Behaviour trees

Andrea Roli
andrea.roli@unibo.it

DISI - Dept. of Computer Science and Engineering
Campus of Cesena
Alma Mater Studiorum Università di Bologna

Intelligent robotic systems

Bibliographic support for these slides

M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI – An Introduction*, CRC Press, 2018.
Available for downloading at:
`https://arxiv.org/abs/1709.00084`

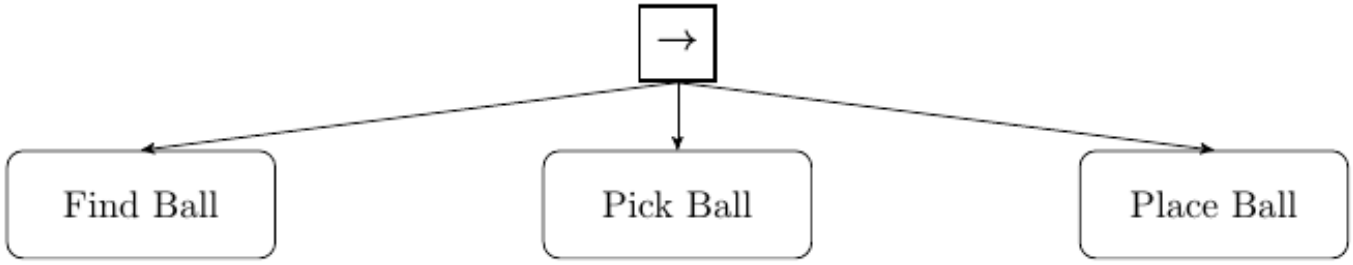
J. Kuckling, A. Ligot, D. Bozhinoski and M. Birattari, Behavior Trees as a Control Architecture in the Automatic Modular Design of Robot Swarms, Swarm Intelligence. ANTS 2018. Lecture Notes in Computer Science, vol 11172. Springer, Cham, 2018.

(Some figures and text in these slides are taken from these references)

Behaviour trees

- A Behavior Tree (BT) is a way to structure the switching between different tasks
- BTs are a very efficient way of creating a control software for robot that is **modular** and **reactive**
- BTs subsume several previous models, such as FSMs and the subsumption architecture

A simple example



A high level BT carrying out a task consisting of first finding, then picking and finally placing a ball

BT definition

- A BT is a tree structure that contains one root node, *control nodes*, and *execution nodes* (actions or conditions)
- For each connected node we use the common terminology of parent and child
- The root is the node without parents; all other nodes have one parent
- The control nodes have at least one child

BT execution (1/2)

- Execution is controlled by a tick generated by the root and propagated through the tree.
- When ticked by its parent, a node is activated. After execution, it returns one of three possible values: *success*, *running*, or *failure*.
- Condition nodes that are ticked observe the world state and return success if their condition is fulfilled; they return failure otherwise.
- Action nodes that are ticked returns success if their action is completed; failure, if their action cannot be completed; and running, if their action is still in progress.

BT execution (2/2)

- Control nodes distribute the tick to their children. Their return value depends on those returned by the children. There are six different types of control nodes (see next slide).

In the basic implementation, the execution process traverses down from the root of the tree every single step, testing each node down the tree to see which is active, rechecking any nodes along the way, until it reaches the currently active node to tick it again.

Control nodes: execution

Name	Symbol	Description
selector	?	Ticks children sequentially as long as they return *failure*.
selector*	$?^*$	Ticks children sequentially as long as they return *failure*. Resumes ticking at last ticked node, if it returned *running*.
sequence	$\rightarrow$	Ticks children sequentially as long as they return *success*.
sequence*	$\rightarrow^*$	Ticks children sequentially as long as they return *success*. Resumes ticking at last ticked node, if it returned *running*.
parallel	$\rightrightarrows$	Ticks all children simultaneously. Returns *success* (or *failure*), if a majority of the children return *success* (or *failure*). Otherwise it returns *running*.
decorator	δ	Executes a custom function on its only child. The function can either manipulate the number of ticks given to the child, or the value returned to the parent.

