FCFM
FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

UANL
UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

Universidad Autónoma de Nuevo León

Facultad de Ciencias Físico Matemáticas

Inteligencia Artificial

PIA

Docente: José Anastacio Hernandez Saldaña

Grupo: 032

Carlos Manuel Valerio Ríos 2049129

21 de noviembre de 2025, San Nicolas de los Garza, N.L.

Torres de Hanoi

Introducción

La planeación automatizada es una rama de la inteligencia artificial que se concentra en la realización de estrategias o secuencias de acción, típicamente para la ejecución de agentes inteligentes. La planeación automatizada necesita un contexto o *mundo* en el cual se definen las acciones posibles a realizar por un agente. A este mundo le llamamos dominio en el contexto del lenguaje *pddl*.

Pddl o mejor conocido como *Planning Domain Definition Language,* es un lenguaje que nos permite definir un dominio, en el cual debemos indicar, los actores dentro del mundo o mejor conocidos como objetos, predicados que son estados que aplican específicamente a ciertos objetos, también podemos referirnos a ellos como propiedades del objeto, y por último, acciones, estas son las transiciones de un estado a otro que puede realizar nuestro agente, estas acciones cuentan con *pre-condiciones* las cuales se deben cumplir para poder realizar la acción y dicha acción tendrá un efecto en el mundo.

Por último, una pieza faltante dentro del mundo de pddl, es el problema mismo, este es el cual el *planeador* intentará resolver utilizando los predicados y acciones definidos en el dominio. El problema debe contar con un estado inicial, es decir, debemos definir las propiedades de los objetos antes de iniciar la planificación. También es necesario un objetivo, o el estado deseado; de esta forma el planeador puede utilizar el estado inicial y el objetivo para desarrollar un plan de acción para llegar al estado final.

Definición del Problema

El problema de las torres de Hanoi es un problema muy conocido dentro del mundo de la computación y la matemática, debido a esto, se han encontrado algoritmos que resuelven dicho problema de forma satisfactoria; no es un reto resolver las torres de Hanoi. Por esta razón, he decidido modificar el problema para añadir una capa de complejidad más, esta modificación consta de añadir colores a los discos, dichos discos tienen prohibido moverse hacia discos de color diferente. El objetivo de este problema es ordenar todos los discos por tamaños y juntarlos con discos de su mismo color.

Dominio

En el dominio definiremos los predicados o propiedades, acciones y tipos de objetos. Cabe recalcar que para este problema especifico se utilizaron dos extensiones del lenguaje pddl que brindan funcionalidad extra, estas son, *strips y typing,* las cuales definen un método de planificación y compatibilidad con tipos de objetos respectivamente.

Predicados

En la sección de predicados definimos 4 relaciones diferentes entre objetos.

- ON, parámetros: x, y

Esta relación indica si el disco “x” se encuentra encima de el objeto “y”, el objeto y puede ser una torre o un disco.

- **Clear, parámetros: x**

Clear nos permite saber si el objeto “x” (ya sea disco o torre) se encuentra libre para moverse, es decir, no tiene un disco posicionado encima del mismo.

- **Same-color, parámetros: x, y**

Define una relación entre dos discos, en la cual se indica si ambos discos tienen el mismo color o no.

- **Smaller, parámetros: d, e**

Define la relación de tamaños entre dos discos, de esta forma podemos saber si el disco d es más pequeño que el disco e.

Acciones

En la sección de acciones podemos definir las transiciones de estados cuando ciertas condiciones se cumplen.

- **Move-to-peg**

Esta acción recibe como parámetros, d, from y peg. D es de tipo disco, from puede ser un disco o una torre y peg es una torre.
Para poder realizar esta acción, se deben cumplir las siguientes condiciones:
- d debe estar sobre from.
- d debe estar libre para moverse, es decir no debe tener discos encima.
- Peg debe estar libre, para que d pueda moverse hacia el.

Esta acción tiene los siguientes efectos:
- d ya no se encontrará en from.
- d ahora estará encima de peg.
- Peg ya no estará libre
- From estará libre.

Debido a que en esta acción el disco se mueve hacia una torre vacía, no es necesario confirmar el color.

- **Move-onto-disc**

Esta acción recibe los objetos, d, from y top. D es un disco, from puede ser un disco o una torre y top es un disco.

Se deben cumplir las siguientes pre-condiciones:

- d debe estar en from.
- d debe estar libre para moverse.
- Top debe estar libre para que d se mueva hacia él.
- d debe ser más chico que top.
- d y top deben tener el mismo color.

Después de realizar esta acción, los siguientes eventos tomarán efecto:

- d ya no se encontrará en from.
- d ahora se encontrará en top.
- Top ya no estará libre.
- From estará libre.

En esta acción se debe confirmar el tamaño y el color de los discos pues en este caso el movimiento si es de disco a disco.

Para ver el dominio completo observe el anexo A.

