Revista Venezolana de Computación
ISSN: 2244-7040
http://www.svc.net.ve/revecom
Vol. 4, No. 2, pp. 46-58, Diciembre 2017
Selección de los Mejores Artículos de CoNCISa 2017

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

Federico Flaviani[1]
fflaviani@usb.ve

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

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.

Abstract: In this paper, new algebraic properties of the predicate transformer wp are presented on the operators $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\forall$, $\exists$, min and max, demonstrated independently of the programming language, using general properties of the denotational semantics of the languages. Additionally, a result that speaks about the decidability and closure of the predicate transformer wp on the GCL programming language with assertions written in the Zermelo-Frankel-Skolem set theory language is shown. In this paper it is shown that calculating wp of a GCL statement and a written assertion in the ZFS language is decidable and is another written assertion in ZFS. It can not necessarily decide the truth value of said assertion resulting from calculating wp, even if all the values of your free variables are given, so that the result does not contradict the undecidability of the halting problem.

Keywords: Weakest Precondition; Denotational Semantics; Decidability; 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

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:*

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

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:*

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

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 $(\overline{x}, \overline{y})$.*

Por ejemplo en la fórmula $Post(\vec{x}, \overline{Y})$, la notación $\vec{x}$ se interpreta como lista, queriendo decir $Post(\overline{x}, \overline{y}, \overline{Y})$, en cambio en una fórmula como $\vec{x} \in Esp$, la notación $\vec{x}$ se interpreta como tupla, queriendo decir $(\overline{x}, \overline{y}) \in Esp$.

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

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.

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

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:

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

$if\ a > -3 \rightarrow$
$\quad b := b/a$
$[]\ a \leq -3 \rightarrow$
$\quad b := 2$
fi,

entonces

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

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>

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

$\Rightarrow$<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>

$P \wedge wp(S, Q)$

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 es una variable no declarada en el programa y ϵ es una expresión en donde S no modifica los valores de sus variables, 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>

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

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 válido 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

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

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. Toda ocurrencia de un conjunto $F(k)$ definido recursivamente en algún predicado $P(\dots, F(k), \dots)$ dentro de una fórmula de ZFK, se entiende como una abreviación de la fórmula $(\forall R | \psi(k, R) : P(\dots, R, \dots))$*

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 en [3], se demostró que

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

es un predicado equivalente a $wp(do\ B_0 \rightarrow S_0\ od, Post)$. Sin embargo, usando el predicado ψ, la fórmula anterior dentro de ZFS es una abreviación de

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

Como $\psi(k, R)$ esta definida como

$$(k \notin \omega \wedge R = \emptyset) \vee$$

$$(k \in \omega \wedge (\exists d, Do | App(d, Do) : k \in d \wedge Do(k) = R))$$

entonces la fórmula anterior se puede escribir equivalentemente como

$$(\exists k | k \in \omega \wedge k \geq 0 : (\exists d, Do | App(d, Do) : k \in d \wedge Do(k)(\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}))$$

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.

Un ejemplo de uso del teorema 2 para mostrar la complejidad que pueden tener las aserciones calculadas es el siguiente, en donde el invariante del ciclo es calculado con la fórmula de wp para Do del teorema 2:

$[Var\ N, i, s :\ Enteros;$
$\quad s, i := 0, 0;$
$\quad \{Inv : (\exists k | k \in \omega \wedge k \geq 0 : (\exists d, Do | App(d, Do) :$
$\qquad\qquad k \in d \wedge Do(k)(\{(N, i, s)\}) \subseteq Rgo_{\overline{Y}}))\}$
$\quad do\ i \neq N \rightarrow$
$\qquad s, i := s + i, i + 1$
$\quad od$
$\{Post :\ s = \sum_{j=0}^{N-1} j\}$
$]$

