Π_ρ: Un lenguaje de programación esotérico basado en el teorema fundamental de la aritmética

Henry Díaz Bordón — henrydiazbordon@gmail.com

23 de mayo de 2022

Resumen

Este artículo describirá la sintaxis básica del lenguaje de programación esotérico Π_ρ, así como las matemáticas detrás del correcto funcionamiento del lenguaje y todas las especificaciones y pasos a seguir para la creación de un intérprete funcional del lenguaje.
Palabras clave: *lenguajes, algoritmos, aritmética*

1. Introducción

La idea fundamental detrás de este lenguaje de programación esotérico[5] parte de una pregunta tan simple como trascendental: ¿Se puede crear un lenguaje de programación Turing completo[19] en el que cada instrucción individual sea representada por un número natural[13]?

Esta idea trabajada y refinada es la que ha dado lugar al lenguaje de programación Π_ρ, el modelo de lenguaje de programación esotérico[5] que trataremos y analizaremos en las secciones posteriores de esta publicación. Este lenguaje, aunque original, bebe mucho del conocido Brainfuck[7], pues las instrucciones de ambos lenguajes[11] son similares, efectivamente, se pueden establecer isomorfismos entre ambas (véase la sección 5). Además, de este último lenguaje se han tomado conceptos como el de interpretar la memoria como un arreglo; sin embargo, hemos de mencionar que el conjunto de instrucciones[11] de Π_ρ puede ser modificado libremente para adaptarse a las necesidades de cualquier usuario. No obstante y como es obvio, en este artículo explicaremos el modelo de lenguaje sin ninguna modificación en particular.

En el presente artículo nos encargaremos de describir la sintaxis básica del lenguaje de programación esotérico[5] Π_ρ, así como las matemáticas detrás del correcto funcionamiento del lenguaje y todas las especificaciones y pasos a seguir para la creación de un intérprete funcional de Π_ρ.

2. Sintaxis del lenguaje Π_ρ

Si bien el lenguaje de programación Π_ρ es capaz de soportar una inmensa cantidad de operaciones descritas en forma de números naturales[13] distintos de cero y uno, un programa Π_ρ debe tener una estructura bien diferenciada, y al mismo tiempo estar compuesto por números que posean un significado específico, ergo, que realicen una determinada operación dentro del propio programa y que, por consecuente, no hayan sido seleccionados al azar, sino que formen parte del propio algoritmo[6].

Un ejemplo de programa ejecutable por el lenguaje Π_ρ es la siguiente sucesión de números:

959 11 89 101 10349

Este programa (veremos el motivo en la siguiente sección del artículo) realiza la siguiente secuencia de instrucciones:

1. Las dos primeras instrucciones del programa piden al usuario que introduzca un carácter cualquiera, el cual es almacenado en la presente celda de memoria[12] (este concepto será explicado con mayor detalle en secciones posteriores de este artículo[4]).

2. Luego, compara el carácter recopilado con 0, pasando al paso 3 si efectivamente el carácter es 0 (esto solo puede suceder si se introduce el símbolo ASCII nulo[16]; el único símbolo cuyo valor ASCII[2] es 0x00[1]). En caso contrario, pasa al paso 4.

3. Si el carácter introducido es efectivamente el símbolo ASCII nulo[16], salta a la primera instrucción ejecutada[1], repitiéndose el proceso desde el principio.

4. En caso contrario, imprimirá el carácter introducido y finalizará el programa, poniéndole fin al ciclo.

Este programa, aun siendo muy simple y básico, nos sirve de ejemplo para darle una primera ojeada a la sintaxis del propio lenguaje.

La sintaxis del lenguaje de programación Π_ρ se basa en un concepto muy simple: Para construir el número entero correspondiente a la instrucción a ejecutar, deberemos de tomar el primo asignado a el operador correspondiente[3] y multiplicarlo por el argumento que va a recibir (argumento que ha de ser dado como un número entero positivo[13]).

Con esto obtendríamos un número entero único que realizará esa y solo esa operación, pues por el teorema fundamental de la aritmética[10] la descomposición de esta instrucción en factores primos será única, tras lo que se tomará el menor de estos como instrucción y el cociente entre la instrucción y el primo del operador como argumento.

Sin embargo, hemos de hacer un pequeño inciso, pues si tomamos una instrucción como 14, el intérprete lo ejecutará como la instrucción que posea 2 como número primo[3] suministrándole 7 como argumento, por lo que si su intención es ejecutar el operador con primo 7, proporcionándole 2 como argumento; habrá de buscar alguna alternativa.

Nótese además que las instrucciones[3] del lenguaje de programación Π_ρ son ejecutadas de izquierda a derecha, solo teniendo caracteres espacio[23] o caracteres de nueva línea[14] como delimitadores; además, como resulta obvio el lenguaje Π_ρ solo es capaz de procesar números como programas, por lo que no se permiten símbolos no numéricos en el programa (aparte de los susodichos caracteres ASCII[2]).

