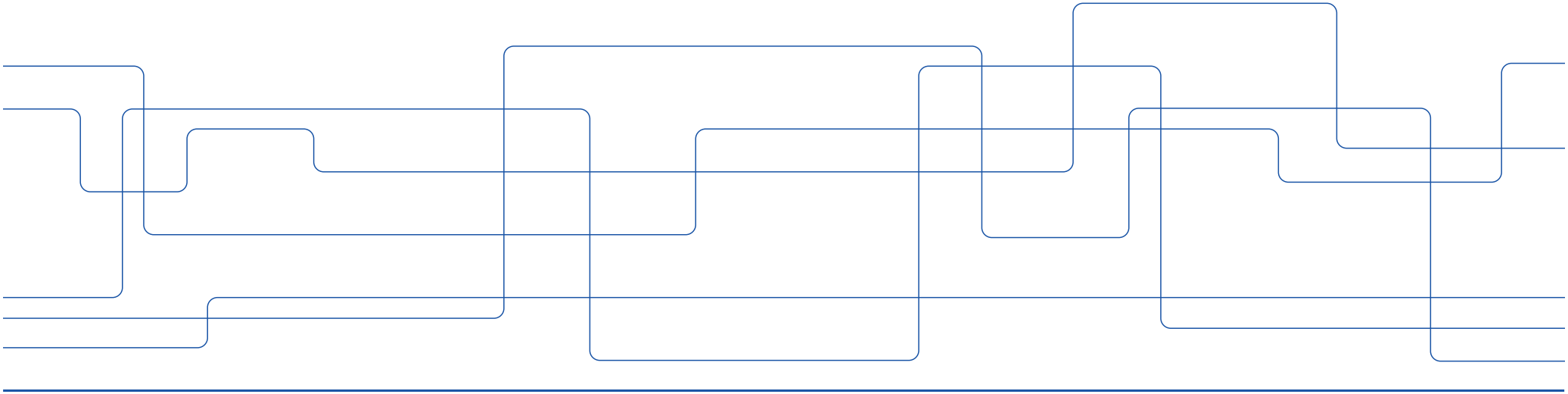
KTH ROYAL INSTITUTE
OF TECHNOLOGY

Lecture 8: Behavior Trees and Task Switching

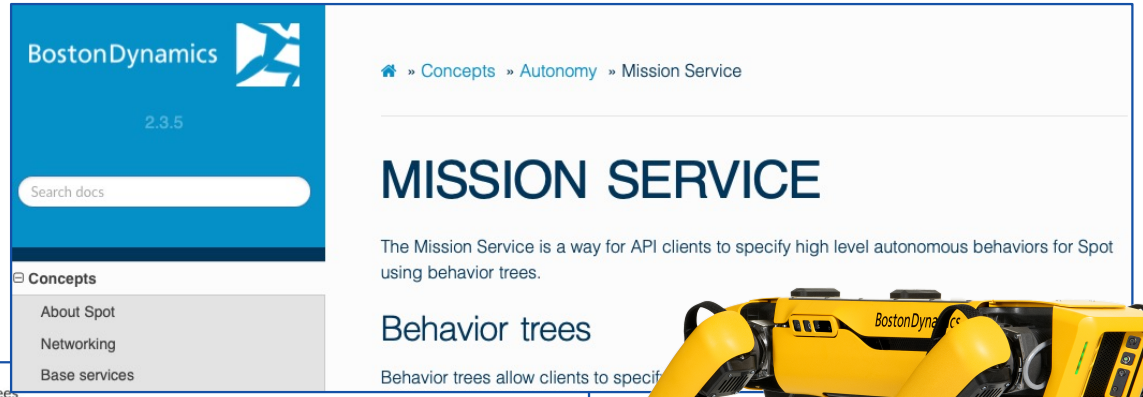
by Petter Ögren

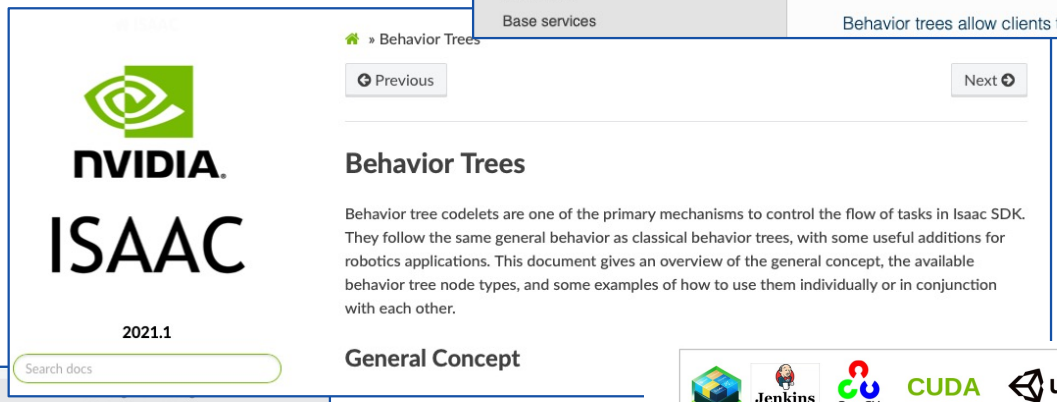


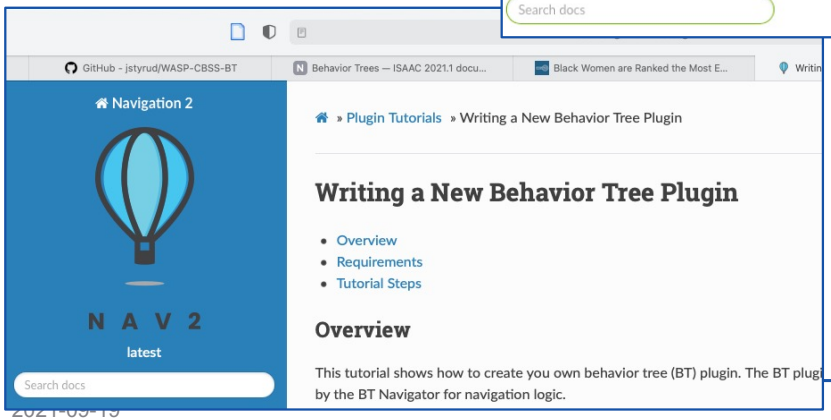
Content

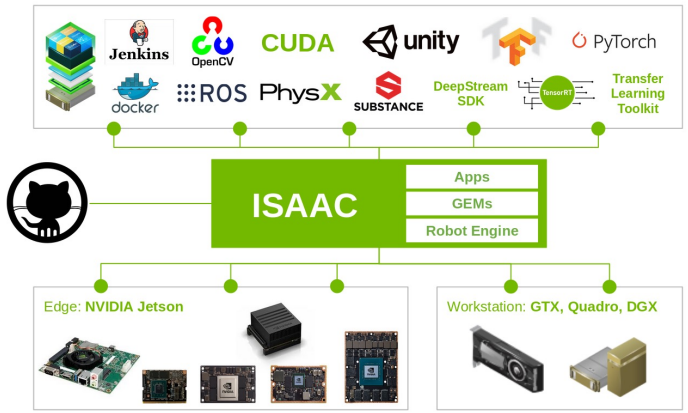
- When to use Behavior Trees (BTs)?
  - Creating complex controllers/policies
- What are BTs?
  - Hierarchically modular policies
- How to create BTs?
  - Improvise
  - Use planning (backward chaining)
- The Big Picture
  - Genetic Algorithms
  - Reinforcement Learning
  - Control Theory (Performance Guarantees)

Behavior trees in use



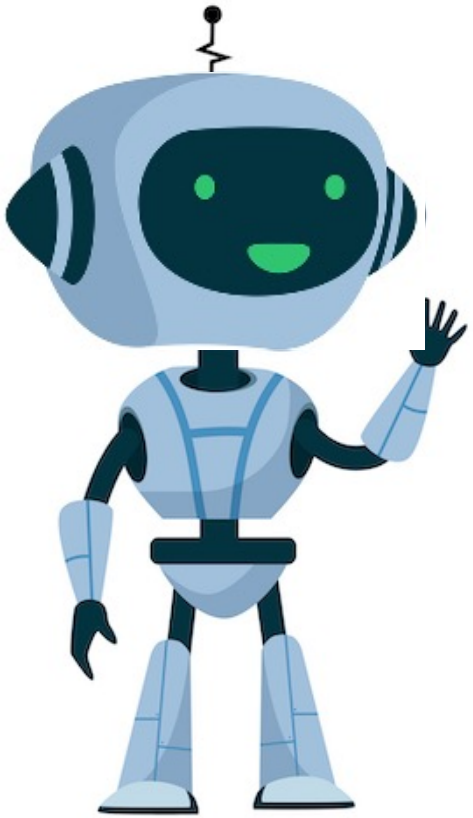






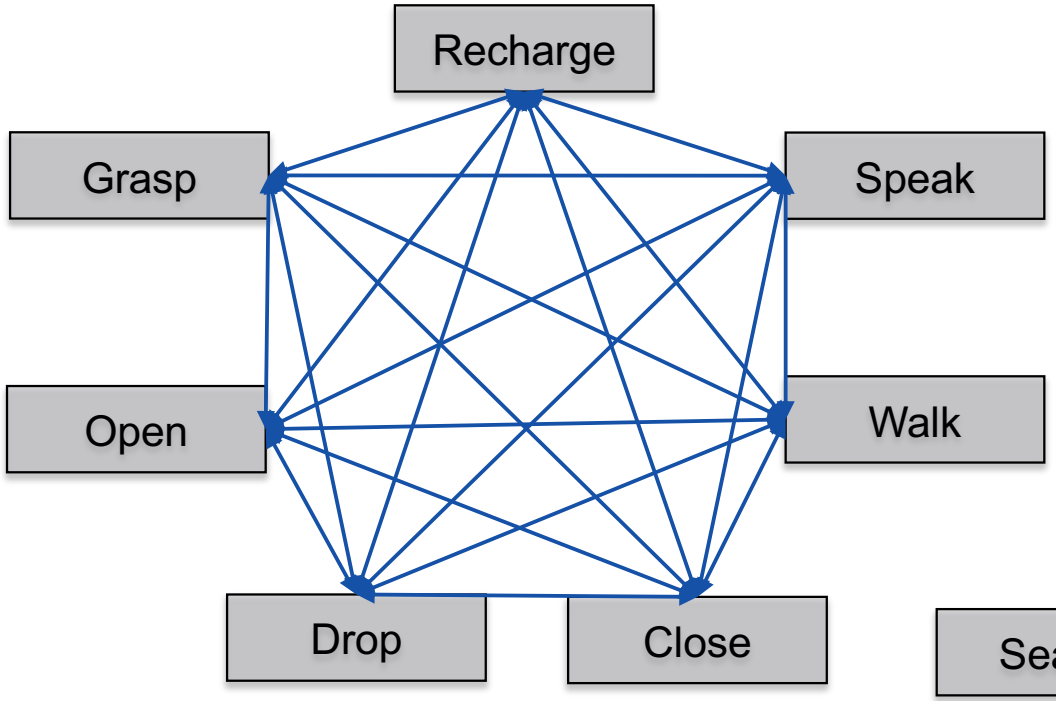
What to do next?

(any autonomous systems needs to answer this question)



Grasp	Recharge
Drop	Lock
Walk	Unlock
Open	Pull
Close	Push
Search	Kick
Clean	Speak
Listen	Idle
Run	Throw

What to do next?

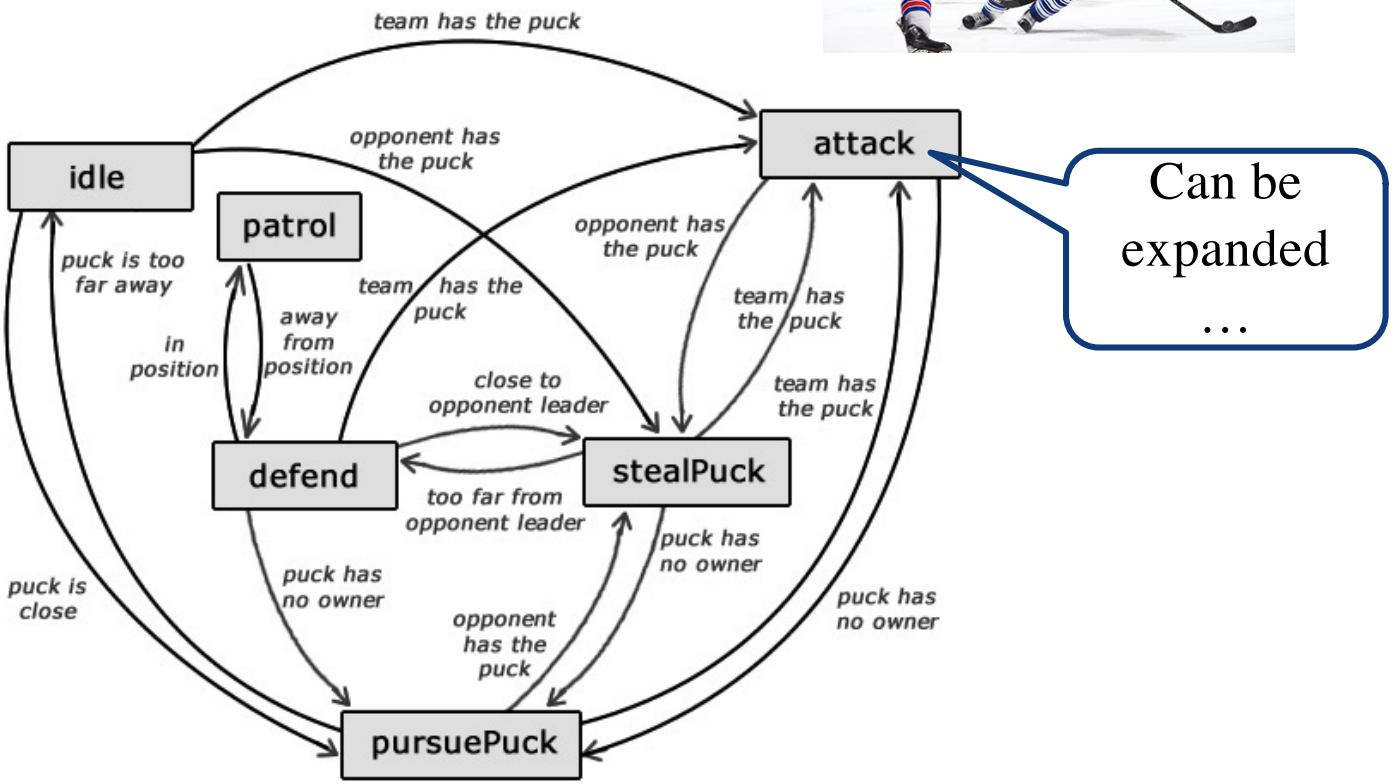


Finite State Machine

Lock
Unlock
Pull
Push
Kick
Search
Speak
Clean
Record

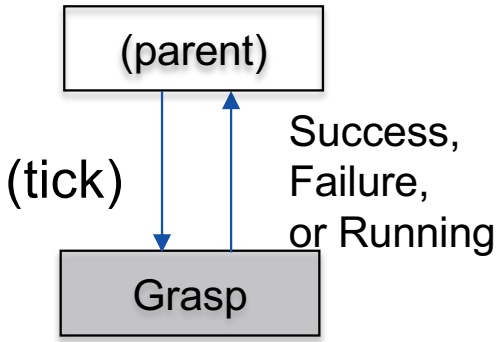
Each action needs to know “What to do next”…

Can you spot the Bug?



Can be expanded …

What to do next?



Behavior Tree

Each action needs to know
“Did I Succeed or Fail?”

Grasp | Recharge
Drop | Lock
Walk | Unlock
Open | Pull
Close | Push
Search | Kick
Clean | Speak
Speak | Record
Run | Throw

Ancestors decide “What do to next?”

