Propiedades Algebraicas y Decidibilidad del Transformador de Predicados wp sobre la Teoría de Conjuntos

Federico Flaviani
Universidad Simón Bolívar
Caracas, Venezuela
fflaviani@usb.ve

***Resumen*—En este trabajo se presentan nuevas propiedades algebraicas del transformador de predicado wp sobre los operadores $\wedge$, $\vee$, $\neg$, $\Rightarrow$ $\forall$, $\exists$, min y max, demostrados independientemente del lenguaje de programación, usando propiedades generales de la semántica denotacional de los lenguajes.**

Adicionalmente se muestra un resultado que habla sobre la decidibilidad y cerradura del transformador de predicados wp sobre el lenguaje de programación GCL y usando aserciones escritas en el lenguaje de la teoría de conjuntos de Zermelo-Frankel-Skolem. En este trabajo se muestra que calcular wp de una instrucción de GCL y una aserción escrita en el lenguaje ZFS, es decidible y es otra aserción escrita en ZFS. No necesariamente se puede decidir el valor de verdad de dicha aserción resultante de calcular wp, aun teniendo todos los valores de sus variables libres, por lo que el resultado no contradice la indecidibilidad del problema de la parada.

Palabras Clave—Precondición más débil; Semántica denotacional; Decidibilidad; GCL.

I. Introducción

La lógica de Dijkstra [1] para la corrección de programas se basa en el transformador de predicados wp (weakest precondition), que es básicamente una función sintáctica de dos variables que devuelve de forma simbólica la precondición más débil de una instrucción $inst$ dado una postcondición $Post$ (usando la notación clásica de funciones de dos variables, la notación $wp(inst, Post)$ se refiere al resultado de aplicarle a la función wp, los argumentos $inst$ y $Post$, este resultado es la precondición más débil, simbólicamente hablando, de la instrucción $inst$ con la postcondición $Post$). El uso sucesivo de wp permite ir calculando precondiciones más débiles entre instrucción e instrucción, desde el final del programa hasta el inicio.

Dijkstra en [1] estableció las reglas que definen la función de transformación sintáctica wp según el párrafo siguiente:

Si $B, B_0, \ldots, B_n$ y $S, S_0, \ldots, S_n$ son expresiones booleanas e instrucciones del lenguaje GCL respectivamente, si se abrevia IF y Do como las instrucciones $if\ B_0 \to S_0 [] \ldots []B_n \to S_n\ fi$ y $do\ B \to S\ od$ respectivamente y si se denota $domain(B_0, \ldots, B_n)$ como un predicado que de satisfacerse en un estado, ninguna de las expresiones B_i, al evaluarse en ese estado, incurren en una operación ilegal (como dividir entre 0), entonces:

- $wp(SKIP, Post) := Post$
- $wp(y_{i_1}, \ldots, y_{i_k} := Exp_1, \ldots, Exp_k, Post) := domain(Exp_1, \ldots, Exp_k) \wedge Post[y_{i_1}, \ldots, y_{i_k} := Exp_1, \ldots, Exp_k]$
- $wp(S_0; S_1, Post) := wp(S_0, wp(S_1, Post))$
- $wp(IF, Post) := domain(B_0, \ldots, B_n) \wedge (B_0 \vee \cdots \vee B_n) \wedge (B_0 \Rightarrow wp(S_0, Post)) \wedge \ldots \wedge (B_n \Rightarrow wp(S_n, Post))$
- $wp(Do, Post) := (\exists k | k \geq 0 : H_k(Post))$

  en donde $H_k(Post)$ es un predicado que satisface las ecuaciones:

$$H_0(Post) \equiv domain(B) \wedge \neg B \wedge Post$$

$$H_k(Post) \equiv$$

$$H_0(Post) \vee (domain(B) \wedge B \wedge wp(S, H_{k-1}(Post)))$$

  para $k \geq 1$

La definición recursiva anterior de $H_k(Post)$, tiene la desventaja de que la regla que define a wp para la instrucción Do esta escrita en lógica de segundo orden, por lo que no es aplicable directamente. Es necesario resolver la recurrencia de fórmulas $H_k(Post)$ primero, antes de aplicar la regla de wp para el Do, y también es necesario demostrar que la solución a esa recurrencia se puede escribir en el mismo lenguaje en que esta escrito $Post$, esto con el objetivo de demostrar que la función sintáctica $wp(S, .)$ es cerrada con respecto al lenguaje en que se escribe las aserciones (Es decir que si una postcondición $Post$ esta escrito en cierto lenguaje, entonces $wp(S, Post)$ es equivalente a una fórmula escrita en el mismo lenguaje).

Por otro lado existe un álgebra del transformador de predicado wp sobre los operadores $\wedge$, $\vee$ y $\Rightarrow$, determinada por las siguientes reglas

- $wp(S, false) \equiv false$
- $wp(S, P \wedge Q) \equiv wp(S, P) \wedge wp(S, Q)$
- $wp(S, P \vee Q) \equiv wp(S, P) \vee wp(S, Q)$ si S es una instrucción determinística
- Si $P \Rightarrow Q$, entonces $wp(S, P) \Rightarrow wp(S, Q)$

En este trabajo se demuestran propiedades del álgebra de wp distintas a las anteriores.

A. Contribución

Asumiendo que las aserciones de este trabajo se escriben en el lenguaje de la teoría de conjuntos de Zermelo-Frankel-Skolem, se demuestra la cerradura de $wp(S,.)$ sobre este lenguaje para cualquier instrucción S de GCL. Adicionalmente se muestra que, usando el teorema de las definiciones recursivas de la teoría de conjuntos, si se tiene un predicado $Post$, se muestra de forma constructiva una fórmula en ZFS que es equivalente a $wp(S, Post)$, demostrando que el problema del calculo de wp es decidible sobre el lenguaje ZFS.

El problema de la parada para la instrucción S, es equivalente al problema de conocer el valor de verdad de la aserción $wp(S, true)$ conociendo todos los valores de las variables libres de $wp(S, true)$ (que corresponden a los valores del estado inicial del programa antes de la ejecución). La decidibilidad del cálculo de $wp(S, Post)$ sobre el lenguaje ZFS no es una contradicción a la indecidibilidad del problema de la parada, ya que el problema de conocer el valor de verdad de la aserción resultante de escribir $wp(S, Post)$ en el lenguaje de la teoría de conjuntos de ZFS, teniendo todos los valores de las variables libres de la aserción, no es decidible, ya que la fórmula puede contener combinaciones de cuantificadores $\forall$ y $\exists$ sobre conjuntos infinitos, obligando a que la única forma de conocer el valor de verdad de la aserción sea haciendo una demostración matemática elaborada, y el problema de decidir si una fórmula es un teorema de ZFS, no es decidible.

Por otro lado en este trabajo usando semántica denotacional se demuestran propiedades algebraicas del transformador de predicados wp sobre los operadores $\wedge$ y $\vee$, cuando uno de los predicados en conjunción o disyunción no es modificado o es modificado determinísticamente por $wp(S,.)$. También se demuestran propiedades algebraicas de wp sobre los operadores $\neg$, $\Rightarrow$ $\forall$, $\exists$, min y max.

B. Trabajos Relacionados

Originalmente en [1] la definición recursiva de wp que se expuso al inicio no incluía la función sintáctica $domain$ en sus reglas, esto fue corregido en [2], donde lo incorpora a la regla de wp de la asignación, pero no en las demás reglas como se definió al inicio de la introducción. Una justificación de la incorporación de $domain$ en las reglas de wp del IF y Do, se encuentra en [3], donde se hace una revisión de la semántica denotacional de GCL incluyendo el estado $abort$. La función sintáctica $domain$, aplica sobre expresiones, pero su incorporación en las reglas de construcción de wp del IF y Do, traen dificultades adicionales que no se tenían en [2]. Para manejar estas dificultades, en [4] se definió la función sintáctica $support$, que viene siendo el análogo a $domain$, pero aplica sobre instrucciones en lugar de expresiones.

Gracias a la incorporación de la función sintáctica $support$ en la teoría de wp, es posible demostrar algunas propiedades algebraicas nuevas del transformador de predicados wp, dichas demostraciones se encuentran en este trabajo usando propiedades generales de la semántica denotacional, de esta forma los resultados de estas propiedades algebraicas no dependen del lenguaje de programación usado. Algunas de las propiedades algebraicas de wp que se enuncian aquí, ya fueron enunciadas en [5], en donde se afirma que las demostraciones se pueden realizar usando inducción estructural sobre el tamaño de la instrucción, pero dichas demostraciones no se encuentran en [5].

Por otro lado en [6] se demuestra un teorema de cerradura del transformador de predicados $wp(S, Post)$ sobre el lenguaje de la aritmética de Peano. La demostración muestra una forma constructiva de resolver la recurrencia $H_k(Post)$ usando las funciones β de Gödel, lo cual demuestra que el cálculo de $wp(S, Post)$ es decidible sobre el lenguaje de la aritmética. En [4] se tomó la misma idea de [6] pero resolviendo la recurrencia $H_k(Post)$ usando la versión numerable del teorema de recursión transfinita que es conocido, como el teorema de las definiciones recursivas, sin embargo en [4] no se toma en cuenta el aspecto constructivo del teorema de las definiciones recursivas y por lo tanto no se mostró un resultado de decibilidad del cálculo de wp. En este trabajo se retoma la demostración de [4] considerando los aspectos constructivos de la demostración con el objetivo de demostrar la decidibilidad del cálculo de wp sobre el lenguaje de la teoría de conjuntos de ZFS.

