Trapping Behavior Trees in Esterel

Alexander Schulz-Rosengarten*, Michael Mendler†, Joaquín Aguado†,
Malte Clement* and Reinhard von Hanxleden*

*Kiel University and †Bamberg University

SYNCHRON 2023, Kiel, Germany

Extension of [DATE'23 WiP]

Praise for Behavior Trees

*"[...]. Sure you could build the very same behaviors with a finite state machine (FSM). But anyone who has worked with this kind of technology in industry knows how fragile such logic gets as it grows. A finely tuned **hierarchical FSM** before a game ships is often a temperamental work of art not to be messed with!"*

Alex J. Champandard
Editor in Chief & Founder AiGameDev.com,
Senior AI Programmer Rockstar Games

This quote and parts of the following material taken from
[Colledanchise Ogren '20]
Michele Colledanchise and Petter Ogren,
Behavior Trees in Robotics and AI - An Introduction, 2020

https://arxiv.org/pdf/1709.00084.pdf

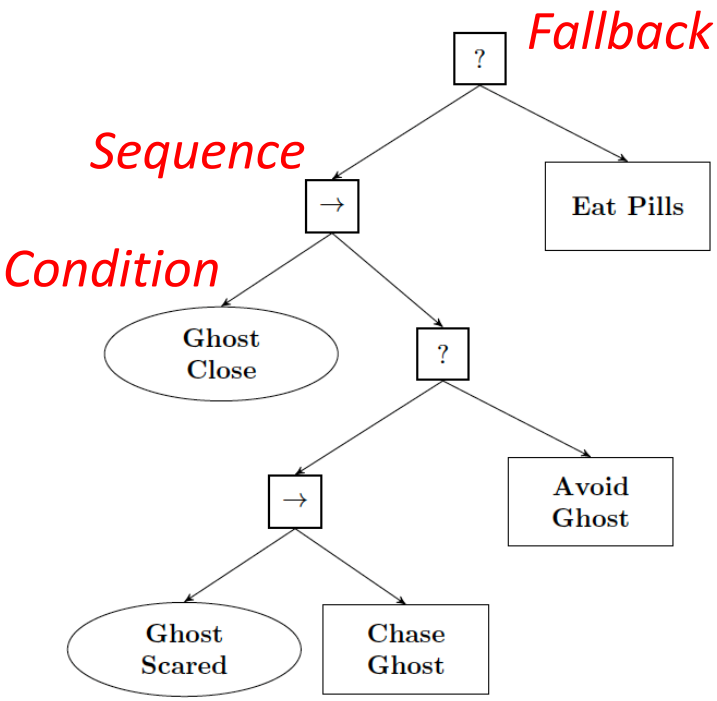
if ghost is close

then if ghost is scared

then chase ghost

else avoid ghost

else eat pills



Key Point:

At every tick, start at **root** of BT

Figs from [Colledanchise Ogren '20]

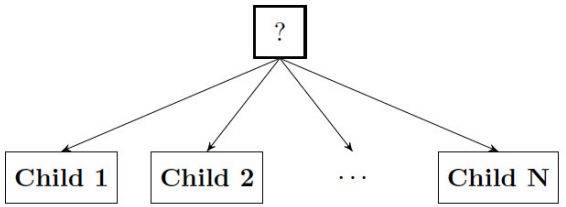


Fig. 1.3: Graphical representation of a Fallback node with N children.

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
```

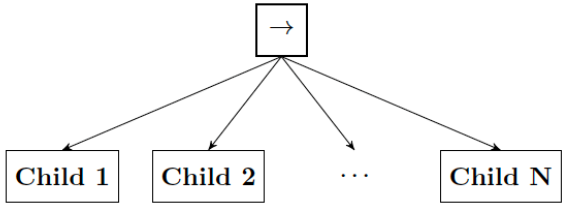


Fig. 1.2: Graphical representation of a Sequence node with N children.

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
```

Behavior Tree Building Blocks

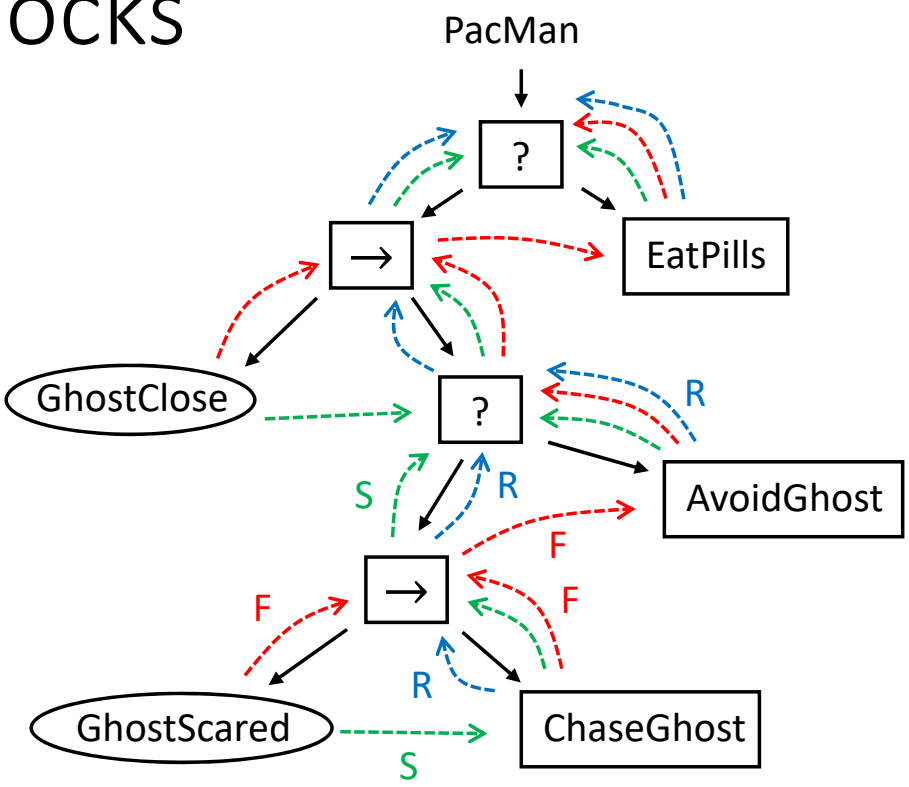
In each reaction cycle (tick) the tree is traversed for activation