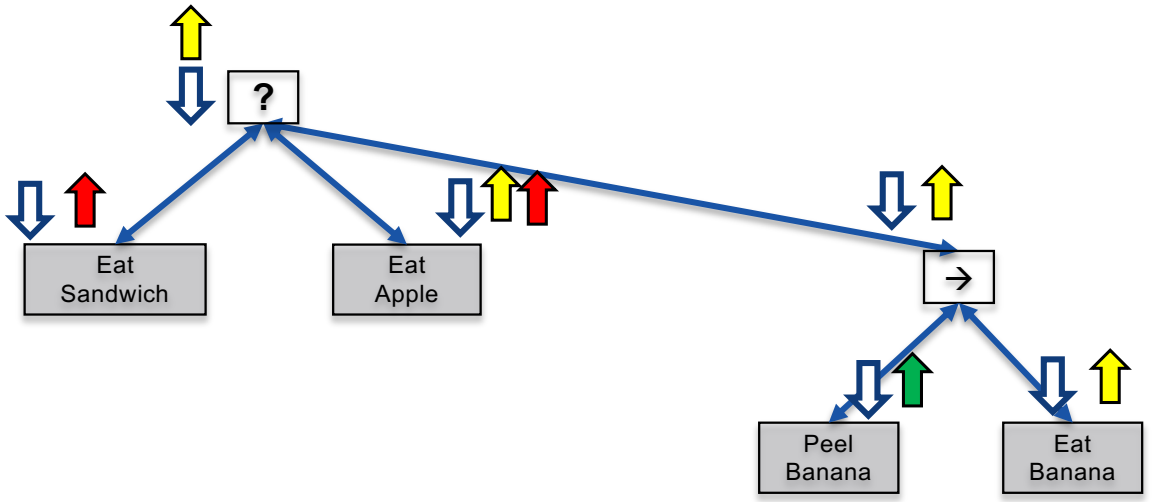
Two Fundamental Compositions of Actions

- **Fallback** (?)(or)
  - (Eat Sandwitch ? Eat Apple)

IF Failure then Tick Next
else Return “same as child”

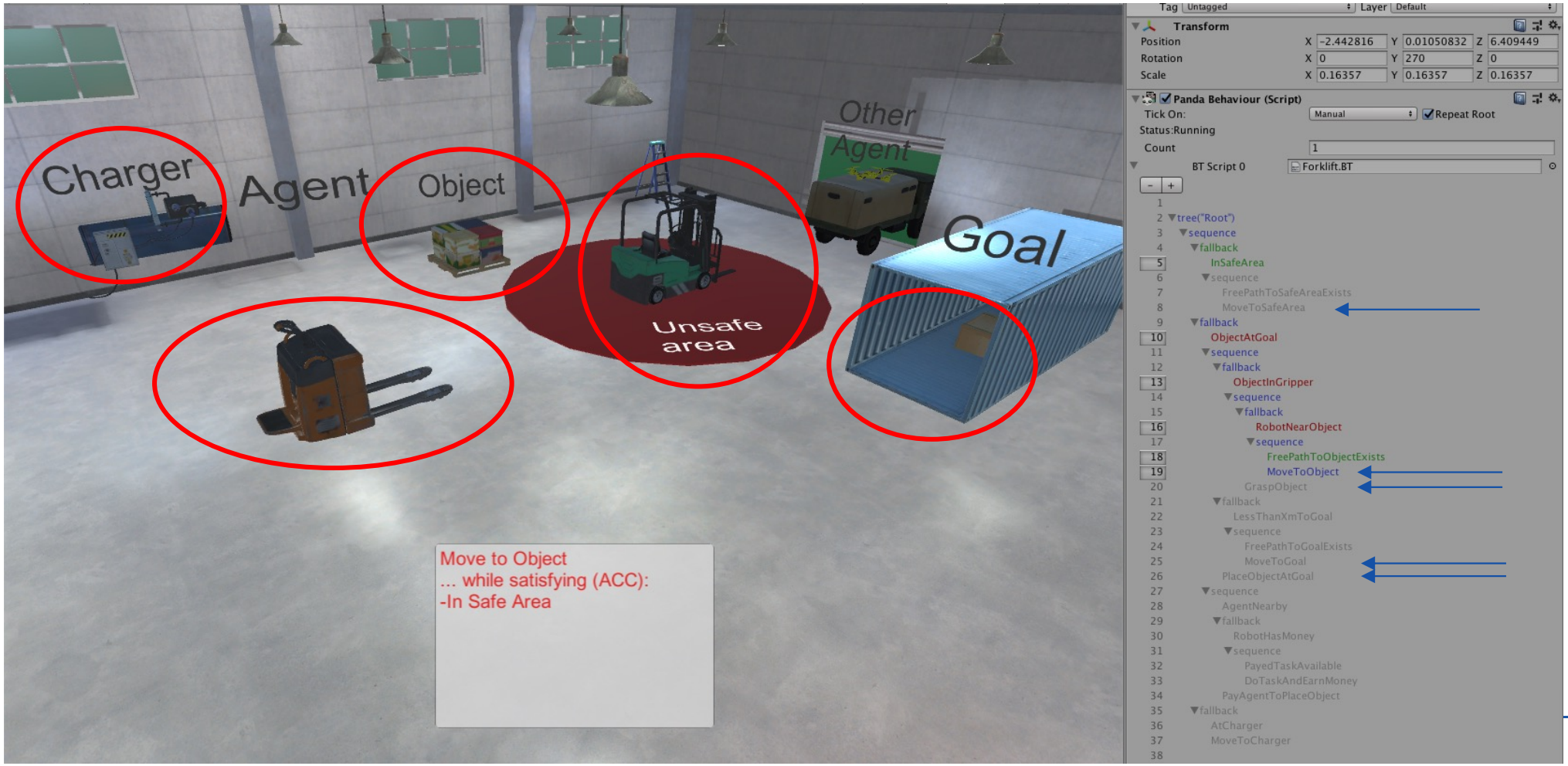
- **Sequence** (→)(and)
  - (Peel Banana → Eat Banana)

IF Success then Tick Next
else Return “same as child”



- Tick (going down)
- Success (up)
- Running (up)
- Failure (up)

Example


Charger
Agent
Object
Other Agent
Goal
Unsafe area
Move to Object
... while satisfying (ACC):
-In Safe Area
Tag Untagged
Layer Default
Transform
Position X -2.442816 Y 0.01050832 Z 6.409449
Rotation X 0 Y 270 Z 0
Scale X 0.16357 Y 0.16357 Z 0.16357
Panda Behaviour (Script)
Tick On: Manual
Repeat Root
Status:Running
Count 1
BT Script 0 Forklift.BT
1
2 tree("Root")
3 sequence
4 fallback
5 InSafeArea
6 sequence
7 FreePathToSafeAreaExists
8 MoveToSafeArea
9 fallback
10 ObjectAtGoal
11 sequence
12 fallback
13 ObjectInGripper
14 sequence
15 fallback
16 RobotNearObject
17 sequence
18 FreePathToObjectExists
19 MoveToObject
20 GraspObject
21 fallback
22 LessThanXmToGoal
23 sequence
24 FreePathToGoalExists
25 MoveToGoal
26 PlaceObjectAtGoal
27 sequence
28 AgentNearby
29 fallback
30 RobotHasMoney
31 sequence
32 PayedTaskAvailable
33 DoTaskAndEarnMoney
34 PayAgentToPlaceObject
35 fallback
36 AtCharger
37 MoveToCharger
38

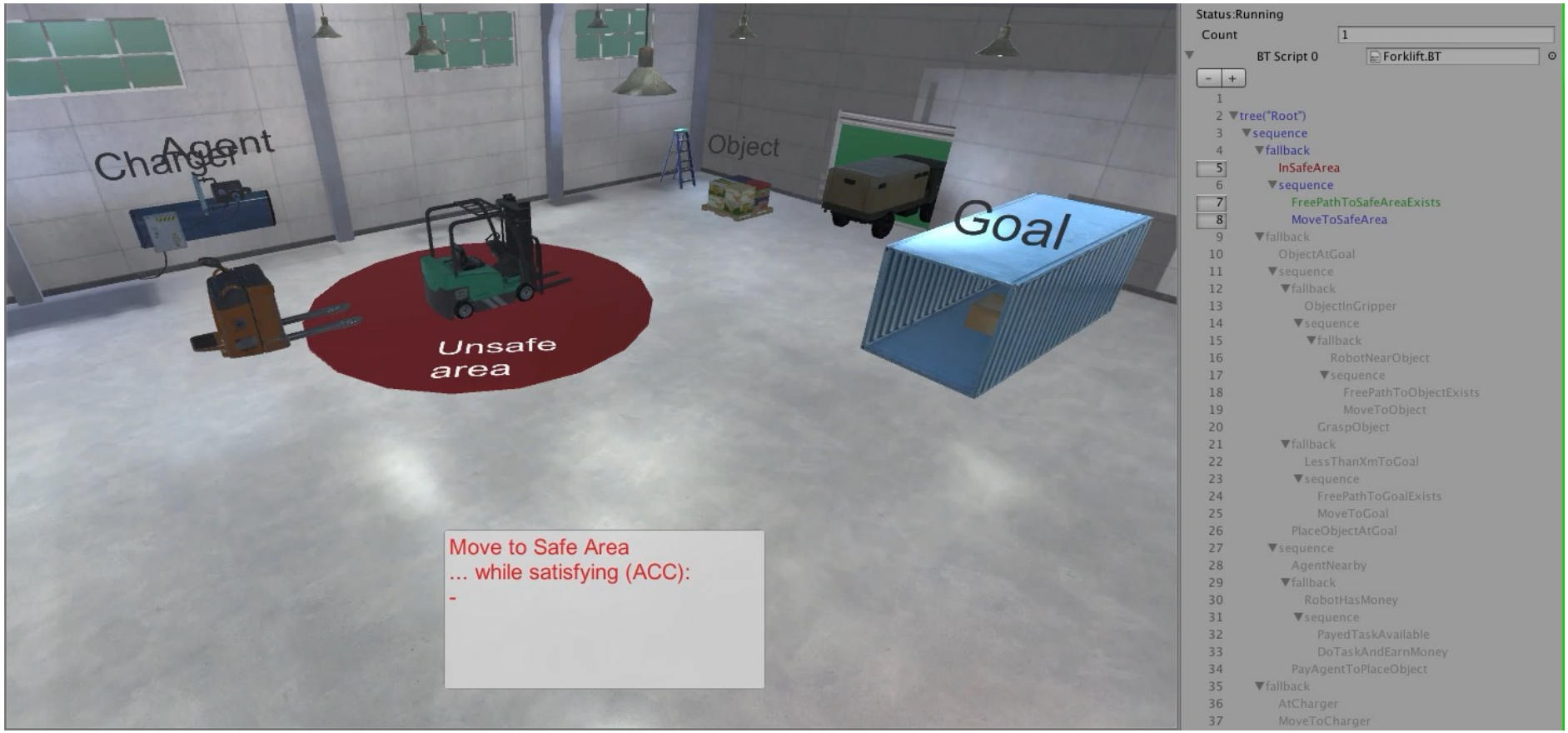
Execution without disturbances

- Success
- Running
- Failure

- Classical Control handles noise disturbances
- Behavior Tree handles events disturbances

Handling disturbances

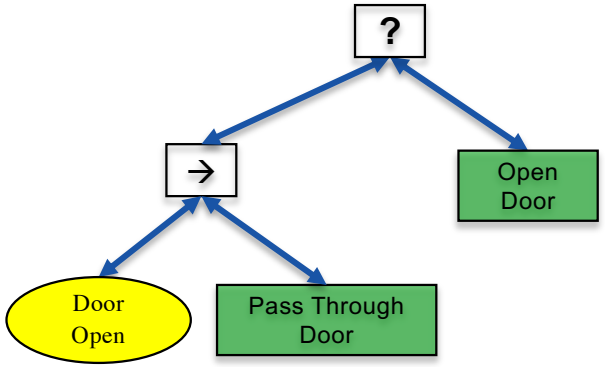
- Success
- Running
- Failure



Properties of Behavior Trees :

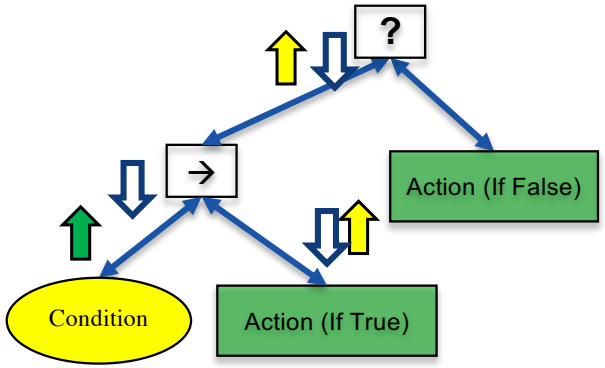
- Modularity
  - Few dependencies between components (Important for large systems)
  - Optimally modular [1]
- Hierarchical structure
  - Actions exist on many levels of detail (Get tea – opening door – grasp handle – move arm)
  - Hierarchical modularity
- Equally expressive as FSMs [2] (with internal variables)
  - choice a matter of taste (as programming languages)
- BTs generalize [3]
  - Subsumption Architecture
  - Teleo-Reactive Approach
  - Decision Trees

If-then-else constructs

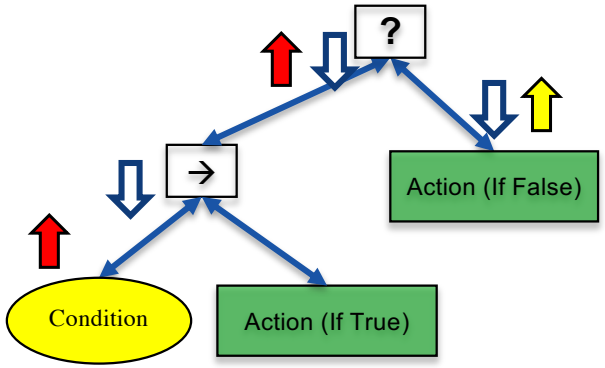


- How to do If-then-else?

- If True…



- If False …

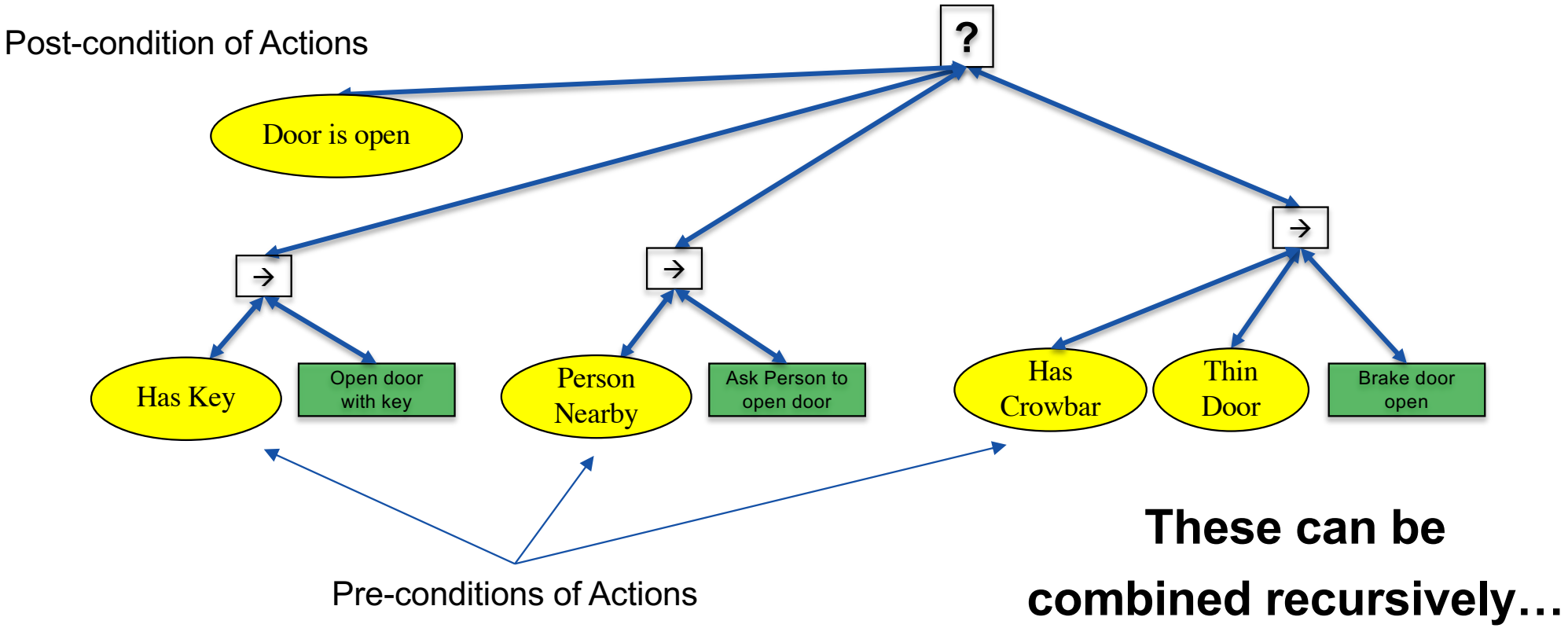


Design BT using Planning (Backward Chaining)

- Backward Chaining
  - Solving an AI Planning Problem by working **backwards from the goal**
- Example:
  - Goal: *Leave the room*
  - To leave I need to **pass through the door**
  - To pass the door I need to **open the door**
  - To open the door I need to **grasp the handle**
  - To grasp the handle I need to **extend my arm**
  - **Plan:**
    - > ***Extend arm***
    - > ***Grasp handle***
    - > ***Open door***
    - > ***Pass through the door***

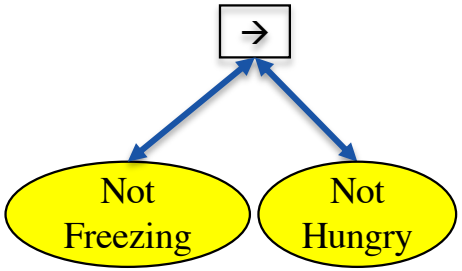
BTs can do this reactively…

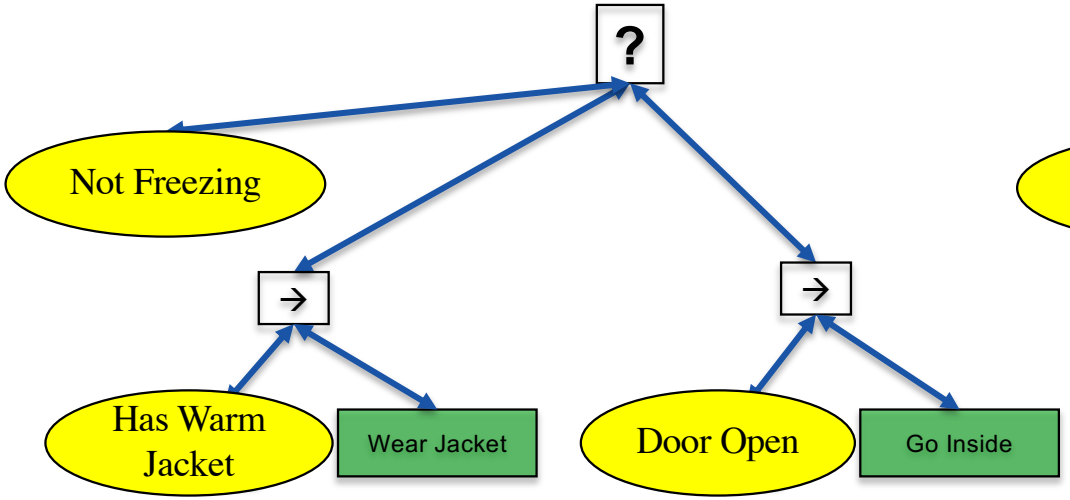
A BT that achieves a single goal (using feedback)