Problema

Para definir el problema debemos indicar cuáles serán los objetos que tomarán parte de nuestro mundo y cuáles son sus estados iniciales. En el ejemplo adjunto a este documento (Anexo B) podrán notar que se definen, 4 discos y 4 torres.

Después de haber definido los objetos se define el estado inicial, donde se indica la relación de tamaños entre los discos, cuales son las torres que se encuentran libres, el arreglo de los discos y la relación de colores entre discos.

Debido a que no está permitido que dos discos de color diferente estén en contacto, el objetivo se define de forma que todos los discos de un color se encuentran juntos en una misma torre. Por ejemplo, en el problema podemos ver que el objetivo es que d1 esté sobre d3 y que d3 se encuentre sobre PegB, organizando así los dos discos del mismo color, por otro lado, se establece que d2 debe estar sobre d4 y d4 debe encontrarse en pegC.

El planeador encontrará una secuencia de acciones para llegar al estado deseado si este problema se puede resolver.

Plan Generado

Found Plan (output)

(move-to-peg d1 d2 pegd)

(move-to-peg d2 d3 pegc)

(move-to-peg d3 d4 pegb)

(move-onto-disc d2 pegc d4)

(move-onto-disc d1 pegd d3)

(move-to-peg d2 d4 pegd)

(move-to-peg d4 pega pegc)

(move-onto-disc d2 pegd d4)

```
(:action move-to-peg
  :parameters (d1 d2 pegd)
  :precondition
    (and
      (on d1 d2)
      (clear d1)
      (clear pegd)
    )
  :effect
    (and
      (not
        (on d1 d2)
      )
      (on d1 pegd)
      (not
        (clear pegd)
      )
      (clear d2)
    )
)
```

Anexo A

Dominio

```
;; Color-Restricted Towers of Hanoi - domain
(define (domain color-hanoi)
  (:requirements :strips :typing)
  (:types
    support ;; parent type of disc and peg
    disc - support
    peg - support
    color
  )
  (:predicates
    ;; Relations
    (on ?d - disc ?x - support)          ;; ?d is on top of ?x  (where ?x is either a disc or a peg)
    (clear ?x - support)                 ;; nothing is on top of ?x (disc or peg)
    (same-color ?x - disc ?y - disc)     ;; disc x and disc y have the same color
    (smaller ?d - disc ?e - disc)        ;; disc ?d is smaller than disc ?e
  )

  (:action move-to-peg
    :parameters (?d - disc ?from - support ?peg - peg)
    :precondition (and
      (on ?d ?from)
      (clear ?d)
      (clear ?peg)
    )
    :effect (and
      (not (on ?d ?from))
      (on ?d ?peg)
      (not (clear ?peg))
      (clear ?from)
    )
  )

  (:action move-onto-disc
    :parameters (?d - disc ?from - support ?top - disc)
    :precondition (and
      (on ?d ?from)
      (clear ?d)
      (clear ?top)
      (smaller ?d ?top)
      ;; color match
      (same-color ?d ?top)
    )
    :effect (and
      (not (on ?d ?from))
      (on ?d ?top)
      (not (clear ?top))
      (clear ?from)
    )
  )
```

)

Anexo B

Problema 4 discos, 4 torres.

```
;; Problem file for Color-Restricted Towers of Hanoi
(define (problem color-hanoi-4)
  (:domain color-hanoi)

  (:objects
    ;; pegs
    pegA pegB pegC pegD - peg

    ;; discs (d1 = smallest, d2, d3, d4 = largest)
    d1 d2 d3 d4 - disc
  )

  ;; Initial state:
  ;; Stack on pegA (bottom -> top): d4 d3 d2 d1
  ;; Colors alternate: d4 red, d3 blue, d2 red, d1 blue
  (:init
    ;; positions
    (on d1 d2)
    (on d2 d3)
    (on d3 d4)
    (on d4 pegA)

    ;; which things are clear: top disc and empty pegs
    (clear d1)
    (clear pegB)
    (clear pegC)
    (clear pegD)
    ;; pegA is not listed clear because it has discs on it

    ;; colors
    (same-color d1 d3)
    (same-color d2 d4)
    (same-color d3 d1)
    (same-color d4 d2)
    (not (same-color d1 d2))
    (not (same-color d1 d4))
    (not (same-color d2 d1))
    (not (same-color d2 d3))
    (not (same-color d3 d2))
    (not (same-color d3 d4))
    (not (same-color d4 d1))
    (not (same-color d4 d3))

    ;; size relation: smaller X Y when X is strictly smaller than Y.
    ;; For 4 discs: d1 < d2 < d3 < d4
    (smaller d1 d2)
    (smaller d1 d3)
    (smaller d1 d4)
```

```
    (smaller d2 d3)
    (smaller d2 d4)
    (smaller d3 d4)
  )

  ;; Goal: all discs stacked on pegC in the same order (d4 bottom ... d1 top)
  (:goal (and
    (on d1 d3)
    (on d3 pegB)
    (on d2 d4)
    (on d4 pegC)
  ))
)
```