



# NetLogo CBR Extension

Doug Salt

The James Hutton Institute

November 16, 2022



# Structure

- Basic concepts
- Core commands
- Other commands
- Simple example
- Example with bespoke lambda



Normally a case base consists of a series of cases, each of these cases consist of:

- state
- decision/activity
- outcome

The state can be anything such as the bank balance of the agent. The decision/activity might be to install central heating. The outcome might be straight forwardly yes or no. It might be probability, or it might some arbitrary decision/activity metric for use elsewhere.



# Basic concepts (continued)

A state and decision/activity are presented to the case base. The case base is searched for the closest match (if there is one) and the outcome of that match is given.

A NetLogo case consists of a state in any of the standard Netlogo variables, such as list, number, string, etc. This is strictly defined by the `cbr:lambda` which is the comparator program used to determine the "distance" the three cases:

- case A
- case B
- referent case R

are relative to each other.



# Basic concepts (continued)

That is, if:

- the case A is 'closer' to the referent case R than the case B using `cbr:lamda` to the referent case R then `cbr:lt` is returned
- the case B is 'closer' to the referent case R than the case A using `cbr:lamda` to the referent case R then `cbr:gt` is returned
- the case B is 'same distance' to the referent case R than the case A using `cbr:lamda` to the referent case R then `cbr:eq` is returned
- the case B is incomparable to the referent case, then `cbr:lamda` to the referent case R then `cbr:incmp` is returned



# Basic concepts (continued)

Now when we need to decide whether a case matches one in the case base, we just bubble through the entire case base until we get the closest match or matches. The comparison method is always the same, i.e. the comparator program, denoted `lambda` herein, iff this routine when presented with three cases, can tell you which is closest to the referent case or whether they are comparable at all.

A default comparator program is provided, but this operates purely on state, does not consider the decision or outcome. For more information then please consult the NetLogo CBR documentation which may be found here.

<https://github.com/DougSalt/cbr>



# Basic concepts (continued)

[fragile]

The lambda can be specified in the code and must have the following parameters:

- case-base
- yes-case
- no-case
- reference-case

So the code for a bespoke comparator might look like this in NetLogo:

```
to-report comparator-pattern [ some-case-base yes-case no-case reference-case]
; ...
report cbr:lt
; ...
report cbr:gt
; ...
report cbr:eq
; ...
report cbr:incmp
end
```



# Core commands

- **cbr:new** - creates a new case base.
- **cbr:add** - adds a case to a case base
- **cbr:match** - returns the closes match
- **cbr:matches** - returns more than one match if it exists.
- **cbr:outcome** - queries a case for its outcome
- **cbr:decision** - queries a case for its decision
- **cbr:lambda** - set the default compartor program





# Other commands

- `cbr:combine` - combines two case bases.
- `cbr:all` - returns all the cases as a list.
- `cbr:state` - gets the state of a particular case.
- `cbr:remove` - removes a particular case.
- `cbr:set-time` - sets the time at which a particular case base was created. This is done automatically at insertion into the case base. This commands just allows a degree of additional flexibility.
- `cbr:get-time` - queries the time stamp for a particular case.
- `cbr:set-earliest` - sets the tick before which all case bases will be "forgotten".



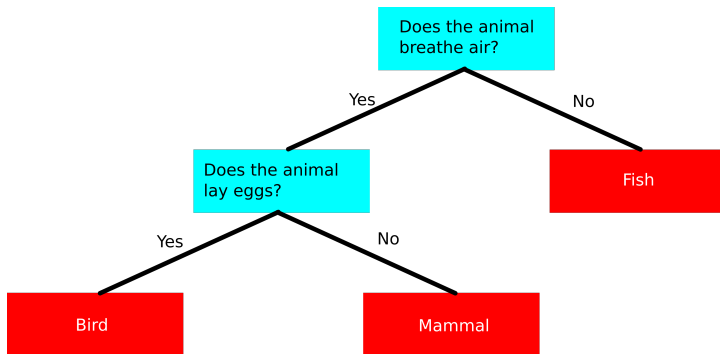
# Other commands (continued)

- `cbr:get-earliest` - allows the querying of the former.
- `cbr:forget` - "forgets" all cases which are too old.
- `cbr:set-rank` - sets the rank in the event of a tie breaker. the former.
- `cbr:get-rank` - allows the querying of the former.
- `cbr:get-size` - reports on the maximum size of the casebase.
- `cbr:set-size` - sets the maximum size of the casebase.
- `cbr:resize` - explicitly tell the case base to resize based on the set-size parameter.