3. Instrucciones y registros del lenguaje

El lenguaje Π_ρ cuenta con un total de 30 instrucciones por defecto[3], junto con un total de 3 registros[3] de propósito general y 3 registros[3] que poseen información extra sobre el propio intérprete y acerca del estado actual del programa; información que de otra manera no podríamos obtener. Las instrucciones con las que consta el lenguaje son las siguientes:

1. **Instrucción** *mover puntero*: Esta instrucción toma un valor entero n y desplaza el puntero de memoria a la posición $n-1$. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $\mu_p n$, y es menester aclarar que el primo asignado a esta instrucción es el número 2.

2. **Instrucción** *mover adelante*: Esta instrucción toma un valor entero n y desplaza el puntero de memoria hacia adelante en n posiciones. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $\rightarrow_p n$, y es menester aclarar que el primo asignado a esta instrucción es el número 3.

3. **Instrucción** *mover atrás*: Esta instrucción toma un valor entero n y desplaza el puntero de memoria hacia atrás en n posiciones. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $\leftarrow_p n$, y es menester aclarar que el primo asignado a esta instrucción es el número 5.

4. **Instrucción** *asignar valor*: Esta instrucción toma un valor entero n y asigna a la celda de memoria actual el valor dado como parámetro decrementado en una unidad (esto es, $n-1$). Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $\Sigma_v n$, y es menester aclarar que el primo asignado a esta instrucción es el número 7.

[1] No confundirse con el carácter ASCII "0", pues este posee un valor de 0x30, valor que efectivamente difiere de 0x00, esto es, 0.

5. **Instrucción** *incrementar valor*: Esta instrucción toma un valor entero n e incrementa el valor de la celda actual de memoria por n unidades. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $+_v n$, y es menester aclarar que el primo asignado a esta instrucción es el número 11.

6. **Instrucción** *decrementar valor*: Esta instrucción toma un valor entero n y decrementa el valor de la celda actual de memoria por n unidades. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $-_v n$, y es menester aclarar que el primo asignado a esta instrucción es el número 13.

7. **Instrucción** *multiplicar valor*: Esta instrucción toma un valor entero n y multiplica el valor de la celda actual de memoria por n unidades. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $\times_v n$, y es menester aclarar que el primo asignado a esta instrucción es el número 17.

8. **Instrucción** *dividir valor*: Esta instrucción toma un valor entero n y divide el valor de la celda actual de memoria por n unidades, guardando la parte entera del cociente y omitiendo el resto. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $\div_v n$, y es menester aclarar que el primo asignado a esta instrucción es el número 19.

9. **Instrucción** *módulo del valor*: Esta instrucción toma un valor entero n y asigna el resto del cociente entre el valor de la celda actual de memoria y el argumento dado, esto es, $v \mod n$. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $\mod_v n$, y es menester aclarar que el primo asignado a esta instrucción es el número 23.

10. **Instrucción** *copiar a registro* ${\Delta_1}^3$: Esta instrucción no toma ningún argumento, y copia el valor de la celda de memoria actual al registro de propósito general número 1, esto es, al registro ${\Delta_1}^3$ (véase el apartado de registros[3] de la sección 3). Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como c_{Δ_1}, y es menester aclarar que el primo asignado a esta instrucción es el número 29.

11. **Instrucción** *copiar a registro* ${\Delta_2}^3$: Esta instrucción no toma ningún argumento, y copia el valor de la celda de memoria actual al registro de propósito general número 2, esto es, al registro ${\Delta_2}^3$ (véase el apartado de registros[3] de la sección 3). Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como c_{Δ_2}, y es menester aclarar que el primo asignado a esta instrucción es el número 31.

12. **Instrucción** *copiar a registro* ${\Delta_3}^3$: Esta instrucción no toma ningún argumento, y copia el valor de la celda de memoria actual al registro de propósito general número 3, esto es, al registro ${\Delta_3}^3$ (véase el apartado de registros[3] de la sección 3). Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como c_{Δ_3}, y es menester aclarar que el primo asignado a esta instrucción es el número 37.

13. **Instrucción** *copiar a celda*: Esta instrucción toma un valor entero n y copia el valor de la celda de memoria actual a la celda de memoria número $n-1$. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $c_c n$, y es menester aclarar que el primo asignado a esta instrucción es el número 41.

14. **Instrucción** *cortar a registro* ${\Delta_1}^3$: Esta instrucción no toma ningún argumento y copia el valor de la celda de memoria actual al registro de propósito general número 1, esto es, al registro ${\Delta_1}^3$ (véase el apartado de registros[3] de la sección 3), para luego asignarle a la celda de memoria actual el valor de 0. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como x_{Δ_1}, y es menester aclarar que el primo asignado a esta instrucción es el número 43.