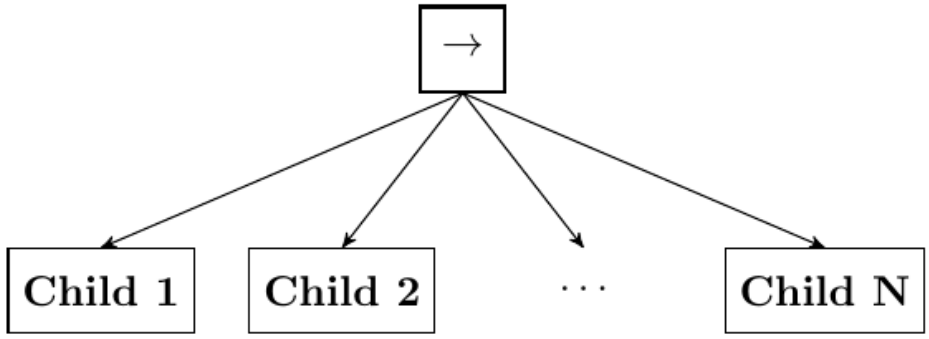
(taken from Kuckling et al. 2018)

Control nodes: return values

Node type	Symbol	Succeeds	Fails	Running
Fallback	?	If one child succeeds	If all children fail	If one child returns Running
Sequence	$\rightarrow$	If all children succeed	If one child fails	If one child returns Running
Parallel	$\rightrightarrows$	If $\geq M$ children succeed	If $> N - M$ children fail	else
Action	text	Upon completion	If impossible to complete	During completion
Condition	text	If true	If false	Never
Decorator	$\Diamond$	Custom	Custom	Custom

Note: Fallback node $\equiv$ Selector node

Sequence node

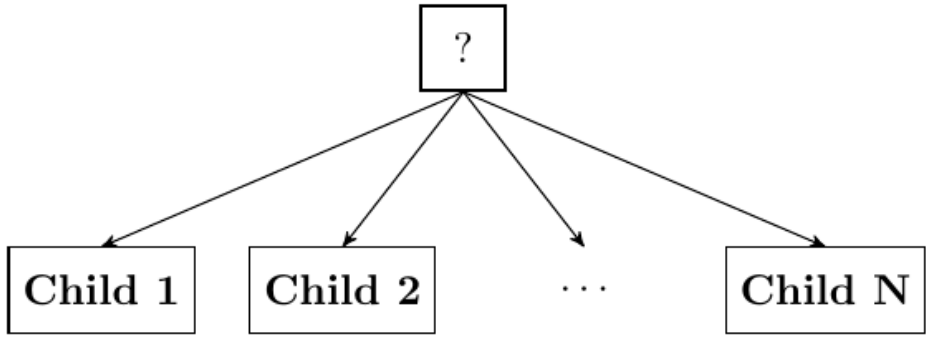


Algorithm 1: Pseudocode of a Sequence node with N children

```
1 for i ← 1 to N do
2     childStatus ← Tick(child(i))
3     if childStatus = Running then
4         return Running
5     else if childStatus = Failure then
6         return Failure
7 return Success
```

