Revista Venezolana de Computación
ISSN: 2244-7040
http://www.svc.net.ve/revecom
Vol. 6, No. 2, pp. 10-20, Diciembre 2019
Fecha recepción: 29/11/2019, Fecha aceptación: 10/04/2020

Precondición Más Débil de Algoritmos de Punto Fijo

Federico Flaviani[1]
fflaviani@usb.ve

[1] Departamento de Computación, Universidad Simón Bolívar, Caracas, Venezuela

Resumen: Dijkstra definió el transformador de predicados wp (*weakest precondition*), y en dicha definición establece para la instrucción de iteración Do un predicado $H_k(\text{Post})$ que describe los estados iniciales que hacen que el ciclo itere a lo sumo k veces, satisfaciendo la postcondición Post. En trabajos recientes se han determinado técnicas para calcular explícitamente $H_k(\text{Post})$, lo cual representa una alternativa a la regla de Hoare del invariante y una forma de calcular precondiciones más débiles de Do. No existen muchas técnicas prácticas, que permitan calcular explícitamente la precondicion más débil wp, por lo tanto historicamente ha sido más versatil usar las reglas de Hoare para obtener precondiciones, con lo cual se sacrifica la generalidad de las aserciones. En este trabajo se muestra como hacer este cálculo explícito de wp para una familia de algoritmos de punto fijo. Esta familia de algoritmos, junto con sus precondiciones más débiles, representan un framework para construir algoritmos de punto fijo correctos.

Palabras Clave: Corrección Formal de Programas; Precondición más Débil; Derivación de Algoritmos.

Abstract: Dijkstra defined the predicate transformer called wp (weakest precondition), and that definition establishes the predicate $H_k(\text{Post})$ for the iteration statement DO, that describes the initial states that forcing the loop to iterate at most k times, with a final state that satisfy the post-condition Post. In recent works, some techniques have been developed to calculate explicitly $H_k(\text{Post})$, which represents an alternative to invariants of Hoare's rule, and a way to calculate weakest precondition of DO. There are not many practical techniques to explicitly calculate the weakest precondition wp, therefore historically it has been more versatile to use Hoare's rules to obtain preconditions, thereby sacrificing the generality of assertions. This work shows how to do this explicit calculation of wp for a family of fixed point algorithms. This family of algorithms, along with their weakest preconditions, represent a framework for building correct fixed point algorithms.

Keywords: Formal Program Verification; Weakest Precondition; Derivation of Algorithms.

I. Introducción

En este trabajo se realiza un análisis de una familia de algoritmos de punto fijo para calcular un conjunto. Dada una postcondición que enuncia que cierta variable contiene el conjunto deseado y un algoritmo de punto fijo, se estudia la precondición más débil y la condición de terminación de los ciclos de algoritmos de punto fijo, usando las técnicas de [1][2].

Para unificar la presentación de los algoritmos en este trabajo, todos los algoritmos serán escritos en pseudolenguaje *GCL* (*Guarded Command Language*) [3], que es un pseudolenguaje definido por Dijkstra, donde se pueden escribir algoritmos no determinísticos. GCL admite una lógica de Hoare [4] y fórmulas para precondiciones más débiles, relativamente simples, que facilitan la actividad de corrección de un programa.

Si Pre y Post son predicados para ser usados como precondición y postcondición de una instrucción S, entonces la lógica de Hoare [4] se basa en la noción de tripletas $\{\text{Pre}\}S\{\text{Post}\}$, que se entienden como un predicado que es verdadero si y sólo si la instrucción S manda los estados del programa que satisfacen Pre a estados que satisfacen Post.

En este trabajo se supone que ninguna expresión tiene valores indefinidos, sino más bien, indeterminados. Por ejemplo si $\{R_i\}_{i=a}^{n}$ es una familia de conjuntos, entonces si i es de tipo entero, se supone que la expresión R_i está definida para todo i entero, solo que para $i < a \vee i > n$ no se conoce el valor de R_i, pero sí existe. De esta forma un predicado de la forma $P(R_i)$ tiene siempre un valor de verdad, solo que para $i < a \vee i > n$ no se conoce. Al modificar el predicado anterior por $a \leq i \leq n \wedge P(R_i)$, se tiene que siempre se puede calcular el valor de verdad, ya que para los valores de i que no se conoce el valor de verdad de $P(R_i)$, sí se conoce que $a \leq i \leq n$ es falso y por lo tanto todo el predicado sería falso.

La abreviación $\text{domain}(Exp_1, \ldots, Exp_n)$ [5] referencia a

un predicado que indica los valores de las variables de las expresiones $Exp_1, \ldots, Exp_n$, en que el valor de todas ellas están determinadas. Por ejemplo $\mathrm{domain}(R_i) \equiv a \leq i \leq n$. Al igual que en el párrafo anterior, un predicado cualquiera $P(Exp_1, \ldots, Exp_n)$ siempre puede ser modificado por $\mathrm{domain}(Exp_1, \ldots, Exp_n) \wedge P(Exp_1, \ldots, Exp_n)$, para que su valor de verdad siempre se pueda calcular, incluso cuando el valor de las expresiones esté indeterminado. En el presente trabajo se sigue la filosofía de usar siempre predicados de este tipo.

Salvo en la Sección XI, las nomenclaturas $\Rightarrow$ y $B \rightarrow S$ denotan la implicación y una guardia con condición B e instrucción S respectivamente. Solamente en la Sección XI, estas nomenclaturas denotarán la relación de derivación y una producción de una gramática libre de contexto respectivamente.

En este trabajo si se abrevian a S_i con $0 \leq i \leq n$ como instrucciones, entonces se denota a $S_i; S_j$ como la secuenciación de instrucciones, se denota IF como la instrucción de selección:

if $\mathtt{B_0} \rightarrow$
 | $\mathtt{S_0}$
[ ] $\mathtt{B_1} \rightarrow$
 | $\mathtt{S_1}$
 | $\vdots$
[ ] $\mathtt{B_n} \rightarrow$
 | $\mathtt{S_n}$
fi

y la abreviación DO como la instrucción de iteración con múltiples guardias:

do $\mathtt{B_0} \rightarrow$
 | $\mathtt{S_0}$
[ ] $\mathtt{B_1} \rightarrow$
 | $\mathtt{S_1}$
 | $\vdots$
[ ] $\mathtt{B_n} \rightarrow$
 | $\mathtt{S_n}$
od

Se denotará como Do a la instrucción de iteración con una sola guardia **do** $B_0 \rightarrow S_0$ **od**.

Se denota SKIP a la instrucción que no cambia el estado de ejecución, se denota $\overline{y}$ y $\overline{Exp}$ como listas de variables y expresiones respectivamente, y $[\overline{y} := \overline{Exp}]$ como el operador que sustituye en paralelo, las variables $\overline{y}$ por las expresiones $\overline{Exp}$ en predicados.

Para hacer derivaciones lógicas sobre tripletas de Hoare, se definen para el lenguaje *GCL*, las siguientes reglas de inferencia, en donde $B_0, \ldots, B_n$ son expresiones booleanas del lenguaje *GCL* y *DG* es una abreviación de $\mathrm{domain}(B_0, \ldots, B_n)$:

$$\{A\}\mathrm{SKIP}\{A\}$$

$$\{\mathrm{domain}(\overline{\mathrm{Exp}}) \wedge B[\overline{y} := \overline{\mathrm{Exp}}]\}\overline{y} := \overline{\mathrm{Exp}}\{B\}$$

$$\frac{\{A\}S_0\{B\} \qquad \{B\}S_1\{C\}}{\{A\}S_0; S_1\{C\}}$$

$$\frac{\begin{array}{ccc} A \Rightarrow DG & \{A \wedge B_0\} & \ldots \{A \wedge B_n\} \\ \wedge (B_0 \vee \cdots \vee B_n) & S_0 & S_n \\ & \{B\} \ldots & \{B\} \end{array}}{\{A\} \text{ if } B_0 \rightarrow S_0[\,]\cdots[\,]B_n \rightarrow S_n \text{ fi } \{B\}}$$

$$\frac{A \Rightarrow A' \qquad \{A'\}S_0\{B'\} \qquad B' \Rightarrow B}{\{A\}S_0\{B\}}$$

$$\frac{\begin{array}{cc} & \{\mathrm{Inv} \wedge B_0 \wedge 0 \leq Exp = Z\} \\ \mathrm{Inv} \Rightarrow \mathrm{domain}(B_0) & S_0 \\ & \{\mathrm{Inv} \wedge 0 \leq Exp < Z\} \end{array}}{\{\mathrm{Inv}\} \text{ do } B_0 \rightarrow S_0 \text{ od } \{\mathrm{Inv} \wedge \neg B_0\}}$$

El predicado Inv de la última regla de inferencia se conoce como invariante, y éste se usa para verificar la correctitud de una Tripleta de Hoare que involucre un ciclo.