- top-down, post-order

with child-skipping modulated by return codes

- SUCCESS (S)
- FAILURE (F)
- RUNNING (R)

that propagate upwards.



BT Control-Flow = Decision Graph

Behavior Tree Building Blocks

Control Flow nodes: Sequence, Fallback/Selector, Parallel, Decorator

Execution nodes: Action/Task, Condition

Possible return values: Success, Running, Failure

Node type	Symbol	Succeeds	Fails	Running
Fallback	?	If one child succeeds	If all children fail	If one child returns Running
Sequence	→	If all children succeed	If one child fails	If one child returns Running
Parallel	⇉	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	◊	Custom	Custom	Custom

Table from [Colledanchise Ogren '20]

Behavior Trees in Lingua Franca

See SYNCHRON ’22!

https://rtsys.informatik.uni-kiel.de/~biblio/downloads/papers/synchron22.pdf

Behavior Trees in Esterel

Preliminary work, barely …

Our Work

Here: Give a simple compositional mapping of BTs into Esterel

- BT conditions and actions ⇒
  Esterel signals (asynchronous actions) or
  Esterel modules (for hierarchical BTs)
- BT control-flow nodes ⇒ Esterel primitive operators (traps!!)

Expected Benefits for BT

- Established modelling & compilation tools for BT programming
- Strong mathematical semantics of Esterel for test & verification
- Constructive semantics gives formal guarantees for determinacy, predictability, convergence, ….

Recall (Some) Basic Esterel Operators

`s1 ; s2`	Run s1, s2 sequentially
`s1 || s2`	Run s1, s2 in parallel
`pause`	Finish tick (terminate with completion code 1)
`trap T in s end`	Declare trap scope
`exit T`	Exit trap (terminate with completion code 2 or higher)

Example:

```
trap T in
  present I then exit T end;      // If I holds in first tick: terminate whole program
  pause;
  emit O                          // Otherwise: emit O in second tick
end
```

Mapping BTs to Esterel – 1st Approach

Observation: return values correspond nicely to completion codes in Esterel.

Esterel in turn can be mapped to hierarchical FSMs,
which should also work for LF modal models

0 – (normal) termination – “Succeeds”

2 (and higher) – throw exception – “Fails”

1 – pause operation – “Running”

Node type	Symbol	Succeeds	Fails	Running
Fallback	?	If one child succeeds	If all children fail	If one child returns Running
Sequence	→	If all children succeed	If one child fails	If one child returns Running
Parallel	⇉	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	◊	Custom	Custom	Custom

[Colledanchise Ogren ’20]

“Sequence” in Esterel – Not

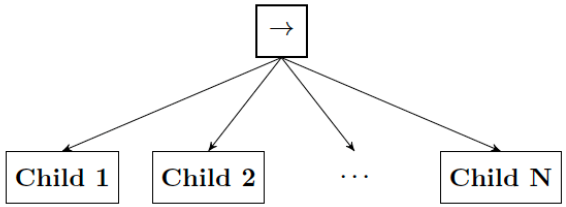


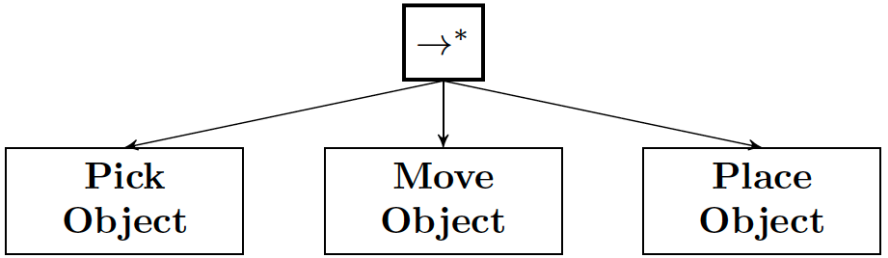
Fig. 1.2: Graphical representation of a Sequence node with N children.

