Computabilidad
Proyecto

Carlos Alexis Barrios Bello
zS23000636@estudiantes.uv.mx

Maestría en Inteligencia Artificial

IIIA Instituto de Investigaciones en Inteligencia Artificial
Universidad Veracruzana
Campus Sur, Calle Paseo Lote II, Sección 2a, No 112
Nuevo Xalapa, Xalapa, Ver., México 91097

27 de junio de 2024

1. Introducción

La máquina de Turing es un elemento de "matemática abstracta" y no un objeto físico. El concepto fue introducido por Alan Turing para tratar un problema muy general, conocido como el Entscheidungsproblem, parcialmente planteado por el gran matemático alemán David Hilbert. La pregunta del problema era: ¿existe algún procedimiento mecánico general que pueda, en principio, resolver uno s otro todos los problemas de las matemáticas, que pertenezcan a alguna clase bien definida? Penrose (1999)
Para contestar esa pregunta, Turing se imaginó un dispositivo que lleve un procedimiento mecánico, por su cabeza ocurrió lo siguiente Penrose (1999): "Queremos que nuestro dispositivo tenga un conjunto discreto de posibles estados diferentes, en número finito. Así, aunque tenga un número finito de estados internos, nuestro dispositivo debe poder manejar un input de cualquier tamaño. Además, el dispositivo dispondrá de un espacio ilimitado de almacenamiento externo para sus cálculos, y podrá también producir un output de tamaño ilimitado. "

Turing representaba los datos externos y el espacio de almacenamiento como una cinta sobre la que se hacen marcas. Esta cinta sería utilizada por

el dispositivo y leída cuando fuera necesario; el dispositivo podría mover la cinta hacia adelante o hacia atrás. También podría hacer nuevas marcas en los lugares de la cinta donde fuera necesario y podría borrar las viejas, permitiendo actuar a la misma cinta como almacenamiento externo y como input.
La cinta seguirá pasando por el dispositivo hacia adelante y hacia atrás mientras sea necesario hacer nuevos cálculos. Cuando el cálculo haya terminado, el dispositivo se detendrá y la respuesta aparecerá en la parte de la cinta que queda a un lado del dispositivo. Supongamos, para ser concretos, que la respuesta aparece siempre a la izquierda, mientras que los datos numéricos del input, junto con los datos del problema a resolver, siempre quedan a la derecha. Penrose (1999).