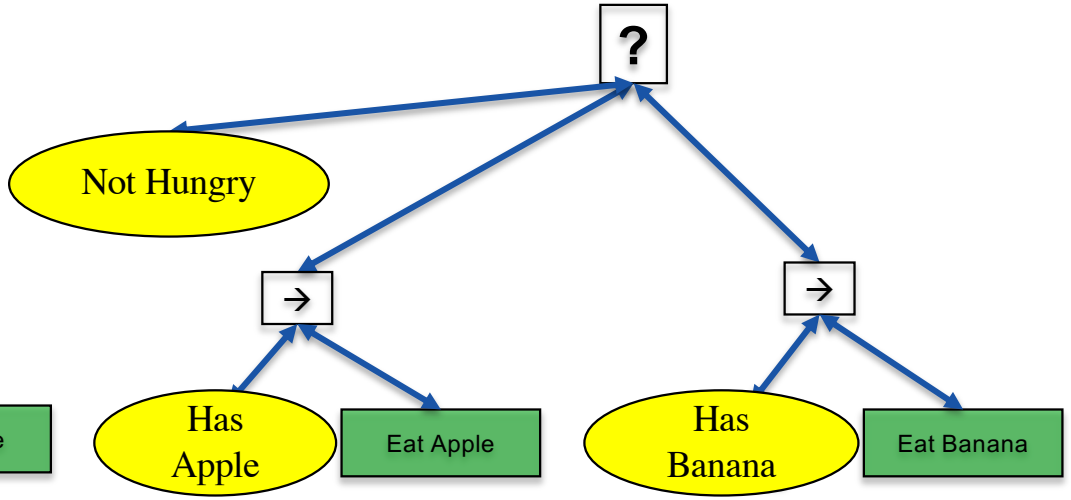


These can be combined recursively…

Backward chaining: starting with 2 goal conditions

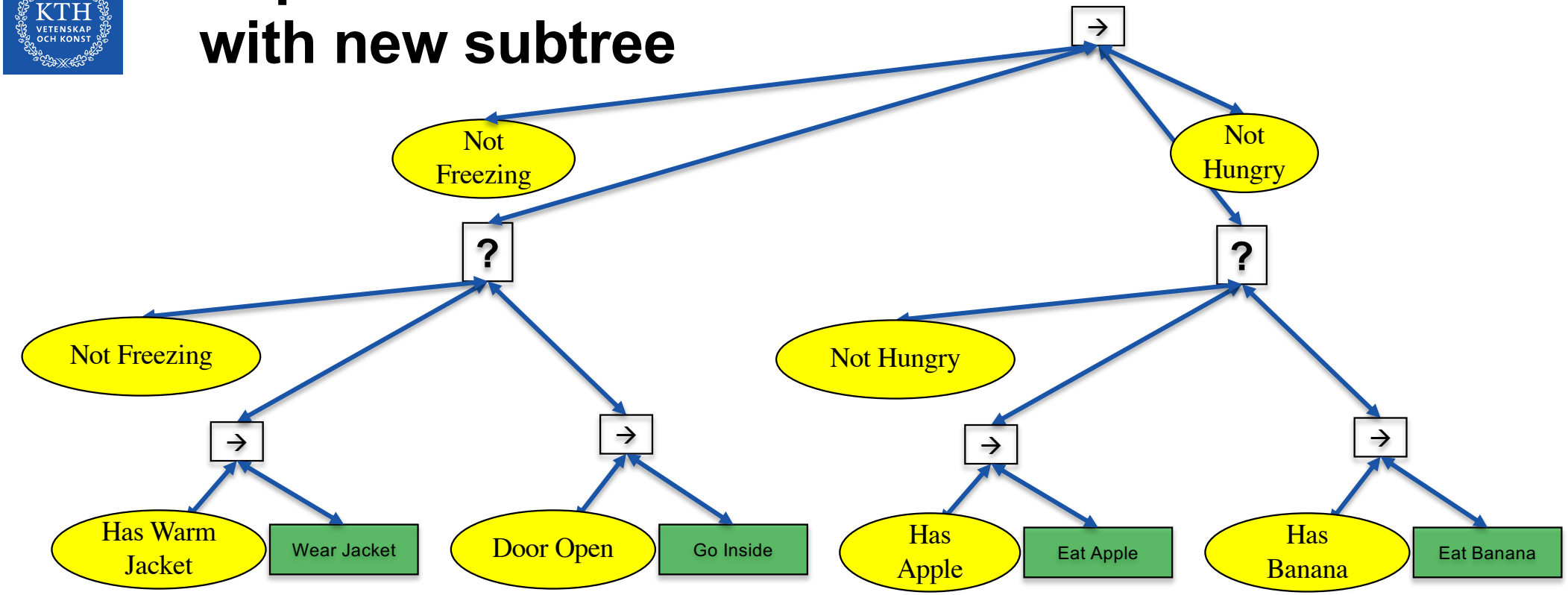






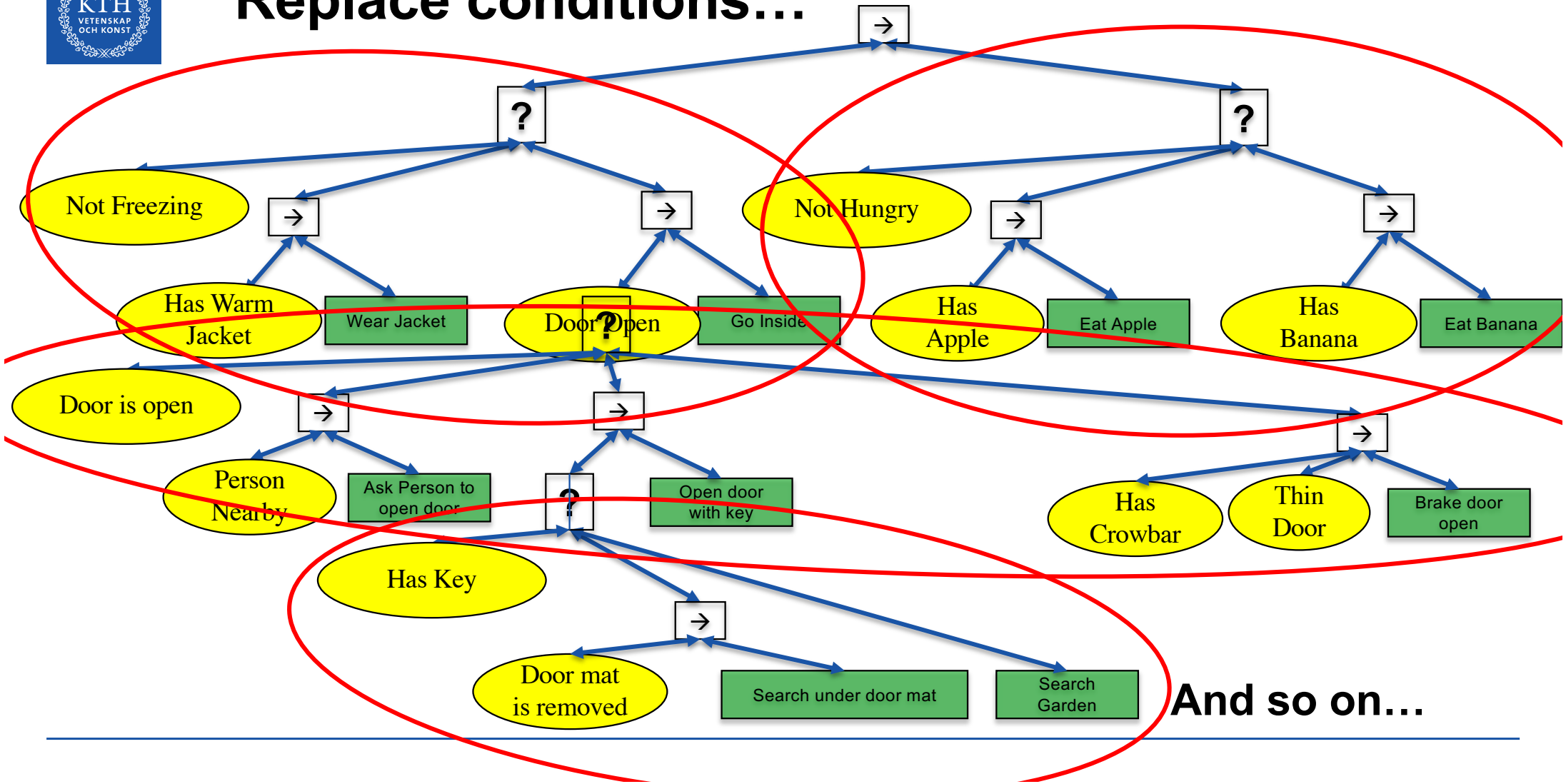
Find BTs that achieve each

Replace condition (with Sequence parent) with new subtree



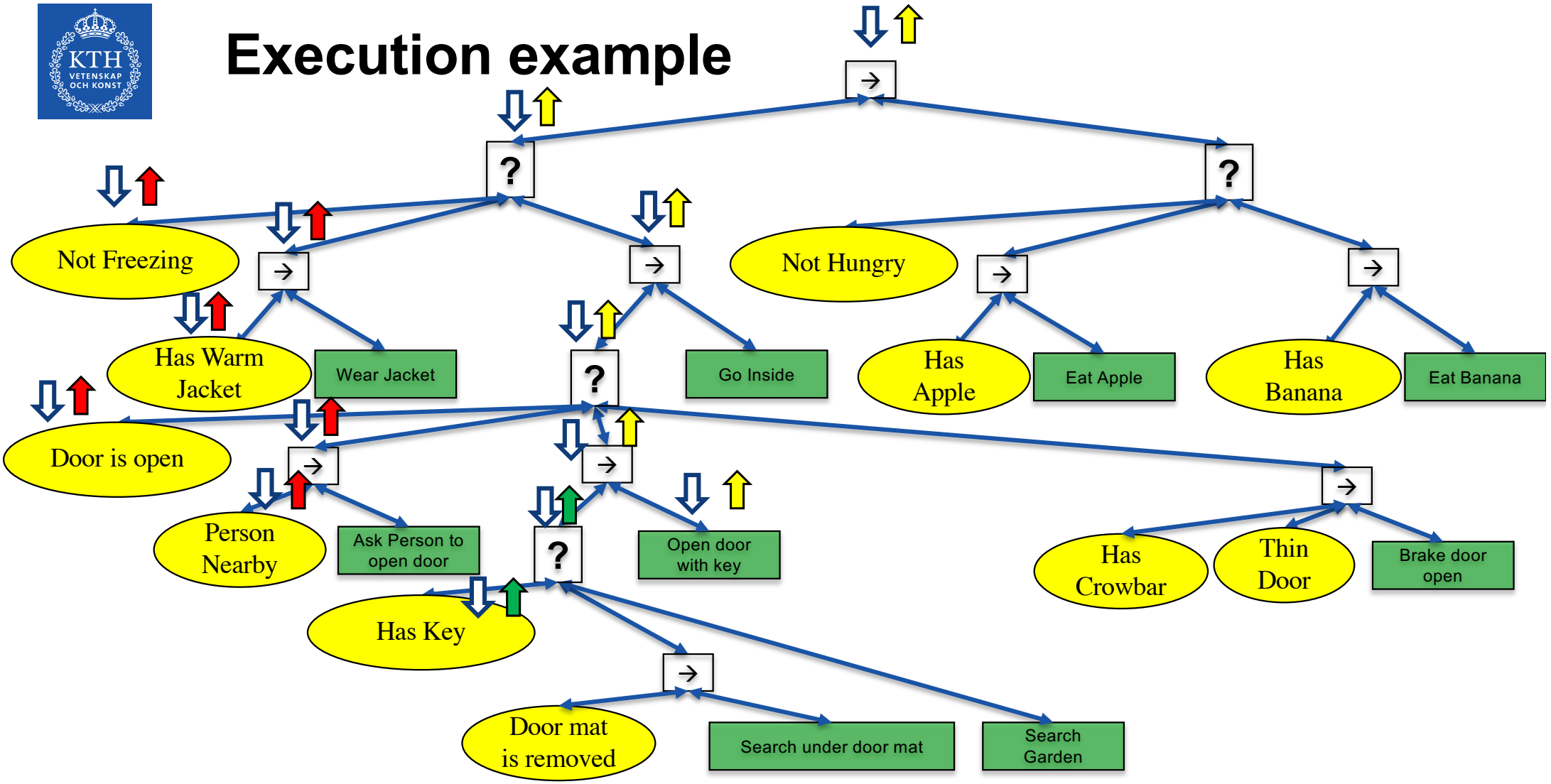
Iterate this…

Replace conditions…

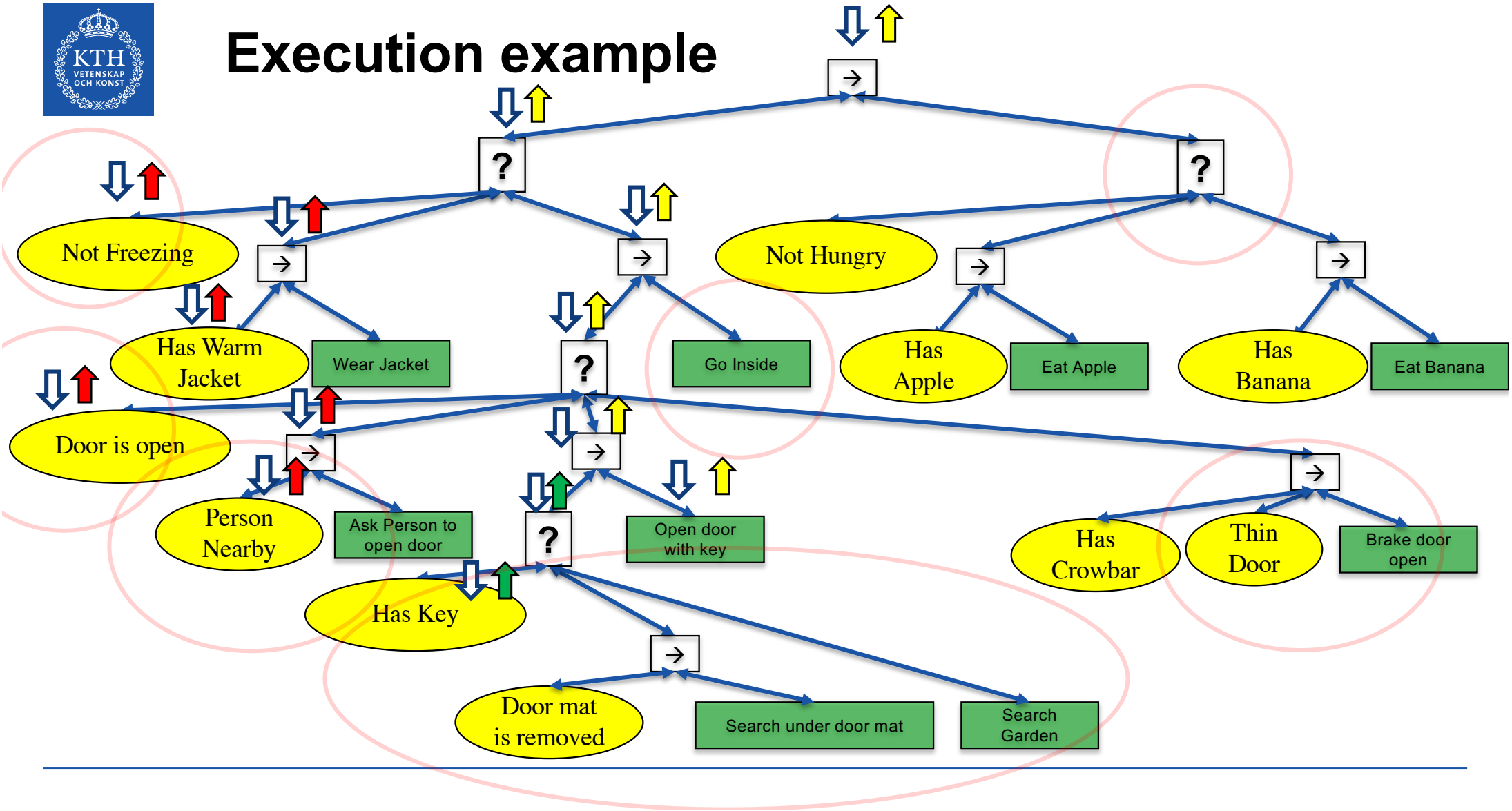


And so on…

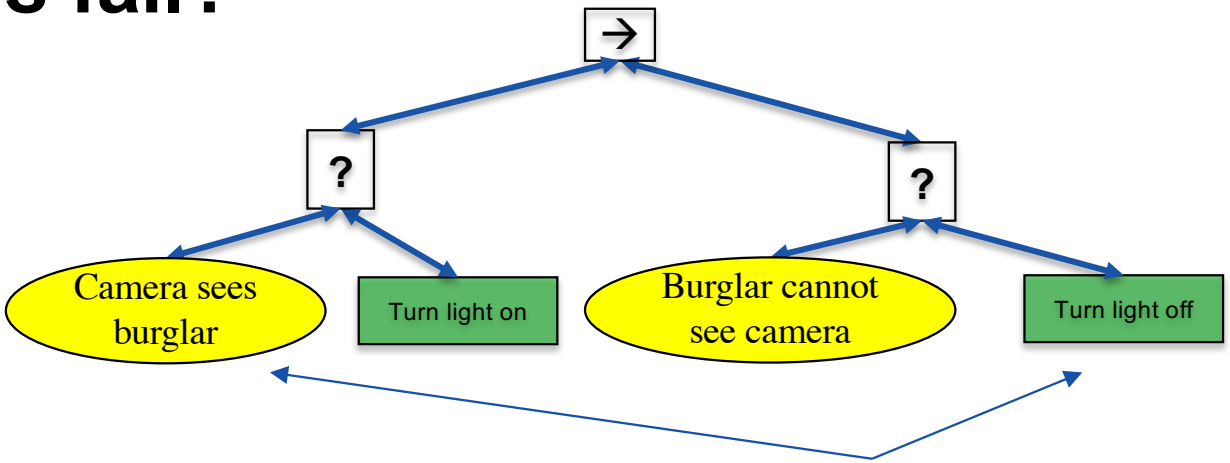
Execution example



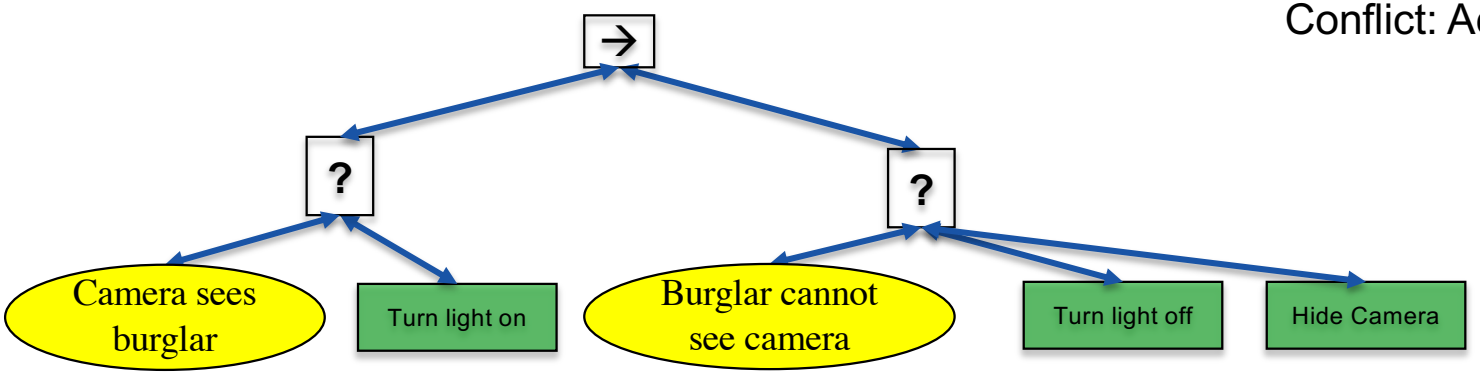
Execution example



When does this fail?