Dijkstra adicionalmente en [3] definió el transformador de predicados wp para el lenguaje *GCL*, que es una función que recibe una instrucción y una postcondición sintácticamente hablando y devuelve la precondición más débil de la instrucción y la postcondición. Si BB es una abreviación del predicado $B_0 \vee \cdots \vee B_n$, entonces las reglas que definen wp son las siguientes:

- $\mathrm{wp}(\mathrm{SKIP}, \mathrm{Post}) := \mathrm{Post}$
- $\mathrm{wp}(y_{i_1}, \ldots, y_{i_k} := \mathrm{Exp}_1, \ldots, \mathrm{Exp}_k, \mathrm{Post}) :=$
  $\mathrm{domain}(\mathrm{Exp}_1, \ldots, \mathrm{Exp}_k) \wedge$
  $\mathrm{Post}[y_{i_1}, \ldots, y_{i_k} := \mathrm{Exp}_1, \ldots, \mathrm{Exp}_k]$
- $\mathrm{wp}(S_0; S_1, \mathrm{Post}) := \mathrm{wp}(S_0, \mathrm{wp}(S_1, \mathrm{Post}))$
- $\mathrm{wp}(\mathrm{IF}, \mathrm{Post}) := \mathrm{domain}(B_0, \ldots, B_n) \wedge$
  $(B_0 \vee \cdots \vee B_n) \wedge (B_0 \Rightarrow \mathrm{wp}(S_0, \mathrm{Post})) \wedge \ldots$
  $\wedge (B_n \Rightarrow \mathrm{wp}(S_n, \mathrm{Post}))$
- $\mathrm{wp}(DO, \mathrm{Post}) := (\exists k | k \geq 0 : H_k(\mathrm{Post}))$ en donde $H_k(\mathrm{Post})$ es un predicado que satisface las ecuaciones

$$H_0(\mathrm{Post}) \equiv \mathrm{domain}(BB) \wedge \neg(BB) \wedge \mathrm{Post}$$
$$H_k(\mathrm{Post}) \equiv H_0(\mathrm{Post}) \vee \mathrm{wp}(\mathrm{IF}, H_{k-1}(\mathrm{Post}))$$

  para $k \geq 1$

El predicado $H_k(\mathrm{Post})$ en la definición de $\mathrm{wp}(DO, \mathrm{Post})$ anterior, describe el conjunto de estados que hacen que el ciclo DO itere a lo sumo k veces satisfaciendo la postcondición Post al final de la ejecución.

Trabajar con $H_k(\mathrm{Post})$ como precondición de un ciclo, tiene la ventaja de que la expresión k determina el número máximo de iteraciones que el ciclo puede realizar, por lo tanto se puede estimar la complejidad en tiempo de ejecución. Por otro lado intentar usar $H_k(\mathrm{Post})$ en lugar de cualquier otro invariante puede ser conveniente ya que en los trabajos [6][1] se ha venido planteando, que en algunos casos es muy fácil calcular $H_k(\mathrm{Post})$ formalmente, sin necesidad de conjeturar un invariante y usar las reglas de Hoare para validar.

Por otro lado Morgan en [7], muestra una forma de abstraer partes del código de un algoritmo y aún así, seguir calculando la precondición más débil del algoritmo dado una postcondición. Estas abstracciones de código reciben el nombre de "instrucción de especificación", ya que son manejadas dentro del algoritmo como si fueran instrucciones, aunque su código aún no se conoce. Por medio de instrucciones de especificación es que en este trabajo se hacen las abstracciones necesarias,

para describir la familia de algoritmos de punto fijo, que es objeto de estudio.

A. Contribución

Dada la existencia de una sucesión de conjuntos $\{R_i\}_{i=a}^{n}$, se define una familia de algoritmos de punto fijo, que computan el punto fijo de la sucesión de conjuntos, la familia se describe por medio de instrucciones de especificación de Morgan [7]. Dada una postcondición que indica que en cierta variable R, se encuentra almacenado el punto fijo de la sucesión (Es decir que $R = R_i = R_{i-1}$), entonces se demuestra que todos estos algoritmos con la postcondición descrita anteriormente, tienen como precondición más débil a $(\exists \epsilon | a \leq \epsilon \leq n-1 : R_\epsilon = R_{\epsilon+1})$. Esta demostración se hace conjeturando un predicado y demostrando que ese predicado es un $H_k(Post)$.

Adicionalmente se muestra que la técnica para conjeturar $H_k(Post)$ definida en [1] aplica para este caso, mostrando cómo fue el procedimiento para conjeturar el predicado que se usó para calcular la precondición más débil.

Durante los cálculos del transformador de predicados wp, se usan las propiedades algebraicas descritas en [2], mostrando cómo el uso de estas propiedades facilita el proceso, dando así una pequeña contribución, en aras de construir un cálculo práctico para precondiciones más débiles.

La familia de algoritmos definida, junto con sus abstracciones de código y precondición más débil representan un framework para construir algoritmos de punto fijo correctos. Como ejemplo del uso de este framework, se instancia la familia en el algoritmo de búsqueda de símbolos anulables, para hacer limpieza de gramáticas libre de contexto. Esto se hace refinando las instrucciones de especificación y demostrando que la precondición más débil del algoritmo para este caso es una tautología.

B. Trabajos Relacionados

Originalmente Dijkstra en [3] definió el transformador de predicados wp con las reglas recursivas anteriores, salvo que no usó el predicado domain. Posteriormente en [5], buscando que las aserciones de *GCL* sean totalmente evaluables, es decir que no existan estados donde el valor de verdad de la aserciones sea indefinido, se definió por primera vez el predicado domain, pero solo en la definición de wp de la asignación. Luego en trabajos recientes como [8][9], se usa domain también en la regla de definición de wp para el IF. Más adelante n [10] se hace una demostración basada en semántica denotacional, que justifica que para que las aserciones de *GCL* sean totalmente evaluables, se debe usar domain también en la regla del wp del DO al definir $H_0(\mathrm{Post})$. De modo que la definición de wp que se usa y se presentó en este trabajo es la de [10].

Morgan [7] define el concepto de refinamiento, instrucción de especificación y cómo calcular wp de esta instrucción bajo ciertas condiciones sintácticas sobre las variables de especificación de las aserciones. Posteriormente en [11] se revisita la definición de la instrucción de especificación desde un punto de vista denotacional, interpretándola como una relación de estados del programa y con eso se estableció una fórmula general para calcular wp de estas instrucciones. Otro trabajo donde se estudia a la instrucción de especificación y el concepto de refinamiento como relaciones de estados del programa, es [12].

En [13] se muestra una técnica semántica llamada relaciones invariantes, para calcular invariantes en el lenguaje de la teoría de relaciones y wp de un ciclo. Análogamente, pero con técnicas sintácticas, en [6] y posteriormente en [1], se muestran teoremas que sirven para calcular $H_k(\mathrm{Post})$ explícitamente y por lo tanto para calcular wp de Do con postcondición Post. Posteriormente en [2], se demuestra propiedades algebraicas del transformador sintáctico wp usando semántica denotacional, estas propiedades pueden facilitar los cálculos de wp. Estas propiedades fueron usadas para demostrar los teoremas de [1], pero hasta el momento no existen trabajos que las usen para hacer cálculos en concreto y mostrando en qué medida representan un aporte a la praxis de corrección. Este trabajo es el primero en que estas propiedades son usadas para realizar cálculos concretos.

II. Preliminares

En esta sección enunciamos las definiciones y teoremas más recientes, según la bibliografía de este trabajo, relacionados al cálculo de wp y de $H_k(Post)$ de un ciclo. Las demostraciones ausentes de los teoremas de esta sección se pueden conseguir en [1][2].

Lema 1. $wp(S, P \wedge Q) \equiv wp(S, P) \wedge wp(S, Q)$

Notación. *Para abreviar* $\mathrm{wp}(S, \mathrm{true})$ *se usará la notación* $\mathrm{support}(S)$*, donde* S *es una instrucción.*

Así tenemos que $\mathrm{support}(S)$ define el conjunto de estados iniciales más grande en el cual la intrucción S no aborta. En el siguiente lema se muestra un caso, en donde es útil usar $\mathrm{support}$ para calcular wp.

Lema 2. *Sea* P *un predicado y* S *una instrucción que no modifica los valores de las variables de* P*, entonces:*

$$\mathrm{wp}(S, P) \equiv \mathrm{support}(S) \wedge P.$$

Lema 3. *Sea* S *una instrucción, entonces:*

$$\mathrm{support}(S; i := i + 1) \equiv \mathrm{support}(S)$$

Lema 4. *Sea* S *una instrucción (determinística o no) tal que se comporta de forma determinística sobre las variables del predicado* $P \vee Q$*, entonces:*

$$wp(S, P \vee Q) \equiv wp(S, P) \vee wp(S, Q)$$

Lema 5. $wp(S, P) \Rightarrow support(S)$

Lema 6. *Sean* P *y* Q *predicados y* S *una instrucción que no modifica los valores de las variables de* P*, entonces:*

$$wp(S, P \wedge Q) \equiv P \wedge wp(S, Q)$$

y:

$$wp(S, P \vee Q) \equiv support(S) \wedge (P \vee wp(S, Q))$$

Lema 7. *Sean P y Q predicados y S una instrucción que se comporta deterministicamente sobre los valores de las variables de P, entonces:*

$$wp(S, P \Rightarrow Q) \equiv support(S) \wedge (wp(S, P) \Rightarrow wp(S, Q))$$

Lema 8. *Sea P un predicado, S una instrucción que se comporta deterministicamente sobre los valores de las variables de P, y ϵ una variable no declarada en el programa, entonces:*

$$wp(S, (\exists \epsilon | : P)) \equiv (\exists \epsilon | : wp(S, P))$$

Definición 1. *Sea K una expresión y Do una instrucción de la forma:*

do B $\rightarrow$
| S
od
{Post}

Sean además k', ϵ, ϵ' variables no declaradas en el programa (es decir no ocurren en Do) y no ocurren en Post*, donde S es una instrucción (determinística o no determinística). Se definen:*

1. *El predicado $domBG$ como un predicado que satisface:*
   - *En $domBG$ ocurren sólo ϵ' y las variables del programa*
   - $domBG[\epsilon' := 0] \equiv \text{domain}(B)$
   - $domBG \equiv \text{wp}(S, domBG[\epsilon' := \epsilon' - 1])$ *suponiendo que* $0 < \epsilon' \leq K \wedge \text{domain}(B) \wedge B \wedge \text{support}(S)$.
2. *El predicado NBG como un predicado que satisface:*
   - *En NBG ocurren sólo ϵ y las variables del programa*
   - $NBG[\epsilon := 0] \equiv \neg B$
   - $NBG \equiv \text{wp}(S, NBG[\epsilon := \epsilon - 1])$ *suponiedo que* $0 < \epsilon \leq K \wedge \text{domain}(B) \wedge B \wedge \text{support}(S)$.
3. *El predicado $T_{k'}$ como:*

$$(\exists \epsilon | 0 \leq \epsilon \leq k' : (\forall \epsilon' | 0 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG).$$

4. *El predicado $TI_{k'}$ como: $T_{k'} \wedge B$*

En el siguiente teorema se establecen las condiciones suficientes para determinar si un predicado es un $H_{k'}(\text{Post})$.