En el área de derivación automática de invariantes ha habido un interés reciente en los últimos años [7][8][9][10][11][12][13][14][15], adicionalmente existen aplicaciones como [16][17] que pueden calcular invariantes para ciclos donde las expresiones de las asignaciones del cuerpo del ciclo son todas lineales o traducibles a sistemas de transición lineales, de igual forma en [18] se encuentra otra técnica que es aplicable sólo a ciclos donde el cuerpo es traducible a una transformación afín de espacios vectoriales. Aplicaciones basadas en lógica de Hoare y separación tenemos a [19][20][21] y basadas en wp se encuentra [22], sólo que funciona para programas no estructurados.

El resultado sobre la decidibilidad del cálculo de wp abre posibilidades en el campo de la derivación automática de invariantes para algunos casos, ya que una precondición más débil de un ciclo es un invariante. Sin embargo, la precondición más débil obtenida del teorema de decidibilidad, no es en general una aserción cuyo valor de verdad es decidible cuando se tienen todos los valores las variables libres de la aserción, de esta forma no siempre es práctico usar este teorema de decidibilidad para derivar automáticamente un invariante, y más aún si se cuenta con alguna otra técnica, que en el caso en cuestión, pueda calcular un invariante equivalente y decidible (que el valor de verdad de la fórmula sea decidible cuando se tienen todos los valores de las variables libres de la aserción).

C. Estructura del Artículo

A continuación se presentan tres secciones de las cuales, en la primera de ellas se exponen todas las definiciones de semántica denotacional de [3] necesarias para demostrar los teoremas de las siguientes secciones. En la sección siguiente, se demuestran nuevas propiedades algebraicas del transformador de predicados wp. En la última sección se demuestra el teorema que afirma que el cálculo de wp sobre GCL y la teoría de conjuntos es decidible sobre el lenguaje de la teoría de conjuntos de ZFS.

II. Semántica Denotacional de un Lenguaje de Programación

Algunas de las propiedades del transformador de predicados wp que se encuentran en la siguiente sección, se enunciaron por primera vez en [5], donde se afirma que su demostración se hace por inducción estructural sobre el tamaño de la instrucción. Demostrar propiedades por inducción estructural tiene la desventaja de que la demostración depende de las instrucciones que tenga el lenguaje GCL, es decir, si en un futuro se extiende el lenguaje GCL con nuevas instrucciones, entonces sería necesario revisar todas las demostraciones por inducción estructural que se han hecho en la teoría, para incluir los casos correspondientes a las nuevas instrucciones.

Demostraciones independiente al lenguaje de programación son posibles usando semántica denotacional, en donde se consideran sólo las hipótesis generales que tiene una interpretación de una instrucción. A continuación se presenta un resumen de la semántica denotacional para lenguajes de programación propuesta en [3].

Definición (Espacio de Estados del Algoritmo)**.** *Se considera el siguiente algoritmo*

$$
\begin{array}{l}
[Const\ \overline{x} : \overline{T}; \\
\ Var\ \overline{y} : \overline{T'}; \\
\ S \\
]
\end{array}
$$

Donde S es el código del algoritmo, $\overline{x}$ es la lista de constantes del algoritmo y $\overline{T}$ es la lista de tipos de cada $\overline{x}$, $\overline{y}$ es la lista de variables del algoritmo y $\overline{T'}$ es la lista de tipos de cada $\overline{y}$.

Si $\overline{T} = T_1, T_2, \ldots, T_n$ y $\overline{T'} = T_{n+1}, T_{n+2}, \ldots, T_{n'}$, entonces se define el espacio de estados del algoritmo anterior como

$$\prod_{i=1}^{n'} T_i$$

Ejemplo. *Se considera el siguiente algoritmo:*

$$
\begin{array}{l}
[Const\ n : Entero; \\
Var\ x : Real; \\
\qquad z : Real; \\
S \\
]
\end{array}
$$

Como las constantes y variables del algoritmo son n, x y z de tipos Entero, Real y Real respectivamente, entonces el espacio de estados del algoritmo anterior es $\mathbb{Z} \times \mathbb{R} \times \mathbb{R}$.

Ejemplo. *Se considera el siguiente algoritmo:*

$$
\begin{array}{l}
[Const\ n : Entero; \\
Var\ A : arreglo\ [3..7)\ de\ Reales; \\
\qquad z : Real; \\
S \\
]
\end{array}
$$

Como un arreglo de tipo T, es una función de una parte de los enteros a T, entonces el espacio de estados del algoritmo del ejemplo es $\mathbb{Z} \times \mathbb{R}^{[3..7)} \times \mathbb{R}$

Notación. *En un algoritmo con espacio de estados Esp se denotará como $\vec{x}$ a la lista de constantes y variables que se encuentra en el orden en que fueron declaradas, es decir, $\vec{x} = \overline{x}||\overline{y}$, donde $\overline{x}$ y $\overline{y}$ son las listas de constantes y variables de la definición de espacio de estados y $||$ es el operador de concatenación de listas.*

Notación. *Para simplificar la notación, el vector $(\vec{x}) = (\overline{x}, \overline{y})$ se denota simplemente como $\vec{x}$, por lo que la notación $\vec{x}$ puede entenderse según el contexto como una lista de variables de tipo sintáctica $\overline{x}||\overline{y}$, ó como una tupla $(\overline{x}, \overline{y})$.*

Definición. *Sea un algoritmo con espacio de estados Esp y se toma un elemento $abort \notin Esp$, entonces se define el espacio de estados extendido al abort como $Esp' := Esp \cup \{abort\}$*

Un algoritmo además de la descripción del espacio de estados consta de frases del lenguaje que se llaman instrucciones y dentro de las instrucciones se encuentran otras frases llamadas expresiones de tipo T (donde T es un conjunto). Dichas frases se interpretan denotacionalmente con la función de interpretación $\mathcal{E}$. Dicha función de interpretación toma una expresión Exp sintácticamente hablando y devuelve una función con rango en T, que se denota $\mathcal{E}[\![Exp]\!]$.

Definición. *En un algoritmo con espacio de estados Esp, una expresión Exp de tipo T, es una frase del lenguaje que se interpreta como una función*

$$\mathcal{E}[\![Exp]\!] : Dom(\mathcal{E}[\![Exp]\!]) \subseteq Esp \to T.$$

Nota. *Se deja abierta a cualquier posibilidad la sintaxis de las expresiones y el valor de su función de interpretación, ya que la teoría que aquí se desarrolla es válida para cualquier tipo de expresión y función de interpretación.*

Notación. *De ahora en adelante para abreviar se denota a la interpretación de la expresión Exp evaluada en $\vec{x}$ por $Exp(\vec{x})$, en lugar de $\mathcal{E}[\![Exp]\!](\vec{x})$ y se denota el dominio de la expresión por $Dom(Exp)$ en lugar de $Dom(\mathcal{E}[\![Exp]\!])$.*

Notación. *Dada una expresión Exp dentro de un algoritmo con espacio de estados Esp, se denota como $domain(Exp)$ a una fórmula con variables libres $\vec{x}$, tal que*

$$Dom(Exp) = \{\vec{x} \in Esp | domain(Exp)\}$$

y si $Exp_1, \ldots, Exp_n$ son expresiones del mismo algoritmo definiremos

$$domain(Exp_1, \ldots, Exp_n) :=$$

$$domain(Exp_1) \wedge \cdots \wedge domain(Exp_n)$$

A continuación se define el concepto de instrucción e interpretación de la misma. Para dar semántica denotacional a las instrucciones, se usará una función de interpretación que se denota $\mathcal{C}$, dicha función recibe una instrucción S sintácticamente hablando y devuelve una interpretación, que

se denota como $\mathcal{C}[\![S]\!]$, que no es más que una relación de la teoría de conjuntos.

Definición. *En un algoritmo con espacio de estados Esp una instrucción S es una frase del lenguaje que se interpreta como una relación*

$$\mathcal{C}[\![S]\!] : Esp' \to Esp'$$

tal que el dominio de $\mathcal{C}[\![S]\!]$ es todo Esp',

$$\mathcal{C}[\![S]\!](\{abort\}) = \{abort\}$$

y

$$\overline{x'} = \overline{x} \text{ si } (\overline{x'}, \overline{y'}) \in \mathcal{C}[\![S]\!](\{(\overline{x}, \overline{y})\}) \text{ y } \overline{x'} \in \overline{T}.$$

Ejecutar esta instrucción para unos valores iniciales $\vec{x_0}$ se entiende como evaluar la relación anterior en los valores $\vec{x_0}$ donde la salida de la ejecución pudiera ser cualquier imagen del punto $\vec{x_0}$.