Como las únicas variables libres que tiene Inv son N, i y s y el ciclo esta precedido de una instrucción de asignación, podemos calcular la precondición más débil de todo el programa aplicando la regla de wp de la asignación obteniendo

$$(\exists k | k \in \omega \wedge k \geq 0 : (\exists d, Do | App(d, Do) : k \in d \wedge Do(k)(\{(N, i, s)\}) \subseteq Rgo_{\overline{Y}}))[s, i := 0, 0]$$
$$\equiv$$
$$(\exists k | k \in \omega \wedge k \geq 0 : (\exists d, Do | App(d, Do) : k \in d \wedge Do(k)(\{(N, 0, 0)\}) \subseteq Rgo_{\overline{Y}}))$$
$\equiv$<Definición de $Rgo_{\overline{Y}}$ >
$$(\exists k | k \in \omega \wedge k \geq 0 : (\exists d, Do | App(d, Do) : k \in d \wedge Do(k)(\{(N, 0, 0)\}) \subseteq \{\langle N, i, s\rangle \in \mathbb{Z}^3 | s = \sum_{j=0}^{N-1} j\})))$$
$\equiv$<Definición de App >
$$(\exists k | k \in \omega \wedge k \geq 0 : (\exists d, Do | esFuncion(Do) \wedge d = Dom(Do) \subseteq \omega \wedge (\forall m | m \in d : m - 1 \subseteq d) \wedge (\forall m | m \in d : \varphi(m, Do \restriction_{m-1}, Do(m))) : k \in d \wedge Do(k)(\{(N, 0, 0)\}) \subseteq \{\langle N, i, s\rangle \in \mathbb{Z}^3 | s = \sum_{j=0}^{N-1} j\})))$$
$\equiv$<Definición de φ >
$$(\exists k | k \in \omega \wedge k \geq 0 : (\exists d, Do | esFuncion(Do) \wedge d = Dom(Do) \subseteq \omega \wedge (\forall m | m \in d : m - 1 \subseteq d) \wedge (\forall m | m \in d : (m = 0 \wedge Do(m) = \mathcal{C}[\![Do_0]\!]) \vee (m > 0 \wedge Do(m) = Do \restriction_{m-1} (m - 1) \circ \mathcal{C}[\![If]\!]) : k \in d \wedge Do(k)(\{(N, 0, 0)\}) \subseteq \{\langle N, i, s\rangle \in \mathbb{Z}^3 | s = \sum_{j=0}^{N-1} j\})))$$

En donde $\mathcal{C}[\![Do_0]\!]$ y $\mathcal{C}[\![If]\!]$ se pueden calcular usando la definición de la semántica denotacional de la instrucción if de GCL definida en [3], con lo que se tiene que

$$\mathcal{C}[\![Do_0]\!] \equiv id_{\{\langle N,i,s\rangle \in \mathbb{Z}^3 | i = N\}} \cup (\{\langle N, i, s\rangle \in \mathbb{Z}^3 | i \neq N\} \times \{abort\}) \cup \{\langle abort, abort\rangle\}$$

y

$$\mathcal{C}[\![If]\!] \equiv \{\langle\langle N, i, s\rangle, \langle N', i', s'\rangle\rangle \in \mathbb{Z}^3 \times \mathbb{Z}^3 | \langle N', i', s'\rangle = \langle N, i + 1, s + i\rangle\} \circ id_{\{\langle N,i,s\rangle \in \mathbb{Z}^3 | i \neq N\}}) \cup id_{\{\langle N,i,s\rangle \in \mathbb{Z}^3 | i = N\}} \cup \{\langle abort, abort\rangle\}$$

El cálculo de esta aserción se hizo aplicando directamente las reglas de wp y las definiciones de la semántica denotacional de GCL, lo cual es automatizable, sin embargo la complejidad y tamaño de la aserción es considerable.