Teorema 1. *Sean K una expresión, Do una instrucción como en la definición 1 y k', ϵ, ϵ' variables como en la Definición 1. Entonces, si S actúa de forma determinística sobre las variables de $domBG$ y NBG, se satisface lo siguiente:*
Si existe un predicado inv *tal que:*

1. $\text{domain}(B) \wedge \neg B \wedge \text{Post} \equiv \text{domain}(B) \wedge \neg B \wedge \text{inv}$.
2. $(\forall k' | 1 \leq k' \leq K : TI_{k'} \Rightarrow (\text{wp}(S, \text{inv}) \equiv \text{inv}))$

Entonces:

$$H_{k'}(\text{Post}) \equiv T_{k'} \wedge \text{inv}$$

para todo k' tal que $0 \leq k' \leq K$.

El predicado en 1 del teorema anterior se le llamará *Obligación de Prueba de Terminación* y al predicado en 2, *Obligación de Prueba de Iteración*.

Corolario 1. *Si un predicado inv cumple con la obligación de prueba de terminación y de iteración usando k' sin ninguna cota superior K que lo restrinja, entonces definiendo T_∞ como:*

$$(\exists \epsilon | 0 \leq \epsilon : (\forall \epsilon' | 0 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG)$$

se tiene que:

$$wp(Do, Post) \equiv T_\infty \wedge inv.$$

Por otro lado si $H_{k'}(\text{Post})$ es de la forma $T_{k'} \wedge inv$ para $k' \leq K + 1$ y $H_{K+1}(\text{Post}) \equiv H_K(\text{Post})$, entonces:

$$wp(Do, Post) \equiv H_K(\text{Post})$$

Teorema 2. *Si B es la guardia del ciclo Do en la definición 1 y $\text{domain}(B) \equiv$ true, entonces $domBG \equiv$ true.*

Proof: Se demuestra que definiendo $domBG$ como true, se satisfacen las reglas recursivas de la definición 1.

$domBG[\epsilon' := 0] \equiv \text{true}[\epsilon' := 0] \equiv \text{true} \equiv \text{domain}(B)$

Se supone que $0 < \epsilon' \leq K \wedge \text{domain}(B) \wedge B \wedge \text{support}(S)$ y se demuestra lo siguiente:

$domBG \equiv wp(S, domBG[\epsilon' := \epsilon' - 1]) \equiv wp(S, \text{true}[\epsilon' := \epsilon' - 1]) \equiv wp(S, \text{true}) \equiv \cancelto{\text{true}}{\text{support}(S)}$ ■

Teorema 3. *Sean $K_1, \ldots, K_n$ expresiones en donde solo ocurren variables del programa que no son modificadas por S y tales que $0 \leq K_1 < \ldots < K_n < K$. Sean $P_0, \ldots, P_n$ predicados en donde solo ocurren ϵ y variables del programa en las que S actúa de forma determinística. Suponiendo que $\epsilon \leq K$, $\text{domain}(B)$, B, $\text{support}(S)$, $P_0[\epsilon := 0] \equiv \neg B$, $wp(S, P_{i-1}[\epsilon := K_i]) \equiv P_i$ y $wp(S, P_i[\epsilon := \epsilon - 1]) \equiv P_i$ si $i = 0 \wedge 1 \leq \epsilon \leq K_1$ o $i \neq 0 \wedge i \neq n \wedge K_i + 1 < \epsilon \leq K_{i+1}$ o $i = n \wedge K_n + 1 < \epsilon$, entonces:*

$$NBG$$

$$\equiv$$

$$(0 \leq \epsilon \leq K_1 \wedge P_0) \vee .. \vee (K_i < \epsilon \leq K_{i+1} \wedge P_i) \vee .. \vee (K_n < \epsilon \wedge P_n)$$

para $0 \leq \epsilon \leq K$

Proof: En la fórmula NBG del enunciado solo ocurre ϵ y las variables del programa. A continuación se demuestra que NBG satisface las condiciones de la definición 1.

$NBG[\epsilon := 0]$
$\equiv$
$\cancelto{true}{(0 \leq 0 \leq K_1 \wedge P_0[\epsilon := 0])} \vee .. \cancelto{false}{(K_i \leq 0 \leq K_{i+1} \wedge P_i[\epsilon := 0])}$
$\vee \cdots \vee \cancelto{false}{(K_n < 0 \wedge P_n[\epsilon := 0])}$
$\equiv$
$P_0[\epsilon := 0]$
$\equiv$
$\neg B$

Para $\epsilon > 0$ se tiene:

$wp(S, NBG[\epsilon := \epsilon - 1])$
$\equiv$
$wp(S, (0 \leq \epsilon - 1 \leq K_1 \wedge P_0[\epsilon := \epsilon - 1])$
$\vee (\bigvee i | 1 \leq i < n : K_i < \epsilon - 1 \leq K_{i+1} \wedge P_i[\epsilon := \epsilon - 1])$
$\vee (K_n < \epsilon - 1 \wedge P_n[\epsilon := \epsilon - 1]))$

$$
\begin{array}{l}
\equiv \\
wp(S, (1 \leq \epsilon \leq K_1 + 1 \wedge P_0[\epsilon := \epsilon - 1]) \\
\vee(\bigvee i | 1 \leq i < n : K_i + 1 < \epsilon \leq K_{i+1} + 1 \wedge P_i[\epsilon := \epsilon - 1]) \\
\vee(K_n + 1 < \epsilon \wedge P_n[\epsilon := \epsilon - 1])) \\
\equiv \\
wp(S, (1 \leq \epsilon \leq K_1 \wedge P_0[\epsilon := \epsilon - 1]) \\
\vee(\epsilon = K_1 + 1 \wedge P_0[\epsilon := K_1]) \\
\vee(\bigvee i | 1 \leq i < n : (K_i + 1 < \epsilon \leq K_{i+1} \wedge P_i[\epsilon := \epsilon - 1]) \\
\vee(\epsilon = K_{i+1} + 1 \wedge P_i[\epsilon := K_{i+1}])) \\
\vee(K_n + 1 < \epsilon \wedge P_n[\epsilon := \epsilon - 1])) \\
\equiv \\
wp(S, (1 \leq \epsilon \leq K_1 \wedge P_0[\epsilon := \epsilon - 1]) \\
\vee(\epsilon = K_1 + 1 \wedge P_0[\epsilon := K_1]) \\
\vee(\bigvee i | 1 \leq i < n : K_i + 1 < \epsilon \leq K_{i+1} \wedge P_i[\epsilon := \epsilon - 1]) \\
\vee(\bigvee i | 1 \leq i < n : \epsilon = K_{i+1} + 1 \wedge P_i[\epsilon := K_{i+1}]) \\
\vee(K_n + 1 < \epsilon \wedge P_n[\epsilon := \epsilon - 1])) \\
\equiv \quad \langle \text{ Lema 4 y 6 e hipótesis } \mathrm{support}(S) \rangle \\
(1 \leq \epsilon \leq K_1 \wedge wp(S, P_0[\epsilon := \epsilon - 1])) \\
\vee(\epsilon = K_1 + 1 \wedge wp(S, P_0[\epsilon := K_1])) \\
\vee(\bigvee i | 1 \leq i < n : K_i + 1 < \epsilon \leq K_{i+1} \wedge \\
\qquad wp(S, P_i[\epsilon := \epsilon - 1])) \\
\vee(\bigvee i | 1 \leq i < n : \epsilon = K_{i+1} + 1 \wedge wp(S, P_i[\epsilon := K_{i+1}])) \\
\vee(K_n + 1 < \epsilon \wedge wp(S, P_n[\epsilon := \epsilon - 1])) \\
\equiv \\
(1 \leq \epsilon \leq K_1 \wedge P_0) \vee (\epsilon = K_1 + 1 \wedge P_1) \\
\vee(\bigvee i | 1 \leq i < n : K_i + 1 < \epsilon \leq K_{i+1} \wedge P_i) \\
\vee(\bigvee i | 1 \leq i < n : \epsilon = K_{i+1} + 1 \wedge P_{i+1}) \\
\vee(K_n + 1 < \epsilon \wedge P_n) \\
\equiv \quad \left\langle \begin{array}{c} \text{Si } n > 1 \text{ se usa separación de rango,} \\ \text{si } n = 1 \text{ se usa idenpotencia del} \vee \text{con } K_2 = \infty \\ \text{y rango vacío} \end{array} \right\rangle \\
(1 \leq \epsilon \leq K_1 \wedge P_0) \\
\vee(\epsilon = K_1 + 1 \wedge P_1) \vee (K_1 + 1 < \epsilon \leq K_2 \wedge P_1) \\
\vee(\bigvee i | 2 \leq i < n : K_i + 1 < \epsilon \leq K_{i+1} \wedge P_i) \\
\vee(\bigvee i | 1 \leq i < n - 1 : \epsilon = K_{i+1} + 1 \wedge P_{i+1}) \\
\vee(\epsilon = K_n + 1 \wedge P_n) \vee (K_n + 1 < \epsilon \wedge P_n) \\
\equiv \\
(1 \leq \epsilon \leq K_1 \wedge P_0) \\
\vee((\epsilon = K_1 + 1 \vee K_1 + 1 < \epsilon \leq K_2) \wedge P_1) \\
\vee(\bigvee i | 2 \leq i < n : K_i + 1 < \epsilon \leq K_{i+1} \wedge P_i) \\
\vee(\bigvee i | 1 \leq i < n - 1 : \epsilon = K_{i+1} + 1 \wedge P_{i+1}) \\
\vee((\epsilon = K_n + 1 \vee K_n + 1 < \epsilon) \wedge P_n) \\
\equiv \\
(1 \leq \epsilon \leq K_1 \wedge P_0) \\
\vee(K_1 + 1 \leq \epsilon \leq K_2 \wedge P_1) \\
\vee(\bigvee i | 2 \leq i < n : K_i + 1 < \epsilon \leq K_{i+1} \wedge P_i) \\
\vee(\bigvee i | 2 \leq i < n : \epsilon = K_i + 1 \wedge P_i) \\
\vee(K_n + 1 \leq \epsilon \wedge P_n) \\
\equiv \\
(1 \leq \epsilon \leq K_1 \wedge P_0) \\
\vee(K_1 + 1 \leq \epsilon \leq K_2 \wedge P_1) \\
\vee(\bigvee i | 2 \leq i < n : (\epsilon = K_i + 1 \wedge P_i) \vee \\
\qquad (K_i + 1 < \epsilon \leq K_{i+1} \wedge P_i)) \\
\vee(K_n + 1 \leq \epsilon \wedge P_n) \\
\equiv \\
(1 \leq \epsilon \leq K_1 \wedge P_0) \\
\vee(K_1 < \epsilon \leq K_2 \wedge P_1) \\
\vee(\bigvee i | 2 \leq i < n : K_i < \epsilon \leq K_{i+1} \wedge P_i) \\
\vee(K_n < \epsilon \wedge P_n) \\
\equiv \\
((\cancel{false} \vee 1 \overset{\epsilon := 0}{\leq} \epsilon \leq K_1) \wedge P_0) \\
\vee(\bigvee i | 1 \leq i < n : K_i < \epsilon \leq K_{i+1} \wedge P_i) \\
\vee(K_n < \epsilon \wedge P_n) \\
\equiv \\
(0 \leq \epsilon \leq K_1 \wedge P_0) \vee \cdots \vee (K_n < \epsilon \wedge P_n) \\
\equiv \\
NBG
\end{array}
$$