15. **Instrucción** *cortar a registro* ${\Delta_2}^3$: Esta instrucción no toma ningún argumento y copia el valor de la celda de memoria actual al registro de propósito general número 2, esto es, al registro ${\Delta_2}^3$ (véase el apartado de registros[3] de la sección 3), para luego asignarle a la celda de memoria actual el valor de 0. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como x_{Δ_2}, y es menester aclarar que el primo asignado a esta instrucción es el número 47.

16. **Instrucción** *cortar a registro* Δ_3: Esta instrucción no toma ningún argumento y copia el valor de la celda de memoria actual al registro de propósito general número 3, esto es, al registro ${\Delta_3}^3$ (véase el apartado de registros[3] de la sección 3), para luego asignarle a la celda de memoria actual el valor de 0. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como x_{Δ_3}, y es menester aclarar que el primo asignado a esta instrucción es el número 53.

17. **Instrucción** *cortar a celda*: Esta instrucción toma un valor entero n y copia el valor de la celda de memoria actual a la celda de memoria número $n-1$, para luego asignarle a la celda de memoria actual el valor de 0. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $x_c n$, y es menester aclarar que el primo asignado a esta instrucción es el número 59.

18. **Instrucción** *intercambiarconregistro*${\Delta_1}^3$: Esta instrucción no toma ningún argumento e intercambia el valor de la celda de memoria actual con el valor del registro de propósito general número 1, esto es, el registro ${\Delta_1}^3$ (véase el apartado de registros[3] de la sección 3). Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como s_{Δ_1}, y es menester aclarar que el primo asignado a esta instrucción es el número 61.

19. **Instrucción** *intercambiarconregistro*${\Delta_2}^3$: Esta instrucción no toma ningún argumento e intercambia el valor de la celda de memoria actual con el valor del registro de propósito general número 2, esto es, el registro ${\Delta_2}^3$ (véase el apartado de registros[3] de la sección 3). Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como s_{Δ_2}, y es menester aclarar que el primo asignado a esta instrucción es el número 67.

20. **Instrucción** *intercambiarconregistro*${\Delta_3}^3$: Esta instrucción no toma ningún argumento e intercambia el valor de la celda de memoria actual con el valor del registro de propósito general número 3, esto es, el registro ${\Delta_3}^3$ (véase el apartado de registros[3] de la sección 3). Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como s_{Δ_3}, y es menester aclarar que el primo asignado a esta instrucción es el número 71.

21. **Instrucción** *intercambiar con celda*: Esta instrucción toma un valor entero n e intercambia el valor de la celda de memoria actual con la celda de memoria número $n-1$. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $s_c n$, y es menester aclarar que el primo asignado a esta instrucción es el número 73.

22. **Instrucción** *imprimir carácter*: Esta instrucción toma un valor entero n y lo imprime en pantalla convertido en su respectivo carácter UTF-8[22]. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $\pi_v n$, y es menester aclarar que el primo asignado a esta instrucción es el número 79.

23. **Instrucción** *imprimir entero*: Esta instrucción toma un valor entero n y lo imprime en pantalla como número entero, sin convertirlo previamente a un carácter individual. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $\pi_i n$, y es menester aclarar que el primo asignado a esta instrucción es el número 83.

24. **Instrucción** *ramificar si hay igualdad*: Esta instrucción toma un valor entero n y

compara el valor de la celda de memoria actual con $n-1$. Si se cumple la igualdad, incrementa el contador del programa[17] en una unidad, mientras que si esto no ocurre, incrementa el contador en 2 unidades. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $?_{=}n$, y es menester aclarar que el primo asignado a esta instrucción es el número 89.

25. **Instrucción** $ramificarsinohayigualdad$: Esta instrucción toma un valor entero n y compara el valor de la celda de memoria actual con $n-1$. Si no se cumple la igualdad, incrementa el contador del programa[17] en una unidad, mientras que si esto no ocurre, incrementa el contador en 2 unidades. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $?_{\neq}n$, y es menester aclarar que el primo asignado a esta instrucción es el número 97.

26. **Instrucción** *saltar a instrucción*: Esta instrucción toma un valor entero n y asigna el valor $n-1$ al contador del programa[17], ergo, desplaza el contador a la instrucción $n-1$. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $\jmath_v n$, y es menester aclarar que el primo asignado a esta instrucción es el número 101.

27. **Instrucción** *saltar hacia adelante*: Esta instrucción toma un valor entero n e incrementa el valor del contador del programa[17] en n unidades, ergo, desplaza el contador hacia delante en n instrucciones. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $\jmath_{\rightarrow} n$, y es menester aclarar que el primo asignado a esta instrucción es el número 103.