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
```

The problem: this translation implements an “un-reactive” *sequence with memory*, where we resume at running children, instead of re-starting each tick at first child again

// Children signal failure with “exit Failure”
// If any child fails, this is propagated out
// Otherwise, terminate normally (success)
Sequence(child1, child2, …, childN):
child1;
child2;
…
childN;

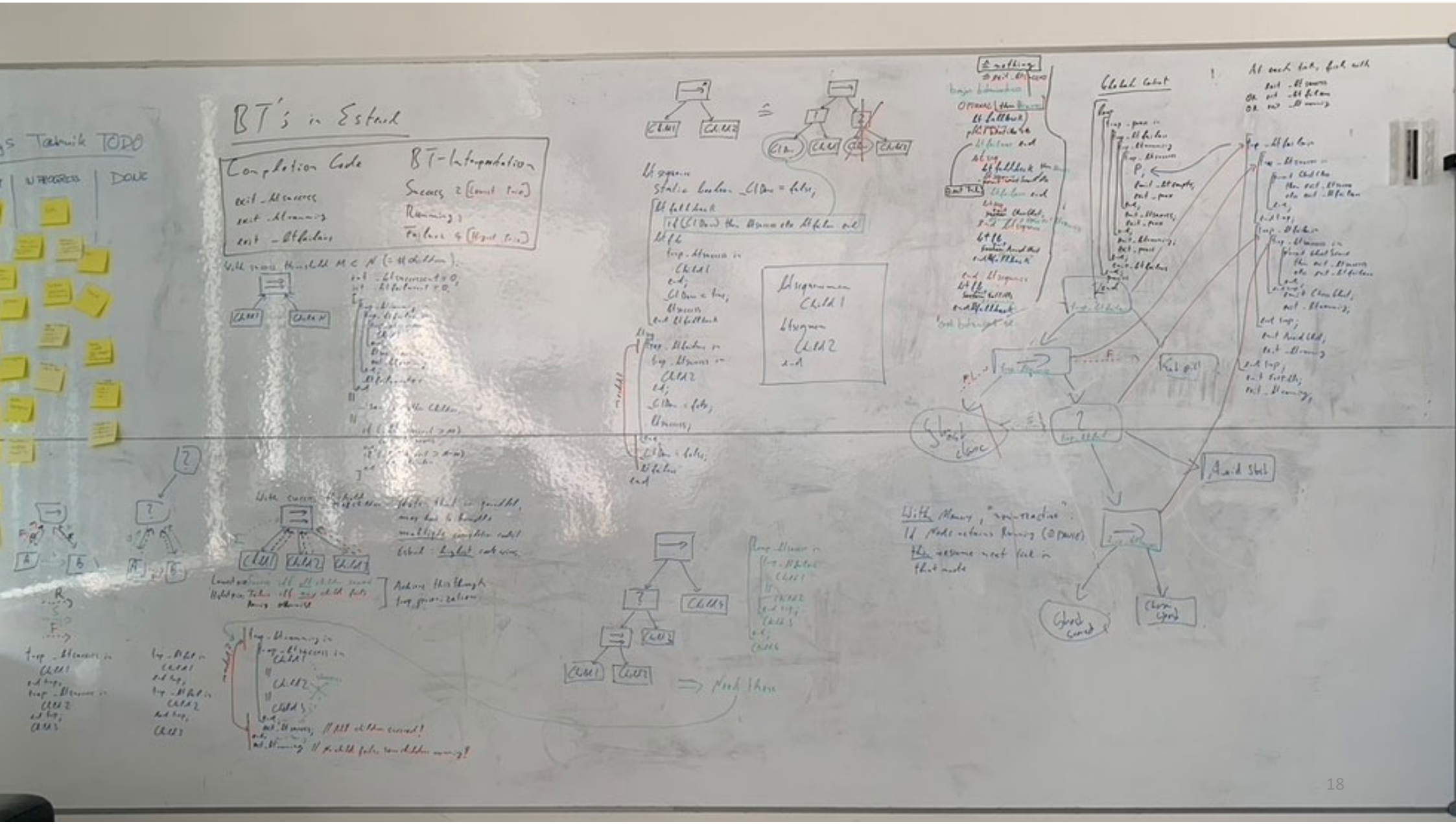


[Colledanchise Ogren ’20]

Behavior Trees in Esterel

~~Preliminary work, barely …~~

The New Thing This Time – and no more beers…


BT's in Estack
Completion Code
BT-Interpretation
Success
Running
Failure
TODO
IN PROGRESS
DONE
Global Cohort
Need them

Operators of BT Esterel

P	::=	`run` a	action call ($a \in \mathcal{A}$)
	|	`exit` T	exception ($T \in \mathcal{T} \cup \{0, 1\}$)
	|	`trap` T `in` P	trap handler ($T \in \mathcal{T}$)
	|	`trap` T `in` P `do` P	trap handler ($T \in \mathcal{T}$)
	|	P ; P	sequential composition
	|	P ‖ P	concurrent parallel

$T \in \mathcal{T}$ trap names $\quad$ $a \in \mathcal{A}$ action/condition names

- run a encapsulates full BT trees or external actions/conditions (may raise traps)
- As in BT we assume the side effects are interference-free (Church-Rosser)

Coding

Completion Code	Esterel	BT
S	User-defined Trap Exit	Success
R	User-defined Trap Exit	Running
F	User-defined Trap Exit	Failure

Conjecture

Memory-free BT coincides with the fragment of BT-Esterel restricted to 3 trap names $\mathcal{T} = \{\mathsf{S}, \mathsf{F}, \mathsf{R}\}$. (exploiting "shadowing")

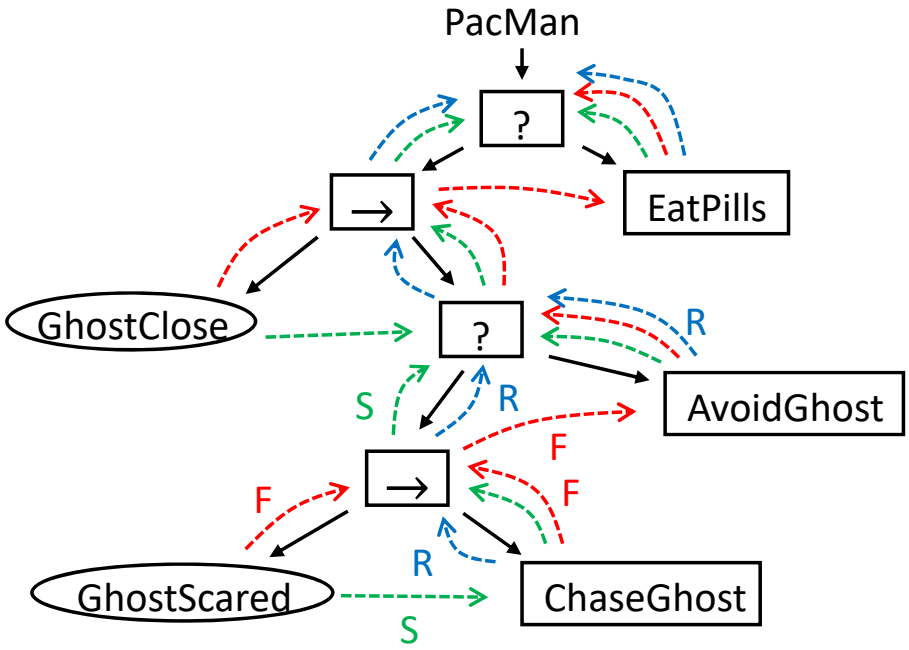
PacMan

Abbreviations

$$P \gg_T Q =_{df} \texttt{trap}\, T\, \texttt{in}\, P\, \texttt{do}\, Q$$

$$P \,?\, Q =_{df} P \gg_{\mathsf{F}} Q$$

$$P \to Q =_{df} P \gg_{\mathsf{S}} Q$$



$$\mathsf{PacMan} =_{df} (\mathsf{GhCl} \gg_{\mathsf{S}} ((\mathsf{GhSc} \gg_{\mathsf{S}} \mathsf{ChGh}) \gg_{\mathsf{F}} \mathsf{AvGh})) \gg_{\mathsf{F}} \mathsf{EatP}$$

Semantics (brief)

Trap context $\tau \in \mathcal{T}^*$

- The trap context fixes the scoping order of free trap names (left-most = inner-most)
- The trap context is identified modulo shadowing, e.g., $[\mathsf{S}, \mathsf{R}, \mathsf{F}] \cong [\mathsf{S}, \mathsf{R}, \mathsf{S}, \mathsf{F}]$

$$\begin{array}{l}\texttt{trap}\, F\, \texttt{in} \\ \quad \texttt{trap}\, R\, \texttt{in} \\ \quad\quad \texttt{trap}\, \mathsf{S}\, \texttt{in} \\ \quad\quad\quad P \text{ ----- } [\mathsf{S}, \mathsf{R}, \mathsf{F}] \\ \quad\quad \texttt{do}\, Q_{\mathsf{S}} \\ \quad \texttt{do}\, Q_{\mathsf{R}} \\ \texttt{do}\, Q_{\mathsf{F}}\end{array}$$