■

III. Instrucción de Especificación

Según [10], dado un algoritmo en que se declaran identificadores de tipos $T_1, \cdots, T_n$ (en ese orden), se considera un estado de la ejecución, a un elemento en el conjunto $\{abort\} \cup \prod_{i=1}^{n} T_i$, en donde $abort$ es un elemento especial para indicar una ejecución abortada. Adicionalmente una instrucción (determinística o no) se considera como un conjunto de pares ordenados de estados de ejecución, los cuales representan entradas y salidas de una ejecución de la instrucción.

Con los conceptos del párrafo anterior se define lo siguiente:

Definición 2 (Instrucción de Especificación)**.** *Sea A un algoritmo en donde $\overline{x}$ y $\overline{y}$ son listas de identificadores de tipos $\overline{T}$ y $\overline{T'}$ respectivamente, tales que $\overline{x}, \overline{y}$ es la lista de todos los identificadores declarados en A, y aquellos identificadores que se declararon constantes, ocurren en $\overline{x}$. Si $\overline{Z}$ es una lista de variables, de tipos $\overline{T''}$, no declaradas en A y* Pdef, Qdef *son predicados que dependen de $\overline{x}, \overline{y}, \overline{Z}$, entonces la nomenclatura $\overline{y} : [\mathrm{Pdef}(\overline{x}, \overline{y}, \overline{Z}), \mathrm{Qdef}(\overline{x}, \overline{y}, \overline{Z})]$ se considera como una instrucción, que se define como el siguiente conjunto de pares ordenados de estados de ejecución:*

$$R' \cup (\mathrm{Dom}(R')^c \times \{abort\})$$

donde $\mathrm{Dom}(R')^c = ((\overline{T} \times \overline{T'}) \cup \{abort\}) \setminus \mathrm{Dom}(R')$ *y*

$$
\begin{array}{rl}
R' := & \bigcup_{\overline{Z} \in \overline{T''}} R_{\overline{Z}} \\
R_{\overline{Z}} := & \{\langle (\overline{x}, \overline{y}), (\overline{x'}, \overline{y'}) \rangle \in \mathrm{Dom}_{\overline{Z}} \times \mathrm{Rgo}_{\overline{Z}} \,|\overline{x} = \overline{x'}\} \\
\mathrm{Dom}_{\overline{Z}} := & \{(\overline{x}, \overline{y}) \in \overline{T} \times \overline{T'} |\, \mathrm{Pdef}(\overline{x}, \overline{y}, \overline{Z})\} \\
\mathrm{Rgo}_{\overline{Z}} := & \{(\overline{x}, \overline{y}) \in \overline{T} \times \overline{T'} |\, \mathrm{Qdef}(\overline{x}, \overline{y}, \overline{Z})\}
\end{array}
$$

En otras palabras la instrucción de especificación $\overline{y} : [\mathrm{Pdef}(\overline{x}, \overline{y}, \overline{Z}), \mathrm{Qdef}(\overline{x}, \overline{y}, \overline{Z})]$, se considera como una abstracción de código. El código que se abstrae corresponde a una instrucción, que se comporta de forma que, cada estado de ejecución inicial que satisface $\mathrm{Pdef}(\overline{x}, \overline{y}, \overline{Z})$ (para valores fijos de $\overline{Z}$), es transformado a un estado que satisface $\mathrm{Qdef}(\overline{x}, \overline{y}, \overline{Z})$ (para los mismos valores de $\overline{Z}$), pero con la restricción de que los valores de los identificadores $\overline{x}$ no cambian en la transformación.

Se puede calcular wp de una instrucción de especificación dado una postcondición de forma general, pero para este trabajo nos interesa solo un caso particular, que es cuando en los predicados no hay variables de especificación $\overline{Z}$. Para este caso se tiene el siguiente teorema de Caroll Morgan [7]:

Teorema 4. *Sea un algoritmo donde $\overline{x}, \overline{y}$ es la lista de identificadores declarados en donde aquellos declarados como constantes ocurren en la lista $\overline{x}$. Si* $\mathrm{Pdef}(\overline{x}, \overline{y})$, $\mathrm{Qdef}(\overline{x}, \overline{y})$ *y* $\mathrm{Post}(\overline{x}, \overline{y})$ *son predicados sin variables de especificación (sólo ocurren los identificadores declarados en el programa), tales que para todo $\overline{x}, \overline{y}$ en el que se satisface* $\mathrm{Pdef}(\overline{x}, \overline{y})$, *existe $\overline{y'}$ tal que* $\mathrm{Qdef}(\overline{x}, \overline{y'})$, *entonces, una precondición más débil para la instrucción de especificación $\overline{y} : [\mathrm{Pdef}(\overline{x}, \overline{y}), \mathrm{Qdef}(\overline{x}, \overline{y})]$ y* $\mathrm{Post}(\overline{x}, \overline{y})$ *es:*

$$\mathrm{Pdef}(\overline{x}, \overline{y}) \wedge (\forall \overline{y} |\, \mathrm{Qdef}(\overline{x}, \overline{y}) : \mathrm{Post}(\overline{x}, \overline{y}))$$

IV. Esquema de Algoritmos de Punto Fijo

Dado una familia de conjuntos $\{R_i\}_{i=a}^{n}$ (donde $n \geq 0$ y puede ser $n = \infty$) se define el esquema de algoritmos de punto fijo a la familia de algoritmos resultantes de refinar la instrucción de especificación del siguiente algoritmo:

```
|[
Var i, Prev, R : Int, Set, Set;

i, R := a, R_a;
do
  | Prev := R;
  | R : [a ≤ i < n ∧ Prev = R_i, R = R_{i+1}];
  | i := i + 1
while Prev ≠ R →
]|
```

Observación 1. *Las instrucciones:*

$$R : [a \leq i < n \wedge Prev = R_i, R = R_{i+1}]$$

y:

$$R : [a \leq i < n \wedge Prev = R_i, a \leq i < n \wedge R = R_{i+1}]$$

son las mismas ya que las variables a, i y n no son modificadas por dicha instrucción. Siguiendo la filosofía de que no deben existir estados en que las aserciones queden indeterminadas, se debería usar la segunda versión de la instrucción, pero al ser ambas iguales, se escogió la primera porque facilita los cálculos.

Usando los teoremas anteriores se calculará la precondición más débil del algoritmo anterior para la postcondición $\mathrm{Post} : a < i \leq n \wedge R = R_i = R_{i-1}$, para esto se descompone el do-while en una instrucción de iteración Do usual de la siguiente forma.

```
|[
Var i, Prev, R : Int, Set, Set;

i, R := a, R_a;
Prev := R;
R : [a ≤ i < n ∧ Prev = R_i, R = R_{i+1}];
i := i + 1;
do Prev ≠ R →
  | Prev := R;
  | R : [a ≤ i < n ∧ Prev = R_i, R = R_{i+1}];
  | i := i + 1
od
]|
```

De ahora en adelante se abreviará con S, a las tres instrucciones dentro del bloque del Do anterior.