Adicionalmente, se define el soporte $supp(\mathcal{C}[\![S]\!])$ como el conjunto de los estados que no resultan en un abort al ejecutar la instrucción, es decir,

$$supp(\mathcal{C}[\![S]\!]) := \{\vec{x} \in Esp | abort \notin \mathcal{C}[\![S]\!](\{\vec{x}\})\}$$

Nota. *Note que como el dominio de $\mathcal{C}[\![S]\!]$ es todo Esp' entonces, $\mathcal{C}[\![S]\!](\{x\}) \neq \emptyset$ para todo $x \in Esp'$*

Definición. *Si $\overline{x}$ y $\overline{y}$ son la lista de constantes y variables de un algoritmo A en el orden en que fueron declaradas, $\overline{Y}$ una lista de variables distintas a $\overline{x}$ y $\overline{y}$, $\overline{Y_0}$ una lista de valores de tipos iguales a las variables $\overline{Y}$, S una instrucción, $Pre(\overline{x},\overline{y},\overline{Y})$ y $Post(\overline{x},\overline{y},\overline{Y})$ predicados que solo tienen como variables libres a $\overline{x}$, $\overline{y}$ y $\overline{Y}$ y las familias de conjuntos*

$$Dom_{\overline{Y}} := \{(\overline{x}, \overline{y}) \in Esp | Pre(\overline{x}, \overline{y}, \overline{Y})\}$$

y

$$Rgo_{\overline{Y}} := \{(\overline{x}, \overline{y}) \in Esp | Post(\overline{x}, \overline{y}, \overline{Y})\},$$

entonces una tripleta de Hoare dentro del algoritmo A es un predicado que tiene la forma

$$\{Pre(\overline{x}, \overline{y}, \overline{Y_0})\} S \{Post(\overline{x}, \overline{y}, \overline{Y_0})\}$$

y es verdadera si y sólo si

$$\mathcal{C}[\![S]\!](Dom_{\overline{Y_0}}) \subseteq Rgo_{\overline{Y_0}}$$

Notación. *En caso en que Pre y $Post$ no tengan valores fijos instanciando las variables libres $\overline{Y}$, entonces la tripleta*

$$\{Pre(\vec{x}, \overline{Y})\} S \{Post(\vec{x}, \overline{Y})\}$$

es una abreviación de

$$(\forall \overline{Y} | : \{Pre(\vec{x}, \overline{Y})\} S \{Post(\vec{x}, \overline{Y})\}),$$

es decir, que para cada instancia $\overline{Y_0}$ de valores fijos para las variables $\overline{Y}$, se tiene que

$$\{Pre(\vec{x}, \overline{Y_0})\} S \{Post(\vec{x}, \overline{Y_0})\}$$

es verdadera.

Nota. *Como $\overline{Y}$ es una lista de variables ligadas en el predicado $\{Pre(\vec{x}, \overline{Y})\}\ S\{Post(\vec{x}, \overline{Y})\}$ según la notación anterior, entonces el nombre de de éstas no importa y pueden cambiarse según convenga.*

Nota. *Como $Dom_{\overline{Y}} := \{\vec{x} \in Esp | Pre(\vec{x}, \overline{Y})\}$ y $Rgo_{\overline{Y}} := \{\vec{x} \in Esp | Post(\vec{x}, \overline{Y})\}$, entonces para evitar redundancia se convendrá que en $Pre(\vec{x}, \overline{Y})$ y $Post(\vec{x}, \overline{Y})$ no aparece el predicado $\vec{x} \in Esp$, y dicho predicado siempre se tomará como una hipótesis sobrentendida.*

Definición. *Sea R una relación de $A \times B$ y un subconjunto $Rgo \subseteq B$, entonces*

$$M := \{x \in A | R(\{x\}) \neq \emptyset \wedge R(\{x\}) \subseteq Rgo\}$$

se denomina como "dominio máximo de R con rango Rgo".

Lema 1. *Dado un algoritmo con espacio de estados Esp, una instrucción S y un predicado $Post(\vec{x}, \overline{Y})$ con $\vec{x} \in Esp$ y $\overline{Y_0}$ lista de valores del mismo tipo que $\overline{Y}$, entonces el dominio máximo de $\mathcal{C}[\![S]\!]$ y $Rgo_{\overline{Y_0}}$ es igual a*

$$\{\vec{x} \in Esp | \vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge \\ (\forall y | : y \in R_S \restriction_{supp(\mathcal{C}[\![S]\!])} (\{\vec{x}\}) \Rightarrow Post(y, \overline{Y_0}))\}$$

Donde $Post(y, \overline{Y_0})$ es una notación simplificada que significa

$$(\exists \overline{y'} | y = (\overline{x}, \overline{y'}) : Post(\overline{x}, \overline{y'}, \overline{Y_0}))$$

Demostración: Ver [3] ■

Definición. *Dado un algoritmo cuyo espacio de estados es Esp con lista de constantes y variables igual a $\vec{x}$ y la tripleta $\{Pre\}S\{Post\}$ tiene como variables libres distintas a las del espacio de estado a $\overline{Y}$, entonces se dice que Pre es una precondición más débil de S y $Post$ si y sólo si $\{\vec{x} \in Esp | Pre(\vec{x}, \overline{Y_0})\}$ es el dominio máximo de la relación $\mathcal{C}[\![S]\!]$ con rango $Rgo_{\overline{Y_0}}$ para cualquier valor $\overline{Y_0}$ de las variables libres $\overline{Y}$.*

Nota. *Según la definición anterior la precondición más débil de la instrucción S y $Post$ es equivalente a*

$$\vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge \\ (\forall y | : y \in R_S \restriction_{supp(\mathcal{C}[\![S]\!])} (\{\vec{x}\}) \Rightarrow Post(y, \overline{Y}))\}$$

De modo que en el lenguaje de la teoría de conjuntos, el predicado $wp(S, Post(\vec{x}, \overline{Y}))$ es equivalente al predicado anterior.

III. No Determinismo y Propiedades de wp y $support$

En esta sección se introduce la función de tipo sintáctica $support$, que es análoga a la función $domain$ salvo, que $support$ aplica a instrucciones mientras que $domain$ aplica a expresiones. Incluir en la teoría del transformador de predicados wp a la función sintáctica $support$, tiene la ventaja de que con su ayuda, pueden enunciarse propiedades de tipo algebraicas de wp sobre operadores como $\neg$, $\Rightarrow$ min y max.

Los siguientes teoremas son propiedades algebraicas del transformador de predicados wp, los lemas principales serán demostrados usando aserciones escritas en el lenguaje de la

teoría de conjuntos de ZFS y usando la fórmula de precondición más débil que se deduce de la semántica denotacional al final de la sección anterior. Dicha fórmula tiene la ventaja de ser independiente del lenguaje de programación, ya que sólo usa el concepto general de interpretación de una instrucción $\mathcal{C}[\![S]\!]$, sin usar hipótesis de como es el comportamiento específico de la instrucción S al ser interpretada.

Lema 2. $wp(S, P \wedge Q) \equiv wp(S, P) \wedge wp(S, Q)$

La interpretación de la instrucción S es una relación $\mathcal{C}[\![S]\!]$, si esta relación es determinística con respecto a las coordenadas $i_1, \ldots, i_k$, entonces la relación se comporta como una función sobre esas coordenadas. Dichas funciones se les pondrá el nombre de "funciones componentes" y se denotará como $\mathcal{C}[\![S]\!]^j$ a la función componente de la coordenada j.

Ejemplo.
$[Const\ x : Real;$
$\quad Var\ y :\ Entero;$
$\quad\quad z :\ Real;$
$\quad if\ y \geq 0\ \rightarrow$
$\quad\quad y := -y;$
$\quad\quad z := z/x$
$\quad [\!]\ y \geq 3\ \rightarrow$
$\quad\quad y := 3;$
$\quad\quad z := z/x$
$\quad fi$
$\quad]$

Denotemos el cuerpo del algoritmo anterior por S. El espacio de estados Esp del algoritmo es $\mathbb{R} \times \mathbb{Z} \times \mathbb{R}$, por lo tanto si $\vec{x} \in Esp$, entonces $\vec{x}$ es de la forma (x, y, z) en donde la tercera coordenada es modificada por $\mathcal{C}[\![S]\!]$ de forma determinística. De esta forma, la tercera coordenada es gobernada por la función componente

$$\mathcal{C}[\![S]\!]^3 : supp(\mathcal{C}[\![S]\!]) \subseteq \mathbb{R} \times \mathbb{Z} \times \mathbb{R} \to \mathbb{R}$$

$$\mathcal{C}[\![S]\!]^3(x, y, z) := \frac{z}{x}$$

y todo elemento perteneciente a $\mathcal{C}[\![S]\!](\{(x, y, z)\})$ con $(x, y, z) \in supp(\mathcal{C}[\![S]\!])$ es de la forma

$$(x, y', \mathcal{C}[\![S]\!]^3(x, y, z))$$

para algún $y' \in \mathbb{Z}$

Lema 3. *Sean $\overline{x}$ y $\overline{y}$ la lista de constantes y variables declaradas en un algoritmo respectivamente y $\overline{Y}$ una lista de variables de especificación. Sea S una instrucción que se comporta determinísticamente sobre los valores de las variables $y_{i_1}, \ldots, y_{i_k}$ de la lista $\overline{y}$. Sea Q un predicado y $P(\overline{x}, y_{i_1}, \ldots, y_{i_k}, \overline{Y})$ un predicado que sólo depende de $\overline{x}, y_{i_1}, \ldots, y_{i_k}, \overline{Y}$, entonces*

$$wp(S, P(\overline{x}, y_{i_1}, \ldots, y_{i_k}, \overline{Y}) \vee Q(\overline{x}, \overline{y}, \overline{Y})) \equiv$$

$$\vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge$$

$$(P(\overline{x}, \mathcal{C}[\![S]\!]^{i_1}(\vec{x}), \ldots, \mathcal{C}[\![S]\!]^{i_k}(\vec{x}), \overline{Y}) \vee wp(S, Q(\overline{x}, \overline{y}, \overline{Y})))$$

Donde $\mathcal{C}[\![S]\!]^j(\vec{x})$ es la función componente de $\mathcal{C}[\![S]\!]$ en la coordenada j