Reduction step $\Sigma \vdash P \twoheadrightarrow_\tau \Sigma' \vdash P' \Downarrow T$

$$P =_\tau Q \text{ iff } [\![P]\!]_\tau = [\![Q]\!]_\tau$$

$$[\![P]\!]_\tau \quad = \quad \{(\Sigma', P', T) \mid \Sigma \vdash P \twoheadrightarrow_\tau \Sigma' \vdash P' \Downarrow T\}$$

BT Algebra

Monoid Structure

$$P \gg_T \texttt{exit}\, T =_\tau P$$ Neutral Element

$$\texttt{exit}\, T \gg_T P =_\tau P$$ Neutral Element

$$P \gg_T (Q \gg_T R) =_\tau (P \gg_T Q) \gg_T R$$ Associativity

Negation („Decorator“)

We can flip (permute) the traps using …

$$\sim P =_\tau ((\texttt{trap}\, \mathsf{S}\, \texttt{in}\, P) \gg_{\mathsf{F}} \texttt{exit}\, \mathsf{S})\ ;\ \texttt{exit}\, \mathsf{F}$$

$$\sim \sim P =_\tau P$$

BT Algebra

Occam-DeMorgan Dualities (1)

Fallback	?	If one child succeeds	If all children fail	If one child returns Running
Sequence	→	If all children succeed	If one child fails	If one child returns Running

Fallback and Sequence are DeMorgan Duals

$$P \,?\, Q = P \gg_{\mathsf{F}} Q =_{\tau} \sim(\sim P \gg_{\mathsf{S}} \sim Q) =_{\tau} \sim(\sim P \rightarrow \sim Q)$$

$$P \rightarrow Q = P \gg_{\mathsf{S}} Q =_{\tau} \sim(\sim P \gg_{\mathsf{F}} \sim Q) =_{\tau} \sim(\sim P \,?\, \sim Q)$$

BT Algebra

Occam-DeMorgan Dualities (2)

Parallel	$\rightrightarrows$	If $\geq M$ children succeed	If $> N - M$ children fail	else

N = 2; M = 2

$$P \rightrightarrows_2 Q =_\tau \mathtt{trap}\,\mathsf{R}\,\mathtt{in}\,((\mathtt{trap}\,\mathsf{S}\,\mathtt{in}\,(P \parallel Q))\,;\,\mathtt{exit}\,\mathsf{S})\,;\,\mathtt{exit}\,\mathsf{R}$$

$$P \rightrightarrows_2 Q =_{[\mathsf{S},\mathsf{R},\mathsf{F}]} P \parallel Q$$

N = 2; M = 1

$$P \rightrightarrows_1 Q =_\tau \mathtt{trap}\,\mathsf{R}\,\mathtt{in}\,((\mathtt{trap}\,\mathsf{F}\,\mathtt{in}\,(P \parallel Q))\,;\,\mathtt{exit}\,\mathsf{F})\,;\,\mathtt{exit}\,\mathsf{R}$$

$$P \rightrightarrows_1 Q =_{[\mathsf{F},\mathsf{R},\mathsf{S}]} P \parallel Q$$

DeMorgan Dualities

$$P \rightrightarrows_2 Q =_\tau \sim(\sim P \rightrightarrows_1 \sim Q) \qquad P \rightrightarrows_1 Q =_\tau \sim(\sim P \rightrightarrows_2 \sim Q)$$

“Sequence” in BT-Esterel

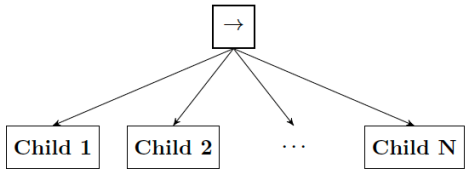


Fig. 1.2: Graphical representation of a Sequence node with N children.

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
```

