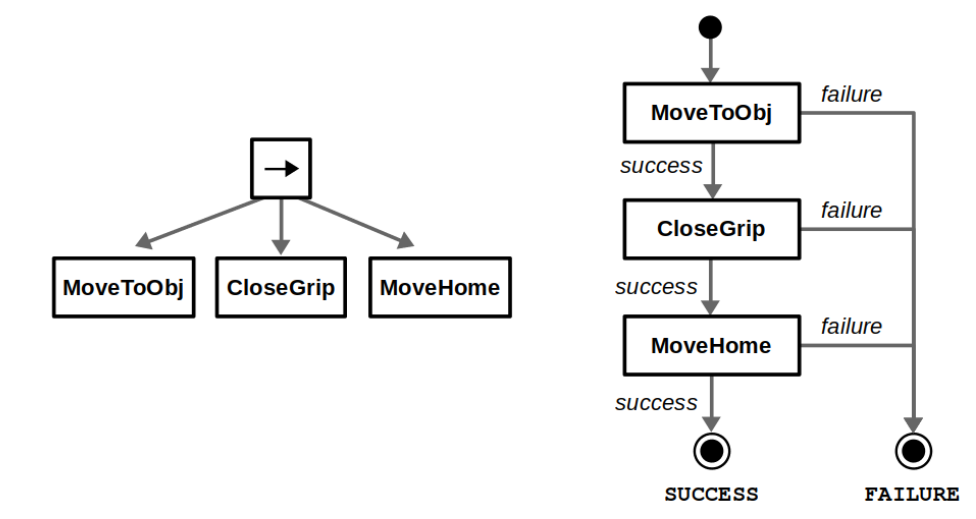
State Machines:

A state machine might be used to describe the different actions that a robot can take, and the conditions under which it will transition between those actions.

In a simple path-following robot, this might mean having a “wait” state, which transitions to a “run” state on receiving a command, and back to a “wait” state once it completes the command.

State machine vs Behavior tree:

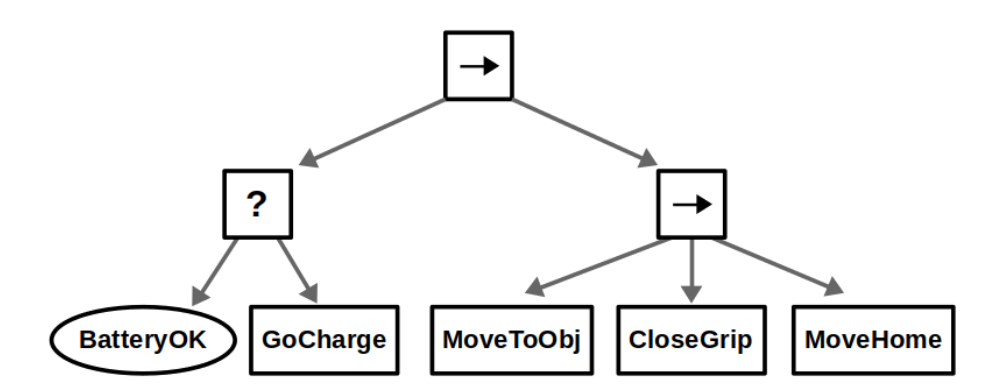
BTs are easier to compose and modify while FSMs have their strength in designing reactive behaviors.