28. **Instrucción** *saltar hacia atrás*: Esta instrucción toma un valor entero n y decrementa el valor del contador del programa[17] en n unidades, ergo, desplaza el contador hacia atrás en n instrucciones, nunca siendo este registro un número negativo (en caso de que n fuese mayor que el contador del programa, se le asignaría 0 al contador y se proseguiría ejecutando las respectivas instrucciones). Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $\jmath_{\leftarrow} n$, y es menester aclarar que el primo asignado a esta instrucción es el número 107.

29. **Instrucción** *ninguna operación*: Esta instrucción no realiza ninguna operación, es decir, no hace nada en particular. Es equivalente a la instrucción *nop*[15] que poseen muchos lenguajes de máquina. Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como $\emptyset$, y es menester aclarar que el primo asignado a esta instrucción es el número 109.

30. **Instrucción** *terminar programa*: Esta instrucción no toma ningún argumento y, una vez ejecutada, termina el programa con código[9] 0 (ningún error). Para mayor simplicidad, en este artículo nos referiremos a la susodicha operación como h, y es menester aclarar que el primo asignado a esta instrucción es el número 113.

Además, como ya he mencionado anteriormente 3, el lenguaje consta de un total de 6 registros (3 registros encargados de suministrar información sobre el estado actual del programa y otros 3 registros de propósito general), los cuales poseen cada uno su propio número primo asignados. Estos son explicados a continuación:

1. **Registro** *índice de la celda*: Este registro contiene y suministra el índice respecto a la memoria que posee la celda de memoria actual, teniendo un valor de 0 si la celda es la primera del arreglo de memoria, 1 si es la segunda... Para mayor simplicidad, en este artículo nos referiremos al susodicho registro como χ, y es menester aclarar que el primo asignado a este registro es el número 127.

2. **Registro** *valor de la celda*: Este registro contiene y suministra el valor entero que posee la celda de memoria actual. Para mayor simplicidad, en este artículo nos referiremos al susodicho registro como v, y

es menester aclarar que el primo asignado a este registro es el número 131.

3. **Registro** *entrada del usuario*: Este registro, una vez llamado le pedirá al usuario que introduzca un carácter, suministrando el valor entero del carácter introducido. Para mayor simplicidad, en este artículo nos referiremos al susodicho registro como γ, y es menester aclarar que el primo asignado a este registro es el número 137.

4. **Registro** *de propósito general* 1: Este registro es el primero de la lista de registros de propósito general, los cuales tienen como función ser contenedores de números enteros fácilmente accedidos y llamados en cualquier instrucción. Para mayor simplicidad, en este artículo nos referiremos al susodicho registro como Δ_1, y es menester aclarar que el primo asignado a este registro es el número 139.

5. **Registro** *de propósito general* 2: Este registro es el segundo de la lista de registros de propósito general, los cuales tienen como función ser contenedores de números enteros fácilmente accedidos y llamados en cualquier instrucción. Para mayor simplicidad, en este artículo nos referiremos al susodicho registro como Δ_2, y es menester aclarar que el primo asignado a este registro es el número 149.

6. **Registro** *de propósito general* 3: Este registro es el tercero y último de la lista de registros de propósito general, los cuales tienen como función ser contenedores de números enteros fácilmente accedidos y llamados en cualquier instrucción. Para mayor simplicidad, en este artículo nos referiremos al susodicho registro como Δ_3, y es menester aclarar que el primo asignado a este registro es el número 151.

4. Funcionamiento del intérprete

El primer paso a realizar con el fin de interpretar un programa escrito en el lenguaje Π_ρ es, evidentemente, obtener el programa en sí; conseguir el arreglo de números enteros distintos de 0 y 1 que representarán el código a ejecutar. Además, para mayor simplicidad, nos referiremos a este arreglo como $\mathcal{S}$ en el código y pseudocódigo utilizado.

Existen multitud de métodos para llevar esta tarea a cabo, pues este proceso puede realizarse o bien pidiéndole a los usuarios la entrada mediante un sistema de escritura de texto (un sistema de "*input*", como lo pueden ser la consola de comandos o una interfaz gráfica dedicada), o leyéndolos y extrayéndolos desde un archivo externo, cuyo nombre puede ser solicitado al usuario al comienzo de la ejecución del intérprete. Además, introducir el programa como argumentos de línea de comandos podría ser otro posible método para hacerse con este conjunto de números. En efecto, esta operación puede ser realizada con una sola línea de código en el lenguaje de programación Python[4] utilizando el primero de los métodos aquí listados:

```
1    S = [int(x) for x in
     ↪  input().split(' ')]
```

Una vez obtenido este arreglo, habremos de construir los demás componentes del propio intérprete de Π_ρ:

1. **Contador de programa**: El contador de programa[17] (también conocido como contador de eventos o puntero de instrucciones) será un valor entero no negativo que, como su propio nombre indica, contenga el índice (esto es, que apunte) a la instrucción actual a ejecutar dentro de $\mathcal{S}$, es decir, que posea el índice de la instrucción a ser procesada dentro del programa dado. Al inicio del programa, este valor será nulo (es decir, 0), y conforme cada instrucción sea ejecutada, esta variable se incrementará en una unidad. Para mayor simplicidad, durante el contenido de este artículo nos referiremos al contador de programa como λ.