Fallback node

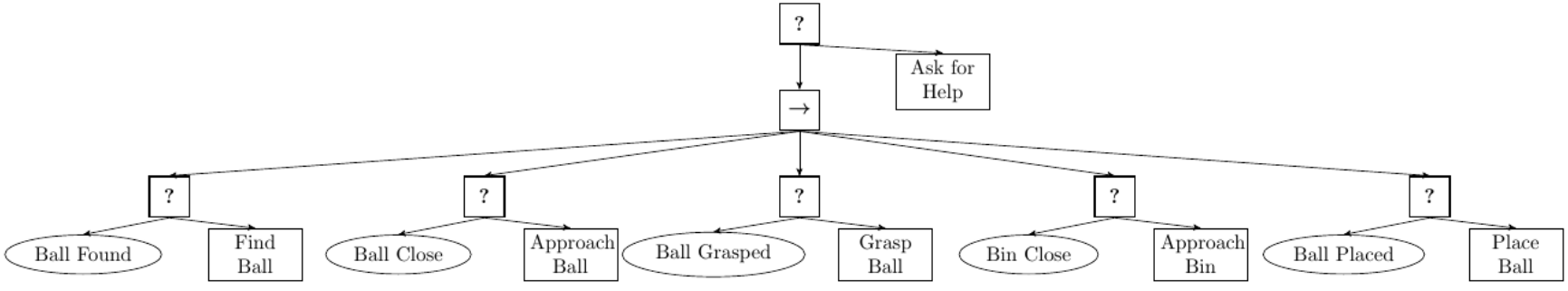
aka: Selector node



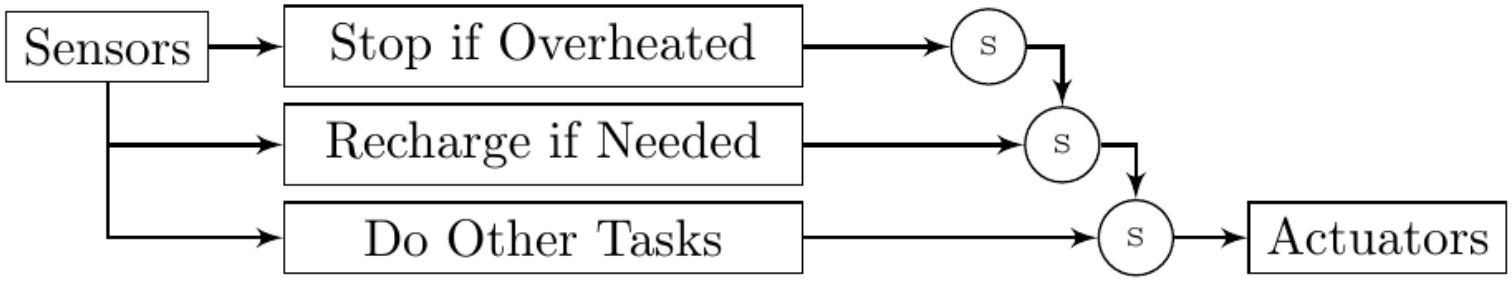
Algorithm 2: Pseudocode of a Fallback node with N children

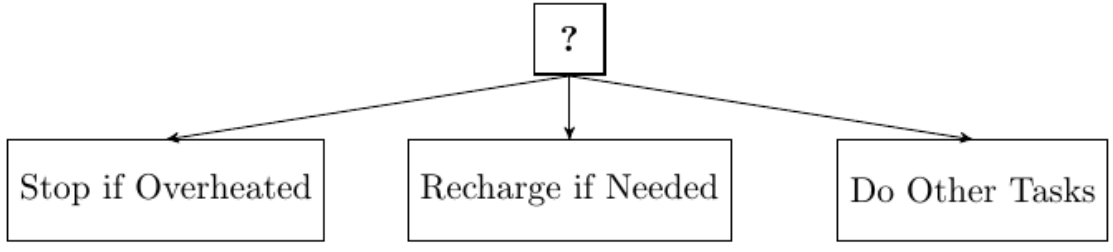
```
1 for i ← 1 to N do
2     childStatus ← Tick (child(i))
3     if childStatus = Running then
4         return Running
5     else if childStatus = Success then
6         return Success
7 return Failure
```