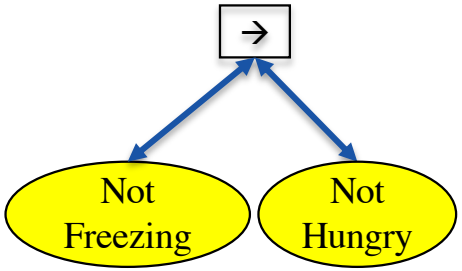


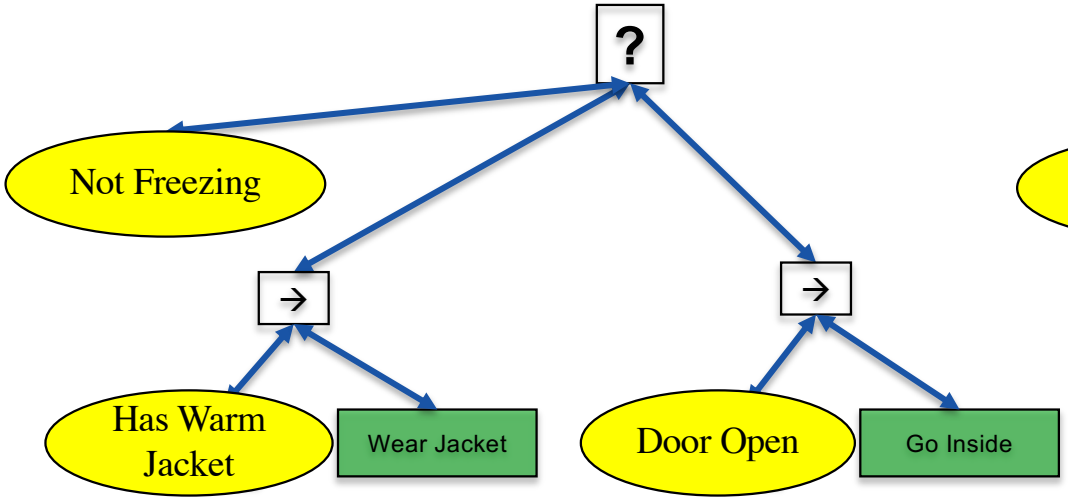
Conflict: Action brakes already satisfied Objective

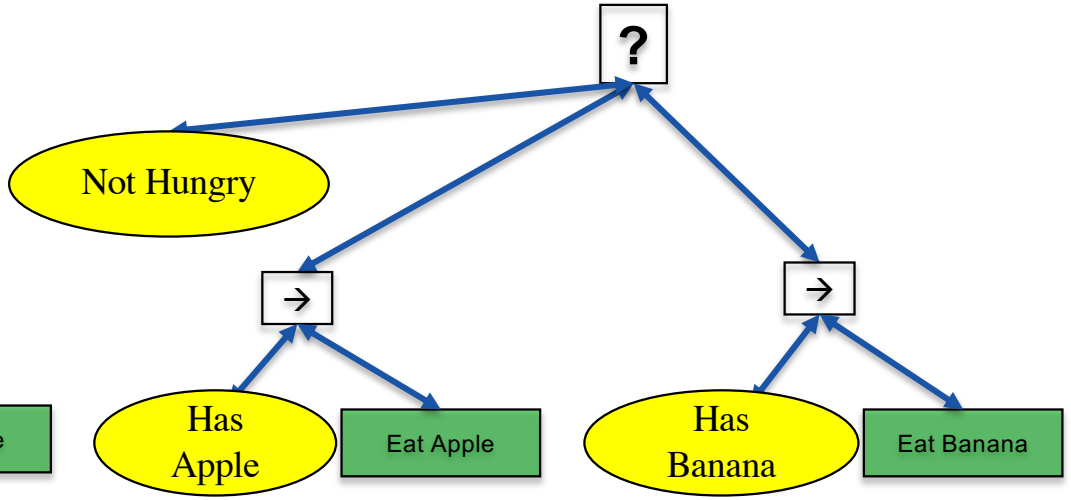


Solution: Avoid braking already achieved goals (if possible)

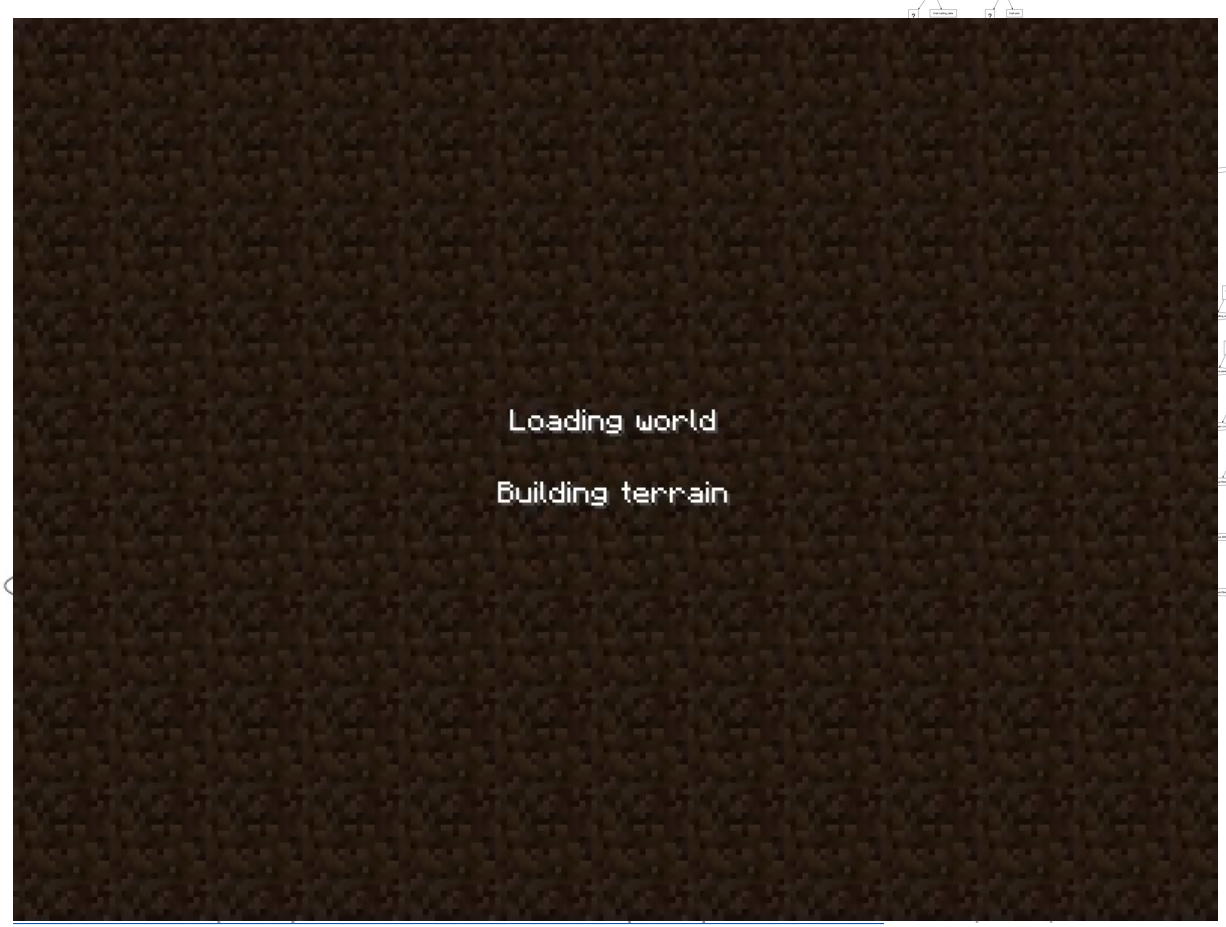
Key idea:
Replace conditions with BT that achieve them!

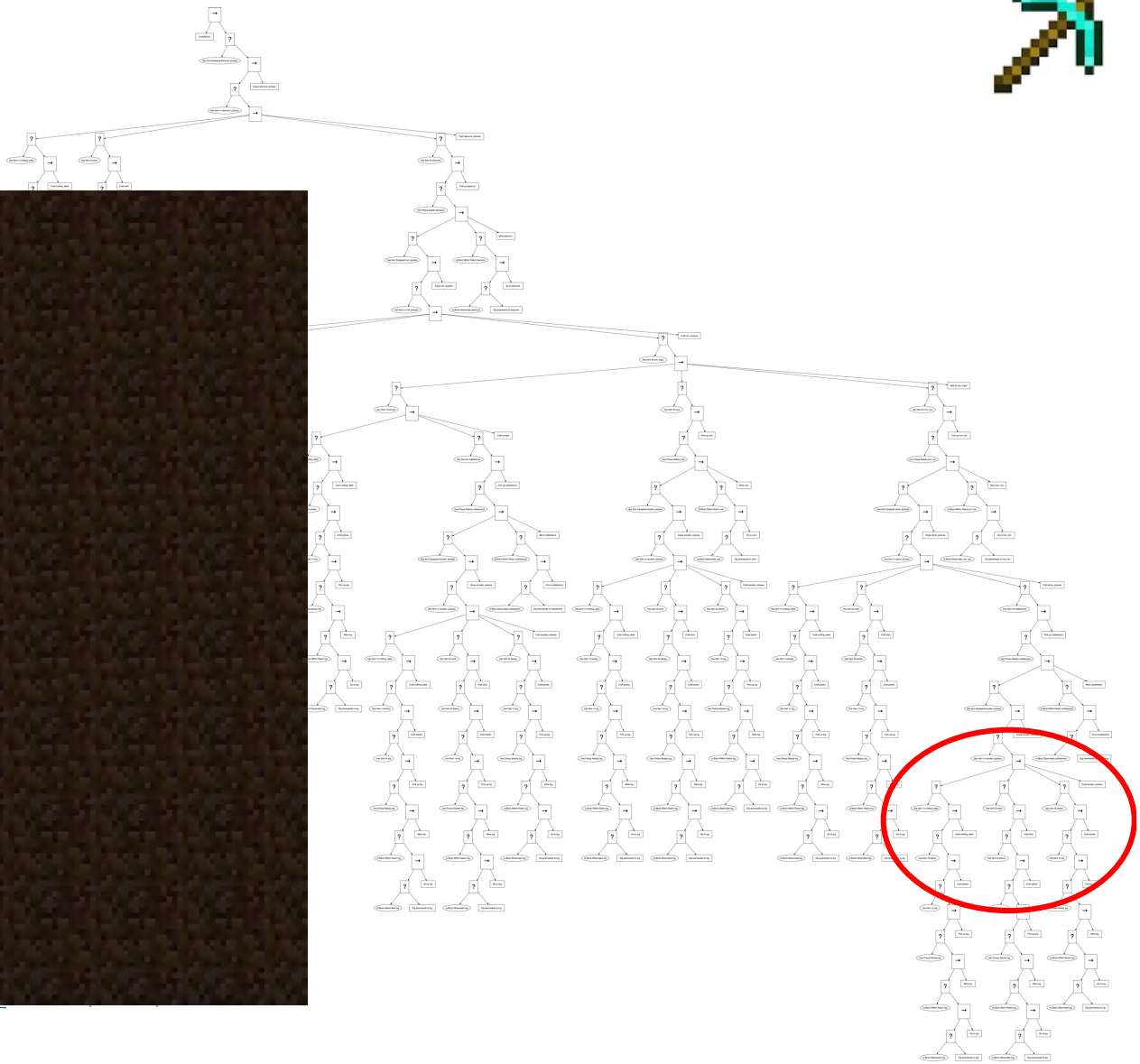






Minecraft AI



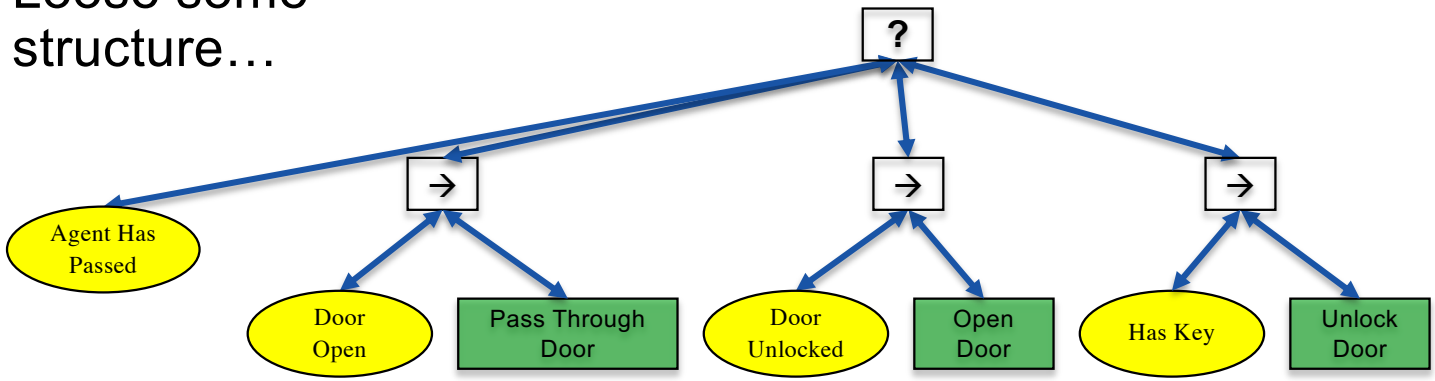


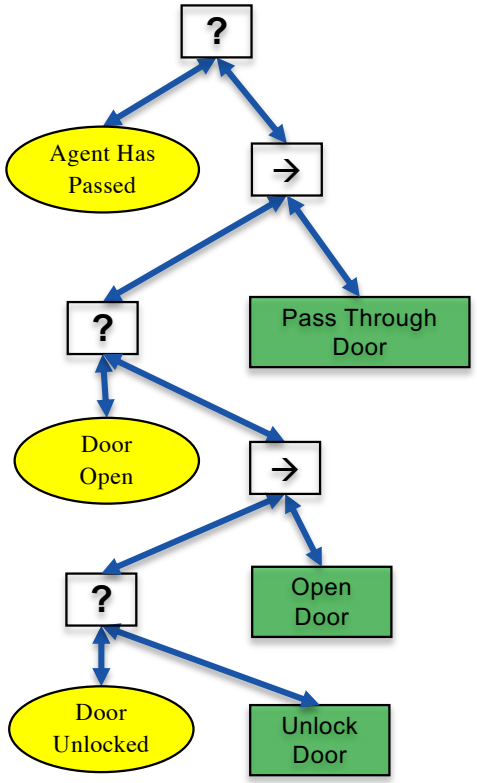
Sometimes we can simplify Backward Chaining (Implicit Sequences)

- Only 2 levels

- Works if
  - Action satisfies Condition to Left

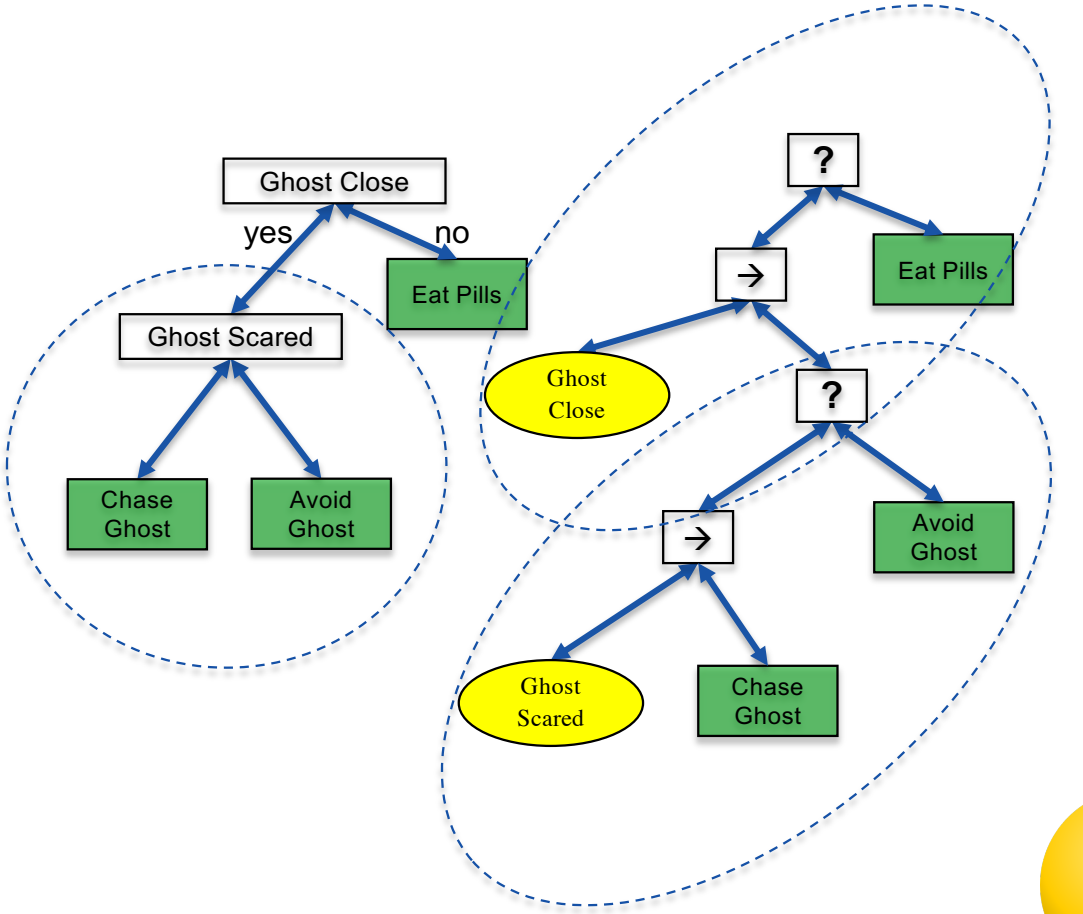
- Loose some structure…

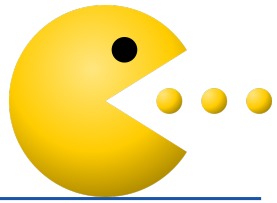




Decision Tree Principle → Divide and Conquer

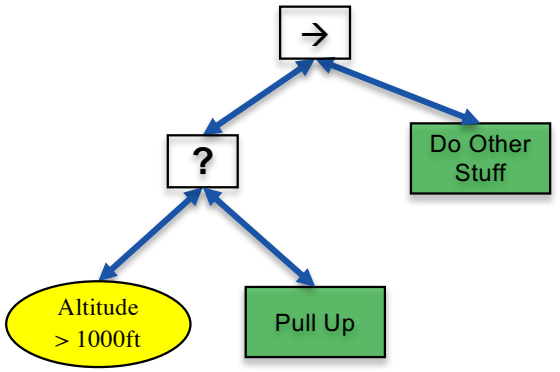
- Can Behavior be divided into Cases? (and sub-cases)
- Think “Decision Tree”
- Use If-then-else…