Demostración: Según la última fórmula de la sección anterior se tiene que la precondición más débil de una instrucción S y la postcondición $Post$ es equivalente a

$$(\overline{x}, \overline{y}) \in supp(\mathcal{C}[\![S]\!]) \wedge$$

$$(\forall y| : y \in \mathcal{C}[\![S]\!] \restriction_{supp(\mathcal{C}[\![S]\!])} (\{(\overline{x}, \overline{y})\}) \Rightarrow$$

$$(\exists \overline{y'} | y = (\overline{x}, \overline{y'}) : Post(\overline{x}, \overline{y'}, \overline{Y}))).$$

Como la interpretación $\mathcal{C}[\![S]\!]$ de la instrucción S modifica las coordenadas $i_1, \ldots, i_k$ de forma determinística con las funciones componentes $\mathcal{C}[\![S]\!]^{i_j}(\vec{x})$, entonces el predicado anterior es equivalente a:

$$(\overline{x}, \overline{y}) \in supp(\mathcal{C}[\![S]\!]) \wedge$$

$$(\forall y| : y \in \mathcal{C}[\![S]\!] \restriction_{supp(\mathcal{C}[\![S]\!])} (\{(\overline{x}, \overline{y})\}) \Rightarrow$$

$$(\exists y'_1, \ldots, y'_{i_1-1}, y'_{i_1+1}, \ldots, y'_{i_k-1}, y'_{i_k+1}, \ldots, y'_m|$$

$$y = (\overline{x}, y'_1, \ldots, y'_{i_1-1}, \mathcal{C}[\![S]\!]^{i_1}(\vec{x}), \ldots, y'_{i_k-1}, \mathcal{C}[\![S]\!]^{i_k}(\vec{x}), \ldots, y'_m) :$$

$$Post(\overline{x}, y'_1, \ldots, y'_{i_1-1}\mathcal{C}[\![S]\!]^{i_1}(\vec{x}), \ldots, y'_{i_k-1}, \mathcal{C}[\![S]\!]^{i_k}(\vec{x}), \ldots, \overline{Y})))$$

Si la postcondición $Post$ es $P(\overline{x}, y_{i_1}, \ldots, y_{i_k}, \overline{Y}) \vee Q(\overline{x}, \overline{y}, \overline{Y})$, entonces la precondición más débil de la instrucción S con esta postcondición es equivalente a:

$$(\overline{x}, \overline{y}) \in supp(\mathcal{C}[\![S]\!]) \wedge$$

$$(\forall y| : y \in \mathcal{C}[\![S]\!] \restriction_{supp(\mathcal{C}[\![S]\!])} (\{(\overline{x}, \overline{y})\}) \Rightarrow$$

$$(\exists y'_1, \ldots, y'_{i_1-1}, y'_{i_1+1}, \ldots, y'_{i_k-1}, y'_{i_k+1}, \ldots, y'_m|$$

$$y = (\overline{x}, y'_1, \ldots, y'_{i_1-1}, \mathcal{C}[\![S]\!]^{i_1}(\vec{x}), \ldots, y'_{i_k-1}, \mathcal{C}[\![S]\!]^{i_k}(\vec{x}), \ldots, y'_m) :$$

$$P(\overline{x}, \mathcal{C}[\![S]\!]^{i_1}(\vec{x}), \ldots, \mathcal{C}[\![S]\!]^{i_k}(\vec{x}), \overline{Y}) \vee$$

$$Q(\overline{x}, y'_1, \ldots, \mathcal{C}[\![S]\!]^{i_1}(\vec{x}), y'_{i_1+1}, \ldots, \mathcal{C}[\![S]\!]^{i_k}(\vec{x}), y'_{i_k+1}, \ldots, y'_m, \overline{Y})))$$

$\equiv$

$$(\overline{x}, \overline{y}) \in supp(\mathcal{C}[\![S]\!]) \wedge$$

$$(\forall y| : y \in \mathcal{C}[\![S]\!] \restriction_{supp(\mathcal{C}[\![S]\!])} (\{(\overline{x}, \overline{y})\}) \Rightarrow$$

$$P(\overline{x}, \mathcal{C}[\![S]\!]^{i_1}(\vec{x}), \ldots, \mathcal{C}[\![S]\!]^{i_k}(\vec{x}), \overline{Y}) \vee$$

$$(\exists y'_1, \ldots, y'_{i_1-1}, y'_{i_1+1}, \ldots, y'_{i_k-1}, y'_{i_k+1}, \ldots, y'_m|$$

$$y = (\overline{x}, y'_1, \ldots, y'_{i_1-1}, \mathcal{C}[\![S]\!]^{i_1}(\vec{x}), \ldots, y'_{i_k-1}, \mathcal{C}[\![S]\!]^{i_k}(\vec{x}), \ldots, y'_m) :$$

$$Q(\overline{x}, y'_1, \ldots, y'_{i_1-1}, \mathcal{C}[\![S]\!]^{i_1}(\vec{x}), \ldots, y'_{i_k-1}, \mathcal{C}[\![S]\!]^{i_k}(\vec{x}), \ldots, y'_m, \overline{Y})))$$

$\equiv$

$$(\overline{x}, \overline{y}) \in supp(\mathcal{C}[\![S]\!]) \wedge$$

$$(\forall y| : y \in \mathcal{C}[\![S]\!] \restriction_{supp(\mathcal{C}[\![S]\!])} (\{(\overline{x}, \overline{y})\}) \Rightarrow$$

$$P(\overline{x}, \mathcal{C}[\![S]\!]^{i_1}(\vec{x}), \ldots, \mathcal{C}[\![S]\!]^{i_k}(\vec{x}), \overline{Y}) \vee (\exists \overline{y'} | y = (\overline{x}, \overline{y'}) : Q(\overline{x}, \overline{y'}, \overline{Y})))$$

$\equiv$

$$(\overline{x},\overline{y}) \in supp(\mathcal{C}[\![S]\!])\wedge$$
$$(\forall y| : y \notin \mathcal{C}[\![S]\!] \restriction_{supp(\mathcal{C}[\![S]\!])} (\{(\overline{x},\overline{y})\})\vee$$
$$P(\overline{x},\mathcal{C}[\![S]\!]^{i_1}(\vec{x}),\ldots,\mathcal{C}[\![S]\!]^{i_k}(\vec{x}),\overline{Y})\vee(\exists\overline{y'}|y=(\overline{x},\overline{y'}) : Q(\overline{x},\overline{y'},\overline{Y})))$$

$\equiv$

$$(\overline{x},\overline{y}) \in supp(\mathcal{C}[\![S]\!])\wedge$$
$$(P(\overline{x},\mathcal{C}[\![S]\!]^{i_1}(\vec{x}),\ldots,\mathcal{C}[\![S]\!]^{i_k}(\vec{x}),\overline{Y})\vee$$
$$(\forall y| : y \notin \mathcal{C}[\![S]\!] \restriction_{supp(\mathcal{C}[\![S]\!])} (\{(\overline{x},\overline{y})\})\vee$$
$$(\exists\overline{y'}|y=(\overline{x},\overline{y'}) : Q(\overline{x},\overline{y'},\overline{Y}))))$$

$\equiv$

$$(\overline{x},\overline{y}) \in supp(\mathcal{C}[\![S]\!])\wedge$$
$$(P(\overline{x},\mathcal{C}[\![S]\!]^{i_1}(\vec{x}),\ldots,\mathcal{C}[\![S]\!]^{i_k}(\vec{x}),\overline{Y})\vee$$
$$(\forall y| : y \in \mathcal{C}[\![S]\!] \restriction_{supp(\mathcal{C}[\![S]\!])} (\{(\overline{x},\overline{y})\}) \Rightarrow$$
$$(\exists\overline{y'}|y=(\overline{x},\overline{y'}) : Q(\overline{x},\overline{y'},\overline{Y}))))$$

$\equiv< p \wedge (q \vee r) \equiv p \wedge (q \vee (p \wedge r)) >$

$$(\overline{x},\overline{y}) \in supp(\mathcal{C}[\![S]\!]) \wedge (P(\overline{x},\mathcal{C}[\![S]\!]^{i_1}(\vec{x}),\ldots,\mathcal{C}[\![S]\!]^{i_k}(\vec{x}),\overline{Y})\vee$$
$$((\overline{x},\overline{y}) \in supp(\mathcal{C}[\![S]\!])\wedge(\forall y| : y \in \mathcal{C}[\![S]\!] \restriction_{supp(\mathcal{C}[\![S]\!])} (\{(\overline{x},\overline{y})\}) \Rightarrow$$
$$(\exists\overline{y'}|y=(\overline{x},\overline{y'}) : Q(\overline{x},\overline{y'},\overline{Y})))))$$

$\equiv$<Lema 1>

$$(\overline{x},\overline{y}) \in supp(\mathcal{C}[\![S]\!])\wedge$$
$$(P(\overline{x},\mathcal{C}[\![S]\!]^{i_1}(\vec{x}),\ldots,\mathcal{C}[\![S]\!]^{i_k}(\vec{x}),\overline{Y}) \vee wp(S,Q(\overline{x},\overline{y},\overline{Y})))$$