2. Código

2.1. Para las instrucciones

Como primera parte, se definieron todas las máquinas conocidas en un diccionario, esto para tener un control y revisar si se están codificando bien. Lo único que se usará aquí, será el número decimal.

```
# Diccionario de máquinas de Turing
turing_machines = {
    "T_0": {
        "instructions": "0 0 R, 0 0 R",
        "binary": "1100110",
        "binary_simplified": "0",
        "decimal_number": "0"
    },
    "T_1": {
        "instructions": "0 0 R, 0 0 L",
        "binary": "1101110",
        "binary_simplified": "1",
        "decimal_number": "1"
    },
    "T_2": {
        "instructions": "0 0 R, 0 1 R",
        "binary": "11010110",
        "binary_simplified": "10",
        "decimal_number": "2"
```

```
},
"T_3": {
    "instructions": "0 0 R, 0 0 STOP",
    "binary": "11011110",
    "binary_simplified": "11",
    "decimal_number": "3"
},
"T_4": {
    "instructions": "0 0 R, 1 0 D",
    "binary": "110100110",
    "binary_simplified": "100",
    "decimal_number": "4"
},
"T_5": {
    "instructions": "0 0 R, 0 1 L",
    "binary": "110101110",
    "binary_simplified": "101",
    "decimal_number": "5"
},
"T_6": {
    "instructions": "0 0 R, 0 0 R, 0 0 R",
    "binary": "110110110",
    "binary_simplified": "110",
    "decimal_number": "6"
},
"T_7": {
    "instructions": "0 0 R, 1 1 R",
    "binary": "110111110",
    "binary_simplified": "111",
    "decimal_number": "7"
},
"T_8": {
    "instructions": "0 0 R,10 0 R",
    "binary": "1101000110",
    "binary_simplified": "1000",
    "decimal_number": "8"
},
"T_9": {
    "instructions": "0 0 R, 1 0 L",
    "binary": "1101001110",
```

```
    "binary_simplified": "1001",
    "decimal_number": "9"
},
"T_10": {
    "instructions": "0 0 R, 1 1 R",
    "binary": "1101010110",
    "binary_simplified": "1010",
    "decimal_number": "10"
},
"T_11": {
    "instructions": "0 0 R, 0 1 STOP",
    "binary": "1101011110",
    "binary_simplified": "1011",
    "decimal_number": "11"
},
"T_12": {
    "instructions": "0 0 R, 0 0 R,  0 0 R",
    "binary": "1101100110",
    "binary_simplified": "1100",
    "decimal_number": "12"
},
"unPlus1" : {
    "instructions": "0 0 R, 1 1 R, 0 1 STOP, 1 1 R",
    "binary": "11010101101011101010110",
    "binary_simplified": "101011010111101010",
    "decimal_number": "177642"
},
"unTimes2" : {
    "instructions": "0 0 R, 1 0 R, 10 1 L, 1 1 R, 11 0 R,
    100 0 R, 0 1 STOP, 11 1 R, 101 1 L, 100 1 R, 10 1 L, 101 1 L",
    "binary": "1101001101001011101010110101001101
    0001101011110101010110100101011101000101101001011101001010101",
    "binary_simplified": "1001101001011101010110101001101
    0001101011110101010110100101011101000101101001011101001010101",
    "decimal_number": "14929234209198720269175476 69"
},
"xnPlus1" : {
    "instructions": "0 0 R, 1 1 L, 0 0 R, 10 1 R, 11 0 L, 10 1 R,
    0 1 STOP, 100 0 L, 101 1 L, 100 1 L, 110 0 R, 10 1 R, 111 1 R,
    11 1 R, 111 0 R",
```

```
        "binary": "11010101101101001011010100111
        01001011010111101000011101001010111010001011101
        01000110100101101101010101101010101101010100110",
        "binary_simplified": "10101101101001011010011101001
        01101011110100001110100101011101000101110101000110100
        10110110101010101101010101101010100",
        "decimal_number": "450813704461563958982113775643437908"
    },
    "xnTimes2" : {
        "instructions": "0 0 R, 1 0 R, 0 1 R, 10 0 R, 11 1 R,
        0 1 STOP, 0 0 R ",
        "binary": "11010011010110100011010101011011011011110",
        "binary_simplified": "10011010110100011010101011011011011",
        "decimal_number": "10389728107"
    }
}
```

También se hace un diccionario de los movimientos permitidos:

```
movimientos = { "0": "0", "10": "1", "R" :"110",
    "L" : "1110", "STOP": "11110"}
```

Se prosigue a convertir un número decimal al binario, agregarle una “R” al inicio y final, también se rellenan los números binarios completos con los movimientos:

```
def decimal_to_binary_simplified(decimal_number):
    """ Convierte un número decimal en una cadena binaria simplificada. """
    return bin(int(decimal_number))[2:]
    # Se usa [2:] para eliminar el prefijo '0b' de la cadena binaria.


def simplified_to_full_binary(simplified_binary):
    """ Añade 'R' al inicio y al final de la codificación
    simplificada y convierte a binario completo. """
    return movimientos["R"] + simplified_binary + movimientos["R"]

def fill_movements(binary_code):
    """ Rellena la codificación binaria con los símbolos
        de movimientos utilizando expresiones regulares. """
```

```
    """ Reemplazar STOP antes de L y R porque es la secuencia más
        larga y evitaría conflictos de superposición. """
    filled_binary = re.sub(r"11110", "STOP", binary_code)
    filled_binary = re.sub(r"1110", "L", filled_binary)
    filled_binary = re.sub(r"110", "R", filled_binary)
    filled_binary = re.sub(r"10", "1", filled_binary)
    filled_binary = re.sub(r"0", "0", filled_binary)
    return filled_binary
```