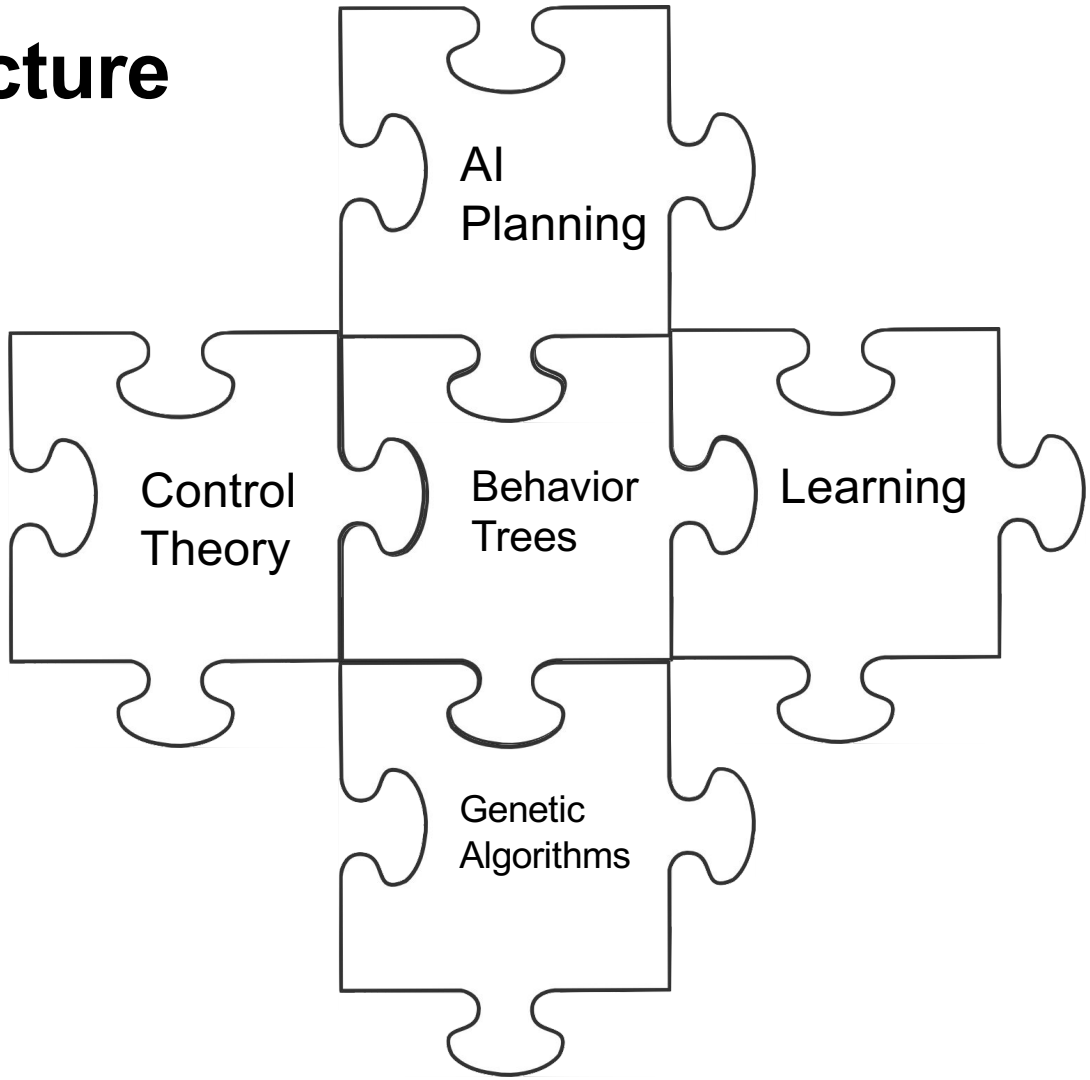


Sequences → Improve Safety

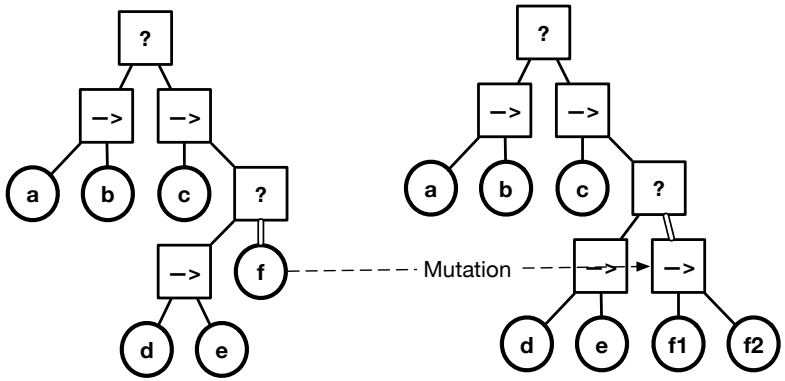
- BTs enable Safety-Guarantees using the following construction...
- If-not-then-else…
- Special case of Backward Chaining

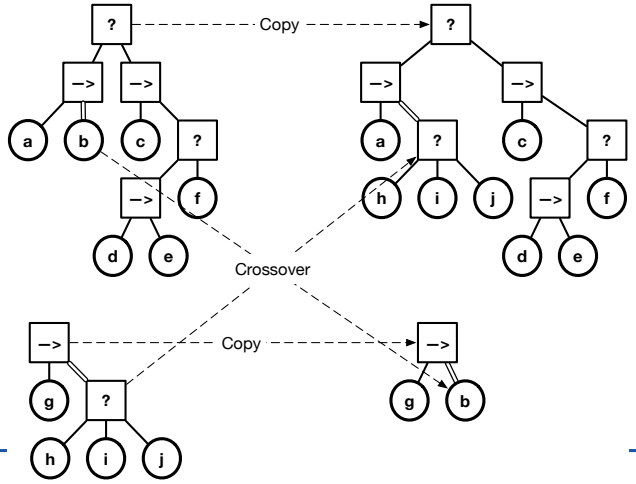


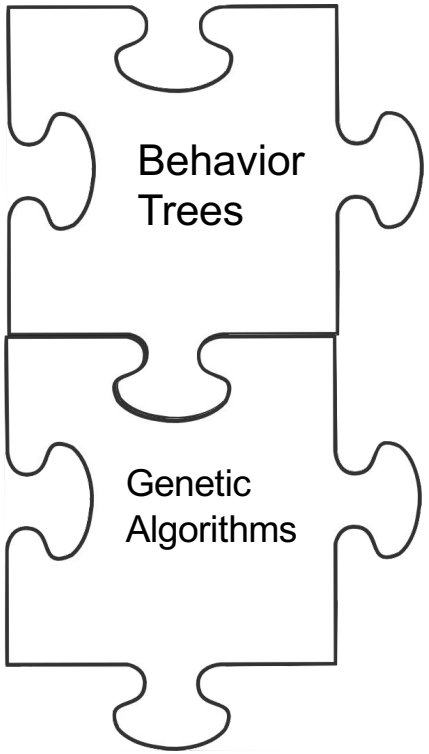
The Big Picture



The Big Picture



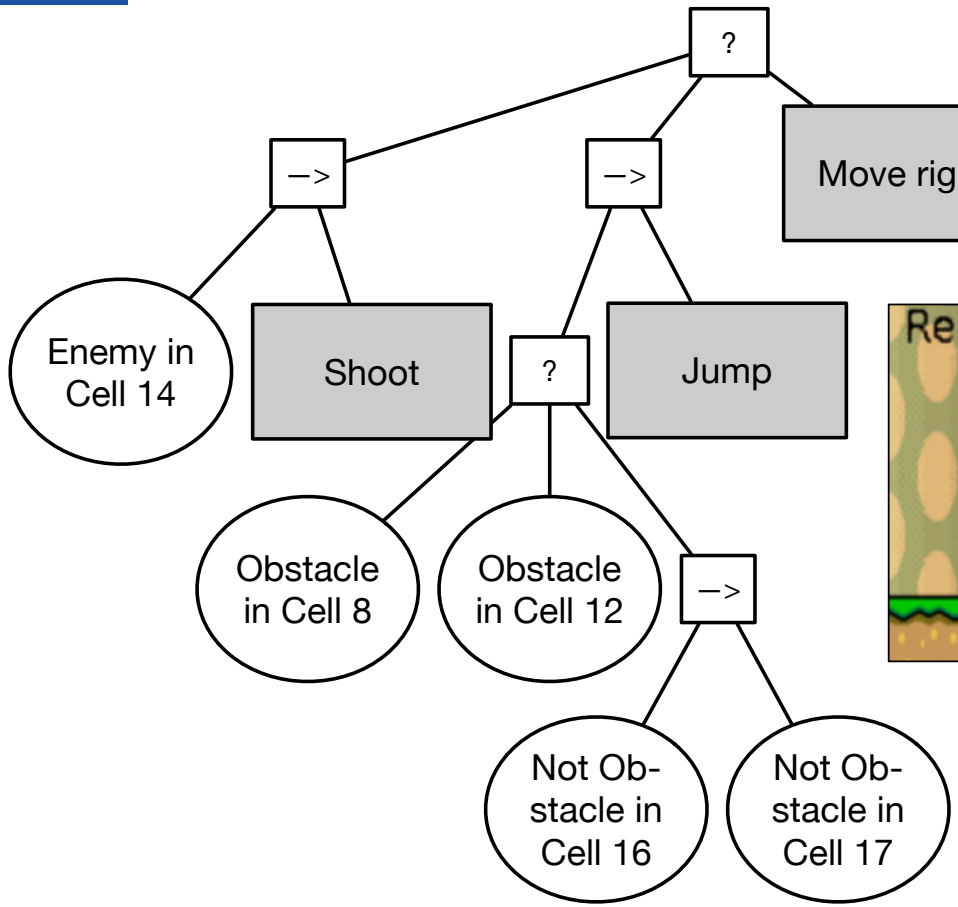


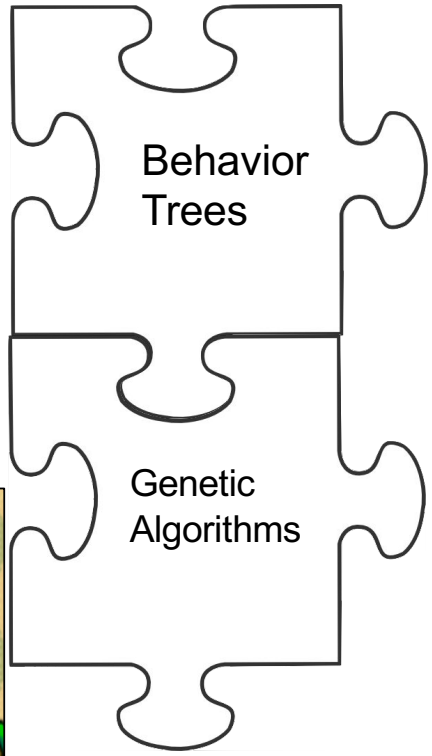


Learning from **Experience**

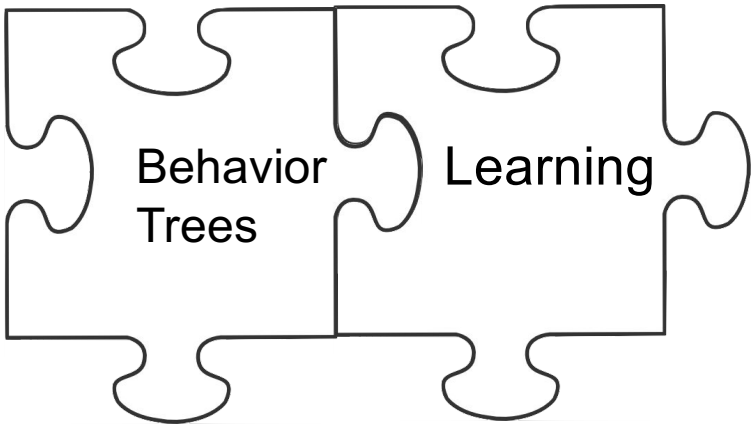
- Apply Genetic Programming
- BT trivially maps to Genes
- Mutation/Crossover easy