[Colledanchise Ogren ’20]

BT-Esterel:

btsequence
child$_1$
btseq
child$_2$
btseq
…
btseq
child$_n$
end [btsequence]

Expands to:

trap _btsucc **in**
child$_1$
end;
trap _btsucc **in**
child$_2$
end;
…
child$_n$

Observations:

- FAILURE and RUNNING exceptions are passed to parent
- For $i < n$, SUCCESS of child$_i$ passes control to child$_{i+1}$
- SUCCESS of child$_n$ passes control to parent, with SUCCESS

- **Invariant:** each child$_i$ terminates tick by throwing exception $e \in$ { _btsucc, _btfail, _btrun }, which encodes SUCCESS / FAILURE / RUNNING
- I.e., each child is instantaneous – only surface, no depth! No pause stmt!
- This implies “reactiveness” in the BT-sense

“Fallback” in BT-Esterel

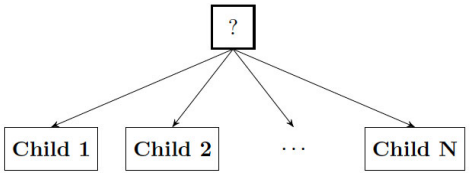


Fig. 1.3: Graphical representation of a Fallback node with *N* children.

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
```

[Colledanchise Ogren ’20]

BT-Esterel:

btfallback
child$_1$
btfb
child$_2$
btfb
…
btfb
child$_n$
end [btfallback]

Expands to:

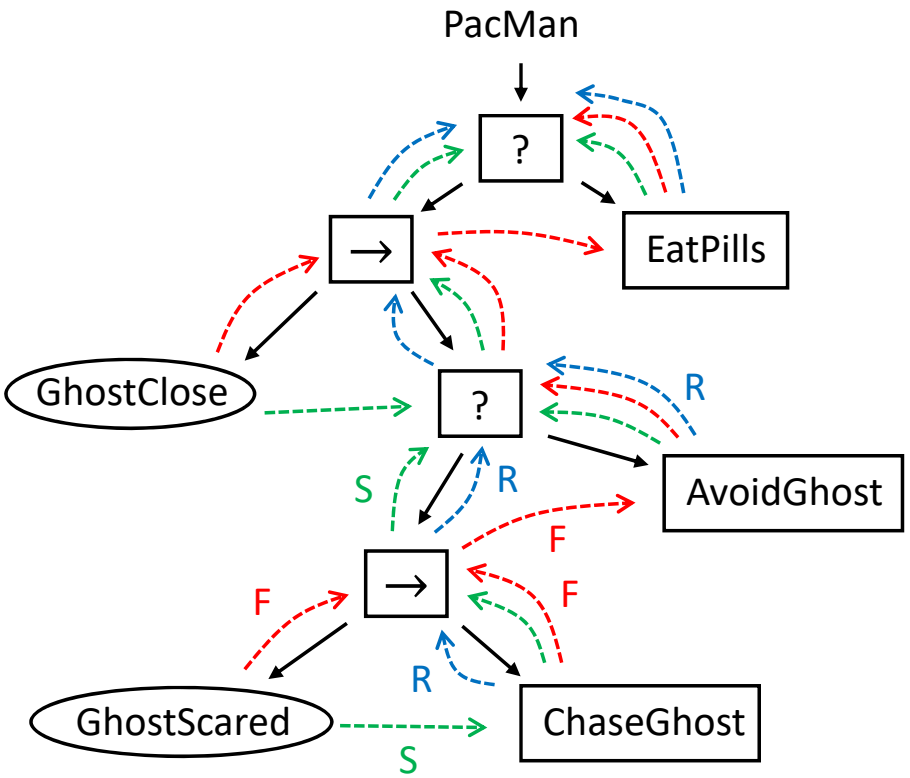
trap _btfail **in**
child$_1$
end;
trap _btfail **in**
child$_2$
end;
…
child$_n$

Observations:

- SUCCESS and RUNNING exceptions are passed to parent
- For $i < n$, FAILURE of child$_i$ passes control to child$_{i+1}$
- FAILURE of child$_n$ passes control to parent, with FAILURE

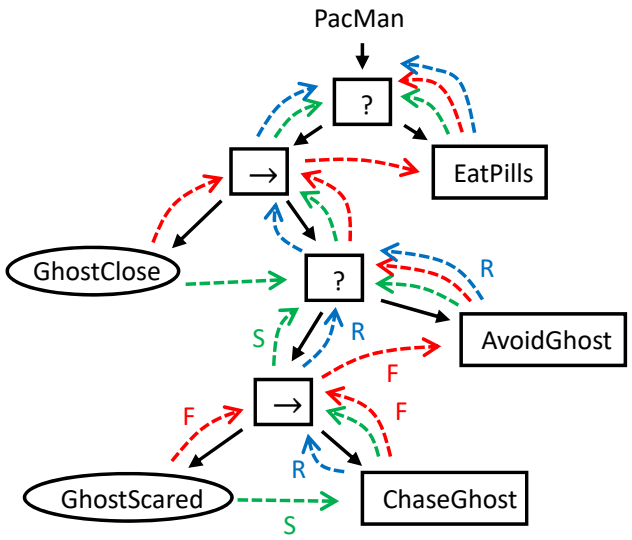
PacMan in “BT-Esterel”

SUCCESS (S)
FAILURE (F)
RUNNING (R)




```
1  module PacMan
2  input GhostClose,
         GhostScared;
3  output ChaseGhost,
         AvoidGhost, EatPills;
4
5  behaviortree
6   btfallback
7    btsequence
8     present GhostClose
9      then exit _btsucc
10     else exit _btfail
11    end;
12   btseq
13    btfallback
14     btsequence
15      present GhostScared
16       then exit _btsucc
17       else exit _btfail
18      end;
19     btseq
20      emit ChaseGhost;
21      exit _btrun
22     end btsequence
23    btfb
24     emit AvoidGhost;