Por último se embellecen los movimientos, dándoles separación con espacios, usando comas, agregar "0 0" si un movimiento está solo y limpiar si hay duplicaciones, además genera un número de movimiento para cada instrucción de cada máquina:

```
def generate_rules(filled_binary):
    """Genera un diccionario de transiciones a partir
    de la cadena binaria llena de movimientos."""
    rules_with_commas = re.sub(r"(R|L|STOP)", r"\1,", filled_binary)
    rules_with_commas = re.sub(r"(R|L|STOP),(?=(R|L|STOP))",
    r"\1,", rules_with_commas)
    rules_cleaned = re.sub(r",+", ",", rules_with_commas).strip(",")
    rules_list = rules_cleaned.split(",")

    program = {}
    state = 0

    for rule in rules_list:
        if rule in ['R', 'L', 'STOP']:
            current_symbol = 0
            new_symbol = 0
            movement = rule
        else:
            match = re.match(r"(\d*)(\d)(R|L|STOP)", rule)
            if match:
                nums, last_digit, movement = match.groups()
                current_symbol = int(last_digit)
                new_symbol = int(last_digit) if nums
                    == "" else int(nums)
            else:
```

```
            continue

        if movement == 'STOP':
            movement = 'H'
        new_state = state + 1 if movement != 'H' else state

        if (state, current_symbol) in program:
            state = max(state, max([s for s, _ in program.keys()])) + 1

        program[(state, current_symbol)] = (new_state,
            new_symbol, movement)

        if movement != 'H':
            state = new_state

    return program
```

2.2. Para la cinta

Se hizo el siguiente código:

- **single instruction format**. Contiene las siguientes tuplas: $state\ \ from\ \ read$: Una tupla que contiene el estado desde el cual se lee y el símbolo leído. $state_to_write_direction$: Una tupla que contiene el estado hacia el que se transita, el símbolo a escribir y la dirección de movimiento.

```
def single_instruction_format(state_from_read,
    state_to_write_direction):
    state_from, read = state_from_read
    state_to, write, direction = state_to_write_direction
    direction_map = {-1: 'L', 0: 'H', 1: 'R', 'L': 'L',
        'H': 'H', 'R': 'R'}
    return {
        "StateFrom": state_from,
        "Read": read,
        "StateTo": state_to,
        "Write": write,
        "Direction": direction_map[direction]
    }
```

- **preprocess program**.Esta función toma un diccionario de transicio-

nes de una máquina de Turing ("program") y devuelve una lista de estas transiciones ordenadas por su clave.

```
def preprocess_program(program):
    sorted_program = sorted(program.items(), key=lambda x: x[0])
    return sorted_program
```

- **turing program format.** Aquí se procesa una máquina de Turing: Primero, utiliza *preprocess program* para ordenar el programa por sus claves.
  Luego, aplica la función *single instruction format* a cada par clave-valor del programa ordenado, transformándolo en un formato más legible y estructurado. Esto resulta en una lista de instrucciones detalladas y formateadas.

```
def turing_program_format(program):
    preprocessed_program = preprocess_program(program)
    return [single_instruction_format(key, value) for key,
        value in preprocessed_program]
```

- **turing machine.** Esta es la función principal que simula la operación de una máquina de Turing:
  'state': Un diccionario que contiene el estado inicial de la máquina y la posición inicial de la cabeza lectora/escritora sobre la cinta.
  'tape': La cinta de la máquina de Turing, representada como una lista de símbolos.
  'transitions': Un diccionario que mapea pares de (estado, símbolo) a tríos de (nuevo estado, símbolo a escribir, movimiento).

```
def turing_machine(state, tape, transitions):
    head_position = state['pos']
    while True:
        current_symbol = tape[head_position]
        action = transitions.get((state['state'],
            current_symbol))
        if action is None:
            break
        new_state, new_symbol, move = action
        tape[head_position] = new_symbol
        state['state'] = new_state
        if move == 'R':
```

```
            head_position += 1
        elif move == 'L':
            head_position -= 1
        elif move == 'H':
            break
        if head_position < 0 or head_position >= len(tape):
            tape.append(0)
    return tape
```