Example: Pick a ball



BT for the subsumption architecture





BT for the subsumption architecture

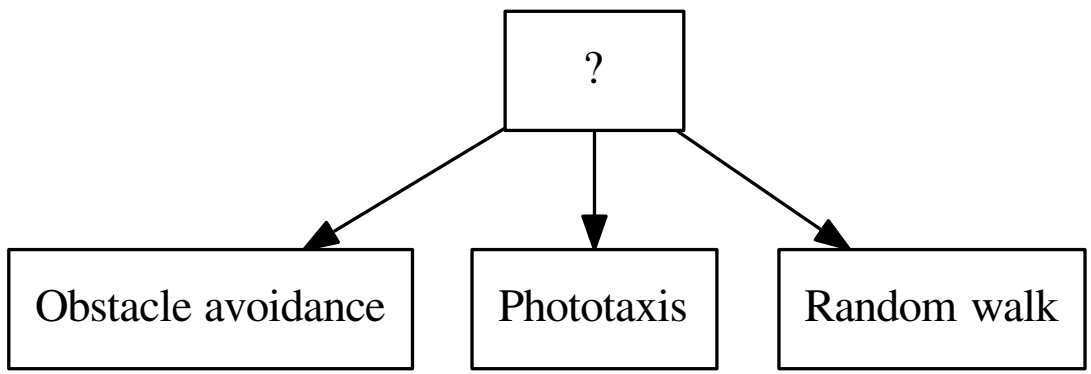
- Given a Subsumption architecture, we can create an equivalent BT by arranging the controllers as actions under a *Fallback* composition, in **from higher to lower priority**, from left to right.
- Furthermore, we let the return status of the actions be *Failure* if they do not need to execute, and *Running* if they do.
- They never return *Success*.
- **Each behaviour contains both its activation conditions and the execution code**.

Example: phototaxis with obstacle avoidance

Level 2: Obstacle avoidance

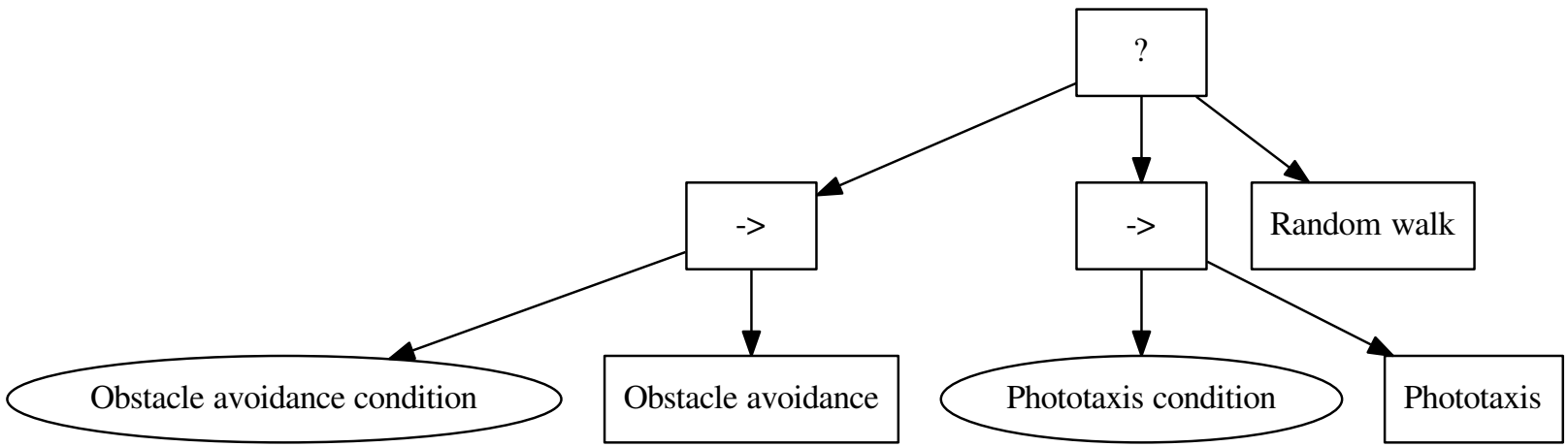
Level 1: Phototaxis

Level 0: Random walk



A variant

- Behaviour nodes are run only if the condition is satisfied.
- This way, we can compose previously coded behaviours just adding suitable execution conditions.
- **Pros**: favours modularity and reuse.
- **Cons**: code might be redundant and care must be taken about coherence between condition and behaviour.



BTs in Lua

A nice implementation of BTs in Lua (suitable to be used with ARGoS) is provided by Michael Allwright:

`github.com/allsey87/luabt`