The Big Picture

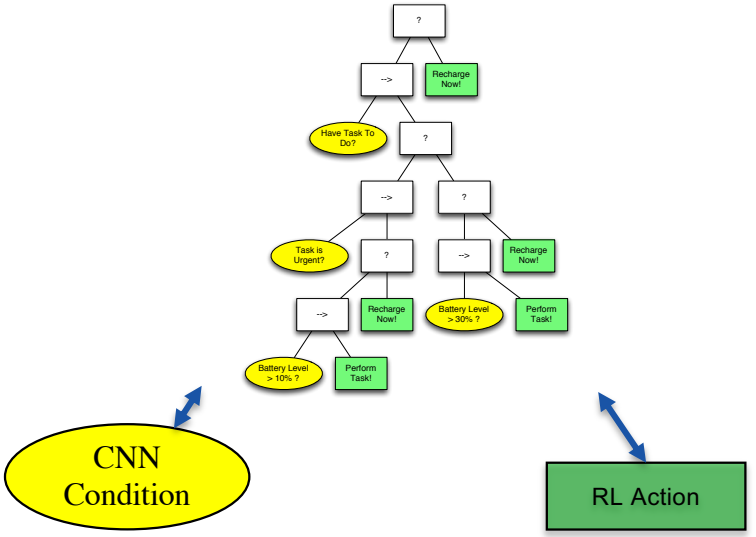




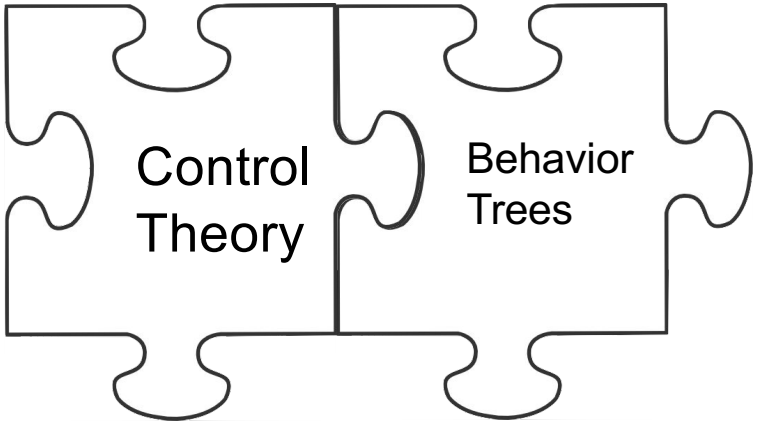
The Big Picture

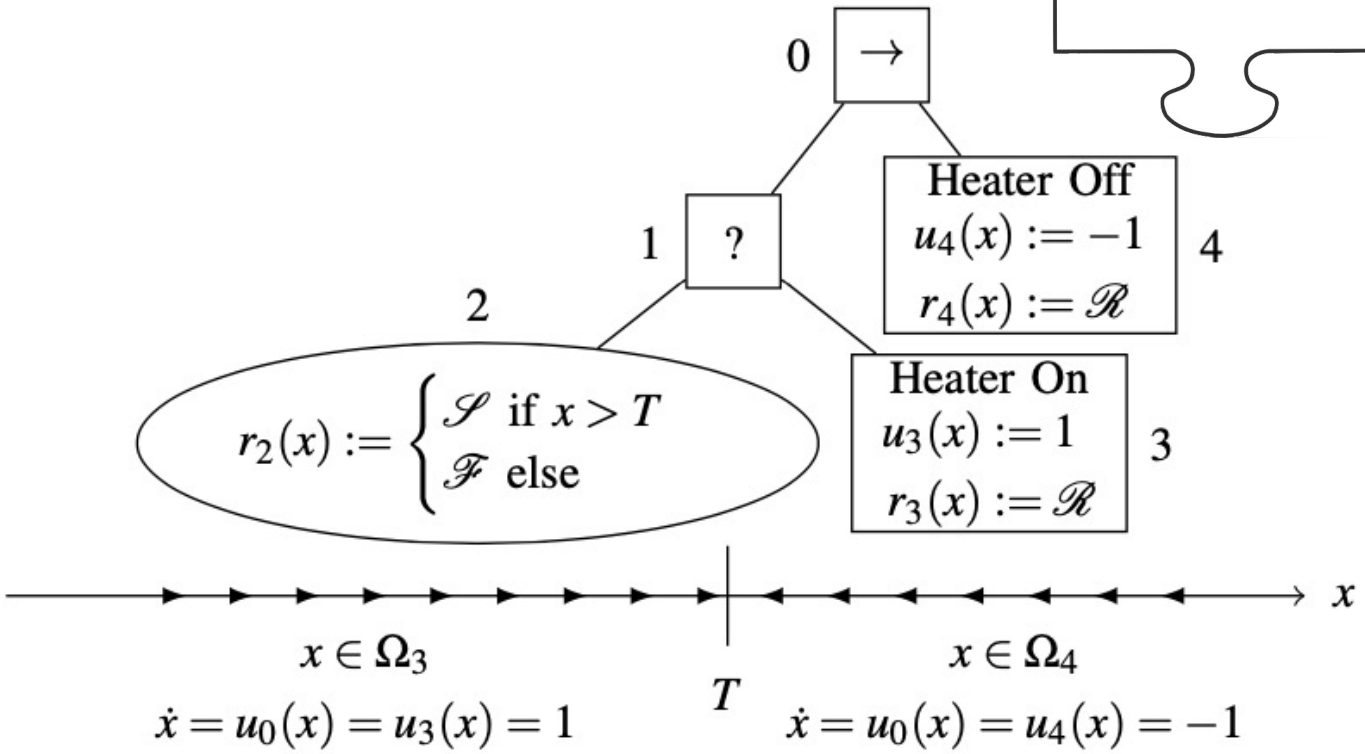


- Use CNN for Conditions (image processing)
- Use RL for Actions

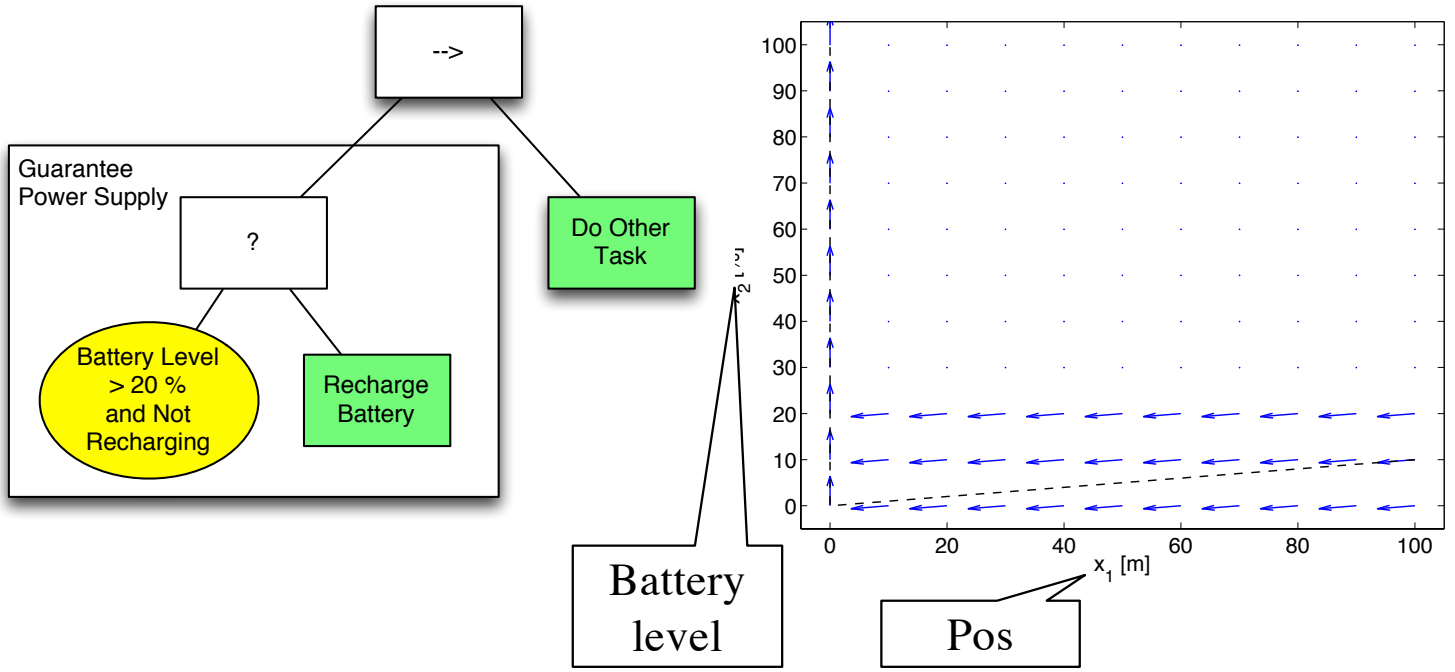


The Big Picture

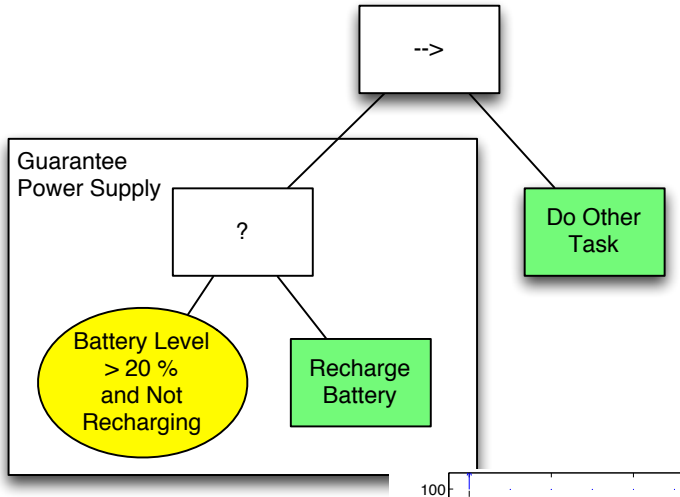


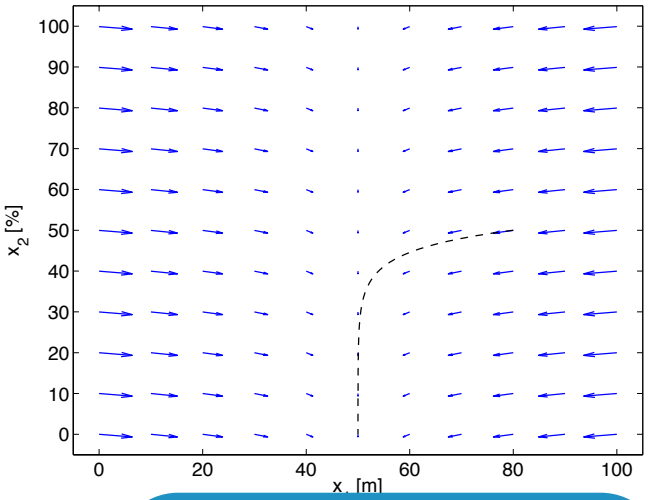


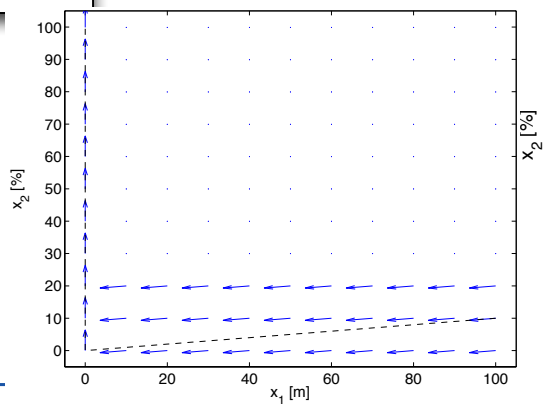
Example: Avoiding Empty Batteries

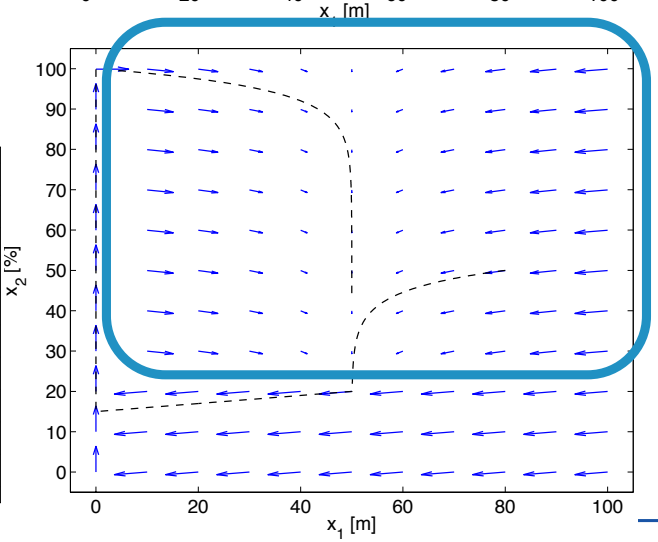


Avoiding Empty Batteries



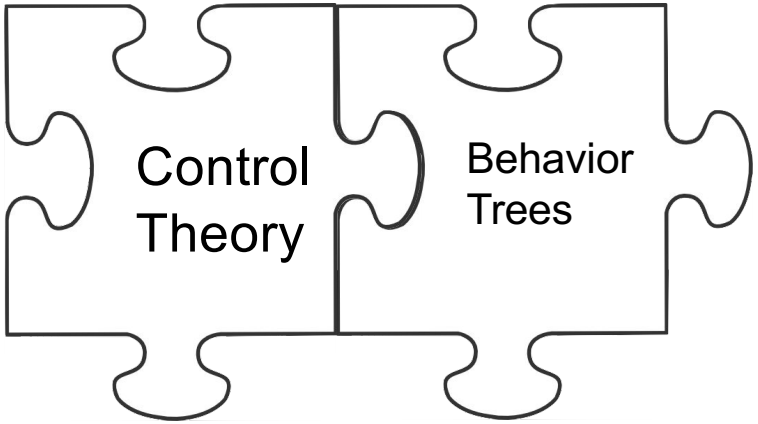






The Big Picture

- Can we give Performance Guarantees?
  - Stability/Convergence?



Convergence Analysis of Hybrid Control Systems in the Form of Backward Chained Behavior Trees

Petter Ögren

***Abstract*—A robot control system is often composed of a set of low level continuous controllers and a switching policy that decides which of those continuous controllers to apply at each time instant. The switching policy can be either a Finite State Machine (FSM), a Behavior Tree (BT) or some other structure. In previous work we have shown how to create BTs using a backward chained approach that results in a reactive goal directed policy. This policy can be thought of as providing disturbance rejection at the task level in the sense that if a disturbance changes the state in such a way that the currently running continuous controller cannot handle it, the policy will switch to the appropriate continuous controller. In this letter we show how to provide convergence guarantees for such policies.**

***Index Terms*—Behavior-based systems, robot safety, control architectures and programming.**

I. Introduction

BEHAVIOR Trees (BTs) were created by computer game programmers as a way to improve modularity and reactivity in the control policies of so-called Non-Player Characters (NPCs) in games [1]. Since then, BTs have been receiving an increasing amount of attention in Robotics [2]–[9]. The reason is that robotics share many high level planning and control problems with game AI, while at the same time, the low

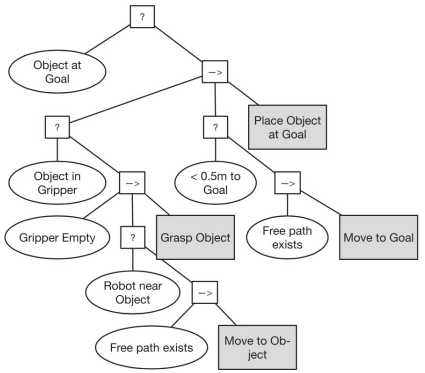
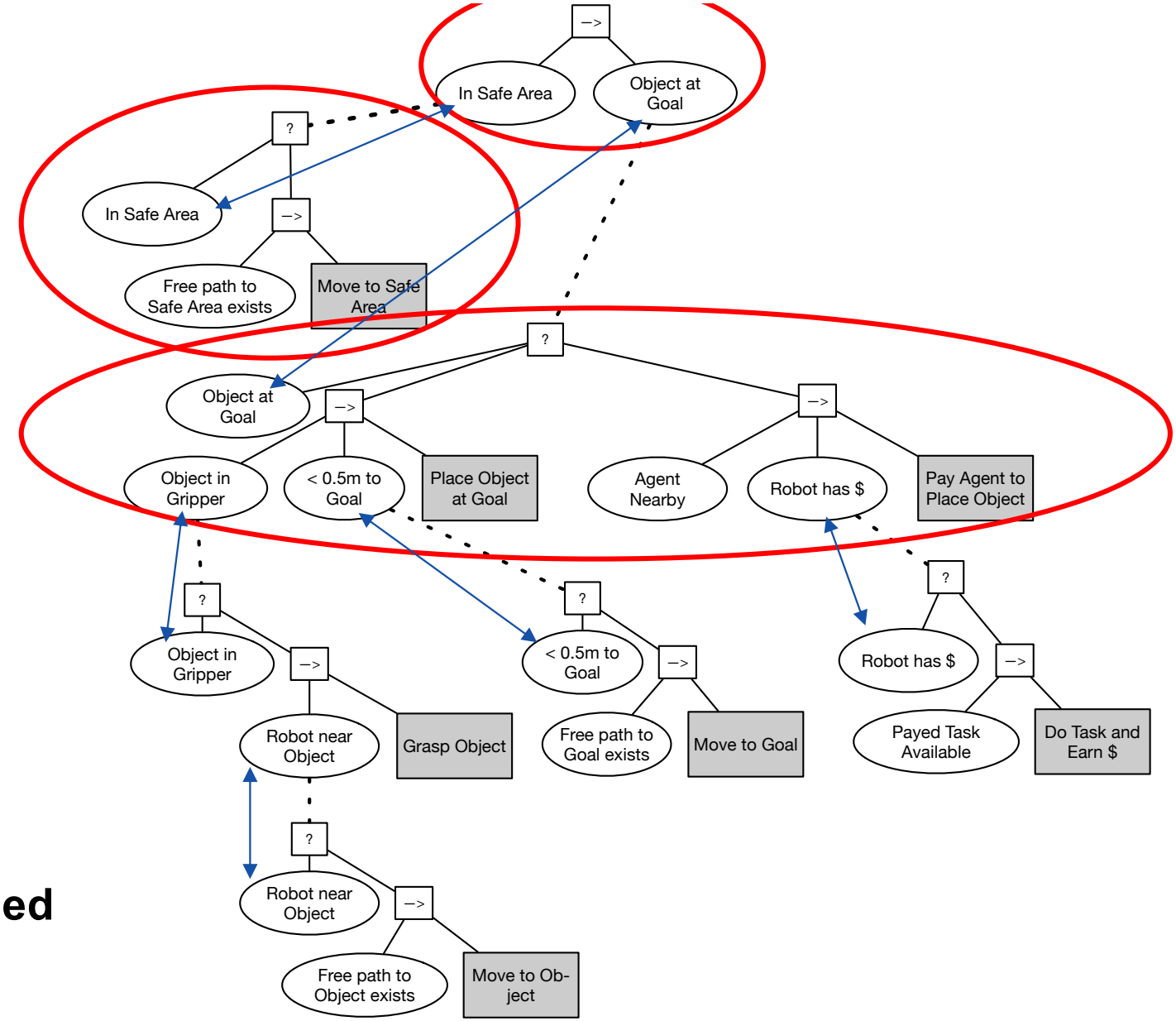


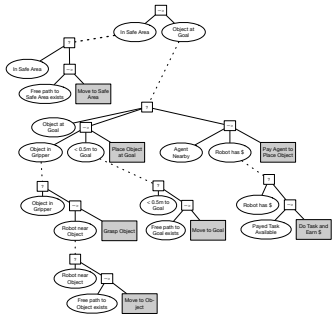
Fig. 1. A BT including the four actions *Move to Object*, *Grasp Object*, *Move to Goal*, *Place Object at Goal*, designed in a way to provide disturbance rejection at the task level. An extended version, including additional objectives and alternative ways to achieve subgoals, can be found in Fig. 5

What are Performance Guarantees?

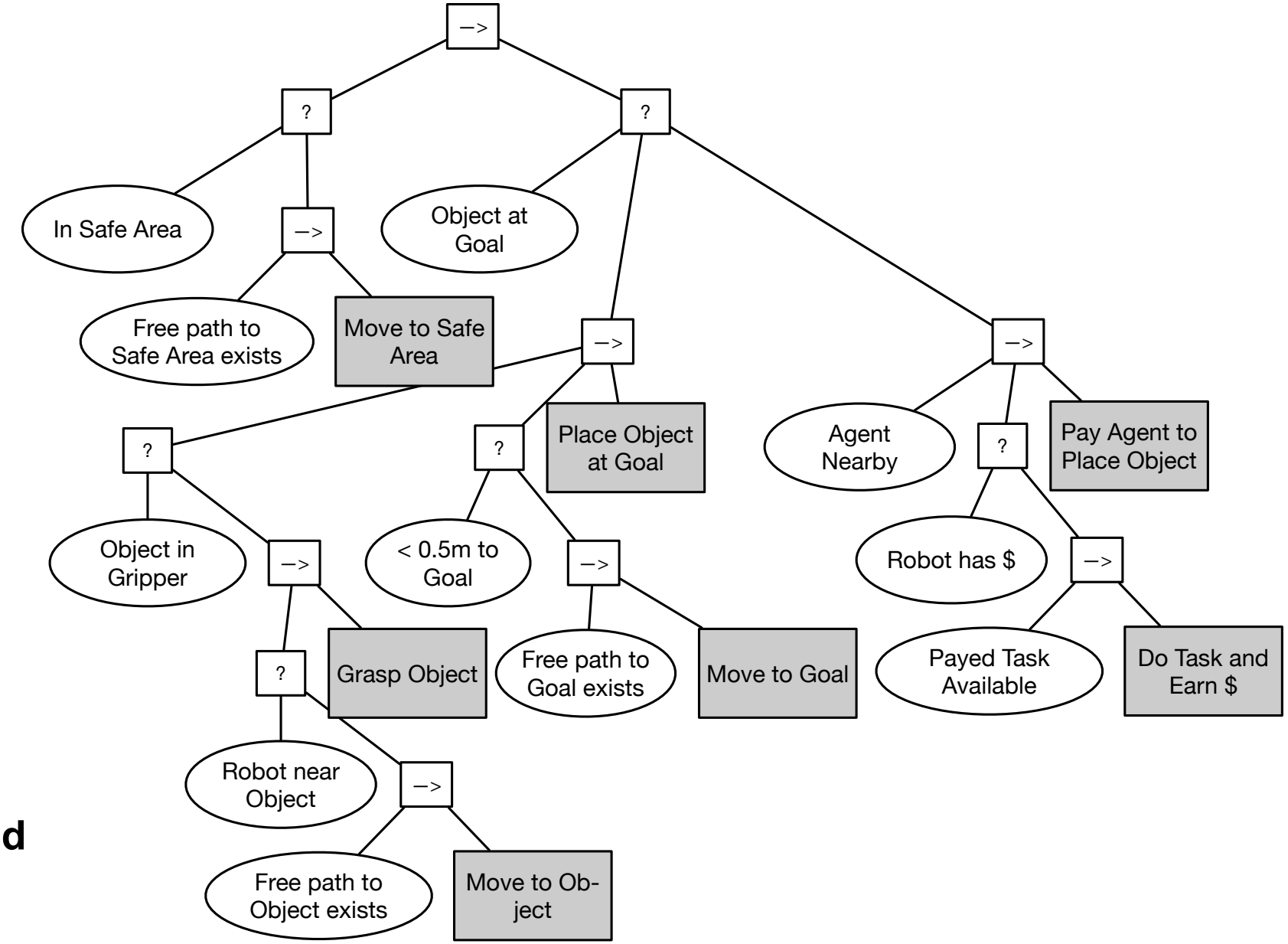
Can we **guarantee Mission Accomplishment**:
InSafeArea AND ObjectAtGoal AND AtCharger?