V. Cálculo de $domBG$ y $\mathrm{support}(S)$

Como $\mathrm{domain}(B) \equiv \mathrm{domain}(Prev \neq R) \equiv$ true, entonces por el teorema 2 se tiene que:

$$domBG \equiv \text{true}.$$

Por otro lado para calcular $\mathrm{support}(S)$ se calcula $wp(S, \text{true})$ como a continuación.

$$wp\left(\begin{array}{c} Prev := R; \\ R : [a \leq i < n \wedge Prev = R_i, R = R_{i+1}]; \quad , \quad \text{true} \\ i := i + 1 \end{array}\right)$$

$\equiv$ ⟨definición de wp⟩

$$wp\left(\begin{array}{c} Prev := R; \\ R : \left[\begin{array}{c} a \leq i < n \\ \wedge \\ Prev = R_i \end{array}\right. , \left. R = R_{i+1} \right], wp(i := i + 1, \text{true}) \end{array}\right)$$

$\equiv$ ⟨definición de wp⟩

$$wp(Prev := R, wp(R : \left[\begin{array}{c} a \leq i < n \\ \wedge \\ Prev = R_i \end{array}\right. , \left. R = R_{i+1} \right], \text{true}))$$

$\equiv$ ⟨teorema 4⟩

$$wp(Prev := R, \begin{array}{c} a \leq i < n \wedge Prev = R_i \wedge \\ (\forall R | R = R_{i+1} : \text{true}) \end{array}^{true})$$

$\equiv$

$$a \leq i < n \wedge R = R_i$$

Con lo que:

$$\mathrm{support}(S) \equiv a \leq i < n \wedge R = R_i$$

VI. Cálculo de NBG

Para calcular NBG se estudian diferentes casos para diferentes valores de ϵ.

Para $\epsilon = 0$

$NBG \equiv Prev = R$ por definición

Para $\epsilon \geq 1$

Para calcular NBG se usará el Teorema 3, para esto se debe suponer como hipótesis $0 < \epsilon \leq K \wedge \mathrm{domain}(B) \wedge B \wedge \mathrm{support}(S)$. El valor de K se determinará más adelante. A continuación se hacen cálculos de $wp(S, .)$ para determinar una fórmula NBG para diferentes valores de ϵ fijos.

Para $\epsilon = 1$

$$wp\left(\begin{array}{c} Prev := R; \\ R : \left[\begin{array}{c} a \leq i < n \\ \wedge \\ Prev = R_i \end{array}\right. , \left. R = R_{i+1} \right]; \quad , \quad Prev = R \\ i := i + 1 \end{array}\right)$$

$\equiv$ ⟨definición de wp⟩

$$wp\left(\begin{array}{c} Prev := R; \\ R : \left[\begin{array}{c} a \leq i < n \\ \wedge \\ Prev = R_i \end{array}\right. , \left. R = R_{i+1} \right], wp(i := i + 1, \begin{array}{c} Prev \\ = \\ R \end{array}) \end{array}\right)$$

$\equiv$ ⟨definición de wp⟩

$$wp(Prev{:=}R, wp(R{:}\begin{bmatrix} a \le i < n \\ \wedge \\ Prev = R_i \end{bmatrix}, \begin{matrix} Prev \\ = \\ R \end{matrix}))$$
$\equiv$ ⟨teorema 4⟩
$$wp\left(Prev := R, \begin{matrix} a \le i < n \wedge Prev = R_i \wedge \\ (\forall R | R = R_{i+1} : Prev = R) \end{matrix}\right)$$
$\equiv$ ⟨regla de un punto⟩
$$wp(Prev := R, a \le i < n \wedge Prev = R_i \wedge Prev = R_{i+1})$$
$\equiv$ ⟨definición de wp⟩
$$a \le i < n \wedge R = R_i \wedge R = R_{i+1}$$
$\equiv$ ⟨hipótesis $\mathrm{support}(S)$⟩
$$a \le i < n \wedge R = R_i \wedge R_i = R_{i+1} \quad \text{true}$$
$\equiv$ ⟨$\epsilon = 1$⟩
$$R_{i+\epsilon-1} = R_{i+\epsilon}$$

Como n es una cota superior para los índices de la familia de conjuntos $\{R_i\}_{i=a}^{n}$, entonces $i + \epsilon \le n$, con lo que $\epsilon \le n - i$ (inecuación válida incluso si $n = \infty$), con lo que se deduce que el valor de K de la definición 1 es $n - i$ y por lo tanto hay que suponer en todo momento que $0 < \epsilon \le n - i$.

A continuación se demuestra que para todo $\epsilon > 1$, se cumple que $wp(S, (R_{i+\epsilon-1} = R_{i+\epsilon})[\epsilon := \epsilon - 1]) \equiv R_{i+\epsilon-1} = R_{i+\epsilon}$

Para $\epsilon > 1$

$$wp\left(\begin{matrix} Prev := R; \\ R : \begin{bmatrix} a \le i < n \\ \wedge \\ Prev = R_i \end{bmatrix}; \ , R_{i+\epsilon-2} = R_{i+\epsilon-1} \\ i := i + 1 \end{matrix}\right)$$
$\equiv$ ⟨definición de wp⟩
$$wp\left(\begin{matrix} Prev := R; \\ R : \begin{bmatrix} a \le i < n \\ \wedge \\ Prev = R_i \end{bmatrix}, wp(\begin{matrix} i & R_{i+\epsilon-2} \\ := & = \\ i+1 & R_{i+\epsilon-1} \end{matrix}) \end{matrix}\right)$$
$\equiv$ ⟨definición de wp⟩
$$wp(Prev := R; R : \begin{bmatrix} a \le i < n \\ \wedge \\ Prev = R_i \end{bmatrix}, \begin{matrix} R_{i+\epsilon-1} \\ = \\ R_{i+\epsilon} \end{matrix})$$
$\equiv$ ⟨Lema 2 y 3 e hipótesis $\mathrm{support}(S)$⟩
$$a \le i < n \wedge R = R_i \wedge R_{i+\epsilon-1} = R_{i+\epsilon} \quad \text{true}$$

Según el Teorema 3, tomando $n = 1$, $K_1 = K_n = 0$, P_0 y P_1 como $Prev = R$ y $R_{i+\epsilon-1} = R_{i+\epsilon}$ respectivamente, se tiene que:

$$NBG \equiv (\epsilon = 0 \wedge Prev = R) \vee (0 < \epsilon \wedge R_{i+\epsilon-1} = R_{i+\epsilon})$$

VII. Condición de Terminación de Algoritmos de Punto Fijo

Según la definición 1 se tiene que:

$$T_{k'} \equiv (\exists \epsilon | 0 \le \epsilon \le k' : (\epsilon = 0 \wedge Prev = R) \vee (0 < \epsilon \wedge R_{i+\epsilon-1} = R_{i+\epsilon}))$$

Lo cual es equivalente a:

$$Prev = R \vee (\exists \epsilon | 1 \le \epsilon \le k' : R_{i+\epsilon-1} = R_{i+\epsilon})$$

ya que por hipótesis $0 \le k'$.

VIII. Cálculo de $H_{k'}(Post)$

En esta sección será de utilidad demostrar el siguiente lema.

Lema 9. *Sea S las tres instrucciones del bloque interno del Do del algoritmo al final de la Sección IV, entonces:*

1. $wp(S, R = R_i) \equiv \mathrm{support}(S)$
2. *Suponiendo* $\mathrm{support}(S)$ *como hipótesis, se tiene que:*
   - $wp(S, Prev = R) \equiv R_i = R_{i+1}$
   - $wp(S, R_{i+\epsilon-2} = R_{i+\epsilon-1}) \equiv R_{i+\epsilon-1} = R_{i+\epsilon}$ *para cualquier* ϵ

Proof: La demostración de la parte 2 se encuentra en los cálculos de wp de la Sección VI, los cálculos hechos no dependían del valor de ϵ, sino solo de la hipótesis $\mathrm{support}(S)$. La parte 1 se demuestra a continuación.

$$wp\left(\begin{matrix} Prev := R; \\ R : \begin{bmatrix} a \le i < n \\ \wedge \\ Prev = R_i \end{bmatrix}; \quad , \quad R = R_i \\ i := i + 1 \end{matrix}\right)$$
$\equiv$
$$wp\left(\begin{matrix} Prev := R; \\ R : \begin{bmatrix} a \le i < n \\ \wedge \\ Prev = R_i \end{bmatrix}, wp(\begin{matrix} i & R \\ := & = \\ i+1 & R_i \end{matrix}) \end{matrix}\right)$$
$\equiv$
$$wp(Prev{:=}R, wp(R : \begin{bmatrix} a \le i < n \\ \wedge \\ Prev = R_i \end{bmatrix}, \begin{matrix} R \\ = \\ R_{i+1} \end{matrix}))$$
$\equiv$ ⟨teorema 4⟩
$$wp\left(Prev := R, \begin{matrix} a \le i < n \wedge Prev = R_i \wedge \\ (\forall R | R = R_{i+1} : R = R_{i+1}) \end{matrix}\right) \quad \text{true}$$
$\equiv$
$$a \le i < n \wedge R = R_i$$
$\equiv$
$$\mathrm{support}(S)$$
■

Observación 2. *La segunda parte del Lema anterior puede ser usada si dentro de una fórmula, se encuentra en conjunción* $\mathrm{support}(S)$ *junto con* $wp(S, Prev = R)$ *o* $wp(S, R_{i+\epsilon-2} = R_{i+\epsilon-1})$.