25     exit _btrun
26    end btfallback
27   end btsequence
28  btfb
29   emit EatPills;
30   exit _btrun
31  end btfallback
32 end behaviortree
```

PacMan in BT-Esterel

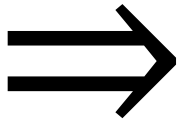


SUCCESS (S)
FAILURE (F)
RUNNING (R)

```
1  module PacMan
2  input GhostClose,
         GhostScared;
3  output ChaseGhost,
         AvoidGhost, EatPills;
4
5  behaviortree
6   btfallback
7    btsequence
8     present GhostClose
9      then exit _btsucc
10     else exit _btfail
11    end;
12   btseq
13    btfallback
14     btsequence
15      present GhostScared
16       then exit _btsucc
17       else exit _btfail
18      end;
19     btseq
20      emit ChaseGhost;
21      exit _btrun
22     end btsequence
23    btfb
24     emit AvoidGhost;
25     exit _btrun
26    end btfallback
27   end btsequence
28  btfb
29   emit EatPills;
30   exit _btrun
31  end btfallback
32 end behaviortree
```

Approach:

Map *all* return values to exceptions!



```
1  module PacMan
2  input GhostClose, GhostScared;
3  output ChaseGhost, AvoidGhost,
         EatPills;
4  output BehaviortreeRunning;
5
6  loop
7   trap _btrun in
8
9    // Application logic
10   trap _btfail in
11    trap _btsucc in
12     present GhostClose
13      then exit _btsucc
14      else exit _btfail
15     end;
16    end trap;
17    trap _btfail in
18     trap _btsucc in
19      present GhostScared
20       then exit _btsucc
21       else exit _btfail
22      end;
23     end trap;
24     emit ChaseGhost;
25     exit _btrun
26    end trap;
27    emit AvoidGhost;
28    exit _btrun
29   end trap;
30   emit AvoidGhost;
31   // End of application logic
32
33  end trap;
34  emit BehaviortreeRunning;
35  pause
36 end loop
```

“Parallel” in BT-Esterel, for *M* = *N*

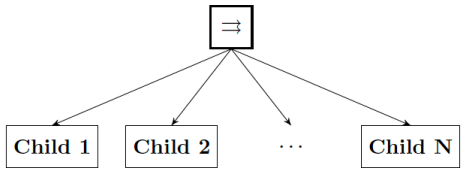


Fig. 1.4: Graphical representation of a Parallel node with N children.

Algorithm 3: Pseudocode of a Parallel node with N children and success threshold M