Backward Chained Behavior Tree

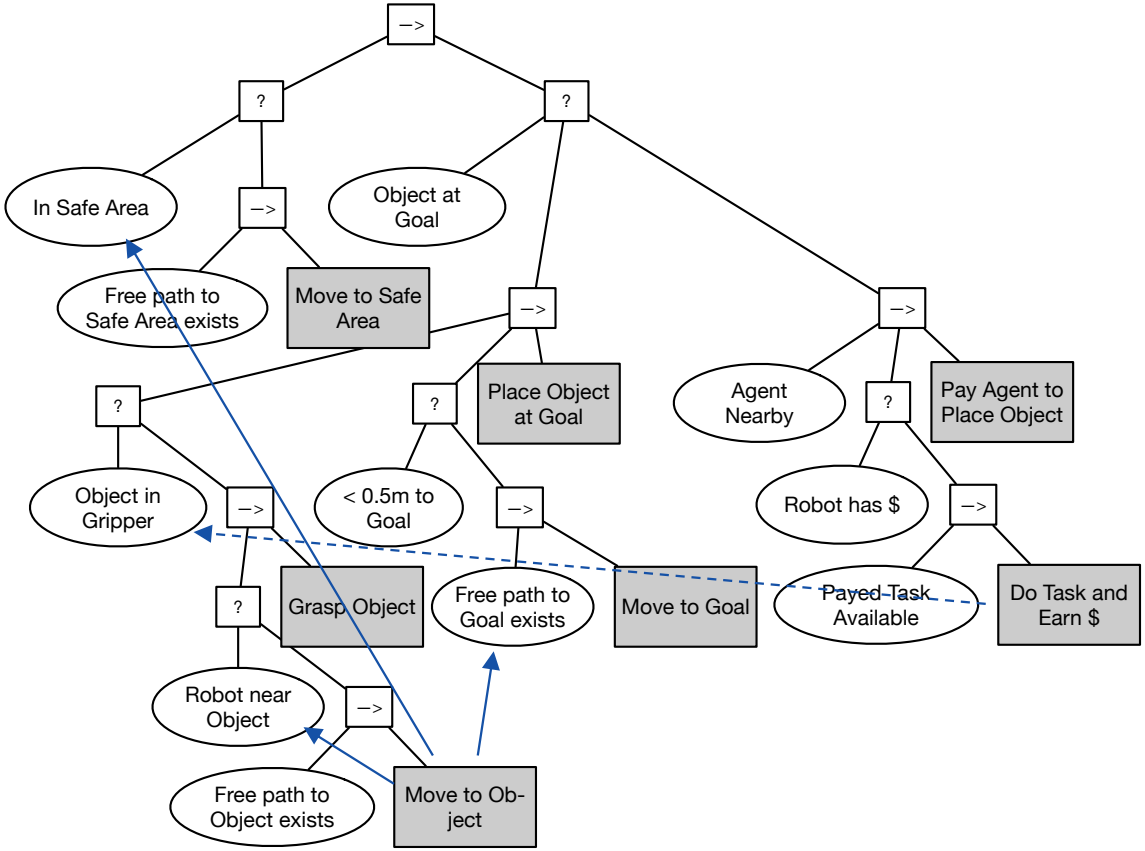


Backward Chained Behavior Tree



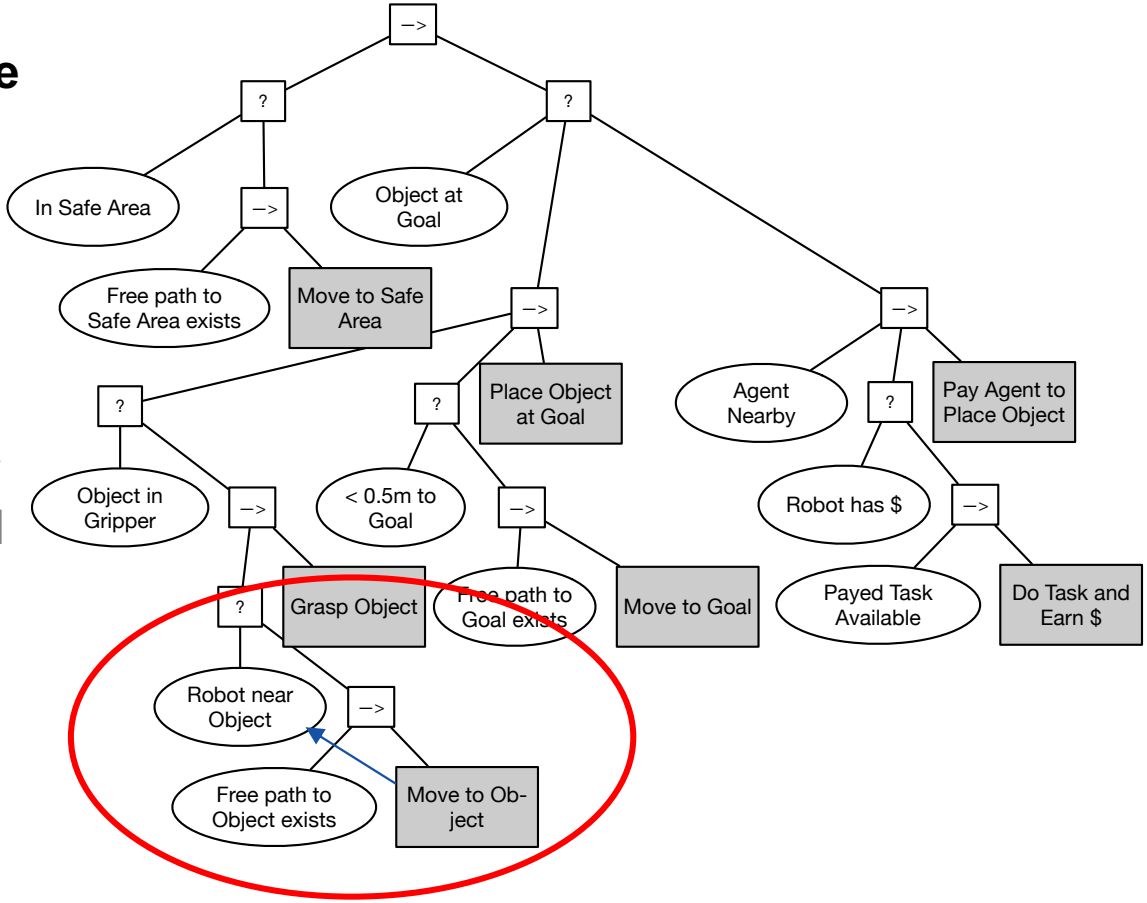
Four Design Specifications for the guarantees to hold

- Action achieves Post-condition in finite time
  - Move to Object satisfies Robot near Object
- Action does not violate key previously achieved subgoals
  - Move to Object does not violate In Safe Area
- Action does not violate conditions needed for later actions
  - Move to Object does not destroy passage to goal area (Free path to object exists)
- Action does not invoke previously failed subtree
  - Explained later…



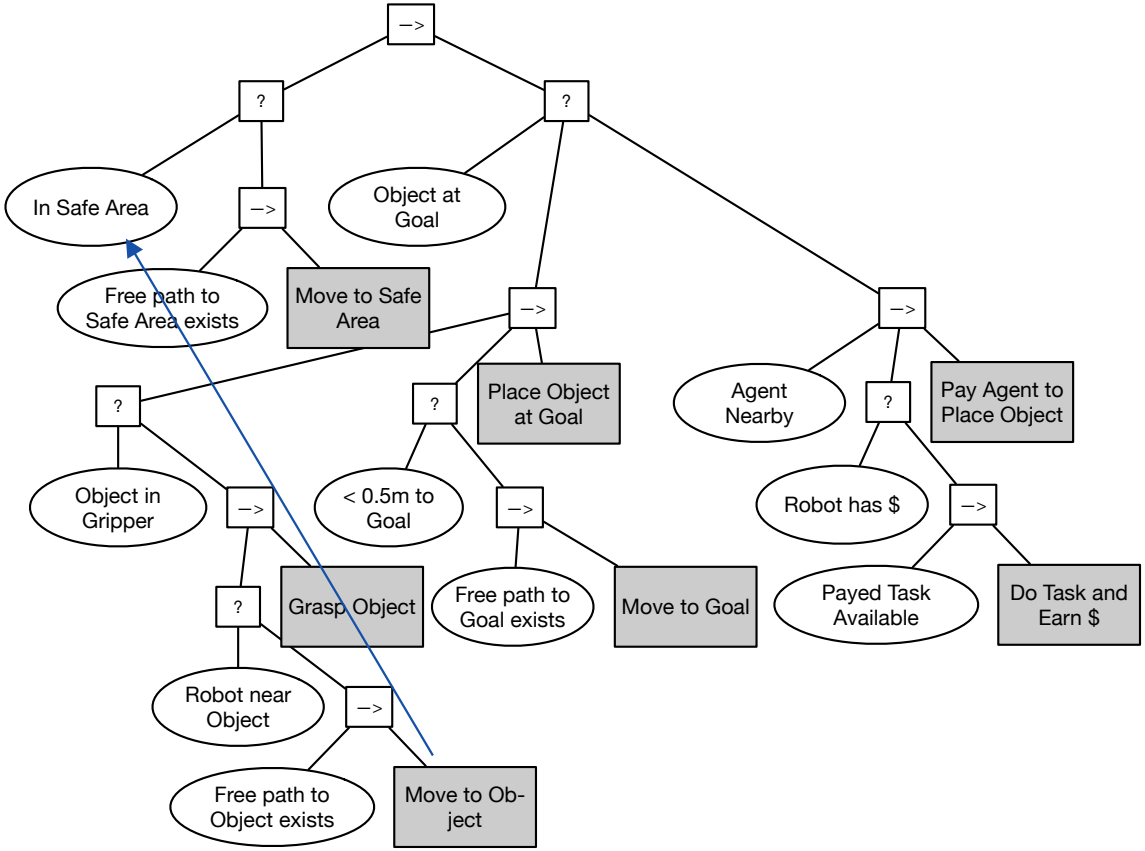
Four Design Specifications for the guarantees to hold

- **Action achieves Post-condition in finite time**
  - **Move to Object satisfies Robot near Object**
- Action does not violate key previously achieved subgoals
  - Move to Object does not violate In Safe Area
- Action does not violate conditions needed for later actions
  - Move to Object does not destroy passage to goal area (Free path to object exists)
- Action does not invoke previously failed subtree
  - Explained later…



Four Design Specifications for the guarantees to hold

- Action achieves Post-condition in finite time
  - Move to Object satisfies Robot near Object
- **Action does not violate key previously achieved subgoals**
  - **Move to Object does not violate In Safe Area**
- Action does not violate conditions needed for later actions
  - Move to Object does not destroy passage to goal area (Free path to object exists)
- Action does not invoke previously failed subtree
  - Explained later…



Preserve (some) earlier subgoals

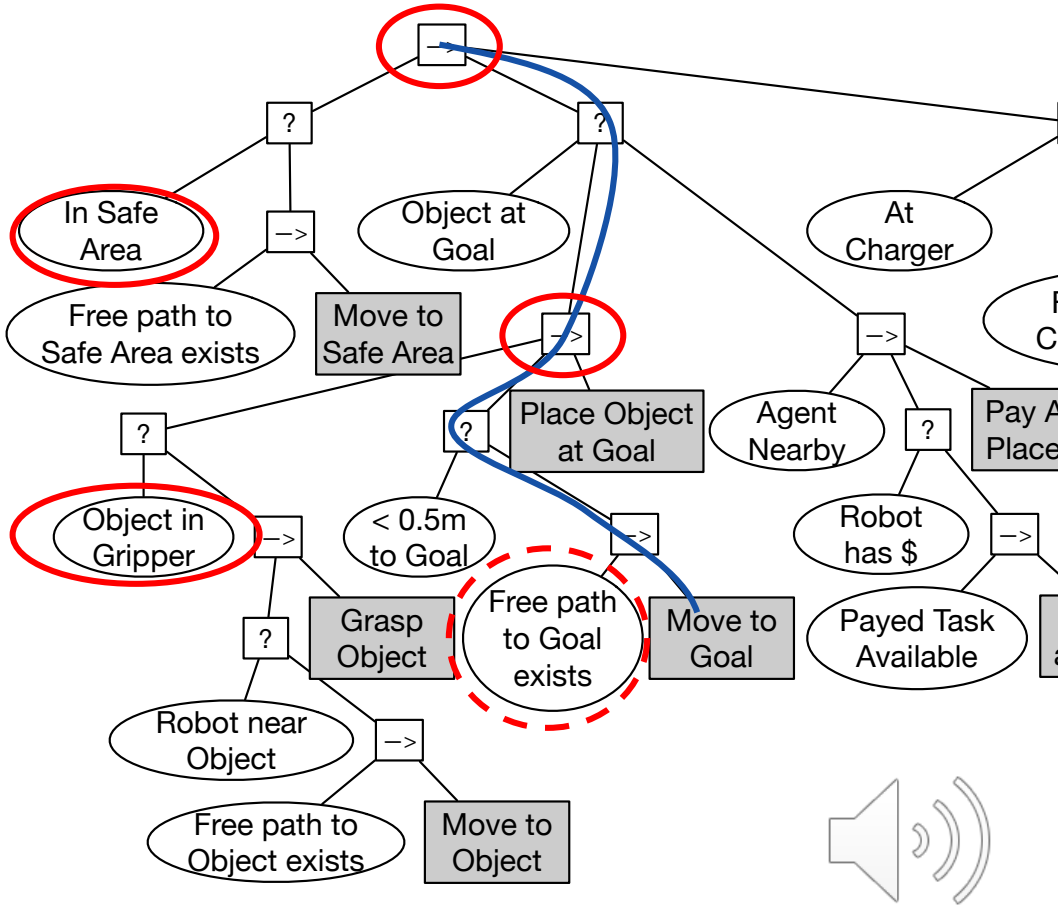
- "The actions satisfy their postconditions, without violating **important achieved subgoals** (ACCs)"
- How can we make this happen?
- Examples
  - Move to Object
    - *plans a path that avoids "Unsafe area"*
  - Move to Goal
    - *plans a path that avoids "Unsafe area"*
    - *moves slowly to keep "Object in Gripper"*
- ACCs can thus be "Obstacles" in
  - Configuration space
  - Velocity/acceleration space
- Assumption gives explicit "Design specifications" for Controllers/Actions

Actions A_i	Postconditions of A_i (Objectives)	Active Constraint Conditions $ACC(i)$
Move to Safe Area	In Safe Area	-
Move to Object	Near Object	In Safe Area
Grasp Object	Object Grasped	In Safe Area
Move to Goal	Near Goal	In Safe Area AND Object in Gripper
Place Object	Object at Goal	In Safe Area
Do Task and Earn $	Robot has $	In Safe Area AND Agent Nearby
Pay Agent to Place Object	Object at Goal	In Safe Area

Which subgoals to Preserve?

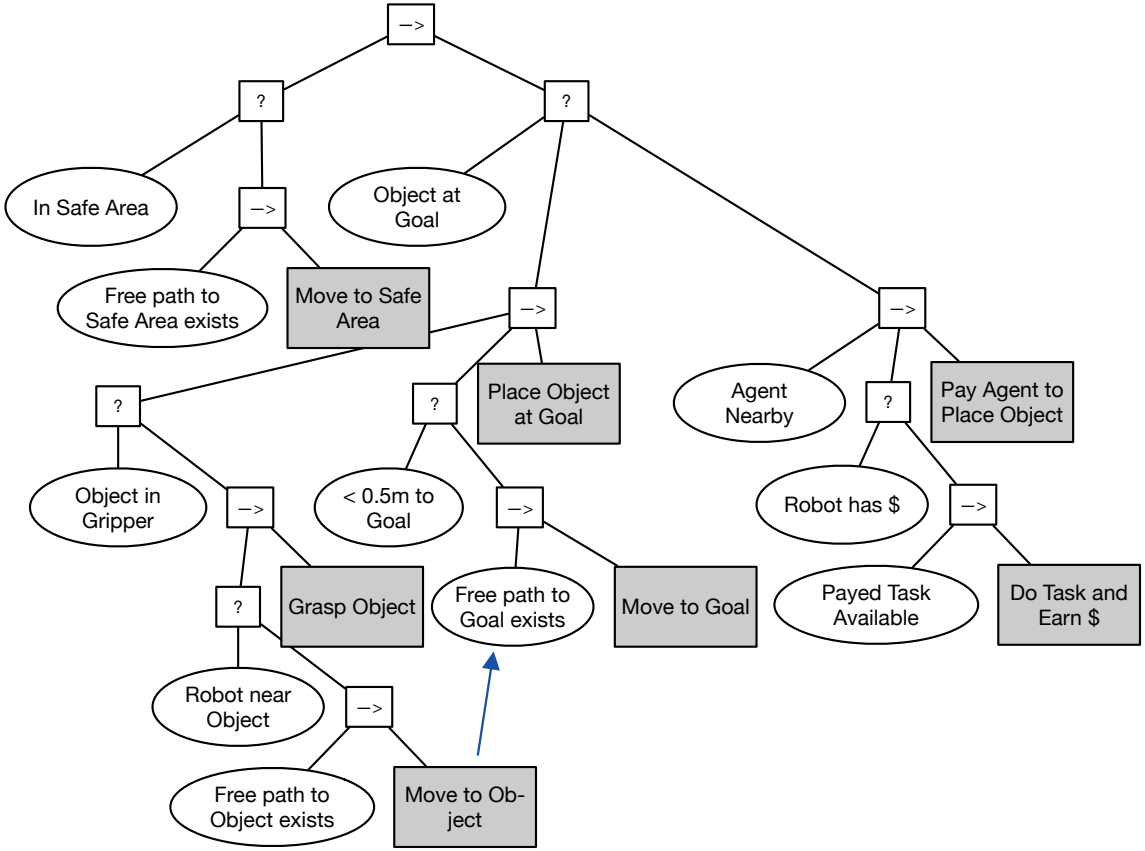
- Two ways to find the subgoals to preserve
  - During execution find non-preconditions returning Success
  - Analytically study BT to find the above
    - > *Check Sequence nodes in the BT between Action and Root*
    - > *Look for older children of those nodes to the left*
    - > *Pre-conditions of the action can be violated at the instant of achieving the postcondition*