A continuación usando el teorema 1, se demostraran las obligaciones de prueba de iteración y de terminación tomando inv como:

$$R = R_i \wedge (Prev = R \Rightarrow a < i \le n \wedge R_i = R_{i-1}) \wedge (Prev \ne R \Rightarrow a \le i < n)$$

Para la obligación de prueba de terminación se tiene:

$$\mathrm{domain}(B) \wedge \neg B \wedge inv$$
$\equiv$
$$\mathrm{domain}(B) \wedge Prev = R \wedge R = R_i \wedge (Prev = R \Rightarrow a < i \le n \wedge R_i = R_{i-1}) \wedge (Prev \ne R \Rightarrow a \le i < n)$$
$\equiv$
$$\mathrm{domain}(B) \wedge Prev = R \wedge R = R_i \wedge a < i \le n \wedge R_i = R_{i-1}$$
$\equiv$
$$\mathrm{domain}(B) \wedge \neg B \wedge a < i \le n \wedge R = R_i = R_{i-1} \quad \text{Post}$$

Para la obligación de prueba de iteración se debe suponer $T_{k'} \wedge Prev \neq R$ con $1 \leq k' \leq n-i$ y hacer lo siguiente:

$wp(S, inv)$
$\equiv$
$wp\left(S, \begin{array}{c}(Prev = R \Rightarrow a < i \leq n \wedge R_i = R_{i-1}) \wedge \\ (Prev \neq R \Rightarrow a \leq i < n) \wedge R = R_i\end{array}\right)$
$\equiv \quad \langle Lema\ 1 \rangle$
$wp(S, Prev = R \Rightarrow a < i \leq n \wedge R_i = R_{i-1}) \wedge$
$wp(S, Prev \neq R \Rightarrow a \leq i < n) \wedge wp(S, R = R_i)$
$\equiv \quad \langle Lema\ 9 \rangle$
$wp(S, Prev = R \Rightarrow a < i \leq n \wedge R_i = R_{i-1}) \wedge$
$wp(S, i < a \vee i \geq n \Rightarrow Prev = R) \wedge \text{support}(S)$
$\equiv \quad \langle Lema\ 7\ y\ 1 \rangle$
$\left(wp(S, Prev = R) \Rightarrow \begin{array}{c} wp(S, a < i \leq n) \wedge \\ wp(S, R_i = R_{i-1})\end{array}\right) \wedge$
$(wp(S, i < a \vee i \geq n) \Rightarrow wp(S, Prev = R)) \wedge \text{support}(S)$
$\equiv \quad \langle Lema\ 9\ y\ \text{support}(S) \Rightarrow a < i+1 \leq n \rangle$
$(R_i = R_{i+1} \Rightarrow a \leq i+1 \leq n \wedge R_{i+1} = R_i) \wedge$ (true)
$(i+1 < a \vee i+1 \geq n \Rightarrow R_i = R_{i+1}) \wedge \text{support}(S)$ ($i+1 < a \vee i+1 \geq n$: false)
$\equiv \quad \left\langle \begin{array}{c} \text{support}(S) \Rightarrow (i+1 \geq n \equiv i = n-1)\ y \\ 1 \leq k' \leq n-i \wedge T_{k'} \wedge Prev \neq R \wedge i = n-1 \\ \Rightarrow \\ R_i = R_{i+1} \end{array}\right\rangle$
$(R_i = R_{i+1} \Rightarrow R_{i+1} = R_i) \wedge (i = n-1 \Rightarrow R_i = R_{i+1}) \wedge$ (true) (true)
$\text{support}(S)$
$\equiv$
$\text{support}(S) \wedge (false \Rightarrow a < i \leq n \wedge R_i = R_{i-1})$
$\equiv \quad \langle hipótesis\ Prev \neq R \rangle$
$a \leq i < n \wedge R = R_i \wedge (Prev = R \Rightarrow a < i \leq n \wedge R_i = R_{i-1})$
$\equiv \quad \langle hipótesis\ Prev \neq R \rangle$
$R = R_i \wedge (Prev = R \Rightarrow a < i \leq n \wedge R_i = R_{i-1})$
$\wedge\ (Prev \neq R \Rightarrow a \leq i < n)$
$\equiv$
inv

Observación 3. *Note que en el cálculo anterior, la sugerencia* $\text{support}(S) \Rightarrow (i+1 \geq n \equiv i = n-1)$ *es cierta si* $n = \infty$*, ya que en este caso* $i+1 \geq n \equiv false$ *y* $i = n-1 \equiv false$*. Por la misma razón es cierto que* $1 \leq k' \leq n-i \wedge T_{k'} \wedge Prev \neq R \wedge i = n-1 \Rightarrow R_i = R_{i+1}$ *cuando* $n = \infty$*.*

Con esto queda demostrado que $T_{k'} \wedge inv$ con $0 \leq k' \leq n-i$ es un $H_{k'}(\text{Post})$.

IX. Cálculo de la Precondición Más Débil del Ciclo

En esta sección se demuestra que $H_{n-i}(\text{Post})$ es la precondición más débil del ciclo. Como se indicó en la Sección VI, si $n = \infty$ entonces $K = \infty$ y por Corolario 1 la precondición más débil del ciclo es $T_\infty \wedge inv$. Pero como $n - i = \infty$, entonces $H_{n-i}(\text{Post}) \equiv T_\infty \wedge inv$.

Para el caso cuando $n \neq \infty$, se muestra que si se define inv' como $T_{n-i} \wedge inv$, entonces inv' satisface la obligación de prueba de terminación y la obligación de prueba de iteración para todo k' sin cota superior K. Haciendo esto, según el Teorema 1, se tiene que $H_{k'}(\text{Post}) \equiv T_{k'} \wedge inv'$ para todo k', lo cual a su vez es equivalente a $T_{k'} \wedge T_{n-i} \wedge inv$. Como $T_{n-i} \Rightarrow T_{k'}$ para todo $k' \geq n-i$, entonces $H_{n-i+1}(\text{Post}) \equiv H_{n-i}(\text{Post})$ y según el Corolario 1, $H_{n-i}(\text{Post})$ sería la precondición más débil del ciclo.

La obligación de prueba de terminación para inv' se demuestra de la siguiente forma:

$\text{domain}(B) \wedge \neg B \wedge inv'$
$\equiv$
$\text{domain}(B) \wedge \neg B \wedge T_{n-i} \wedge inv$
$\equiv \quad \langle \text{absorción} \rangle$
$\text{domain}(B) \wedge \neg B \wedge inv$
$\equiv \quad \langle \text{obligación de prueba de terminación de } inv \rangle$
$\text{domain}(B) \wedge \neg B \wedge \text{Post}$

Se hace notar que:

$$T_{n-i} \equiv Prev = R \vee (\exists \epsilon | 1 \leq \epsilon \leq n-i : R_{i+\epsilon-1} = R_{i+\epsilon})$$

y a su vez, esto último es equivalente a (incluso si $n = \infty$):

$$Prev = R \vee (\exists \epsilon | : i < \epsilon \leq n \wedge R_{\epsilon-1} = R_\epsilon)$$

Con esto último se demuestra la obligación de prueba de iteración para inv' asumiendo $T_{k'} \wedge Prev \neq R$ para $k' \geq 1$ y haciendo lo siguiente:

$wp(S, inv')$
$\equiv$
$wp(S, T_{n-i}(\text{Post}) \wedge inv)$
$\equiv \quad \langle \text{Lema } 1 \rangle$
$wp(S, T_{n-i}(\text{Post})) \wedge wp(S, inv)$
$\equiv \quad \langle \text{Lema } 4,\ 8 \text{ y } 6 \rangle$
$(wp(S, Prev = R)$
$\vee (\exists \epsilon | : wp(S, i < \epsilon \leq n) \wedge R_{\epsilon-1} = R_\epsilon)) \wedge wp(S, inv)$
$\equiv \quad \langle \text{Lema } 5 \rangle$
$(wp(S, Prev = R)$
$\vee (\exists \epsilon | : wp(S, i < \epsilon \leq n) \wedge R_{\epsilon-1} = R_\epsilon)) \wedge wp(S, inv) \wedge$
$\text{support}(S)$
$\equiv \quad \langle \text{Lema } 9,\ 2\ y\ 3 \rangle$
$(R_i = R_{i+1}$
$\vee (\exists \epsilon | i+1 < \epsilon \leq n : R_{\epsilon-1} = R_\epsilon)) \wedge wp(S, inv) \wedge$
$\text{support}(S)$
$\equiv \quad \langle \text{Lema 5 y separación de rango} \rangle$
$(\exists \epsilon | i < \epsilon \leq n : R_{\epsilon-1} = R_\epsilon) \wedge wp(S, inv)$
$\equiv \quad \langle \text{hipótesis } Prev \neq R \rangle$
$(Prev = R \vee (\exists \epsilon | i < \epsilon \leq n : R_{\epsilon-1} = R_\epsilon)) \wedge wp(S, inv)$ (T_{n-i})
$\equiv \quad \langle \text{obligación de prueba de iteración de } inv \rangle$
$T_{n-i} \wedge inv$
$\equiv$
inv'

X. Cálculo de la Precondición Más Débil del Algoritmo

Según lo demostrado hasta el momento el algoritmo planteado tiene las siguientes aserciones.