# Meaningless example

We are going to implement this simple decision tree (well kinda)



# Meaningless example (continued)

`cbr:new` to set a new case base, so:

```
set simple-case-base cbr:new
```

To add a new case then:

```
set some-case cbr:add simple-case-base ["lays eggs" "breathes  
air"] "bird" .01
```

where the first field is the case base object, the second is the state  
["lays eggs" "breathes air"]; the third is the decision,  
colorredbird, and the last is the outcome, which in this case is a  
probability of 0.1.



# Meaningless example (continued)

Add repeated multiple cases and then query using `cbr:match` or `cbr:matches`, thus:

```
let some-creature [ "lays eggs" "breathes air" ]  
let result cbr:match simple-case-base some-creature "bird"
```

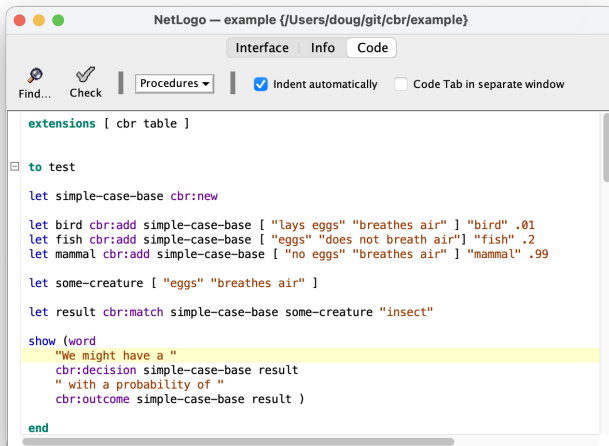
So this constructs a "reference" case base to match, consisting of the statesome-creature, and the decision "bird".

This is using the standard, in-build comparator. For more details on this please see the documentation in the github repository <https://github.com/DougSalt/cbr>.



# Meaningless example (continued)

So this here we another exmaple the code looks like eventually:

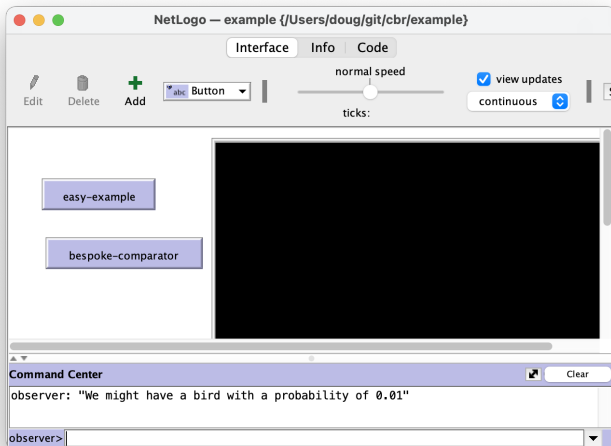


```
NetLogo — example {/Users/doug/git/cbr/example}  
  
Interface  Info  Code  
  
Find...  Check  Procedures ▾  Indent automatically  Code Tab in separate window  
  
extensions [ cbr table ]  
  
to test  
  
  let simple-case-base cbr:new  
  
  let bird cbr:add simple-case-base [ "lays eggs" "breathes air" ] "bird" .01  
  let fish cbr:add simple-case-base [ "eggs" "does not breath air" ] "fish" .2  
  let mammal cbr:add simple-case-base [ "no eggs" "breathes air" ] "mammal" .99  
  
  let some-creature [ "eggs" "breathes air" ]  
  
  let result cbr:match simple-case-base some-creature "insect"  
  
  show (word  
    "We might have a "  
    cbr:decision simple-case-base result  
    " with a probability of "  
    cbr:outcome simple-case-base result )  
  
end
```



# Meaningless example (continued)

And this is the result of clicking the "simple example" button.