■

Lema 4. *Sean $\overline{x}$ y $\overline{y}$ la lista de constantes y variables declaradas en un algoritmo respectivamente y $\overline{Y}$ una lista de variables de especificación. Sea S una instrucción que se comporta determinísticamente sobre los valores de las variables $y_{i_1},\ldots,y_{i_k}$ de la lista $\overline{y}$. Sea $P(\overline{x},y_{i_1},\ldots,y_{i_k},\overline{Y})$ un predicado que sólo depende de $\overline{x},y_{i_1},\ldots,y_{i_k},\overline{Y}$, entonces*

$$wp(S,P(\overline{x},y_{i_1},\ldots,y_{i_k},\overline{Y})) \equiv$$
$$\vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge P(\overline{x},\mathcal{C}[\![S]\!]^{i_1}(\vec{x}),\ldots,\mathcal{C}[\![S]\!]^{i_k}(\vec{x}),\overline{Y})$$

Donde $\mathcal{C}[\![S]\!]^j(\vec{x})$ es la función componente de $\mathcal{C}[\![S]\!]$ en la coordenada j

Demostración: Inmediato tomando Q como $false$ en el Lema 3 y usando $wp(S,false) \equiv false$ ■

Lema 5. *Sea S una instrucción (determinística o no) tal que se comporta de forma determinística sobre las variables del predicado $P \vee Q$, entonces*

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

Demostración: Si P depende de $\overline{x}$, $\overline{Y}$ y de las variables $y_{i_1},\ldots,y_{i_k}$ y Q depende de $\overline{x}$, $\overline{Y}$ y de las variables $y_{j_1},\ldots,y_{j_{k'}}$, entonces:

$$wp(S,P(\overline{x},y_{i_1},\ldots,y_{i_k},\overline{Y}) \vee Q(\overline{x},y_{j_1},\ldots,y_{j_{k'}},\overline{Y}))$$

$\equiv$<Lema 4>

$$\vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge (P(\overline{x},\mathcal{C}[\![S]\!]^{i_1}(\vec{x}),\ldots,\mathcal{C}[\![S]\!]^{i_k}(\vec{x}),\overline{Y})\vee$$
$$Q(\overline{x},\mathcal{C}[\![S]\!]^{j_1}(\vec{x}),\ldots,\mathcal{C}[\![S]\!]^{j_{k'}}(\vec{x}),\overline{Y}))$$
$\equiv$
$$(\vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge P(\overline{x},\mathcal{C}[\![S]\!]^{i_1}(\vec{x}),\ldots,\mathcal{C}[\![S]\!]^{i_k}(\vec{x}),\overline{Y}))\vee$$
$$(\vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge Q(\overline{x},\mathcal{C}[\![S]\!]^{j_1}(\vec{x}),\ldots,\mathcal{C}[\![S]\!]^{j_{k'}}(\vec{x}),\overline{Y}))$$

$\equiv$<Lema 4>

$$wp(S,P) \vee wp(S,Q)$$ ■

Lema 6. *Sea P un predicado y S una instrucción que no modifica los valores de las variables de P, entonces*

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

donde $support(S)$ es un predicado que depende de las constantes y variables declaradas en el programa, tal que un estado lo satisface si y sólo si la instrucción S no aborta al ser ejecutado en dicho estado.

Por ejemplo $true$ es un predicado que para cualquier S, se tiene que S no modifica sus variables, por lo que una forma de calcular $support(S)$ es calculando $wp(S,true) \equiv support(S) \wedge true \equiv support(S)$. Por ejemplo si S es la instrucción

$$\begin{array}{l} if\ a > -3 \rightarrow \\ \quad b := b/a \\ [\!]\ a \leq -3 \rightarrow \\ \quad b := 2 \\ fi, \end{array}$$

entonces

$$\begin{array}{l} wp(S,true) \\ \equiv \\ (a > -3 \Rightarrow domain(b/a) \wedge true[b := b/a])\wedge \\ (a \leq -3 \Rightarrow true[b := 2]) \\ \equiv \\ (a > -3 \Rightarrow a \neq 0) \wedge true \end{array}$$

Con lo que $support(S) \equiv a > -3 \Rightarrow a \neq 0$.
A continuación se demostrará el Lema 6.

Demostración: Si P depende de $\overline{x}$, $\overline{Y}$ y de las variables $y_{i_1},\ldots,y_{i_k}$, entonces

$$wp(S,P(\overline{x},y_{i_1},\ldots,y_{i_k},\overline{Y}))$$

$\equiv$<Lema 4>

$\vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge P(\overline{x}, \mathcal{C}[\![S]\!]^{i_1}(\vec{x}), \ldots, \mathcal{C}[\![S]\!]^{i_k}(\vec{x}), \overline{Y})$

Como la instrucción S no modifica los valores de las variables $y_{i_1}, \ldots, y_{i_k}$, entonces las funciones componentes $\mathcal{C}[\![S]\!]^{i_1}(\vec{x}), \ldots, \mathcal{C}[\![S]\!]^{i_k}(\vec{x})$ son funciones identidad y por lo tanto la expresión anterior es equivalente a:

$\vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge P(\overline{x}, y_{i_1}, \ldots, y_{i_k}, \overline{Y})$

$\equiv$<Notación>

$support(S) \wedge P$ ■

Lema 7. $wp(S, P) \Rightarrow support(S)$

Demostración:
$wp(S, P)$

$\equiv$<Lema 1>

$\vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge$
$(\forall y| : y \in R \restriction_{supp(\mathcal{C}[\![S]\!])} (\{\vec{x}\}) \Rightarrow P(y, \overline{Y}))$

$\Rightarrow$<Debilitamiento>

$\vec{x} \in supp(\mathcal{C}[\![S]\!])$

$\equiv$<Notación>

$support(S)$ ■

Lema 8. *Sean P y Q predicados y S una instrucción que se comporta determinísticamente sobre los valores de las variables de P, entonces*

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

Demostración: Si P depende de $\overline{x}$, $\overline{Y}$ y de las variables $y_{i_1}, \ldots, y_{i_k}$, entonces

$wp(S, P(\overline{x}, y_{i_1}, \ldots, y_{i_k}, \overline{Y}) \vee Q(\overline{x}, \overline{y}, \overline{Y}))$

$\equiv$<Lema 3>

$$\vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge$$
$$(P(\overline{x}, \mathcal{C}[\![S]\!]^{i_1}(\vec{x}), \ldots, \mathcal{C}[\![S]\!]^{i_k}(\vec{x}), \overline{Y}) \vee wp(S, Q(\overline{x}, \overline{y}, \overline{Y})))$$

$\equiv$

$$(\overline{x}, \overline{y}) \in supp(\mathcal{C}[\![S]\!]) \wedge$$
$$(((\overline{x}, \overline{y}) \in supp(\mathcal{C}[\![S]\!]) \wedge P(\overline{x}, \mathcal{C}[\![S]\!]^{i_1}(\vec{x}), \ldots, \mathcal{C}[\![S]\!]^{i_k}(\vec{x}), \overline{Y})) \vee$$
$$wp(S, Q(\overline{x}, \overline{y}, \overline{Y})))$$

$\equiv$<Lema 4>

$$(\overline{x}, \overline{y}) \in supp(\mathcal{C}[\![S]\!]) \wedge$$
$$(wp(S, P(\overline{x}, \overline{y}, \overline{Y})) \vee wp(S, Q(\overline{x}, \overline{y}, \overline{Y}))$$

$\equiv$<Notación>

$$support(S) \wedge$$
$$(wp(S, P(\overline{x}, \overline{y}, \overline{Y})) \vee wp(S, Q(\overline{x}, \overline{y}, \overline{Y})))$$

■

Lema 9. *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

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

Demostración:
$wp(S, P \wedge Q)$

$\equiv$<Lema 2>

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

$\equiv$<Lema 6>

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

$\equiv$<Lema 7>

$P \wedge wp(S, Q)$

Por otro lado si S no modifica los valores de las variables de P, entonces S se comporta determinísticamente sobre los valores de las variables de P, por lo tanto se puede aplicar el Lema 8 de la siguiente manera:

$wp(S, P \vee Q)$

$\equiv$<Lema 8>

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

$\equiv$<Lema 6>

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

Lema 10. *Sea P un predicado y S una instrucción que se comporta determinísticamente sobre los valores de las variables de P, entonces*

$$wp(S, \neg P) \equiv support(S) \wedge \neg wp(S, P)$$

Demostración: Si P depende de $\overline{x}$, $\overline{Y}$ y de las variables $y_{i_1}, \ldots, y_{i_k}$, entonces:

$wp(S, \neg P(\overline{x}, y_{i_1}, \ldots, y_{i_k}, \overline{Y}))$

$\equiv$<Lema 4>

$\vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge \neg P(\overline{x}, \mathcal{C}[\![S]\!]^{i_1}(\vec{x}), \ldots, \mathcal{C}[\![S]\!]^{i_k}(\vec{x}), \overline{Y})$

$\equiv$<Absorción>

$\vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge (\vec{x} \notin supp(\mathcal{C}[\![S]\!]) \vee \neg P(\overline{x}, \mathcal{C}[\![S]\!]^{i_1}(\vec{x}), \ldots, \mathcal{C}[\![S]\!]^{i_k}(\vec{x}), \overline{Y}))$
$\equiv$
$\vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge \neg(\vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge P(\overline{x}, \mathcal{C}[\![S]\!]^{i_1}(\vec{x}), \ldots, \mathcal{C}[\![S]\!]^{i_k}(\vec{x}), \overline{Y}))$

$\equiv$<Lema 4>

$\vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge \neg wp(S, P(\overline{x}, y_{i_1}, \ldots, y_{i_k}, \overline{Y}))$

$\equiv$<Notación>

$support(S) \wedge \neg wp(S, P)$ ■

Lema 11. *Sean P y Q predicados y S una instrucción que se comporta determinísticamente sobre los valores de las variables de P, entonces*