$|[$

```
Var i, Prev, R : Int, Set, Set;

i, R := a, Ra;
{wp(S, T_{n-i} ∧ inv)}
Prev := R;
R : [a ≤ i < n ∧ Prev = Ri, R = Ri+1];
i := i + 1;
{H_{n-i}(Post) : T_{n-i} ∧ inv}
do Prev ≠ R →
   Prev := R;
   R : [a ≤ i < n ∧ Prev = Ri, R = Ri+1];
   i := i + 1
od
{Post : a < i ≤ n ∧ R = R_i = R_{i-1}}
]|
```

Por lo tanto el próximo paso consiste en calcular $wp(S, T_{n-i} \wedge inv)$

$$
\begin{aligned}
& wp(S, T_{n-i} \wedge inv) \\
\equiv & \quad \left\langle wp(S, inv') \equiv \begin{array}{c} (\exists \epsilon | i < \epsilon \leq n : R_{\epsilon-1} = R_\epsilon) \\ \wedge \\ wp(S, inv) \end{array} \right\rangle \\
& (\exists \epsilon | i < \epsilon \leq n : R_{\epsilon-1} = R_\epsilon) \wedge wp(S, inv) \\
\equiv & \quad \left\langle \begin{array}{c} 5 \text{ primeros pasos del cálculo de } wp(S, inv) \\ \text{de la Sección } VIII \end{array} \right\rangle \\
& (\exists \epsilon | i < \epsilon \leq n : R_{\epsilon-1} = R_\epsilon) \wedge \\
& (i + 1 \geq n \Rightarrow R_i = R_{i+1}) \wedge \text{support}(S) \\
\equiv & \quad \left\langle \begin{array}{c} \text{support}(S) \Rightarrow (i + 1 \geq n \equiv i = n - 1) \\ i = n - 1 \wedge (\exists \epsilon | i < \epsilon \leq n : R_{\epsilon-1} = R_\epsilon) \\ \Rightarrow \\ R_i = R_{i+1} \end{array} \right\rangle \\
& (\exists \epsilon | i < \epsilon \leq n : R_{\epsilon-1} = R_\epsilon) \wedge \\
& (i = n - 1 \Rightarrow \overset{\text{true}}{\cancel{R_i = R_{i+1}}}) \wedge \text{support}(S) \\
\equiv & \\
& (\exists \epsilon | i \leq \epsilon \leq n - 1 : R_\epsilon = R_{\epsilon+1}) \wedge a \leq i \wedge \overset{\text{true}}{\cancel{i < n}} \wedge R = R_i
\end{aligned}
$$

De esta forma se tiene que el predicado anterior, es correcto colocarlo como una aserción en el algoritmo, de la siguiente manera:

```
|[
Var i, Prev, R : Int, Set, Set;

i, R := a, Ra;
{(∃ε | i ≤ ε ≤ n − 1 : R_ε = R_{ε+1}) ∧ a ≤ i ∧ R = R_i}
Prev := R;
R : [a ≤ i < n ∧ Prev = Ri, R = Ri+1];
i := i + 1;
do Prev ≠ R →
   Prev := R;
   R : [a ≤ i < n ∧ Prev = Ri, R = Ri+1];
   i := i + 1
od
{Post : a < i ≤ n ∧ R = R_i = R_{i-1}}
]|
```

Que es equivalente a escribir:

```
|[
Var i, Prev, R : Int, Set, Set;

i, R := a, Ra;
{(∃ε | i ≤ ε ≤ n − 1 : R_ε = R_{ε+1}) ∧ a ≤ i ∧ R = R_i}
do
   Prev := R;
   R : [a ≤ i < n ∧ Prev = Ri, R = Ri+1];
   i := i + 1
while Prev ≠ R →
{Post : a < i ≤ n ∧ R = R_i = R_{i-1}}
]|
```

Calculando la precondición más débil se tiene:

```
|[
Var i, Prev, R : Int, Set, Set;

{Pre : (∃ε | a ≤ ε ≤ n − 1 : R_ε = R_{ε+1}) ∧ a ≤ a ∧ R_a = R_a}   (a ≤ a → true; R_a = R_a → true)
i, R := a, Ra;
do
   Prev := R;
   R : [a ≤ i < n ∧ Prev = Ri, R = Ri+1];
   i := i + 1
while Prev ≠ R →
{Post : a < i ≤ n ∧ R = R_i = R_{i-1}}
]|
```

Con esto se concluye que la precondición más débil del algoritmo y Post es $(\exists \epsilon | a \leq \epsilon \leq n - 1 : R_\epsilon = R_{\epsilon+1})$.

XI. Ejemplos

La tripleta de Hoare encontrada al final de la sección anterior, induce un marco de trabajo para conseguir algoritmos de punto fijo concretos. Para construir un algoritmo de punto fijo, basta con definir una familia de conjuntos $\{R_i\}_{i=a}^{n}$, de tal forma que la precondición Pre sea una tautología, y refinar la instrucción de especificación en un trozo de código St que satisfaga $\{a \leq i < n \wedge Prev = R_i\} St \{R = R_{i-1}\}$.

En esta sección se construye el algoritmo de búsqueda de símbolos anulables de una gramática libre de contexto. Una gramática G libre de contexto es una tupla $G = (V, \Sigma, P, S)$ donde V es un conjunto de símbolos no terminales, Σ es un conjunto de símbolos terminales, $P \subseteq V \times (\Sigma \cup V)^*$ y $S \in V$ (dado un conjunto de símbolos Γ, se denota Γ^* y Γ^+ a los conjuntos de las frases construidas con los símbolos de Γ de longitud ≥ 0 y longitud > 0 respectivamente). Los pares $\langle A, w \rangle \in P$ se le llaman producciones, y se suelen denotar como $A \to w$.

Definición 3. *Dado una gramática libre de contexto G, se define la relación de derivación $\Rightarrow \subseteq (\Sigma \cup V)^+ \times (\Sigma \cup V)^*$ de forma que $t \Rightarrow w$ si y sólo si existen $A \in V$ y $t_1, t_2, w_1 \in (\Sigma \cup V)^*$ tales que $t = t_1 A t_2$, $w = t_1 w_1 t_2$ y $A \to w_1 \in P$. Por otro lado se define la relación de derivación en paralelo $\Rightarrow_p \subseteq (\Sigma \cup V)^+ \times (\Sigma \cup V)^*$ de forma que $t \Rightarrow_p w$ si y sólo si t es de la forma $t_0 A_1 t_1 A_2 \ldots t_{k-1} A_k t_k$ con $A_i \in V$, $t_i \in (\Sigma \cup V)^*$ y w es de la forma $t_1 w_1 t_2 w_2 \ldots t_k w_k$ en donde $A_i \Rightarrow w_i$.*

Definición 4. *Dada una relación binaria R, la notación $tR^* w$ y $tR^{\leq i} w$ significan $(\exists m | m \in \mathbb{N} : tR^m w)$ y $(\exists m | m \leq i :$*

$tR^m w$) *respectivamente, en donde* R^m *es la composición de* R *consigo misma* m *veces.*

El conjunto de los símbolos anulables de una gramática es $NULL := \{A \in V | A \Rightarrow^* \lambda\}$, donde λ es la frase vacía. Para diseñar un algoritmo de punto fijo primeramente se define una familia de conjuntos de la siguiente forma:

Definición 5. *Dada una gramática libre de contexto* $G = (\Sigma, V, P, S)$*, se define para* $i \geq 1$

$$NULL_i := \{A \in V | A \Rightarrow_p^{\leq i} \lambda\}.$$

El siguiente teorema justifica el uso de un algoritmo de punto fijo para conseguir el conjunto de símbolos anulables.

Teorema 5. *Si existe* $\epsilon \geq 1$ *tal que* $NULL_\epsilon = NULL_{\epsilon+1}$*, entonces* $NULL = NULL_\epsilon$

Por esta razón para calcular $NULL$ basta con encontrar el punto fijo de $\{NULL_i\}_{i=1}^{\infty}$.

Para instanciar la familia de algoritmos en un algoritmo que compute el punto fijo de la familia $\{NULL_i\}_{i=1}$, se debe demostrar que Pre es una tautología.

Teorema 6.

$$(\exists \epsilon | 1 \leq \epsilon : NULL_\epsilon = NULL_{\epsilon+1})$$

Proof: Por la definición de $\Rightarrow^{\leq i}$ se tiene que si $A \Rightarrow^{\leq i} \lambda$ entonces $A \Rightarrow^{\leq i+1} \lambda$, por lo tanto $NULL_1 \subseteq NULL_2 \subseteq \cdots \subseteq NULL_i$. Si el enunciado del teorema no fuera cierto, entonces se tendría para todo i que:

$$NULL_1 \subset NULL_2 \subset \cdots \subset NULL_i,$$

con lo que aumentando i se pudiera conseguir un $NULL_i$ de cardinalidad arbitrariamente grande, pero esto es imposible ya que $NULL_i \subseteq V$ para todo i y por lo tanto $|NULL_i| \leq |V|$, con lo que $|NULL_i|$ está acotado por ser V finito. ■

A continuación se refinará la instrucción de especificación:

$$R : [1 \leq i \wedge Prev = NULL_i, R = NULL_{i+1}] \qquad (*)$$

del esquema de algoritmos, en una instrucción concreta.

Teorema 7. *La siguiente definición recursiva es equivalente a la defición 5:*

- $NULL_1 := \{A \in V | A \to \lambda \in P\}$
- $NULL_{i+1} = \{A \in V | A \to w \in P \wedge w \in NULL_i^*\}$
  $\cup\, NULL_i$ *para* $i \geq 1$

Proof: Para demostrar el primer item se hace lo siguiente:

$NULL_1 = \{A \in V | A \Rightarrow_p^{\leq 1} \lambda\} =$
$\{A \in V | (\exists m | m \leq 1 : A \Rightarrow_p^m \lambda)\} = \{A \in V | A \to \lambda\}$ en donde la última igualdad es cierta debido a que es siempre falso que $A \to^0 \lambda$ si $A \in V$.

Para demostrar el segundo item se usa doble contención. La demostración en el sentido $\subseteq$ es la siguiente:

Si $A \in NULL_{i+1}$ entonces por definición $A \Rightarrow_p^{\leq i+1} \lambda$ y por lo tanto existe $m \leq i+1$ tal que $A \Rightarrow_p^{\leq m} \lambda$. Si $m \leq i$ entonces $A \in NULL_i$ con lo que se cumple la contención. Por otro lado si $m = i+1$ entonces $A \Rightarrow_p w \Rightarrow_p^i \lambda$ donde w debe ser de la forma $A_1 \ldots A_n$ y $A_k \Rightarrow_p^{\leq i} \lambda$, con lo que $A_k \in NULL_i$ y por lo tanto $w \in NULL_i^*$, de modo que $A \in \{A \in V | A \to w \in P \wedge w \in NULL_i^*\}$.