# Bespoke lambda

So the default comparator is not that brilliant, so we can implement our own:

```
cbr:lambda simple-case-base some-comparator
```

Where the comparator starts with:

```
to-report [a-case-base src-case obj-case ref-case]
```

Where this routine must return `cbr:lt`, `cbr:gt`, `cbr:eq` or `cbr:incmp`. And that is it.

The only small problem being that there is a bug in the comparator by the looks of things which needs fixing.





# Bespoke lambda (continued)

And this is the result of clicking the "bespoke comparator" button.

```
Command Center Clear  
observer: "So we have some creature with the following attributes"  
observer: "[lays eggs breathes air flies stings]\n"  
observer: "Is our mystery creature a fish?"  
observer: true  
observer: {{cbr:case <[lays eggs, breathes air, flies, stings] | java.lang.String(insect?) | true | -1.0>}}  
observer: "Now let's try again, with our own comparator."  
observer: "Adding our version of the comparator..."  
observer: "Is our mystery creature a fish?"  
observer: {{cbr:case <[lays eggs, breathes air, flies, stings] | java.lang.String(fish?) | false | -1.0>}}  
observer: false
```



# Bespoke lambda (continued)

So how is this done? We set up the case base.

```
let simple-case-base cbr:new

let bird cbr:add simple-case-base [ "lays eggs" "breathes air" "flies" "no sting" ] "bird?" true
let insect cbr:add simple-case-base [ "lays eggs" "breathes air" "flies" "stings" ] "insect?" true
let reptile cbr:add simple-case-base [ "lays eggs" "breathes air" "does not fly" "stings" ] "reptile?" true
let mammal cbr:add simple-case-base [ "no eggs" "breathes air" "no flying" "no sting" ] "mammal?" true
let fish cbr:add simple-case-base [ "lays eggs" "does not breath air" "does not fly" "stings" ] "fish?" true
set fish cbr:add simple-case-base [ "lays eggs" "does not breath air" "does not fly" "no sting" ] "fish?" true
set fish cbr:add simple-case-base [ "lays eggs" "breathes air" "flies" "no sting" ] "fish?" false
set fish cbr:add simple-case-base [ "lays eggs" "breathes air" "flies" "stings" ] "fish?" false
set fish cbr:add simple-case-base [ "lays eggs" "breathes air" "does not fly" "stings" ] "fish?" true
set fish cbr:add simple-case-base [ "no eggs" "breathes air" "no flying" "no sting" ] "fish?" true
```



# Bespoke lambda (continued)

We code our comparator:

```
to-report some-comparator [ some-case-base src-case obj-case ref-case ]
  let src-state cbr:state some-case-base src-case
  let src-decision cbr:decision some-case-base src-case
  let src-hits 0
  let obj-state cbr:state some-case-base obj-case
  let obj-decision cbr:decision some-case-base obj-case
  let obj-hits 0
  let ref-state cbr:state some-case-base ref-case
  let ref-decision cbr:decision some-case-base ref-case
  foreach n-values length ref-state [ i -> i ] [ i ->
    if item i ref-state = item i src-state [set src-hits src-hits + 1]
    if item i ref-state = item i obj-state [set obj-hits obj-hits + 1]
  ]
  if src-decision != ref-decision and obj-decision = ref-decision [report cbr:gt]
  if src-decision != ref-decision [report cbr:incmp]
  if obj-decision != ref-decision [report cbr:lt]
  (ifelse
    src-hits > obj-hits [report cbr:lt]
    src-hits < obj-hits [report cbr:gt]
    [report cbr:eq]
  )
end
```



Any questions?

Thank you very much



Scottish Government  
Riaghaltas na h-Alba  
gov.scot