UNIVERSIDAD NACIONAL DE COLOMBIA

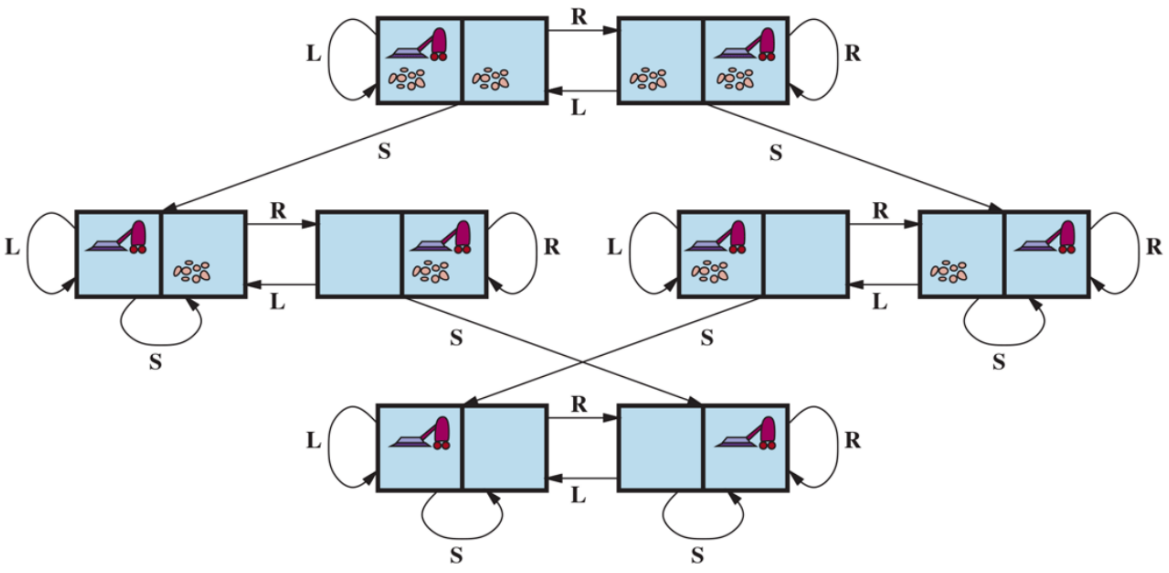
MINI PROYECTO 1

13 / 02 / 2023
Asignatura: Inteligencia artificial

Estudiantes	**Correo**
Andrés Camilo Peralta Fragozo	aperaltaf@unal.edu.co
Daniel Felipe Zuniga Hurtado	dfzunigah@unal.edu.co
Sergio Andres Guzman Carrascal	seguzmanc@unal.edu.co

PROGRAMA DE AGENTE DE SOLUCIÓN DE PROBLEMAS SENCILLOS

Figure 3.2



Tenemos un agente capaz de realizar dos acciones, *aspirar* si detecta mugre y *moverse* si ya está limpio.

```
class SimpleProblemSolvingAgentProgram:
    """
    [Figure 3.1]
```

```
    Abstract framework for a problem-solving agent.
    """

    def __init__(self, initial_state=None):
        """State is an abstract representation of the state
        of the world, and seq is the list of actions required
        to get to a particular state from the initial
state(root)."""
        self.state = initial_state
        self.seq = []

    def __call__(self, percept):
        """[Figure 3.1] Formulate a goal and problem, then
        search for a sequence of actions to solve it."""
        self.state = self.update_state(self.state, percept)
        if not self.seq:
            goal = self.formulate_goal(self.state)
            problem = self.formulate_problem(self.state, goal)
            self.seq = self.search(problem)
            if not self.seq:
                return None
        return self.seq
        # return self.seq.pop(0)


    def update_state(self, state, percept):
        raise NotImplementedError

    def formulate_goal(self, state):
        raise NotImplementedError

    def formulate_problem(self, state, goal):
        raise NotImplementedError

    def search(self, problem):
        raise NotImplementedError
```

Este es un programa de un agente de resolución de problemas simple en Python. Se utiliza una estructura abstracta para representar un agente que resuelve problemas y se define una clase llamada *SimpleProblemSolvingAgentProgram* que implementa esta estructura.

La clase tiene cuatro métodos que deben ser implementados para que el agente pueda funcionar correctamente:

- *update_state:* Este método actualiza el estado del agente basado en la percepción recibida.

- *formulate_goal:* Este método permite al agente formular una meta basada en su estado actual.