2. **Arreglo de memoria**: Este componente del lenguaje será el encargado de proveer almacenamiento para los datos que el programa a ejecutar genere. Al igual que en el lenguaje de programación Brainfuck[7] o en una misma Máquina de Turing[20], trataremos a esta estructura como un

arreglo de números enteros. Como hemos mencionado en la sección previa (véase sección 3), existen multitud de instrucciones capaces de modificar esta estructura (un total de 15, para ser exactos), además de que la totalidad del conjunto de instrucciones del lenguaje posee acceso a este componente (y lee de él). Al inicio del programa, este arreglo estará compuesto de un número indefinido de ceros (este número habrá de ser decidido por el desarrollador aunque, en concepto, el arreglo de memoria tiene una longitud ilimitada), además, para mayor simplicidad, durante el contenido de este artículo nos referiremos a la memoria como $\mathcal{M}$.

3. **Puntero de memoria**: El puntero de memoria será un valor entero no negativo que, como su propio nombre indica, contenga el índice (esto es, que apunte) a la celda de memoria que esté siendo utilizada en ese momento, es decir, que posea el índice de la celda de memoria que se esté empleando durante el ciclo actual. Al inicio del programa, este valor será nulo (es decir, 0), y podrá variar según se ejecuten determinadas instrucciones del lenguaje (en concreto, aquellas ideadas para modificar este valor de algún modo). Para mayor simplicidad, durante el contenido de este artículo nos referiremos al puntero de memoria como χ. Además, es posible percatarse de que este valor es a su vez equivalente al registro χ, pues ambos representan el índice de la celda de memoria actual, ergo, son el mismo componente del lenguaje con la ligera diferencia de que el puntero de memoria en sí no puede ser accedido directamente desde un programa (abarcaremos este tema más adelante). Este es a su vez el motivo por el que hemos designado a ambos componentes con el mismo nombre.

Una vez construidos estos tres elementos, habremos de proseguir con el punto de partida de el intérprete, pero antes de enfrentarnos a esta tarea, habremos de idear una función que, dada una instrucción del lenguaje (un número entero distinto de 0 y 1), nos genere efectivamente el número primo de la instrucción y el valor de su argumento. Para ello, tendremos que realizar la descomposición factorial del propio número y, tras ello, tomar el menor de ellos (este valor será el primo de la instrucción) y multiplicar todos los valores mayores (este será su argumento). En el caso de que no hayan argumentos propiamente definidos (el número de la instrucción sea primo), se tomará el valor 1 como argumento. El pseudocódigo para este algoritmo sería el siguiente:

Algoritmo 1: Separación entre instrucción y argumentos

Datos: $n \in \mathbb{N}_0$ y $n \neq 0, 1$
Resultado: $y \in (\mathbb{N}_0 \setminus \{0, 1\}) \times \mathbb{N}$
$\mathcal{F} \leftarrow [\,]$;
para $k \leftarrow 2$ *hasta* n **hacer**
  si $(0 \equiv n \mod k)$ *&* $(k \in \mathbb{P})$ **entonces**
    $\mathcal{F} \leftarrow k$;
  fin
fin
$\mathcal{F} \leftarrow 1$;
$y \leftarrow \{\mathcal{F}[0], \prod_{j \in (\mathcal{F} \setminus \mathcal{F}[0])} j\}$;

Sin embargo, este algoritmo no está del todo completo, pues hemos de recordar que las instrucciones pueden tomar registros como argumentos, los cuales son referidos como números primos (véase el apartado de registros[3] de la sección 3), por lo que habremos de crear otro algoritmo que convierta argumentos (números enteros positivos) en su respectivo valor, sean registros o valores literales. Esta tarea se podría simplificar enormemente utilizando un bloque *switch* en lenguajes de programación como C, C#, JavaScript e incluso el previamente mencionado lenguaje Python[4]. Además, he de aclarar que en el algoritmo mostrado a continuación, los registros $\chi, \upsilon, \gamma, \Delta_1, \Delta_2$ y Δ_3 han de estar definidos en el programa y deben de iniciar la ejecución como 0 salvo el registro γ, el cual (como relatamos previamente) tomará la entrada de un solo carácter del usuario y del cual se leerá el valor entero ASCII del carácter introducido. Esto se puede conseguir utilizando funciones como *getch* en los lenguajes de programación C y C++, o utilizando paquetes como $py - getch$[1]. Otro aspecto importante a destacar es que en la conversión del argumento a valor literal, se leería de los registros si fuese necesario, dejando su valor intacto salvo que

la instrucción a ejecutar realice una transformación sobre alguno (un suceso virtualmente seguro). Sin más que objetar, este sería el algoritmo para la conversión de argumento a valor literal:

Algoritmo 2: Conversión de argumento a valor literal

Datos: $n \in \mathbb{N}$
Resultado: $y \in \mathbb{Z}$
si $(n = 127)$ **entonces**
  $y \leftarrow \chi$;
si no, si $(n = 131)$ **entonces**
  $y \leftarrow \upsilon$;
si no, si $(n = 137)$ **entonces**
  $y \leftarrow \gamma$;
si no, si $(n = 139)$ **entonces**
  $y \leftarrow \Delta_1$;
si no, si $(n = 149)$ **entonces**
  $y \leftarrow \Delta_2$;
si no, si $(n = 151)$ **entonces**
  $y \leftarrow \Delta_3$;
en otro caso
  $y \leftarrow n$;
fin

Una vez teniendo los argumentos para extraer la instrucción y el argumento de la operación a realizar, podemos continuar con la propia ejecución del programa. Esta tarea no requerirá de gran complejidad, sin embargo ha de ser bien implementada por el propio desarrollador del intérprete, pues de ella depende el buen funcionamiento de los programas. La estrategia a realizar consistiría en recorrer toda $\mathcal{S}$ utilizando un bucle for, analizando cada instrucción imperativamente. El algoritmo para este procedimiento sería el siguiente:

Algoritmo 3: Algoritmo para la ejecución del programa

para $k \leftarrow 0$ *hasta* $|\mathcal{S}|$ **hacer**
  $\alpha \leftarrow \text{sep}(\mathcal{S}[k])_4$;
  si $(\alpha[0] = 2)$ **entonces**
    $\chi \leftarrow \text{con}(\alpha[1])_4 - 1$;
  si no, si $(\alpha[0] = 3)$ **entonces**
    $\chi \leftarrow \chi + \text{con}(\alpha[1])_4 - 1$;
  si no, si ... **entonces**
    ...
  en otro caso
    devolver error;
  fin
fin

Y una vez hayamos construido el punto de partida del intérprete y el analizador del programa, solo habremos de enlazar todos los algoritmos listados en esta sección y traducirlos a un lenguaje de programación en específico; con lo que habríamos construido un intérprete funcional del lenguaje Π_ρ.

5. Completitud de Turing del lenguaje

La siguiente sección del artículo estará dedicada exclusivamente a la demostración formal de la completitud de Turing[19] del lenguaje de programación Π_ρ, demostración que requerirá de la certeza de que el lenguaje Brainfuck[7] es Turing completo[19]; hecho que podemos enunciar mediante un lema previamente establecido y demostrado:

Lema 0.1. *El lenguaje de programación Brainfuck[7] es Turing completo[19], ergo, es computacionalmente equivalente a una máquina de Turing[20].*

Demostración. La demostración de este lema es demasiado extensa en comparación a la longitud del presente artículo, por este motivo, no la incluiremos, pues resultaría en una publicación innecesariamente extensa. Una prueba formal y libre de errores de este enunciado podría ser la dada por Franciscus Johannes Faase en el artículo "$BrainfuckisTuringcomplete$"[3] de su sitio web. ■

Una vez teniendo la certeza de que, efectivamente, el lenguaje Brainfuck es equivalente a una máquina de Turing[20], podremos demostrar la completitud de Turing del lenguaje Π_ρ:

Teorema 1. *El lenguaje de programación Π_ρ es Turing completo[19], ergo, es computacionalmente equivalente a una máquina de Turing[20].*

Demostración. La estrategia que seguiremos en esta demostración para atacar el problema impuesto consistirá en establecer isomorfismos entre ciertas instrucciones de Π_ρ y la totalidad de los operadores del lenguaje Brainfuck[7], pues, si es posible establecer las susodichas igualdades podremos enunciar con total seguridad que Brainfuck[7] es emulable en el lenguaje Π_ρ (y viceversa, pues al Brainfuck[7] ser Turing

completo[19], es decir, equivalente a una Máquina Universal de Turing[21], puede ejecutar en sí cualquier algoritmo descrito, por consecuente y al estar compuesto el lenguaje Π_ρ de algoritmos yuxtapuestos entre sí, podremos afirmar con total seguridad que el lenguaje Π_ρ podrá ser también emulable en Brainfuck[7]), y al ser Brainfuck[7] equivalente a una Máquina Universal de Turing[21] por su condición de Turing completo[19], estaremos demostrando a su vez por la propiedad transitiva[18] que se puede emular una Máquina Universal de Turing[21] dentro del lenguaje de programación Π_ρ. La lista de isomorfismos que relacionan a los operadores de Brainfuck[7] con los de Π_ρ es la siguiente:

1. $> \cong \rightarrow_p 1$: Este isomorfismo es dado puesto que, como es definida la instrucción $\rightarrow_p n$ en el apartado de instrucciones[3] de la sección 3, el puntero de memoria se desplazará hacia su derecha en n posiciones. Al ser en este caso $n = 1$, el puntero de memoria será desplazado hacia su derecha en una sola posición, resultando en exactamente el mismo comportamiento de la instrucción $>$ de Brainfuck[7].