3. Resultados

3.1. Instrucciones

Para computar y obtener las instrucciones, se hizo uso de este código, que es llamar todo lo anterior:

```
all_machines = turing_machines
for machine_id, info in all_machines.items():
    binary_simplified = decimal_to_binary_simplified
        (info["decimal_number"])
    full_binary = simplified_to_full_binary(binary_simplified)
    filled_binary = fill_movements(full_binary)
    rules = generate_rules(filled_binary)
    decimal_value = binary_to_decimal(binary_simplified)

    print(f"Machine {machine_id}:")
    print(f"  Generated rules: {rules}")
    print(f"  Full binary: {full_binary}")
    print(f"  Binary simplified: {binary_simplified}")
    print(f"  Filled binary: {filled_binary}")
    print(f"  Decimal of simplified binary: {decimal_value}")
```

Y este fue el resultado:

```
output:
"""
Machine T_0:
  Generated rules: {(0, 0): (1, 0, 'R'), (1, 0): (2, 0, 'R')}
  Full binary: 1100110
  Binary simplified: 0
```

```
  Filled binary: ROR
  Decimal of simplified binary: 0
Machine T_1:
  Generated rules: {(0, 0): (1, 0, 'R'), (1, 0): (2, 0, 'L')}
  Full binary: 1101110
  Binary simplified: 1
  Filled binary: RL
  Decimal of simplified binary: 1
Machine T_2:
  Generated rules: {(0, 0): (1, 0, 'R'), (1, 1): (2, 1, 'R')}
  Full binary: 11010110
  Binary simplified: 10
  Filled binary: R1R
  Decimal of simplified binary: 2
Machine T_3:
  Generated rules: {(0, 0): (1, 0, 'R'), (1, 0): (1, 0, 'H')}
  Full binary: 11011110
  Binary simplified: 11
  Filled binary: RSTOP
  Decimal of simplified binary: 3
Machine T_4:
  Generated rules: {(0, 0): (1, 0, 'R'), (1, 0): (2, 1, 'R')}
  Full binary: 110100110
  Binary simplified: 100
  Filled binary: R10R
  Decimal of simplified binary: 4
Machine T_5:
  Generated rules: {(0, 0): (1, 0, 'R'), (1, 1): (2, 1, 'L')}
  Full binary: 110101110
  Binary simplified: 101
  Filled binary: R1L
  Decimal of simplified binary: 5
Machine T_6:
  Generated rules: {(0, 0): (1, 0, 'R'), (1, 0): (2, 0, 'R'),
  (2, 0): (3, 0, 'R')}
  Full binary: 110110110
  Binary simplified: 110
  Filled binary: RRR
  Decimal of simplified binary: 6
Machine T_7:
```

```
  Generated rules: {(0, 0): (1, 0, 'R'), (1, 1): (1, 1, 'H')}
  Full binary: 110111110
  Binary simplified: 111
  Filled binary: R1STOP
  Decimal of simplified binary: 7
Machine T_8:
  Generated rules: {(0, 0): (1, 0, 'R'), (1, 0): (2, 10, 'R')}
  Full binary: 1101000110
  Binary simplified: 1000
  Filled binary: R10OR
  Decimal of simplified binary: 8
Machine T_9:
  Generated rules: {(0, 0): (1, 0, 'R'), (1, 0): (2, 1, 'L')}
  Full binary: 1101001110
  Binary simplified: 1001
  Filled binary: R10L
  Decimal of simplified binary: 9
Machine T_10:
  Generated rules: {(0, 0): (1, 0, 'R'), (1, 1): (2, 1, 'R')}
  Full binary: 1101010110
  Binary simplified: 1010
  Filled binary: R11R
  Decimal of simplified binary: 10
Machine T_11:
  Generated rules: {(0, 0): (1, 0, 'R'), (1, 1): (1, 1, 'H')}
  Full binary: 1101011110
  Binary simplified: 1011
  Filled binary: R1STOP
  Decimal of simplified binary: 11
Machine T_12:
  Generated rules: {(0, 0): (1, 0, 'R'), (1, 0): (2, 0, 'R'),
        (2, 0): (3, 0, 'R')}
  Full binary: 1101100110
  Binary simplified: 1100
  Filled binary: RR0R
  Decimal of simplified binary: 12
Machine unPlus1:
  Generated rules: {(0, 0): (1, 0, 'R'), (1, 1): (2, 1, 'R'),
    (2, 1): (2, 1, 'H'), (3, 1): (3, 1, 'R')}
  Full binary: 110101011010111101010110
```

```
  Binary simplified: 101011010111101010
  Filled binary: R11R1STOP11R
  Decimal of simplified binary: 177642
Machine unTimes2:
  Generated rules: {(0, 0): (1, 0, 'R'), (1, 0): (2, 1, 'R'),
    (2, 1): (3, 10, 'L'), (3, 1): (4, 1, 'R'), (4, 0):
    (5, 11, 'R'), (5, 0): (6, 100, 'R'), (6, 1): (6, 1, 'H'),
    (7, 1): (7, 11, 'R'), (8, 1): (8, 101, 'L'), (9, 1):
    (9, 100, 'R'), (10, 1): (10, 10, 'L'), (11, 1): (11, 101, 'L')}
  Full binary: 1101001101001011101010110101001101000011010111101
    0101101001010111010001011010010111010010101110
