A BFS implementation in Scheme

General purpose BFS implementation:

```
(bfs root-node at-goal? get-children)
```

where:

`(at-goal? node)` returns `#t` if node is the goal

`(get-children node)` returns a list of child *nodes*

The algorithm:

- put root node on a queue Q
- Repeat:
  - if Q is empty, return failure
  - remove first node N from Q
  - if N is the goal, return success
  - add children of N to end of Q

Missionaries & Cannibals problem in Scheme

State: `(boat-side L-mis L-can R-mis R-can)`
`boat-side` = `'left` or `'right`

Node: `(state parent-node)`
parent of root node is `'()`

```
(define mc-start '((left 3 3 0 0) ()))

(define (mc-goal? node)
  (equal? (car node) '(right 0 0 3 3)))
```

Approach to `(mc-children node)`:

- use `(mc-child-states state)`
- convert list of child states to nodes

Approach to `(mc-child-states state)`:

- If boat is on left, compute child states
- If boat is on right
  - switch left and right sides
  - compute child states
  - switch left and right sides
- Enforce constraint (`#Can` ≤ `#Mis`)

Heuristic searches

heuristic: a "rule of thumb," for searching we a heuristic function $h(n)$ that gives an *estimate* of the cost from node n to the goal.

A simple example is *Greedy Search*:

- Put the root node on a queue Q
- Repeat:
  - if Q is empty, return failure
  - remove the node N *with the lowest* $h(\cdot)$ *value* from Q
  - if N is the goal, return success
  - add children of N to Q

Analysis:

- Optimal?
- Complete?
- Time complexity?
- Space complexity?

The A* search

A queue implementation:

- Put the root node on a queue Q
- Repeat:
  - if Q is empty, return failure
  - remove the node N *with the lowest* $f(\cdot) = g(\cdot) + h(\cdot)$ *value* from Q
  - if N is the goal, return success
  - add children of N to Q

where $g(n)$ is the cost from the root node to node n

Important properties:

- if $h(\cdot)$ is *admissible*, A* is optimal
- if $h(\cdot)$ is also *monotonic*, A* is *optimally efficient*

admissibility: A heuristic $h(n)$ is admissible if it *never overestimates* the cost to the goal from node n

monotonicity: A heuristic $h(n)$ is monotonic if for any nodes A and B, $h(B) \geq h(A) + c(A, B)$

A different formulation of the A* algorithm

- Put the start node on a list OPEN
- Create an empty list CLOSED
- Repeat:
  - If OPEN is empty, return failure
  - Select the node N from OPEN with lowest $f(\cdot)$ value
  - Remove N from OPEN and add to CLOSED
  - If N is the goal, return success
  - Find the children C of N
  - For each child $c \in$ C:
    - if c is not on OPEN or CLOSED, add to OPEN
    - if c is on OPEN, update $f(c)$ if necessary
    - if c is on CLOSED and must be updated, remove c from CLOSED and add to OPEN