Actions A_i	Postconditions of A_i (Objectives)	Active Constraint Conditions $ACC(i)$
Move to Safe Area	In Safe Area	-
Move to Object	Near Object	In Safe Area
Grasp Object	Object Grasped	In Safe Area
Move to Goal	Near Goal	In Safe Area AND Object in Gripper
Place Object	Object at Goal	In Safe Area
Do Task and Earn $	Robot has $	In Safe Area AND Agent Nearby
Pay Agent to Place Object	Object at Goal	In Safe Area



Four Design Specifications for the guarantees to hold

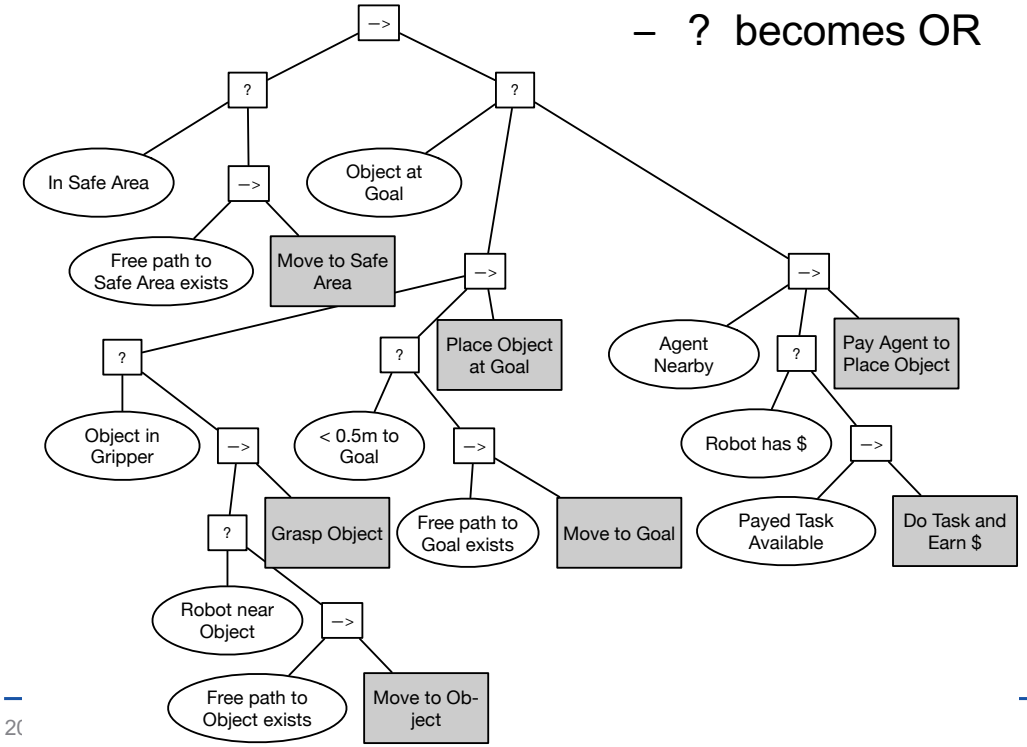
- Action achieves Post-condition in finite time
  - Move to Object satisfies Robot near Object
- Action does not violate key previously achieved subgoals
  - Move to Object does not violate In Safe Area
- **Action does not violate conditions needed for later actions**
  - **Move to Object does not destroy passage to goal area (Free path to object exists)**
- Action does not invoke previously failed subtree
  - Explained later…

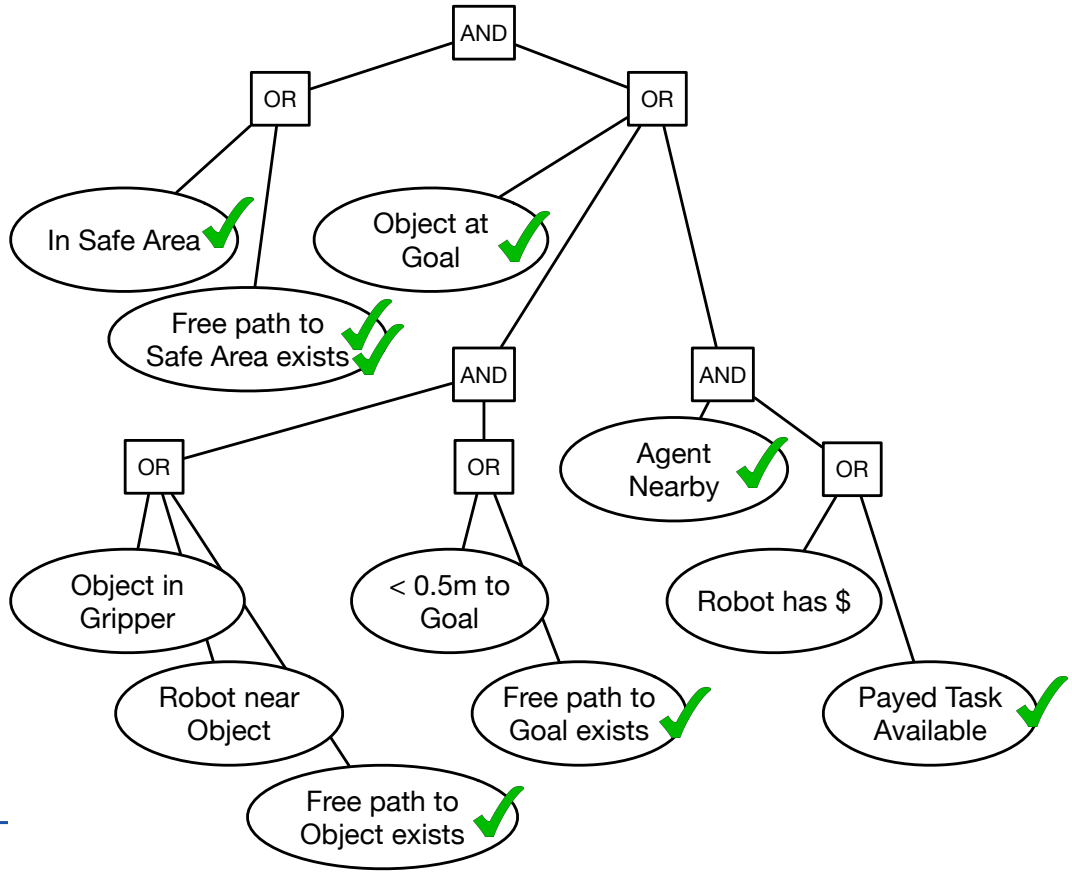


The region of attraction of the BT

Shows **what kind of disturbances can be handled**

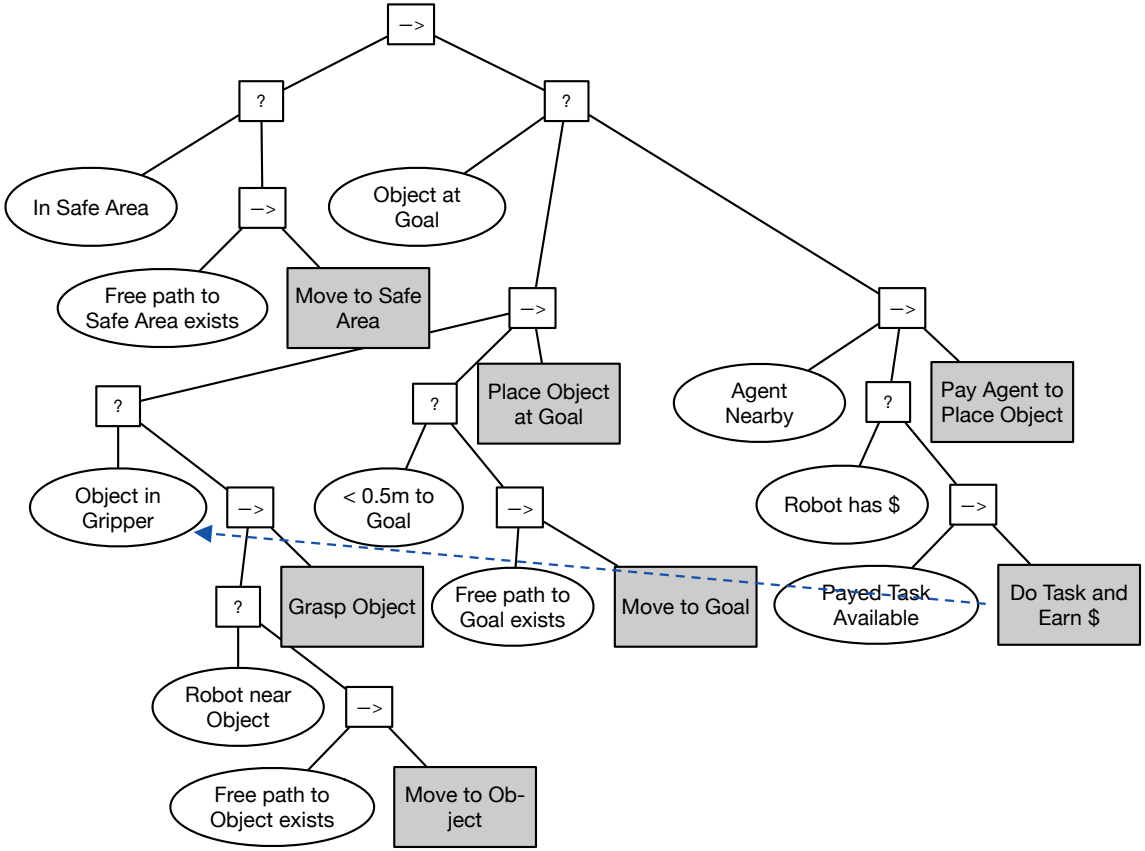
- Create And-Or-Tree
  - Remove Actions
  - → becomes AND
  - ? becomes OR





Four Design Specifications for the guarantees to hold

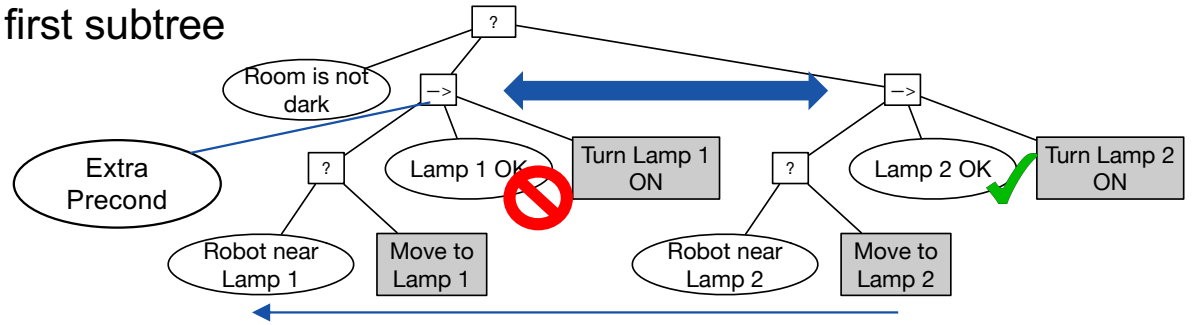
- Action achieves Post-condition in finite time
  - Move to Object satisfies Robot near Object
- Action does not violate key previously achieved subgoals
  - Move to Object does not violate In Safe Area
- Action does not violate conditions needed for later actions
  - Move to Object does not destroy passage to goal area (Free path to object exists)
- **Action does not invoke previously failed subtree**
  - **Explained now…**

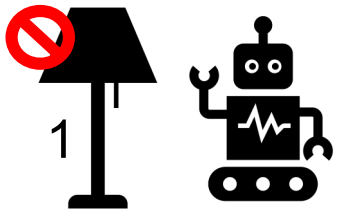


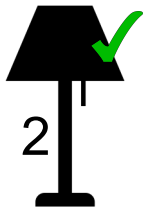
Example: Turn the light on

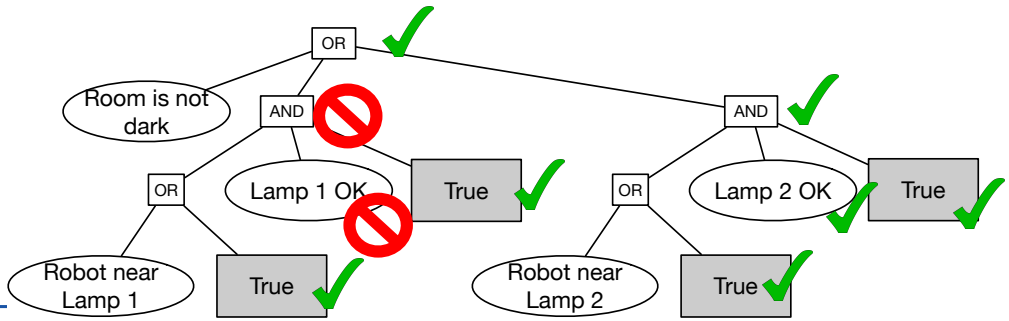
Avoid starting a sub-plan that is doomed to fail

- If Lamp 1 is broken, the policy will still try to move to Lamp 1…
- Solution:
  - Swap order of Fallbacks (so Lamp 2 is first option after initial fail)
  - Add precondition to deactivate first subtree
- When?
  - Use AndOr-tree
  - Count # fails



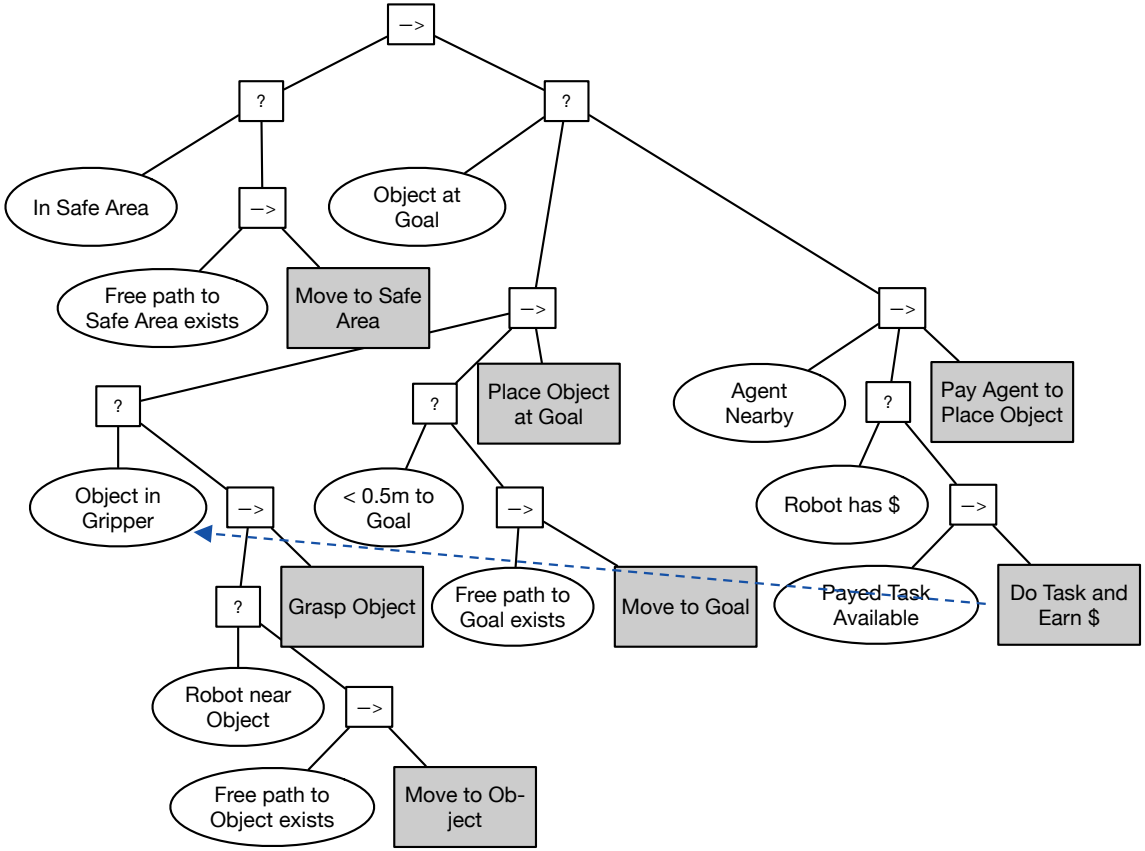






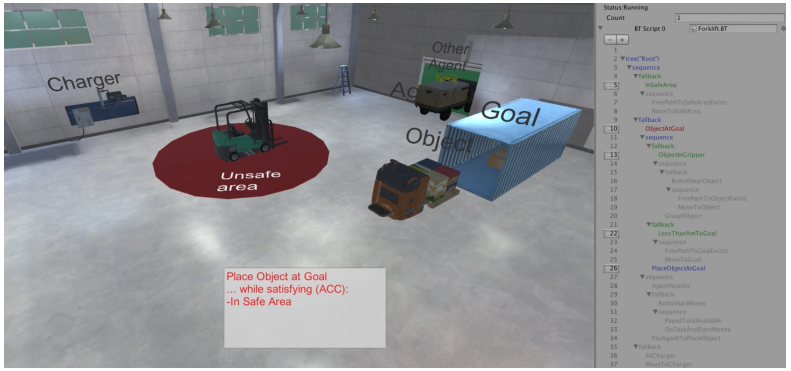
Four Design Specifications for the guarantees to hold

- Action achieves Post-condition in finite time
  - Move to Object satisfies Robot near Object
- Action does not violate key previously achieved subgoals
  - Move to Object does not violate In Safe Area
- Action does not violate conditions needed for later actions
  - Move to Object does not destroy passage to goal area (Free path to object exists)
- **Action does not invoke previously failed subtree**
  - **Explained now…**



Content

- When to use Behavior Trees (BTs)?
  - Creating complex controllers/policies
- What are BTs?
  - Hierarchically modular policies
- How to create BTs?
  - Improvise
  - Use planning (backward chaining)
- The Big Picture
  - Genetic Algorithms
  - Reinforcement Learning
  - Control Theory (Performance Guarantees)



References

- [1] Biggar, Oliver, Mohammad Zamani, and Iman Shames. "On modularity in reactive control architectures, with an application to formal verification." *arXiv preprint arXiv:2008.12515* (2020).
- [2] Biggar, Oliver, Mohammad Zamani, and Iman Shames. "An expressiveness hierarchy of Behavior Trees and related architectures." *IEEE Robotics and Automation Letters* 6.3 (2021): 5397-5404.
- [3] Colledanchise, Michele, and Petter Ögren. "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees." *IEEE Transactions on robotics* 33.2 (2016): 372-389.

Questions