2. $< \cong \leftarrow_p 1$: Este isomorfismo es dado puesto que, como es definida la instrucción $\leftarrow_p n$ en el apartado de instrucciones[3] de la sección 3, el puntero de memoria se desplazará hacia su izquierda en n posiciones. Al ser en este caso $n = 1$, el puntero de memoria será desplazado hacia su izquierda en una sola posición, resultando en exactamente el mismo comportamiento de la instrucción $<$ de Brainfuck[7].

3. $+ \cong +_v 1$: Este isomorfismo es dado puesto que, como es definida la instrucción $+_v n$ en el apartado de instrucciones[3] de la sección 3, el valor de la celda actual de memoria será incrementado en n unidades. Al ser en este caso $n = 1$, el valor de la celda actual de memoria será incrementado en una sola unidad, resultando en exactamente el mismo comportamiento de la instrucción $+$ de Brainfuck[7].

4. $- \cong -_v 1$: Este isomorfismo es dado puesto que, como es definida la instrucción $-_v n$ en el apartado de instrucciones[3] de la sección 3, el valor de la celda actual de memoria será decrementado en n unidades. Al ser en este caso $n = 1$, el valor de la celda actual de memoria será decrementado en una sola unidad, resultando en exactamente el mismo comportamiento de la instrucción $-$ de Brainfuck[7].

5. $\cdot \cong \pi_v v$: Este isomorfismo es dado puesto que, como es definida la instrucción $\pi_v n$ en el apartado de instrucciones[3] de la sección 3, se imprime el valor de la celda actual de memoria convertido en su respectivo carácter UTF-8[22]. Al ser en este caso $n = v$ (es decir, el registro cuyo valor es el de la celda actual de memoria), el valor de la susodicha celda será impreso en la línea de comandos convertido en su respectivo carácter UTF-8[22], resultando en exactamente el mismo comportamiento de la instrucción $\cdot$ de Brainfuck[7].

6. $, \cong \{\Sigma_v \gamma, +_v 1\}$: Este isomorfismo es dado puesto que, como es definida la instrucción $\Sigma_v n$ en el apartado de instrucciones[3] de la sección 3, se asigna el argumento proporcionado en la instrucción a la celda actual de memoria decrementado en una unidad. Al ser en este caso $n = \gamma$ (es decir, el registro cuyo valor es el de el carácter individual proporcionado por el usuario), el valor de la susodicha celda será el carácter ASCII[2] predecesor al introducido. Sin embargo, para que el comportamiento de esta instrucción sea isomorfo al de el operador , de Brainfuck[7]; hemos de incrementar la celda de memoria actual en una unidad mediante la instrucción $+_v 1$, resultando en exactamente el mismo comportamiento de la instrucción , de Brainfuck[7].

7. $[\cong \{?_= 1, \jmath_\rightarrow (x+3)\}$, donde x es el número de instrucciones adjuntadas entre los caracteres delimitadores [y] excluyendo las instrucciones , más el doble del número de instrucciones , entre los caracteres delimitadores [y]: Este isomorfismo es dado puesto que, como son definidas las instrucciones $?_= n$ y $\jmath_\rightarrow m$ en el apartado de

instrucciones[3] de la sección 3, se comparará el valor de la celda actual de memoria con n y, si esta igualdad se cumple, ejecutará la instrucción $\jmath_{\rightarrow}m$, que incrementará el puntero de memoria en m unidades. Al ser en este caso $n = 1$ y $m = (x + 3)$, se comparará el susodicho valor con 0 y, si la dicha igualdad se cumple, se avanzará hasta el final del bucle, resultando en exactamente el mismo comportamiento de la instrucción [de Brainfuck[7].

8.] $\cong$ $\{?_{\neq}1, \jmath_{\leftarrow}x\}$, donde x es el número de instrucciones adjuntadas entre los caracteres delimitadores [y] excluyendo las instrucciones , más el doble del número de instrucciones , entre los caracteres delimitadores [y]: Este isomorfismo es dado puesto que, como son definidas las instrucciones $?_{\neq}n$ y $\jmath_{\leftarrow}m$ en el apartado de instrucciones[3] de la sección 3, se comparará el valor de la celda actual de memoria con n y, si esta igualdad no se cumple, ejecutará la instrucción $\jmath_{\leftarrow}m$, que decrementará el puntero de memoria en m unidades. Al ser en este caso $n = 1$ y $m = x$, se comparará el susodicho valor con 0 y, si la dicha desigualdad no se cumple, retrocederá hasta el comienzo del bucle nuevamente, resultando en exactamente el mismo comportamiento de la instrucción] de Brainfuck[7].