Nota. *Como una muestra de lo innecesariamente compleja que puede resultar una aserción calculada usando el teorema 2, observe que se puede demostrar por inducción sobre N, que esta última precondición más débil calculada, es equivalente a*

$0 \leq N$, *que es un predicado mucho más simple. Esto muestra que es recomendable usar el teorema 2 sólo cuando no se puede calcular la precondición más débil del programa con otras técnicas.*

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 aserciones decidibles. 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, vol. 18, no. 8, pp. 453-457, 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*, en las memorias de la Conferencia Nacional de Computación, Informática y Sistemas (CoNCISa 2015), Valencia, Venezuela, Noviembre 2015, pp. 153-164.

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

[5] F. Flaviani. *Calculation of Invariants Assertions*, en las memorias de la XLIII Conferencia Latinoamericana en Informática (CLEI 2017), Córdoba, Argentina, Septiembre 2017.

[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*, in proceedings of the 34th Annual Symposium on Principles of Programming Languages, Nice, France, January 2007.

[8] E. Rodríguez Carbonnell and D. Kapur. *Program Verification using Automatic Generation of Invariants*, in proceedings of the First International Colloquium on Theoretical Aspects of Computing, Guiyang, China, September 2004.

[9] J. Carette and R. Janicki. *Computing Properties of Numeric Iterative Programs by Symbolic Computation*. Fundamentae Informatica, vol. 80, no. 1, pp. 125-146, March 2007.

[10] M. A. Colon, S. Sankaranarayana, and H. B. Sipma. *Linear Invariant Generation using non Linear Constraint Solving*. Computer Aided Verification, vol. 2725, pp. 420-432, 2003.

[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*, in proceedings of the 11th High Assurance Systems Engineering Symposium (HASE 2008), pp. 333-342, Nanjing, China, December 2008.

[13] L. Kovacs and T. Jebelean. *Automated Generation of Loop Invariants by Recurrence Solving in Theorema*, in proceedings of proceedings of the 6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2004), pp. 451-464, Timisoara, Romania, September 2004.

[14] L. Kovacs and T. Jebelean. *An Algorithm for Automated Generation of Invariants for Loops with Conditionals*, in 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), pp. 16-19, Timisoara, Romania, September 2005.

[15] S. Sankaranarayana, H. B. Sipma, and Z. Manna. *Non Linear Loop Invariant Generation Using Groebner Bases*, in proceedings of the ACM SIGPLAN Symposium on Principles of Programming Languages (POLP 2004), pp. 381-329, Venice, Italy, January 2004.

[16] A. Gupta and A. Rybalchenko. *InvGen: An Efficient Invariant Generator*, in proceedings of the International Conference on Computer Aided Verification (CAV 2009), Grenoble, France, June 2009.

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

[18] E. Rodríguez-Carbonell and D. Kapur. *Generating all Polynomial Invariants in Simple Loops*. Journal Symbolic Computation, vol. 42, no. 4, pp. 443-476, April 2007.

[19] S. Magill, A. Nanevski, E. Clarke, and P. Lee. *Inferring Invariants in Separation Logic for Imperative List-processing Programs*, in proceedings of the Third Workshop on Semantics, Program Analysis, and Computing Environments for Memory Management (SPACE 2006), Charleston, SC, USA, January 2006.

[20] J. Berdine, B. Cook, and S. Ishtiaq, *SLAyer: Memory Safety for Systems-Level Code*, in proceedings of the 23rd International Conference on Computer Aided Verification (CAV 2011), Snowbird, UT, USA, July 2011.

[21] C. Varming and L. Birkedal. *Higher-order Separation Logic in Isabelle/HOLCF*. Electronic Notes in Theoretical Computer Science, vol. 218, pp. 371-389, October 2008.

[22] M. Barnett, K. Rustan, and M. Leino, Microsoft Research. *Weakest-Precondition of Unstructured Programs*, in proceedings of the 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, (PASTE 2005), pp. 82-87, Lisbon, Portugal, September 2005.

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