  Binary simplified: 1001101001011101010110101001101000011010111101
    0101011010010101110100010110100101110100101
  Filled binary: R10R101L11R110R1000R1STOP111R1011L1001R101L1011L
  Decimal of simplified binary: 14929234209198720269175476 69
Machine xnPlus1:
  Generated rules: {(0, 0): (1, 0, 'R'), (1, 1): (2, 1, 'R'),
    (2, 0): (3, 0, 'R'), (3, 1): (4, 10, 'R'), (4, 0):
    (5, 11, 'L'), (5, 1): (6, 10, 'R'), (6, 1): (6, 1, 'H'),
    (6, 0): (7, 100, 'L'), (7, 1): (8, 101, 'L'), (8, 1):
    (9, 100, 'L'), (9, 0): (10, 110, 'R'), (10, 1):
    (11, 10, 'R'), (11, 0): (12, 0, 'R'), (12, 1):
    (13, 111, 'R'), (13, 1): (14, 11, 'R'), (14, 0): (15, 111, 'R')}
  Full binary: 1101010110110100101101010011101001011010101
    1110100001110100101011101000101110101000110100101101 1
    0101010101101010101101010100110
  Binary simplified: 1010110110100101101010011101001011010111101
    0000111010010101110100010111010100011010010110110101 0
    1010110101010110101010100
  Filled binary: R11RR101R110L101R1STOP1000L1011L1001L110
    0R101RR1111R111R1110R
  Decimal of simplified binary: 4508137044615639589821 13
    7756434379 08
Machine xnTimes2:
  Generated rules: {(0, 0): (1, 0, 'R'), (1, 0): (2, 1, 'R'),
    (2, 1): (3, 1, 'R'), (3, 0): (4, 10, 'R'), (4, 1):
    (5, 11, 'R'), (5, 0): (6, 0, 'R'), (6, 1): (6, 1, 'H')}
  Full binary: 1101001101011010001101010101101101011110
  Binary simplified: 1001101011010001101010101101101011
  Filled binary: R10R1R100R111RR1STOP
```

```
  Decimal of simplified binary: 10389728107
Machine u:
  Generated rules: {(0, 0): (1, 0, 'R'), (1, 1): (2, 10000000, 'L'),
  (2, 0): (3, 1, 'R'), (3, 1): (4, 1001011, 'R'), (4, 0): (5, 10, 'R'),
  (5, 0): (6, 10011000, 'R'), (6, 0): (7, 11, 'R'), (7, 1):
  (8, 1001110, 'R') ...
  Full binary: 11010000000010111010011010001001010101
  10100011010001010000011010100110100010101001011010000
  11010001010010101101001001110100101001001011101010001110 10...
  Binary simplified: 10000000010111010011010001001010101101000110100
  01010000011010100110100010101001011010000110100010100 1001010
  11010010011101001010010010111010100011101010100 1...
  Filled binary: R100000001L10R10010111R100R100110000R110R100111
  01R1000R10011011R1010L10110101L1100L11101011L1110R100110011...
  Decimal of simplified binary: 72448553353393175771983950396157
  11237952360672556559631108144796606505059404241090310483
  61363235936564444345838222688327876762655614469281411771
  50178425517075540856576897533463569424784885970469347257
  39988582283827795294683460521061169835945938791885546326
  44092552550582055598945189071653741489603309675302043155
  36250349845298323206515830476641421307088193297172341510
  56980262734686429921838172157333482823073453713421475059
  74034518437235959309064002432107734217885149276079759763
  44151230795863963544922691594796546147113457001450481673
  37562172573464522731054482980784965126988788964569760906
  63420447798902191443793283001949357096392170390483327088
  25962013017737272027186259199144282754374223513556751340
  84222299889374410534305471044366958764051781280194375 30
  81387063994277282315642528923751456544389905278079324114
  48261423572861931183326106561227555318102075110853376338
  06031082361675045635852164214869542347187426437544428790
  06248582709124042207653875426445413345174856629157429990
  95026230097337381377241621727477236102067868540028935660
  85696822620141982486216989026091309402985706001743006700
  86896759034473417412787425581201549366393899690581773859
  16540553567040928213322216314109787108145997866959970450
  96818419062994436560151454904880922084480034822492077304
  03043188429899393135266882349662101947161910701461968523
  19284748203449589770955356110702758174873332729667899879
  84732840981907648512726310017401667873634776058572450369
```

```
64434897992034489997455662402937487668839751404451665707
75006051388399166881407254554466522205072426239237921152
53181625125363050931728631422004064571305275802307665183
35199568913974813750492642960501001365198018694563949 8
"""
```