- *formulate_problem*: Este método permite al agente formular un problema a partir de su estado actual y su meta.

- *search:* Este método permite al agente buscar una secuencia de acciones para resolver el problema formulado.

Estos métodos se utilizan en el método *__call__* para formular un objetivo, formular un problema y buscar una secuencia de acciones para resolver el problema.

Nótese que en la declaración del método de llamada *__call__* el comentario `# return self.seq.pop(0)` se deja a modo de comparación con instrucción *return* que ya está. Fue necesario omitir el meteoro *.pop(0)* para que se permita el retorno completo de la lista que contiene la secuencia de acciones que nuestro agente realizará desde el respectivo estado que se le pasa como parámetro.

Si necesitas implementar un agente que resuelva problemas, debes crear una clase que herede de SimpleProblemSolvingAgentProgram e implemente estos cuatro métodos.

Considerando lo anterior se define la subclase *vacuumAgent* que define al agente que interactúa con el entorno y hereda los métodos de la clase base SimpleProblemSolvingAgentProgram.

```
class vacuumAgent(SimpleProblemSolvingAgentProgram):
        def update_state(self, state, percept):
            return percept

        def formulate_goal(self, state):
            goal = [state7, state8]
            return goal

        def formulate_problem(self, state, goal):
            problem = state
            return problem

        def search(self, problem):
            if problem == state1:
                seq = ["Suck", "Right", "Suck"]
            elif problem == state2:
                seq = ["Suck", "Left", "Suck"]
            elif problem == state3:
                seq = ["Right", "Suck"]
            elif problem == state4:
                seq = ["Suck"]
            elif problem == state5:
                seq = ["Suck"]
            elif problem == state6:
                seq = ["Left", "Suck"]
            return seq
```

Ahora cada método de la clase *vacuumAgent* es una **sobreescritura** de un método de la clase padre que es necesario para que el programa funcione correctamente. Las actualizaciones de tales métodos se describe como sigue:

- *update_state:* este método actualiza el estado actual del agente con los percepciones recibidas del entorno. En este caso, simplemente devuelve el percepción sin realizar ninguna operación adicional.

- *formulate_goal*: este método forma una meta a partir del estado actual del agente. En este caso, la meta es una lista con dos estados posibles: state7 y state8.

- *formulate_problem:* este método forma un problema a partir del estado actual y la meta. En este caso, el problema es simplemente el estado actual del agente.

- *search*: este método implementa la búsqueda de la solución al problema. Aquí se hacen comparaciones con diferentes estados y se devuelve una secuencia de acciones que se deben ejecutar para llegar a un estado final objetivo. Cada comparación y secuencia es diferente dependiendo del estado actual del agente.

Un estado se define de la siguiente manera:

```
state = [(0,0) , [(0,0), ”<Dirty, Clean>”] , [(1,0), [”<Dirty, Clean>”]]]
```

Como una 3-tupla en donde:

- El primer elemento define la posición actual de la aspiradora.
- La segunda y la tercera tupla (que a su vez, son duplas/2-tuplas) indican el estado de cada una de las dos habitaciones y su estado.
- El estado de cada habitación viene dado por las palabras “Dirty” o “Clean”, señalando si el cuarto está sucio o limpio, respectivamente.
- La posición *(0, 0)* hace referencia a la habitación de la izquierda . La posición *(1, 0)* hace referencia a la habitación de la derecha.

Se define, a manera de ejemplo, un estado de la siguiente forma:

```
state1 = [(1,0) , [(0,0), ”Dirty”] , [(1,0), [”Clean”]]]
```

Donde la aspiradora se encuentra en la habitación de la derecha, el cuarto de la izquierda *(0, 0)* se encuentra sucio y el cuarto a la derecha *(1, 0)* se encuentra limpio. La secuencia entregada al utilizar el algoritmo sobre este estado, sería:

```
[“Left”, “Suck”]
```

Siendo que la aspiradora se encuentra en una habitación limpia, es decir, la de la derecha, procederá a moverse a la izquierda y una vez ahí, evaluará el estado de dicha habitación; al encontrarla sucia, procederá a succionar. Finalmente evaluando el estado de ambas habitaciones y siendo esté, el que corresponde al estado objetivo, no realizará más acciones, pues ambas habitaciones están limpias.

Finalmente para hacer que el código se ejecute se crea una instancia de la case *vacuumAgent* y se llama a la instancia con un estado inicial, por ejemplo:

```
agent = vacuumAgent(state1)

print(agent(state1))
```