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

Demostración: Como S es una instrucción que se comporta determinísticamente sobre los valores de las variables de P, entonces S es una instrucción que se comporta determinísticamente sobre los valores de las variables de $\neg P$, entonces:

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

$\equiv$<Lema 8>

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

$\equiv$<Lema 10>

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

Lema 12. *Sean P y Q predicados y S una instrucción que no modifica los valores de las variables de P, entonces*

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

Demostración: Como S es una instrucción que no modifica los valores de las variables de P, entonces S es una instrucción que se comporta determinísticamente sobre las variables de P, entonces:

$wp(S, P \Rightarrow Q)$

$\equiv$<Lema 11>

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

$\equiv$<Lema 6>

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

Lema 13. *Sea P un predicado, S una instrucción que se comporta determinísticamente sobre los valores de las variables de P, y ϵ una variable no declarada en el programa, entonces*

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

Demostración: Si P depende de $\overline{x}$, $\overline{Y}$ y de las variables $y_{i_1}, \ldots, y_{i_k}$, entonces:

$wp(S, (\exists \epsilon | : P(\overline{x}, y_{i_1}, \ldots, y_{i_k}, \overline{Y})))$

$\equiv$<Lema 4>

$\vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge (\exists \epsilon | : P(\overline{x}, \mathcal{C}[\![S]\!]^{i_1}(\vec{x}), \ldots, \mathcal{C}[\![S]\!]^{i_k}(\vec{x}), \overline{Y}))$

$\equiv <$ ϵ no ocurre en el vector de variables declaradas $\vec{x} >$

$(\exists \epsilon | : \vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge P(\overline{x}, \mathcal{C}[\![S]\!]^{i_1}(\vec{x}), \ldots, \mathcal{C}[\![S]\!]^{i_k}(\vec{x}), \overline{Y}))$

$\equiv$<Lema 4>

$(\exists \epsilon | : wp(S, P(\overline{x}, y_{i_1}, \ldots, y_{i_k}, \overline{Y})))$ ■

Lema 14. *Sean P y R predicados, S una instrucción que se comporta determinísticamente sobre los valores de las variables de P y no modifica los valores de las variables de R, y ϵ una variable no declarada en el programa, entonces*

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

Demostración: Como S no modifica los valores de las variables de R, entonces S se comporta determinísticamente sobre los valores de las variables de R y como adicionalmente S se comporta determinísticamente sobre los valores de las variables de P, entonces S se comporta determinísticamente sobre los valores de las variables de $R \wedge P$.

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

$\equiv$<Lema 13>

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

$\equiv$<Lema 9>

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

Lema 15. *Sea S una instrucción, P un predicado y ϵ una variable no declarada en el programa, entonces*

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

Demostración: Dado que $\mathcal{C}[\![S]\!] \restriction_{supp(\mathcal{C}[\![S]\!])}$ es una relación, es fácil demostrar que la fórmula de $wp(S, P)$ del final de la sección anterior es equivalente a

$$(\overline{x}, \overline{y}) \in supp(\mathcal{C}[\![S]\!]) \wedge$$
$$(\forall \overline{y'}| : (\overline{x}, \overline{y'}) \in \mathcal{C}[\![S]\!] \restriction_{supp(\mathcal{C}[\![S]\!])} (\{(\overline{x}, \overline{y})\}) \Rightarrow$$
$$P(\overline{x}, \overline{y'}, \overline{Y}))$$

Como ϵ no ocurre en la lista de variables libres $\overline{x}, \overline{y}$, entonces de la fórmula $(\forall \epsilon| : wp(S, P))$ se puede sacar del $\forall \epsilon$ el lado izquierdo del $\wedge$, el $\forall \overline{y'}$ y el antecedente de $\Rightarrow$ quedando

$$(\overline{x}, \overline{y}) \in supp(\mathcal{C}[\![S]\!]) \wedge$$
$$(\forall \overline{y'}| : (\overline{x}, \overline{y'}) \in \mathcal{C}[\![S]\!] \restriction_{supp(\mathcal{C}[\![S]\!])} (\{(\overline{x}, \overline{y})\}) \Rightarrow$$
$$(\forall \epsilon| : P(\overline{x}, \overline{y'}, \overline{Y})))$$
$$\equiv$$
$$wp(S, (\forall \epsilon| : P))$$

■

Lema 16. *Sean P y R predicados, S una instrucción que no modifica los valores de las variables de R, y ϵ una variable no declarada en el programa. Si $(\exists \epsilon| : R) \equiv true$, entonces*

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

Demostración:
$wp(S, (\forall \epsilon | R : P))$
$\equiv$
$wp(S, (\forall \epsilon| : \neg R \vee P))$

$\equiv$<Lema 15>

$(\forall \epsilon| : wp(S, \neg R \vee P))$

$\equiv$<Lema 9>

$(\forall \epsilon| : support(S) \wedge (\neg R \vee wp(S, P)))$
$\equiv$
$support(S) \wedge (\forall \epsilon| : \neg R \vee wp(S, P))$
$\equiv$
$support(S) \wedge (\forall \epsilon | R : wp(S, P))$

$\equiv< (\exists \epsilon| : R) \equiv true >$

$(\forall \epsilon | R : support(S) \wedge wp(S, P))$

$\equiv$<Lema 7>

$(\forall \epsilon | R : wp(S, P))$ ■

Lema 17. *Sea R un predicado y S una instrucción que se comporta determinísticamente sobre los valores de las variables de R. Si i_f y ϵ son variables no declarada en el programa, entonces*

$$wp(S, \epsilon \neq (min\ i_f | R : i_f)) \equiv$$
$$support(S) \wedge \epsilon \neq (min\ i_f | wp(S, R) : i_f)$$

y

$$wp(S, \epsilon = (min\ i_f | R : i_f)) \equiv$$
$$support(S) \wedge \epsilon = (min\ i_f | wp(S, R) : i_f)$$

Demostración: $\epsilon \neq (min\ i_f | R : i_f)$ es equivalente a

$(\neg(\exists i_f| : R) \Rightarrow \epsilon \neq \infty) \wedge$
$((\exists i_f| : R) \Rightarrow \neg(R[i_f := \epsilon]) \vee (\exists i_f| : R \wedge i_f < \epsilon))$.

De modo que:

$wp(S, \epsilon \neq (min\ i_f | R : i_f))$

$\equiv< wp(S, P) \Rightarrow support(S)$ para cualquier $P >$

$support(S) \wedge wp(S, \epsilon \neq (min\ i_f | R : i_f))$
$\equiv$
$support(S) \wedge wp(S, (\neg(\exists i_f| : R) \Rightarrow \epsilon \neq \infty) \wedge$
$((\exists i_f| : R) \Rightarrow \neg(R[i_f := \epsilon]) \vee (\exists i_f| : R \wedge i_f < \epsilon)))$

$\equiv$<Lema 2>

$support(S) \wedge wp(S, \neg(\exists i_f| : R) \Rightarrow \epsilon \neq \infty) \wedge$
$wp(S, (\exists i_f| : R) \Rightarrow \neg(R[i_f := \epsilon]) \vee (\exists i_f| : R \wedge i_f < \epsilon))$

$\equiv$<Lema 11>

$support(S) \wedge (wp(S, \neg(\exists i_f| : R)) \Rightarrow wp(S, \epsilon \neq \infty)) \wedge$
$(wp(S, (\exists i_f| : R)) \Rightarrow wp(S, \neg(R[i_f := \epsilon]) \vee$
$(\exists i_f| : R \wedge i_f < \epsilon)))$

$\equiv$<Lema 8>

$support(S) \wedge (wp(S, \neg(\exists i_f| : R)) \Rightarrow wp(S, \epsilon \neq \infty)) \wedge$
$(wp(S, (\exists i_f| : R)) \Rightarrow wp(S, \neg(R[i_f := \epsilon])) \vee$
$wp(S, (\exists i_f| : R \wedge i_f < \epsilon)))$

$\equiv$<Lema 10>

$support(S) \wedge (\neg wp(S, (\exists i_f| : R)) \Rightarrow wp(S, \epsilon \neq \infty)) \wedge$
$(wp(S, (\exists i_f| : R)) \Rightarrow \neg wp(S, R[i_f := \epsilon]) \vee$
$wp(S, (\exists i_f| : R \wedge i_f < \epsilon)))$

$\equiv$<Lemas 13>

$support(S) \wedge (\neg(\exists i_f| : wp(S, R)) \Rightarrow wp(S, \epsilon \neq \infty)) \wedge$
$((\exists i_f| : wp(S, R)) \Rightarrow \neg wp(S, R[i_f := \epsilon]) \vee$
$(\exists i_f| : wp(S, R \wedge i_f < \epsilon)))$

$\equiv$<Lema 2>

$support(S) \wedge (\neg(\exists i_f| : wp(S, R)) \Rightarrow wp(S, \epsilon \neq \infty)) \wedge$

$((\exists i_f| : wp(S,R)) \Rightarrow \neg wp(S, R[i_f := \epsilon]) \vee$
$(\exists i_f| : wp(S,R) \wedge wp(S, i_f < \epsilon)))$

$\equiv$<Lema 6>

$support(S) \wedge (\neg(\exists i_f| : wp(S,R)) \Rightarrow \epsilon \neq \infty) \wedge$
$((\exists i_f| : wp(S,R)) \Rightarrow \neg wp(S, R[i_f := \epsilon]) \vee$
$(\exists i_f| : wp(S,R) \wedge i_f < \epsilon))$