Para demostrar el sentido $\supseteq$ se hace lo siguiente:

Si $A \in \{A \in V | A \to w \in P \wedge w \in NULL_i^*\}$, entonces $A \Rightarrow_p w$ donde w es de la forma $A_1 \ldots A_n$ con $A_k \in V$, en donde $(\exists m'_k | m'_k \leq i : A_k \Rightarrow_p^{m'_k} \lambda)$. Si se toma m' como el máximo de los m'_k, se tiene que $(\exists m' | m' \leq i : w = A_1 \ldots A_k \Rightarrow_p^{m'} \lambda)$ y por lo tanto $A \Rightarrow_p w \Rightarrow_p^{\leq i} \lambda$, con lo cual $A \in NULL_{i+1}$. Por otro lado como en la demostración de 6 se determinó que $NULL_i \subseteq NULL_{i+1}$, entonces si $A \in NULL_i$, la demostración es inmediata. ■

Denotando $\mathtt{R} := \{\mathtt{A} \in \mathtt{V} | \mathtt{A} \to \mathtt{w} \in \mathtt{P} \wedge \mathtt{w} \in \mathtt{Prev}^*\} \cup \mathtt{Prev}$; como St, se tiene que:

$1 \leq i \wedge Prev = NULL_i$
$\Rightarrow \quad \langle$Teorema 7$\rangle$
$NULL_{i+1} = \{A \in V | A \to w \in P \wedge w \in Prev^*\} \cup Prev$
$\equiv$
$wp(St, 1 \leq i \wedge R = NULL_{i+1})$

de modo que la tripleta:

$\{1 \leq i \wedge Prev = NULL_i\}$
$\quad \mathtt{R} := \{\mathtt{A} \in \mathtt{V} | \mathtt{A} \to \mathtt{w} \in \mathtt{P} \wedge \mathtt{w} \in \mathtt{Prev}^*\} \cup \mathtt{Prev};$
$\{1 \leq i \wedge R = NULL_{i+1}\},$

es cierta y St es una realización de la especificación $(*)$.
Por último para demostrar el teorema 5 es necesario demostrar los siguiente lemas.

Lema 10. *Si* $NULL_\epsilon = NULL_{\epsilon+1}$ *con* $\epsilon \geq 1$*, entonces* $NULL_\epsilon = NULL_{\epsilon+n}$ *para todo* $n \geq 1$.

Proof: Por inducción natural sobre n, en donde el caso base $NULL_\epsilon = NULL_{\epsilon+1}$ está dado por hipótesis. Asumiendo $NULL_\epsilon = NULL_{\epsilon+n}$ como hipótesis inductiva (H.I.) se tiene que $NULL_{\epsilon+n+1} = NULL_{\epsilon+n} \cup \{A \in V | A \to w \in P \wedge w \in NULL_{\epsilon+n}^*\} \overset{H.I.}{=} NULL_\epsilon \cup \{A \in V | A \to w \in P \wedge w \in NULL_\epsilon^*\} = NULL_{\epsilon+1} = NULL_\epsilon$ ■

Lema 11. *Si para todo* $m' > \epsilon$ *se tiene que* $NULL_\epsilon = NULL_{m'}$ *entonces:*

$$(\exists m' | : A \Rightarrow_p^{m'} \lambda) \equiv A \Rightarrow_p^{\leq \epsilon} \lambda$$

Proof: Por doble implicación.

Si $A \Rightarrow_p^{\leq \epsilon} \lambda$ entonces por definición:
$(\exists m' | m' \leq \epsilon : A \Rightarrow_p^{m'} \lambda)$ y por lo tanto $(\exists m' | : A \Rightarrow_p^{m'} \lambda)$.

Por otro lado si existe m' tal que $A \Rightarrow_p^{m'} \lambda$ y $m' \leq \epsilon$, entonces por definición $A \Rightarrow_p^{\leq \epsilon} \lambda$, pero si $m' > \epsilon$ entonces se tendría que $A \in NULL_{m'} = NULL_\epsilon$ y por lo tanto $A \Rightarrow_p^{\leq \epsilon} \lambda$. ■

Ahora se puede demostrar el teorema 5.

Proof: Sea $A \in V$ entonces:

$$A \in NULL \overset{\text{definición de } NULL}{\equiv} A \Rightarrow^* \lambda$$
$$\overset{\text{definición de } \Rightarrow^*}{\equiv} (\exists m | : A \Rightarrow^m \lambda) \equiv (\exists m' | : A \Rightarrow_p^{m'} \lambda)$$
$$\overset{\text{Lema 11}}{\equiv} A \Rightarrow_p^{\leq \epsilon} \lambda \overset{\text{definición de } NULL_\epsilon}{\equiv} A \in NULL_\epsilon \quad \blacksquare$$

Con esto se concluye que el algoritmo de punto fijo para calcular símbolos anulables es:

$||[\texttt{Var i}, \texttt{Prev}, \texttt{R} : \texttt{Int}, \texttt{Set}, \texttt{Set};$

$\{Pre : true\}$
$\texttt{i}, \texttt{R} := 1, \{\texttt{A} \in \texttt{V} | \texttt{A} \rightarrow \lambda \in \texttt{P}\};$
do
  $\texttt{Prev} := \texttt{R};$
  $\texttt{R} := \{\texttt{A} \in \texttt{V} | \texttt{A} \rightarrow \texttt{w} \in \texttt{P} \wedge \texttt{w} \in \texttt{Prev}^*\} \cup \texttt{Prev};$
  $\texttt{i} := \texttt{i} + 1$
while $\texttt{Prev} \neq \texttt{R} \rightarrow$
$\{\text{Post} : 1 < i \wedge R{=}NULL_i{=}NULL_{i-1} \overset{\text{Teo 5}}{=} NULL\}$
$]||$

XII. Conclusiones

La técnica que se usa en este trabajo es efectiva para encontrar la precondición más débil de un algoritmo. Por otro lado en [1][6], existen ejemplos en que esta técnica, además de ser efectiva, también es más eficiente para corregir un algoritmo, que si se usara la lógica de Hoare. Sin embargo, en vista de la cantidad de cálculos que se tuvieron que hacer para corregir los algoritmos de punto fijo, se puede conjeturar, que era más eficiente hacer la corrección de la familia de algoritmos planteados, usando la lógica de Hoare.

Aunque para el tipo de algoritmos visto en este trabajo, la técnica propuesta resulte más complicada que las clásicas, aún existe una ventaja, la de poder demostrar que una aserción es la precondición más débil de todo el algoritmo. Esto es algo que no se puede garantizar con las reglas de Hoare.

Adicionalmente si se tiene la precondición más débil WP, se puede validar si cualquier otro predicado Pre es válido para ser usado como una precondición del algoritmo, verificando $Pre \Rightarrow WP$. De esta forma otra justificación de los cálculos realizados, es que la familia de algoritmos definida, junto con la familia de preconciones más débiles calculada, puede usarse como framework para construir algoritmos de punto fijo corretos. Esto es porque para corregir el algoritmo Alg, con la precondición Pre (y la postcondición $Post$ usada en el trabajo), basta con verificar que Alg es una instancia de la familia y que Pre implica la precondición más débil que se calculó aquí para Alg.

Referencias

[1] F. Flaviani, *Calculation of Invariants Assertions*, Electronic Notes in Theoretical Computer Science, vol. 339, pp. 63–83, 2018.

[2] F. Flaviani, *Propiedades Algebraicas y Decidibilidad del Transformador de Predicados wp Sobre la Teoría de Conjuntos*, Revista Venezolana de Computación (ReVeCom), vol. 4, no. 2, pp. 46–58, Diciembre 2017.

[3] E. W. Dijkstra, *Guarded Commands, Nondeterminacy and Formal Derivation of Programs*, Commun. ACM, vol. 18, pp. 453–457, August 1975.

[4] C. A. R. Hoare, *An Axiomatic Basis for Computer Programming*, Commun. ACM, vol. 12, pp. 576–580, October 1969.

[5] D. Gries, *The Science of Programming*, Springer New York, 1981.

[6] F. Flaviani, *Cálculo de Precondiciones más Débiles*, Revista Venezolana de Computación (ReVeCom), vol. 3, no. 2, pp. 68–80, Diciembre 2016.

[7] C. Morgan, *The Specification Statement*, ACM Trans. Program. Lang. Syst., vol. 10, pp. 403–419, July 1988.

[8] M. Todorova and D. A. Orozova, *The Predicate Transformer and its Application in Introduction to Programming Courses*, Burgas Free University Yearbook, vol. 32, no. 1, pp. 194–207, 2015.

[9] G. O'Regan, *Giants of Computing: A Compendium of Select, Pivotal Pioneers*, Springer London, 2013.

[10] F. Flaviani, *Inference of the Definition of the Predicate Transformer wp with Occurrences of the Predicate domain Based on Denotational Semantics of GCL on ZF Set Theory*, in proceedings of the XLII Latin American Computing Conference, Sao Paulo, Brazil, Octubre 2018.

[11] F. Flaviani, *Modelo Relacional de la Teoría Axiomática del Lenguaje GCL de Dijkstra*, en las memorias de la Conferencia Nacional de Computación, Informática y Sistemas (CoNCISa 2015), Valencia, Venezuela, Noviembre 2015, pp. 153-164.

[12] L. Jilani, O. Mraihi, A. Louhichi, W. Ghardallou, K. Bsaies, and A. Mili, *Invariant Functions and Invariant Relations: An Alternative to Invariant Assertions*, Journal of Symbolic Computation, vol. 48, pp. 1–36, January 2013.

[13] O. Mraihi, W. Ghardallou, A. Louhichi, L. L. Jilani, K. Bsaies, and A. Mili, *Computing Preconditions and Postconditions of While Loops*, in proceedings of the 8th International Colloquium on Theoretical Aspects of Computing, Johannesburg, South Africa, pp. 173–193, Springer, September 2011.