```
1 for i ← 1 to N do
2     childStatus(i) ← Tick(child(i))
3 if Σ_{i:childStatus(i)=Success} 1 ≥ M then
4     return Success
5 else if Σ_{i:childStatus(i)=Failure} 1 > N − M then
6     return Failure
7 return Running
```

[Colledanchise Ogren ’20]

BT-Esterel:

btparallel
child$_1$
btpar
child$_2$
btpar
...
btpar
child$_n$
end [btparallel]

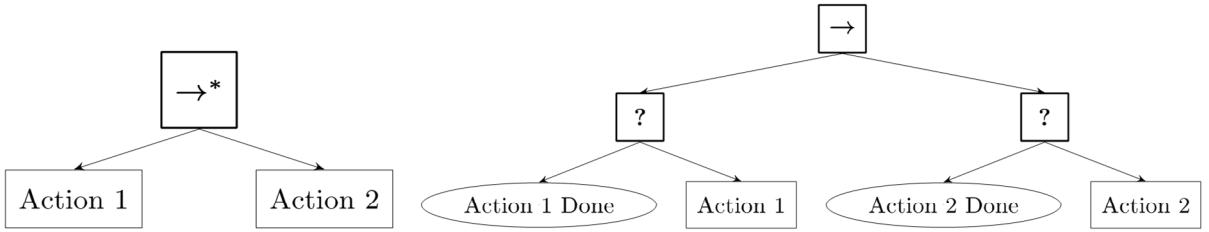
Expands to:

trap _btrun **in**
trap _btsucc **in**
child$_1$
||
...
||
child$_n$
end trap;
exit _btsucc
end trap;
exit _btrun

Observations:

- FAILURE of any child$_i$ passes control to parent, with FAILURE
- If all child$_i$ succeed, return SUCCESS
- Otherwise, return RUNNING

“Nodes with Memory”



(a) Sequence composition with memory.

(b) BT that emulates the execution of the Sequence composition with memory using nodes without memory.

Fig. 1.8: Relation between memory and memory-less BT nodes.

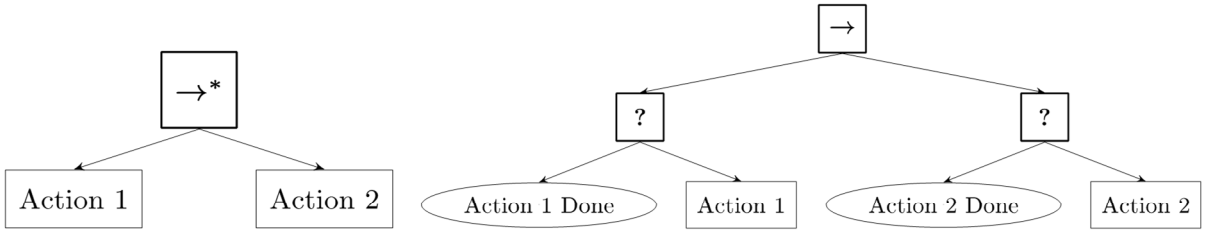
“Nodes with memory [Millington and Funge, 2009] have been introduced to enable the designer to avoid the unwanted re-execution of some nodes. Control flow nodes with memory always remember whether a child has returned Success or Failure, avoiding the re-execution of the child until the whole Sequence or Fallback finishes in either Success or Failure. In this book, nodes with memory are graphically represented with the addition of the symbol “” (e.g. a Sequence node with memory is graphically represented by a box with a “*”). The memory is cleared when the parent node returns either Success or Failure, so that at the next activation all children are considered. Note however that every execution of a control flow node with memory can be obtained with a non-memory BT using some auxiliary conditions as shown in Figure 1.8. Hence nodes with memory can be considered to be syntactic sugar.”*

[Colledanchise Ogren ’20]

“Nodes with Memory” in BT-Esterel

BT-Esterel:

btsequencemem
Action$_1$
btseq
Action$_2$
end [btsequencemem]



(a) Sequence composition with memory.

(b) BT that emulates the execution of the Sequence composition with memory using nodes without memory.

Fig. 1.8: Relation between memory and memory-less BT nodes. [Colledanchise Ogren ’20]

Expands to:

btsequence
btfallback
ActionDone$_1$
btfb
Action$_1$
end
btseq
btfallback
ActionDone$_2$
btfb
Action$_2$
end
end

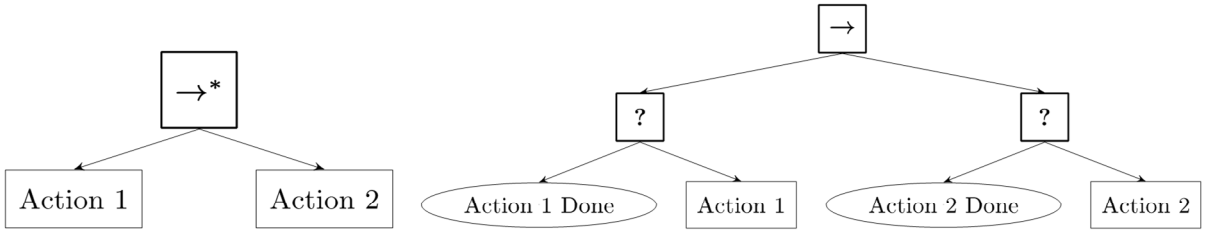
Observations:

- If Action$_1$ fails, then the whole sequence fails.
  Thus in that case we do want to execute Action$_1$ in the next tick again.
  We only want to memorize a success status of Action$_1$, not a failure status.
  Conversely, for fallback, we only want to memorize a failure status.
- According to [Colledanchise Ogren ’20], if Action$_2$ returns success or failure, the memory (including ActionDone$_2$) is cleared. Thus, what sense does ActionDone$_2$ make?
  It appears that this memory is only useful for the siblings **before** the last child of a sequence.

“Nodes with Memory” in BT-Esterel