$\equiv <$ S no modifica las variables $i_f, \epsilon >$

$support(S) \wedge (\neg(\exists i_f| : wp(S,R)) \Rightarrow \epsilon \neq \infty) \wedge$
$((\exists i_f| : wp(S,R)) \Rightarrow \neg (wp(S, R)[i_f := \epsilon]) \vee$
$(\exists i_f| : wp(S,R) \wedge i_f < \epsilon))$
$\equiv$
$support(S) \wedge \epsilon \neq (min\ i_f | wp(S,R) : i_f)$

Por otro lado

$wp(S, \epsilon = (min\ i_f | R : i_f))$
$\equiv$
$wp(S, \neg(\epsilon \neq (min\ i_f | R : i_f)))$

$\equiv$<Lema 10>

$support(S) \wedge \neg wp(S, \epsilon \neq (min\ i_f | R : i_f))$

$\equiv$<Resultado anterior>

$support(S) \wedge \neg(support(S) \wedge \epsilon \neq (min\ i_f | wp(S,R) : i_f))$
$\equiv$
$support(S) \wedge (\neg support(S) \vee \epsilon = (min\ i_f | wp(S,R) : i_f))$

$\equiv$<Absorción>

$support(S) \wedge \epsilon = (min\ i_f | wp(S,R) : i_f)$ ■

Lema 18. *Sea R un predicado y S una instrucción que se comporta determinísticamente sobre los valores de las variables de R. Si i_f y ϵ son variables no declarada en el programa, entonces*

$$wp(S, \epsilon \neq (max\ i_f | R : i_f)) \equiv$$
$$support(S) \wedge \epsilon \neq (max\ i_f | wp(S,R) : i_f)$$

y

$$wp(S, \epsilon = (max\ i_f | R : i_f)) \equiv$$
$$support(S) \wedge \epsilon = (max\ i_f | wp(S,R) : i_f)$$

Demostración: La demostración es análoga a la del Lema 17 pero sustituyendo la expresión $i_f < \epsilon$ por $i_f > \epsilon$. ■

Nota. *las demostraciones de estas propiedades con aserciones escritas en el lenguaje de la teoría de conjuntos de ZFS, implican la veracidad de las mismas en un fragmento de un lenguaje que incluya la teoría de ZFS y que sea cerrado sobre los operadores $\wedge$, $\vee$, $\neg$, $\Rightarrow$ $\forall$, $\exists$, min, max y $wp(S,.)$.*

Por ejemplo el lenguaje de las aserciones decidibles de [2] (aserciones donde conocer el valor de verdad, si se tienen todos los valores de las variables libres de la aserción, es decidible) es tal, que se puede extender para poder escribir todos los axiomas de ZFS consistentemente, a modo que el lenguaje de [2] pudiera entenderse, como un fragmento de ZFS. En este lenguaje de [2] las propiedades de esta sección son ciertas para toda instrucción S tal que $wp(S,.)$ sea cerrado, ya que el fragmento es cerrado sobre $\wedge$, $\vee$, $\neg$, $\Rightarrow$ $\forall$, $\exists$, min y max.

Para mostrar la afirmación de la nota anterior para el Lema 5 se supone que S es una instrucción que se comporta determinísticamente sobre las variables de P y Q, que son predicados escritos en el lenguaje de las aserciones decidibles de [2] y $wp(S,.)$ es cerrado en ese lenguaje. Como puedo escribir consistentemente los axiomas de ZFS con la sintaxis del lenguaje de [2], entonces este lenguaje puede entenderse como un fragmento de ZFS y por lo tanto el lema 5 es cierto obteniendo

$wp(S, P \vee Q)$

$\equiv$<Lema 5>

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

Como $wp(S,.)$ es cerrado sobre el lenguaje de [2], entonces $wp(S,P)$ y $wp(S,Q)$ son fórmulas del lenguaje de [2], y como dicho lenguaje es cerrado sobre el operador $\vee$, entonces $wp(S,P) \vee wp(S,Q)$ es una fórmula en el lenguaje de [2]. Esto demuestra que el lema 5 es cierto si se restringen las aserciones a este lenguaje y $w(S,.)$ es cerrado en el fragmento.

IV. Cerradura y Decidibilidad del Cálculo de wp

Del lenguaje y axiomatización de la teoría de conjuntos de ZFS no es directo que todo conjunto definido recursivamente exista, sin embargo todo conjunto que quiera definirse de forma recursiva, puede definirse con una fórmula en el lenguaje de ZFS que es equivalente a la recursión inicial. El proceso de conseguir la fórmula del lenguaje de primer orden de ZFS, que define equivalentemente el conjunto que inicialmente se encontraba definido recursivamente, se conoce como "meta-teorema de recursión transfinita". Dado una definición de un conjunto hecho de forma recursiva, dicho metateorema muestra de forma constructiva, cuál es la fórmula dentro del lenguaje de ZFS, que define al mismo conjunto.

Una demostración detallada del metateorema de recursión transfinita se encuentra en [23], sin embargo es más general de lo que se necesita para esta sección, ya que es valido para hacer recursión sobre cualquier ordinal. En esta sección se usará una versión de dicho teorema restringido a ω, cuyo enunciado es:

Teorema 1. *Si se tiene un predicado φ tal que satisface $(\forall k, F| : (\exists! y| : \varphi(k,F,y)))$. Definiendo $G(k,F)$ como el único y tal que $\varphi(k,F,y)$. Entonces se puede escribir una fórmula ψ donde lo siguiente es demostrable:*

1) $(\forall k| : (\exists! y| : \psi(k,y)))$*, es decir ψ define una función F tal que $\psi(k, F(k))$*
2) $(\forall k | k \in \omega| : F(k) = G(k, F \restriction_{k-1}))$

Una explicación verbosa del teorema anterior sería que la expresión $F(k) = G(k, F\restriction_{k-1})$ es una definición recursiva del conjunto $F(k)$ y la fórmula $\psi(k, y)$ es una versión en el lenguaje de ZFS, que define por comprensión un conjunto y que viene siendo igual $F(k)$. La idea de la demostración es la siguiente:

Demostración: Se define $\psi(k, y)$ como

$$(k \notin \omega \wedge y = \emptyset) \vee (k \in \omega \wedge (\exists d, h)(App(d, h) \wedge k \in d \wedge h(k) = y))$$

En donde $App(d, h)$ es un predicado definido como

$$esFuncion(h) \wedge$$
$$d = Dom(h) \subseteq \omega \wedge (\forall m)(m \in d \Rightarrow m - 1 \subseteq d) \wedge$$
$$(\forall m)(m \in d \Rightarrow \varphi(m, h\restriction_{m-1}, h(m)))$$

El resto de la demostración consiste en demostrar $(\exists! y| : \psi(k, y))$ por inducción fuerte y luego que la función $F(k)$ existe usando el axioma de reemplazo. ■

Nota. *Note que la demostración del teorema anterior dice que si se tiene la fórmula del predicado φ, entonces se tiene la fórmula para $\psi(k, y)$, que es una fórmula escrita en el lenguaje de primer orden de la teoría de conjuntos de ZFS.*

Teorema 2. *Si el lenguaje que se usa para escribir los predicados de las aserciones en GCL, es el lenguaje de primer orden de la teoría de conjuntos de Zermelo-Frankel-Skolem, entonces por cada caso particular de predicado $Post$, de expresión B_0 e instrucción S_0, existe una fórmula escrita en el lenguaje de primer orden de la teoría de conjuntos de Zermelo-Framkel-Skolem, que es equivalente a $wp(do\ B_0 \to S_0\ od, Post)$*

Demostración:
Se definen recursivamente las siguientes instrucciones

$$If := if B_0 \to S_0 [] \neg B_0 \to SKIP fi$$
$$Do_0 := if \neg B_0 \to SKIP fi$$
$$Do_{k+1} := If; Do_k.$$

Como la interpretación de una secuenciación de instrucciones es la composición de las interpretaciones, entonces la interpretación de la instrucción Do_k satisface la siguiente recurrencia:

$$\mathcal{C}[\![Do_0]\!] := \mathcal{C}[\![if \neg B_0 \to SKIP fi]\!]$$

$$\mathcal{C}[\![Do_{k+1}]\!] := \mathcal{C}[\![Do_k]\!] \circ \mathcal{C}[\![If]\!]$$

Por otro lado la fórmula definida por

$$\varphi(k, F, y) :=$$
$$(k = 0 \wedge y = \mathcal{C}[\![Do_0]\!]) \vee$$
$$(k \neq 0 \wedge k \in \omega \wedge esFuncion(F) \wedge y = F(k-1) \circ \mathcal{C}[\![If]\!]) \vee$$
$$(\neg(k = 0 \vee (k \neq 0 \wedge k \in \omega \wedge esFuncion(F))) \wedge y = F)$$

satisface que:

$$(\forall k, F| : (\exists! y| : \varphi(k, F, y))).$$

Si se define a $G(k, F)$ como el único y que satisface $\varphi(k, F, y)$, entonces por el teorema 1, se tiene que existe una formula ψ que satisface que:

1) $(\forall k| : (\exists! y| : \psi(k, y)))$, es decir ψ define una función F tal que $\psi(k, F(k))$
2) $(\forall k | k \in \omega| : F(k) = G(k, F\restriction_{k-1}))$

Como $G(k, F)$ en notación de llaves es la función a trozos

$$G(k, F) = \begin{cases} \mathcal{C}[\![Do_0]\!] & si \quad k = 0 \\ F(k-1) \circ \mathcal{C}[\![If]\!] & si \quad k \in \omega\ y \\ & \quad esFuncion(F) \\ F & sino \end{cases}$$

entonces la fórmula

$$(\forall k | k \in \omega| : F(k) = G(k, F\restriction_{k-1}))$$

es equivalente a la recurrencia que define a $\mathcal{C}[\![Do_k]\!]$ arriba y por lo tanto, $F(k)$ debe ser igual a $\mathcal{C}[\![Do_k]\!]$. Por esta razón, como se tiene que F es una función tal que $F(k)$ es el único valor en el que $\psi(k, F(k))$ es verdad, entonces cuando el predicado

$$\psi(k, R)$$

sea cierto, debe ocurrir que $R = \mathcal{C}[\![Do_k]\!]$.