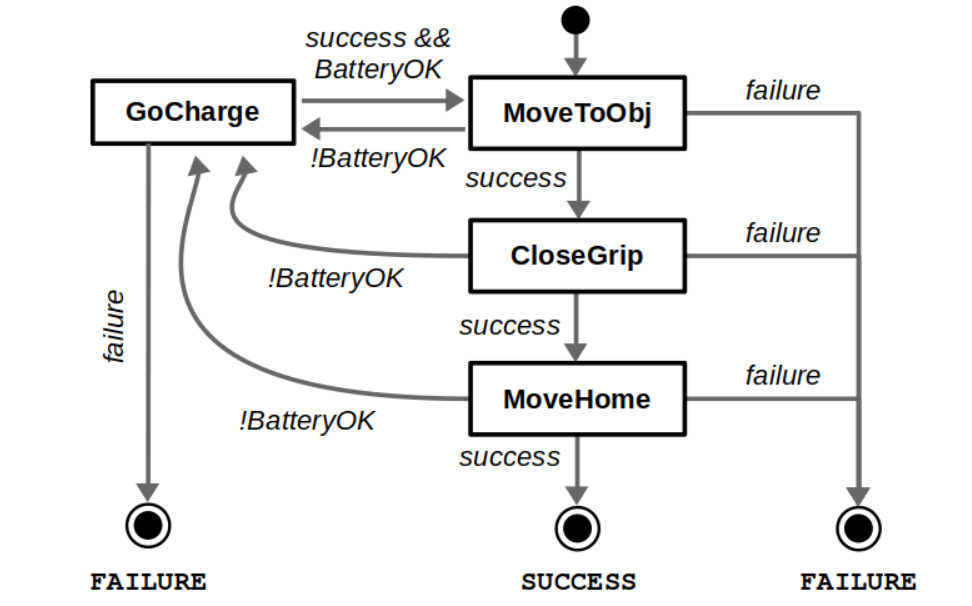


As we can see in terms of modularity, BTs are so much simpler.

This example shows how a sequence of actions are written using BT on the left, and SM on the right.

In terms of reactivity, imagine that the robot needs to charge, and the robot is in the phase of sequence in BT, we’ll need to add check for the charging state in the beginning of the sequence, which is not reactive (since if it is in the beginning, it has to be done first it can’t just go back to it whenever it wants), in SM, this is not a problem, since SM allows this reactivity by allowing the definition of transitions between any two states.





BT libraries:

LibFT

BehaviorTree.CPP

BTFramework

PyBehaviorTree

Aiobehavior

py\_trees

2- Applications:

Autonomous Robots & AI Decision-Making:

Behavior Trees are used in robotic decision-making for autonomous navigation, object manipulation, and human-robot interaction.

Example:

ROS2 Navigation Stack (Nav2) uses BehaviorTree.CPP for high-level decision-making in robot movement. Robots use BTs to select routes, avoid obstacles, or decide whether to replan paths dynamically.

Example Implementation: TurtleBot 4 (ROS2) uses BTs to navigate and interact with the environment. Boston Dynamics Spot robot integrates BTs for movement and task execution.

Industrial Automation & Factory Robots:

Behavior Trees help in automating industrial robots for tasks like assembly, inspection, and picking/placing objects.

Ex:

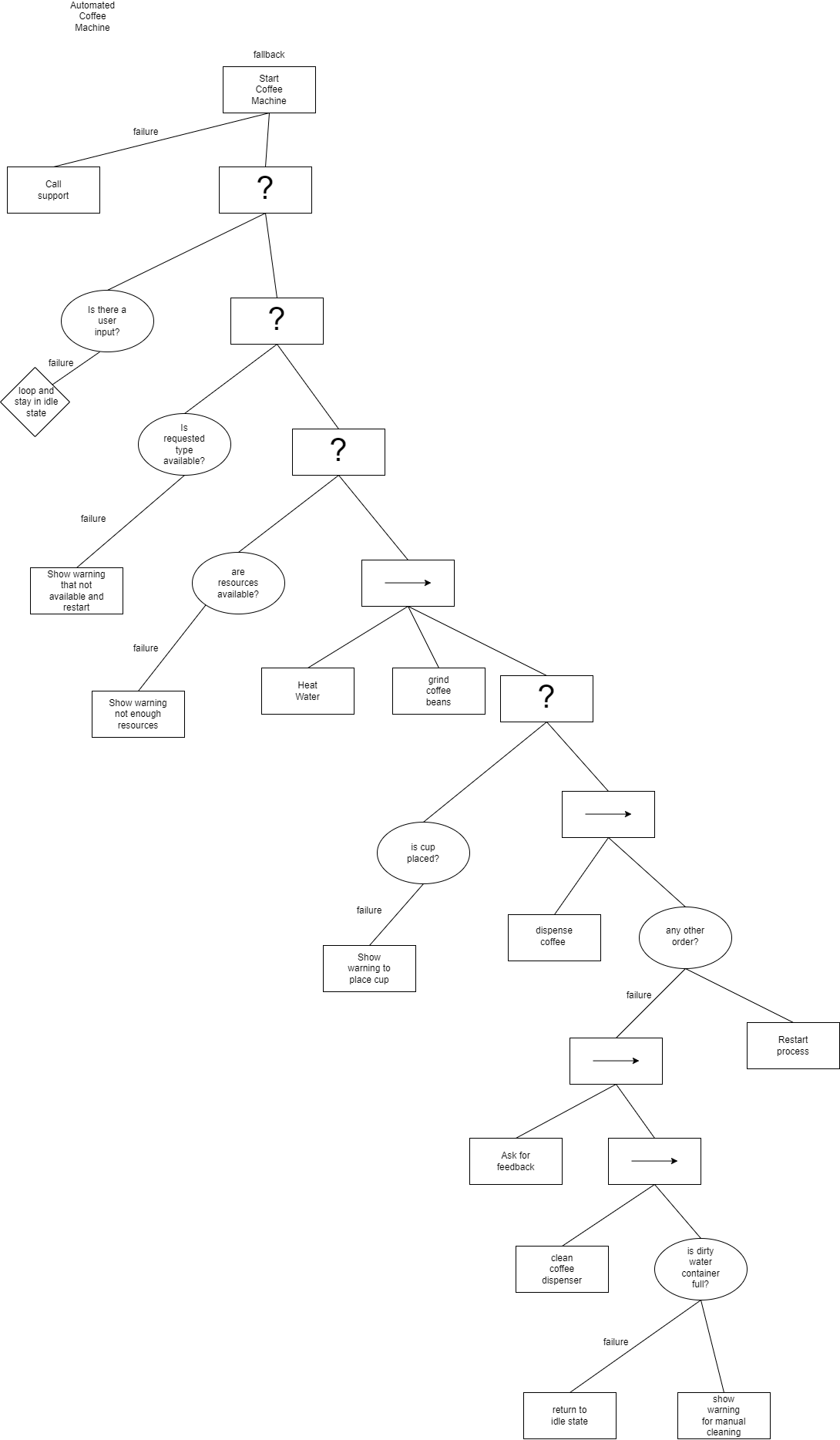
Siemens & ABB Robotics use BTs for manufacturing robots that dynamically adjust behaviors based on sensor inputs

Example Implementation:

A pick-and-place robot using BTs to check if an object is available, grasp it, and place it in the correct location.

Robots in Tesla’s Gigafactories dynamically adjust to conveyor belt speeds.

Part 2:



In this Behavior Tree, I implemented the automated coffee machine behavior map,

The flow is as follows:

First of all the customer starts the machine, if it returns failure, it automatically calls support to assist, if it returns success, a condition takes place, a loop takes place for this condition, the condition is if the user insert his input or not, if he did another condition takes place that checks if there are enough resources to create the requested item by the customer, if it fails an error shows that there is no such item and the process restarts, otherwise, a sequence of actions take place, first the water is heated, then the coffee is grinded and then a condition takes place to check if the cup is placed, if not, a messages is shown for the customer to place the cup and the condition loops until it succeed, when it does another sequence of actions take place, first the coffee is dispensed, and then it asks the customer if there is any other orders, if yes, the process is restarted, otherwise, the customer is asked for his feedback and the coffee dispenser is cleaned and it checks if the dirty water’s container is full, if it is not, the machine returns to idle state, otherwise, it shows warning that manual emptying of water container is required!

References:

[1] <https://robohub.org/introduction-to-behavior-trees/>

[2] <https://www.polymathrobotics.com/blog/state-machines-vs-behavior-trees>

[3] <https://www.behaviortree.dev/docs/4.0.2/nodes-library/decoratornode/>