Sólo se dejó todo el número decimal ya que ese se puede comparar con el decimal que está en el libro de Penrose (1999) o al menos los unos cuantos dígitos que estén, ya que del libro se descubrió que no ponen todo el número decimal.

¿Cómo se sabe que generó las instrucciones de la verdadera máquina universal de Turing? Para contestar esta pregunta, es muy difícil hacerlo si se compara movimiento por movimiento, lo que se hizo fue confiar ciegamente que del número decimal se consiguió las instrucciones correctas y además, se leyó la longitud de las instrucciones dando un resultado de 402, resultado que coincidió con el de mis compañeros.

3.2. Cinta

Desglose de la simulación de xnPlus1 para la cinta '0 0 0 1 1 0 1':

1. Estado inicial: 0, Cabeza: en el primer 0. Regla: (0, 0) → (0, 0, 'R') Acción: Escribe 0, se mueve a la derecha.
2. Estado: 0, Cabeza: en el segundo 0. Regla: (0, 0) → (0, 0, 'R') Acción: Escribe 0, se mueve a la derecha.
3. Estado: 0, Cabeza: en el tercer 0. Regla: (0, 0) → (0, 0, 'R') Acción: Escribe 0, se mueve a la derecha.
4. Estado: 0, Cabeza: en el primer 1. Regla: (0, 1) → (1, 1, 'R') Acción: Escribe 1, cambia a estado 1, se mueve a la derecha.
5. Estado: 1, Cabeza: en el segundo 1. Regla: (1, 1) → (2, 1, 'R') Acción: Escribe 1, cambia a estado 2, se mueve a la derecha.
6. Estado: 2, Cabeza: en el tercer 1. Regla: (2, 1) → (2, 1, 'R') Acción: Escribe 1, se mueve a la derecha.

