11A. Wolf-Sheep-Cabbage Problem Pseudocode

CARRY WOLF. SHEEP, AND CABBAGE FROM A TO B WITH ONE PLACE IN THE BOAT

1. CARRY SHEEP TO B
2. GO BACK TO A
3. CARRY WOLF TO B
4. GO BACK A WITH SHEEP
5. CARRY CABBAGE TO B
6. GO BACK A
7. CARRY SHEEP TO B

```
from enum import Enum
from collections import namedtuple
from functools import partial

def breadth_first_search(*, start, is_goal, get_neighbors):
    parent = dict()
    to_visit = [start]
    discovered = set([start])

    while to_visit:
        vertex = to_visit.pop(0)

        if is_goal(vertex):
            path = []
            while vertex is not None:
                path.insert(0, vertex)
                vertex = parent.get(vertex)
            return path

        for neighbor in get_neighbors(vertex):
            if neighbor not in discovered:
                discovered.add(neighbor)
                parent[neighbor] = vertex
                to_visit.append(neighbor)

State = namedtuple("State", ["man", "cabbage", "goat", "wolf"])
Location = Enum("Location", ["A", "B"])
```

```
start_state = State(
    man=Location.A,
    cabbage=Location.A,
    goat=Location.A,
    wolf=Location.A,
)

goal_state = State(
    man=Location.B,
    cabbage=Location.B,
    goat=Location.B,
    wolf=Location.B,
)

def is_valid(state):
    goat_eats_cabbage = (
        state.goat == state.cabbage
        and state.man != state.goat
    )
    wolf_eats_goat = (
        state.wolf == state.goat and state.man != state.wolf
    )
    invalid = goat_eats_cabbage or wolf_eats_goat
    return not invalid
```

```python
def next_states(state):
    if state.man == Location.A:
        other_side = Location.B
    else:
        other_side = Location.A

    move = partial(state._replace, man=other_side)

    candidates = [move()]

    for thing in ["cabbage", "goat", "wolf"]:
        if getattr(state, thing) == state.man:
            candidates.append(move(**{thing: other_side}))

    yield from filter(is_valid, candidates)
```

```python
solution = breadth_first_search(
    start = start_state,
    is_goal = goal_state.__eq__,
    get_neighbors = next_states,
)
```

```python
def describe_solution(path):
    for old, new in zip(path, path[1:]):
        boat = [
            thing
            for thing in ["man", "cabbage", "goat", "wolf"]
            if getattr(old, thing) != getattr(new, thing)
        ]
        print(old.man, "to", new.man, boat)
describe_solution(solution)
```

```
Location.A to Location.B ['man', 'goat']
Location.B to Location.A ['man']
Location.A to Location.B ['man', 'cabbage']
Location.B to Location.A ['man', 'goat']
Location.A to Location.B ['man', 'wolf']
Location.B to Location.A ['man']
Location.A to Location.B ['man', 'goat']
```

11B.

1. Creating the Knowledge Base

initial state

```
[6]  from aima3.logic import FolKB
     from aima3.utils import expr

     kb = FolKB([
           expr('At(Wolf, Left)'),
           expr('At(Sheep, Left)'),
           expr('At(Cabbage, Left)'),
           expr('At(Farmer, Left)'),
           expr('NotAt(Wolf, Right)'),
           expr('NotAt(Sheep, Right)'),
           expr('NotAt(Cabbage, Right)'),
           expr('NotAt(Farmer, Right)')
         ])
```

2. Adding New Information

```
[19] kb.tell(expr('At(Farmer, Left) & At(Sheep, Left) ==> At(Farmer, Right) & At(Shee
     kb.tell(expr('At(Farmer, Right) & At(Sheep, Right) ==> At(Farmer, Left) & At(Shee
```

```
kb.tell(expr('Hates(Sheep, Wolf)'))
kb.tell(expr('Hates(Cabbage, Sheep)'))
kb.tell(expr('Hates(Farmer, Wolf)'))
```

3. Removing Information

```
[9]  kb.retract(expr('Hates(Farmer, Wolf)'))
```

4. Querying the Knowledge Base

```
[16] is_safe = kb.ask(expr('Unsafe'))
     print(is_safe)
```

```
False
```

```
[33] kb.ask(expr('At(Farmer, Right) & At(Sheep, Left) ==> At(Farmer, Left) & At(Sheep,
```

```
False
```

```
kb.ask(expr('Hates(Sheep,Cabbage)'))
```

```
False
```

Making Inferences

The power of a FOL KB lies in its ability to make inferences based on the rules and facts it contains. In this example, the KB can infer that current state is safe because it knows: everybody is at the right side so there is no any unsafe condition.