Y una vez dadas estas igualdades, podemos enunciar con total seguridad que, efectivamente, el lenguaje de programación Π_ρ es Turing completo[19] y que por ende, es capaz de emular a una Máquina Universal de Turing[21] y ejecutar cualquier algoritmo[6] formalmente descrito. ■

Y habiendo demostrado la completitud de Turing[19] del modelo de lenguaje de programación propuesto en este artículo, el propósito de esta sección ha concluido y podemos proseguir con la conclusión final de esta publicación.

6. Conclusiones finales

Luego del contenido explicado durante esta publicación, podemos responder con total seguridad a la pregunta planteada en el inicio de este artículo[1], pues bien es posible crear un lenguaje de programación Turing completo[19] en el que cada instrucción individual sea un número entero positivo distinto de 1; y este artículo es prueba de ello, pues el modelo de lenguaje relatado durante la totalidad de la publicación cumple con todos los requisitos establecidos, aportando una nueva y refrescante idea al campo de los lenguajes de programación esotéricos[5].

Sin embargo, el lenguaje de programación Π_ρ posee, por su propio concepto, un grave problema intrínseco, pues debido a la conmutatividad[8] de la multiplicación, sacar el máximo provecho del amplio y extenso conjunto de instrucciones[3] del lenguaje se torna una tarea sumamente compleja, pues si el argumento de la instrucción dada contiene en sus factores primos a un número primo menor al de la instrucción, el compilador tendrá problemas en identificar cuál era la instrucción pensada para ser ejecutada.

Debido a estos serios problemas y al mero hecho de ser un lenguaje de programación esotérico[5], recomendamos utilizar el lenguaje de programación Π_ρ con fines puramente educativos y de investigación, aun estando demostrada la completitud de Turing[19] del lenguaje.

Referencias

Joe Esposito. Portable getch() for python. `https://github.com/joeyespo/py-getch`.

ASCII Code The extended ASCII table. Ascii character list. `https://www.ascii-code.com/`.

Franciscus Johannes Faase. Brainfuck is turing-complete. `https://web.archive.org/web/20081025012833/http://www.iwriteiam.nl/Ha_bf_Turing.html`.

The Python Software Foundation. Python programming language. `https://www.python.org/`.

Esolang: the esoteric programming languages wiki. Esoteric programming language. `https://esolangs.org/wiki/Esoteric_programming_language`.

Wikipedia: the free encyclopedia. Algorithm. https://en.wikipedia.org/wiki/Algorithm.

Wikipedia: the free encyclopedia. Brainfuck programming language. https://en.wikipedia.org/wiki/Brainfuck.

Wikipedia: the free encyclopedia. Commutative property. https://en.wikipedia.org/wiki/Commutative_property.

Wikipedia: the free encyclopedia. Exit status. https://en.wikipedia.org/wiki/Exit_status.

Wikipedia: the free encyclopedia. Fundamental theorem of arithmetic. https://en.wikipedia.org/wiki/Fundamental_theorem_of_arithmetic.

Wikipedia: the free encyclopedia. Instruction set architecture. https://en.wikipedia.org/wiki/Instruction_set_architecture.

Wikipedia: the free encyclopedia. Memory cell (computing). https://en.wikipedia.org/wiki/Memory_cell_(computing).

Wikipedia: the free encyclopedia. Natural number. https://en.wikipedia.org/wiki/Natural_number.

Wikipedia: the free encyclopedia. Newline character. https://en.wikipedia.org/wiki/Newline.

Wikipedia: the free encyclopedia. Nop (code). https://en.wikipedia.org/wiki/NOP_(code).

Wikipedia: the free encyclopedia. Null character. https://en.wikipedia.org/wiki/Null_character.

Wikipedia: the free encyclopedia. Program counter. https://en.wikipedia.org/wiki/Program_counter.

Wikipedia: the free encyclopedia. Transitive relation. https://en.wikipedia.org/wiki/Transitive_relation.

Wikipedia: the free encyclopedia. Turing completeness. https://en.wikipedia.org/wiki/Turing_completeness.

Wikipedia: the free encyclopedia. Turing machine. https://en.wikipedia.org/wiki/Turing_machine.

Wikipedia: the free encyclopedia. Universal turing machine. https://en.wikipedia.org/wiki/Universal_Turing_machine.

Wikipedia: the free encyclopedia. Utf-8 encoding. https://en.wikipedia.org/wiki/UTF-8.

Wikipedia: the free encyclopedia. Whitespace character. https://en.wikipedia.org/wiki/Whitespace_character.