7. Estado: 2, Cabeza: en el primer 0. Regla: (2, 0) → (3, 0, 'L') Acción: Escribe 0, cambia a estado 3, se mueve a la izquierda.

8. Estado: 3, Cabeza: en el tercer 1. Regla: (3, 1) → (4, 0, 'L') Acción: Escribe 0, cambia a estado 4, se mueve a la izquierda.

9. Estado: 4, Cabeza: en el segundo 1. Regla: (4, 1) → (4, 1, 'L') Acción: Escribe 1, se mueve a la izquierda.

10. Estado: 4, Cabeza: en el primer 1. Regla: (4, 1) → (4, 1, 'L') Acción: Escribe 1, se mueve a la izquierda.

11. Estado: 4, Cabeza: en el primer 0. Regla: (4, 0) → (5, 1, 'L') Acción: Escribe 1, cambia a estado 5, se mueve a la izquierda.

12. Estado: 5, Cabeza: en el segundo 0. Regla: (5, 0) → (6, 0, 'R') Acción: Escribe 0, cambia a estado 6, se mueve a la derecha.

13. Estado: 6, Cabeza: en el primer 1. Regla: (6, 1) → (7, 1, 'R') Acción: Escribe 1, cambia a estado 7, se mueve a la derecha.

14. Estado: 7, Cabeza: en el tercer 1. Regla: (7, 1) → (7, 0, 'R') Acción: Escribe 0, cambia a estado 7, se mueve a la derecha.

15. Estado: 7, Cabeza: en el primer 0. Regla: (7, 0) → (3, 1, 'R') Acción: Escribe 1, cambia a estado 3, se mueve a la derecha.

16. Estado: 3, Cabeza: en el segundo 1. Regla: (3, 1) → (3, 1, 'R') Acción: Escribe 1, se mueve a la derecha.

Resultado de la cinta: '0, 0, 1, 0, 1, 1, 1'

Ahora probando con la cinta $UN + 1$, quedaría lo siguiente si se aplica a la cinta 0 0 0 1 1 0 0:

Cinta inicial [0, 0, 0, 1, 1, 0, 0]

1. Estado 0, Cabeza en 0, Leyendo 0 ->Estado 0, Escribiendo 0, Moviendo R

2. Estado 0, Cabeza en 1, Leyendo 0 ->Estado 0, Escribiendo 0, Moviendo R

3. Estado 0, Cabeza en 2, Leyendo 0 ->Estado 0, Escribiendo 0, Moviendo R

4. Estado 0, Cabeza en 3, Leyendo 1 ->Estado 1, Escribiendo 1, Moviendo R

5. Estado 1, Cabeza en 4, Leyendo 1 ->Estado 1, Escribiendo 1, Moviendo R

6. Estado 1, Cabeza en 5, Leyendo 0 ->Estado 0, Escribiendo 1, Moviendo H

Resultado de la cinta: [0, 0, 0, 1, 1, 1, 0]

4. Conclusión

Para las instrucciones, se tuvieron muy buenos resultados para las primeras máquinas, sin embargo, mientras más aumentaban los estados a los cuales moverse, más errores aparecían, por ejemplo las máquinas especiales como UN y XN. A pesar de diferentes métodos y bastantes programas hechos, no se pudo resolver estos problemas, sin embargo, se entienden más o menos bien, si se compara con los movimientos originales de las máquinas. Otra observación es que se obtuvieron las instrucciones con enumeración, pero no fue la enumeración correcta. Con respecto a la cinta, se entiende que se tuvo un resultado totalmente correcto, siendo este funcional y bueno.

Referencias

Penrose, R. (1999). *Emperor's New Mind.* Oxford University Press UK.