BT-Esterel:

btsequencemem
Action$_1$
btseq
Action$_2$
end [btsequencemem]



(a) Sequence composition with memory.

(b) BT that emulates the execution of the Sequence composition with memory using nodes without memory.

Fig. 1.8: Relation between memory and memory-less BT nodes. [Colledanchise Ogren ’20]

Expands to:

btsequence
btfallback
ActionDone$_1$
btfb
Action$_1$
end
btseq
~~**btfallback**~~
~~ActionDone$_2$~~
~~**btfb**~~
Action$_2$
~~**end**~~
end

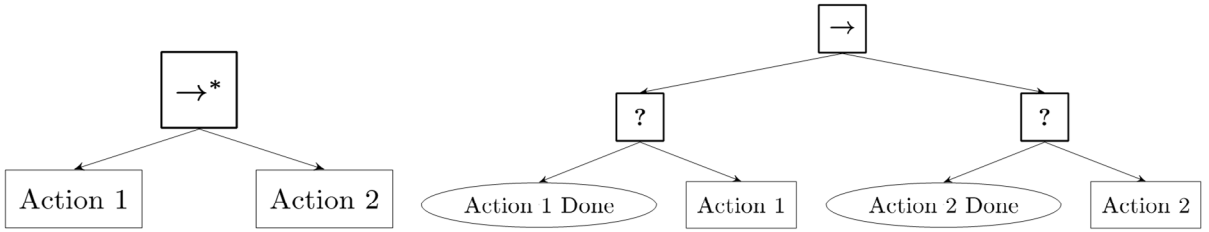
Observations:

- If Action$_1$ fails, then the whole sequence fails.
  Thus in that case we do want to execute Action$_1$ in the next tick again.
  We only want to memorize a success status of Action$_1$, not a failure status.
  Conversely, for fallback, we only want to memorize a failure status.
- According to [Colledanchise Ogren ’20], if Action$_2$ returns success or failure, the memory (including ActionDone$_2$) is cleared. Thus, what sense does ActionDone$_2$ make?
  It appears that this memory is only useful for the siblings **before** the last child of a sequence.

“Nodes with Memory” in BT-Esterel

BT-Esterel:

btsequencemem
Action$_1$
btseq
Action$_2$
end [btsequencemem]



(a) Sequence composition with memory.

(b) BT that emulates the execution of the Sequence composition with memory using nodes without memory.

Fig. 1.8: Relation between memory and memory-less BT nodes.
[Colledanchise Ogren ’20]

Expands to:

btsequence
btfallback
ActionDone$_1$
btfb
Action$_1$
end
btseq
Action$_2$
end

Observations:

- If Action$_1$ fails, then the whole sequence fails.
  Thus in that case we do want to execute Action$_1$ in the next tick again.
  We only want to memorize a success status of Action$_1$, not a failure status.
  Conversely, for fallback, we only want to memorize a failure status.
- According to [Colledanchise Ogren ’20], if Action$_2$ returns success or failure, the memory (including ActionDone$_2$) is cleared. Thus, what sense does ActionDone$_2$ make?
  It appears that this memory is only useful for the siblings **before** the last child of a sequence.

“Nodes with Memory” in BT-Esterel

BT-Esterel:

```
btsequencemem
  Action1
btseq
 Action2
end [btsequencemem]
```

Alternative expansion, with *internal* bookkeeping:

```
btsequence
 static boolean Action1Done = false;
 btfallback
  if (Action1Done)
   then exit _btsucc
   else exit _btfail
  end
 btfb
  trap _btsucc in
   Action1
  end;
  Action1Done = true;
  exit _btsucc
 end btfallback
```

```
btseq
 trap _btfail in
  trap _btsucc in
   Action2;
  end trap;
  Action1Done = false;
  exit _btsucc
 end trap;
 Action1Done = false;
 exit _btfail;
end
```

Wrap-Up – BTs in Esterel

- As in Esterel, individual BT nodes do maintain (internal) state
- However, “reactive” BT does **not** maintain state;
  e.g., sequence always starts at first child
- BT return values resemble Esterel completion codes
- Have presented trap-based mapping of BT constructs to plain Esterel

Thanks!