Por otro lado usando las notaciones de [3], las cuales son: $\vec{x}$ para indicar el vector de constantes y variables declaradas en un algoritmo, Esp para indicar el espacio de estados del algoritmo y $Rgo_{\overline{Y}}$ para referirse al conjunto $\{\vec{x} \in Esp | Post(\vec{x}, \overline{Y})\}$, se demostró que

$$(\exists k | k \geq 0 : \mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}})$$

es un predicado equivalente a $wp(do\ B_0 \to S_0\ od, Post)$. Sin embargo, usando el predicado ψ se puede reescribir la fórmula anterior como

$$(\exists k | k \geq 0 : \psi(k, R) \wedge R(\{\vec{x}\}) \subseteq Rgo_{\overline{Y}})$$

la cual está escrita en el lenguaje de primer orden de la teoría de conjuntos. ■

Como puede observarse la fórmula de la precondición más débil de la demostración del teorema anterior, se consigue de forma constructiva, ya que el predicado $\psi(k, R)$ se extrae del metateorema de recursión transfinita, y como se tiene la fórmula explícita para φ, se puede construir la fórmula de $\psi(k, R)$ y por ende la fórmula de la precondición más débil dentro del lenguaje de primer orden de la teoría de conjuntos.

Por lo dicho en el párrafo anterior, este último teorema muestra que el cálculo de $wp(Do, .)$ para cualquier instrucción Do es decidible dentro del lenguaje de la teoría de conjuntos y por ende, como las reglas de cálculo de wp (que se enunciaron en la introducción) para las instrucciones distintas de Do, son aplicables directamente sobre lenguajes de primer orden, entonces el cálculo de $wp(S, .)$ para cualquier instrucción S es decidible sobre ZFS.

De la demostración se observa que la aserción $wp(Do, Post)$ que se extrae del teorema 2 es muy complicada, incluso la verificación del valor de verdad de dicha aserción teniendo los valores de las variables libres puede ser no decidible. Esto es debido a que el problema de la parada

se puede escribir equivalentemente, como el problema de verificar el valor de verdad de la precondición $wp(S, true)$ para valores iniciales de las variables y constantes del programa S, es decir, para no contradecir la indecidibilidad del problema de la parada, debe ocurrir que verificar el valor de verdad de la aserción $wp(S, true)$, no es decidible en general.

V. Conclusiones

El teorema de decidibilidad del cálculo de wp provee una alternativa para calcular precondiciones más débiles de forma automática, sin embargo estas precondiciones, en muchos casos, no serían una aserción decidible. Por esta razón el teorema sugiere una aplicación de software para el cálculo de wp que maneje aserciones no decidibles de forma simbólica.

La idea práctica es usar todas las técnicas conocidas para calcular precondiciones o invariantes, y en caso de que estas técnicas fallen en el cálculo de una aserción decidible, la aplicación arroja como última opción, la aserción resultante del teorema de decidibilidad aquí mostrado.

Una aplicación de este estilo trabajaría en muchas ocasiones con aserciones no decidibles, por lo que no tiene sentido manejar estas aserciones dentro del lenguaje de programación en cuestión, porque en estos casos las aserciones no serían programables. Por esta razón sugiero que una aplicación que haga uso del teorema de decibilidad aquí mostrado, maneje las aserciones como comentarios al código. Por ejemplo se pudiera desarrollar un IDE que inserte a modo de comentario, de forma automática la precondición más débil de cada instrucción. Un IDE de este tipo siempre puede computar una aserción entre todas las instrucciones del programa, aunque algunas de ellas sean no decidibles, pudiera ser provechoso, porque es posible que en el cálculo sucesivo de wp, las aserciones se simplifiquen y se obtenga, al final del proceso, una precondición más débil de todo el programa, que sea una aserción decidible.

Por otro lado desde el punto de vista teórico, los resultados aquí presentados muestran que la teoría de wp de Dijkstra es aplicable sobre la teoría de conjuntos, y por ende también el teorema de la invariancia y todas las reglas de Hoare derivadas de wp. De esta forma se pueden usar las técnicas de corrección formal sobre algoritmos con tipos de datos pertenencientes a la teoría de conjuntos. Por ejemplo, se puede usando wp o reglas de Hoare, corregir algoritmos donde los tipos de las variables son objetos como ordinales, cardinales, filtros, ultrafiltros, espacios topológicos, etc.

Referencias

[1] E. W. Dijkstra. *Guarded Commands, Nondeterminacy and Formal Derivation of Programs*. Communications of the ACM, 18(8):453457, 1975.

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

[3] F. Flaviani. *Modelo Relacional de la Teoría Axiomática del Lenguaje GCL de Dijkstra*. CoNCISa 2015, Valencia, Venezuela, Noviembre 2015, pp. 153-164.

[4] F. Flaviani. *Cálculo de Precondiciones Más Débiles*. ReVeCom, Diciembre 2016, Vol. 3, No. 2, pp. 68-80.

[5] F. Flaviani. *Calculation of Invariants Assertions*. CLEI, Septiembre 2017, http://www.clei2017-46jaiio.sadio.org.ar/sites/default/files/Mem/SLTC/sltc-04.pdf

[6] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993.

[7] J. Berdine, A. Chawdhary, B. Cook, D. Distefano, and P. OHearn. *Variance Analyses from Invariance Analyses*. Proceedings of the 34th Annual Symposium on Principles of Programming Languages, Nice, France, 2007.

[8] E. Rodriguez Carbonnell and D. Kapur. *Program Verification using Automatic Generation of Invariants*. International Conference on Theoretical Aspects of Computing, 2004, Vol. 3407, pp. 325340.

[9] J. Carette and R. Janicki. *Computing Properties of Numeric Iterative Programs by Symbolic Computation*. Fundamentae Informatica, 80(1-3):125146, March 2007.

[10] M. A. Colon, S. Sankaranarayana, and H. B. Sipma. *Linear Invariant Generation using non Linear Constraint Solving*. Computer Aided Verification, 2003, Vol. 2725, pp. 420432.

[11] M. D. Ernst, J. H Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao. *The Daikon System for Dynamic Detection of Likely Invariants*. Science of Computer Programming, 2006.

[12] J.C. Fu, F. B. Bastani, and I-L. Yen. *Automated Discovery of Loop Invariants for High Assurance Programs Synthesized using ai Planning Techniques*. HASE 2008: 11th High Assurance Systems Engineering Symposium, 2008, pp. 333342, Nanjing, China.

[13] L. Kovacs and T. Jebelean. *Automated Generation of Loop Invariants by Recurrence Solving in Theorema*. In D. Petcu, V. Negru, D. Zaharie, and T. Jebelean, editors, Proceedings of the 6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC04), pages 451464, Timisoara, Romania, 2004. Mirton Publisher.

[14] L. Kovacs and T. Jebelean. *An Algorithm for Automated Generation of Invariants for Loops with Conditionals*. D. Petcu, editor, Proceedings of the Computer-Aided Verification on Information Systems Workshop (CAVIS 2005), 7th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2005), pages 1619, Department of Computer Science, West University of Timisoara, Romania, 2005.

[15] S. Sankaranarayana, H. B. Sipma, and Z. Manna. *Non Linear Loop Invariant Generation Using Groebner Bases*. In Proceedings, ACM SIGPLAN Principles of Programming Languages, POPL 2004, pages 381329, 2004.

[16] A. Gupta, A. Rybalchenko. *InvGen: An Efficient Invariant Generator*. International Conference on Computer Aided Verification, 2009, pp. 634-640.

[17] Stanford Invariant Generator, 2006, http://theory.stanford.edu/ srirams/Software/sting.html

[18] E. Rodrguez-Carbonell and D. Kapur. *Generating all Polynomial Invariants in Simple Loops*. J. Symbolic Comput. 42 (2007), no. 4, 443476.

[19] S. Magill, A. Nanevski, E. Clarke, and P. Lee. *Inferring Invariants in Separation Logic for Imperative List-processing Programs*. SPACE, 1(1):57, 2006.

[20] J. Berdine, B. Cook, and S. Ishtiaq, *SLAyer: Memory Safety for Systems-Level Code*. Gopalakrishnan, Springer, Heidelberg 6806:178183, 2011.

[21] C. Varming, L. Birkedal. *Higher-order Separation Logic in Isabelle/holcf*. Electronic Notes in Theoretical Computer Science, 218:371389, 2008.

[22] M. Barnett, K. Rustan, and M. Leino, Microsoft Research. *Weakest-Precondition of Unstructured Programs*. Proceedings of the 6th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, 31(2006):1, pp. 82-87.

[23] K. Kunen. *Set Theory*. College Publications, London, UK, 2013.