Inference of the Definition of the Predicate Transformer wp with Occurrences of the Predicate $domain$ Based on Denotational Semantics of GCL on ZF Set Theory

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

***Abstract*—Dijkstra recursively defined the predicate transformer wp. Then Gries for each expression Exp of the language, defined $domain(Exp)$, which is a predicate that indicates the states in which Exp is defined. This predicate Gries added it to the recursive formula that defines wp for assignment, and subsequently other authors added it to the rule that recursively defines wp for IF, so that in the bibliography there are several versions of the definition of wp, with and without occurrence of $domain$. The present work shows an inference of the definition of wp, demonstrating that the occurrence of $domain$ is necessary for wp in assignement, IF and DO. This inference is done through the GCL denotational semantics over the set theory ZF, showing that the classical formulas of Dijkstra to define wp in GCL using $domain$, are valid if the language of set theory is used to write the assertions.**

I. Introduction

The Dijkstra [1] logic for program correction is based on the predicate transformer wp (weakest precondition), which is basically a syntactic function of two variables that returns in a symbolic way the weakest precondition of an statement $inst$ given a postcondition $Post$.

Dijkstra established in [1] the rules that define the syntactic transformation function wp. In this paper an improvement to these rules is given, and justified using a denotational semantics of GCL (Guard Command Language) that contemplates the $abort$ state. The obtained definition of wp is found in the conclusions section of this work.

The main idea of the study is to identify in which rules of the construction of wp, the syntactic function $domain$ must be used, which applies to expressions and returns a predicate that describes the domain of said expression.

The use of $domain$ in these rules is a way to discriminate states that can cause the program to abort because of an illegal operation such as dividing by zero. With the syntactic function $domain$ you can ensure that assertions are fully assessable, that is, that they can be evaluated as true or false, for any state of the program, or equivalently, that there are no program states in which the predicates in the assertions are indefinite.

For example the assertion $\{y = \frac{1}{x}\}$ is undefined in the state $x, y := 0, 1$, so it is not a fully assessable assertions. But as $domain(\frac{1}{x}) \equiv x \neq 0$, and if $\wedge$ is the Boolean operator 'and' with a short circuit, the assertion $\{domain(\frac{1}{x}) \wedge y = \frac{1}{x}\}$ is totally assessable.

When assertions are used within a program, it is done with the intention that if the program aborts at the height of an assertion, it is because the assertion is false when evaluated in a running state, and not because the assertion itself, when evaluated, incurs in an illegal or undefined evaluation. For this reason the use of fully assessable assertions is recommended.

In most of the works on the wp transformer (classic or recent), the recursive rules that define wp differ slightly from those obtained in this work in the DO rule. Specially with regards to the occurrences of $domain$ i.e. in most of the literature, it is only used in the assignment statement rule. This is because in these works it is permissible to use assertions that are not fully assessable. Although this is not incorrect, it is unadvisable, because it is easy to make mistakes. For example, the triplet of Hoare [2] below is correct for states in which $x = 0$:

$\{Pre : \frac{1}{x} = \frac{3}{x^2}\}$
$y := 1/x$
$\{Post : \frac{1}{x} = \frac{3}{x} * y\}$

since Pre cannot be evaluated as true for these states because the Pre assertion is undefined. However, an error is made in pretending to write equivalently the precondition by clearing and obtaining $\{PreAux : x^2 = 3x\}$, which in turn is equivalent to $\{PreAux : x = 0 \vee x = 3\}$. But in the states where $x = 0$, although the precondition $PreAux$ would be true, the statement $y := 1/x$ would abort, so the triplet using $PreAux$ as a precondition is false. This error would not have been possible if fully assessable assertions had been used. For instance,

$\{Pre : x \neq 0 \wedge \frac{1}{x} = \frac{3}{x^2}\}$
$y := 1/x$
$\{Post : x \neq 0 \wedge \frac{1}{x} = \frac{3}{x} * y\}$

if the same clearance is made on the precondition, there would be $\{PreAux : x \neq 0 \wedge (x = 0 \vee x = 3)\}$ which is equivalent to $\{PreAux : x = 3\}$.

In this paper it is shown that if we want to use fully assessable assertions, it is necessary to use the syntactic function $domain$ in the recursive rules of wp, for the assignment, selection and iteration statements.

A. Contribution

In this work we find a demonstration based on denotational semantics of the calculation rules of wp with the proper use of the syntactic function $domain$. This is done in order to guarantee that if the P assertion is totally assesable then so also is $wp(S, P)$. The latter cannot be guaranteed with any of the definitions of wp found in the bibliography listed here, which includes recent works.

On the other hand, all the work is done assuming that the assertions of the programs are written in Zermelo-Fraenkel-Skolem set theory (ZFS), but with the notation used for the quantifiers $\forall$ and $\exists$ of [3]. The data types of the GCL variables are interpreted as objects of the ZFS set theory, and the algorithms in GCL are interpreted as relationships on these objects. For this reason, the theory here developed for the wp transformer is as general as the set theory, and ensures that algorithm correction and calculation of wp can be made on algorithms with data types as general as the topological spaces, ultrafilters, non-measurable sets, etc. For this reason the semantics here defined constitute a way to embed the theory of algorithms and program correction within ZFS.

The theory of wp for GCL written with assertions in other languages (such as the language of decidable assessable assertions of the books [4], [5]), can be considered as a fragment of ZFS, so they are contemplated within this theory. That is, as long as they are closed fragments with respect to the recursive calculation rules of wp as stated above.

B. Related Works

Originally in [1] the definition of wp did not include the function $domain$ in its rules. The first time that $domain$ was used as an improvement was in [5], where it is incorporated into the wp rule for assignment, but he does not extend its formulation for the IF nor for the DO rules, which is necessary, if you want fully assessable assertions. Then came [6], [7], [8], [9] which extended the use of $domain$ to the IF statement. In none of them however, has it been applied to the DO statement. Hence, we have not seen a complete definition of wp with the purpose of making fully assessable assertions. In [10], [11] use of $domain$ is made in the wp thread for the standard Do, but no mention is made of what the rule is for the DO with multiple guards. For this reason it is important to revisit the topic and make a formal derivation of these rules from non-axiomatic semantics, in order to validate and have a solid version of the definition of wp with the correct and total use of $domain$.

The wp rule for the statement DO has not been frequently used, because it presupposes solving a recurrence of predicates, which is complicated, and there are other more famous methods to correct a loop, such as the use of an invariant and bound function. This lack of practical use of the wp rule for the DO, could explain the lack of bibliography with the correct use of $domain$ with this rule. However, methods to calculate $wp(Do, P)$ explicitly, it has been investigated recently. One of the methods [12]-[15], is using invariant relations and another method in [10], [11] by calculating wp of the cycle body repeatedly to get a general predicate. For the method of invariant relations fully assessable assertions are not used, so $domain$ is not used at all. The second method contemplates the use of $domain$ as part of the technique, which makes it more accurate and secure. This recent interest for $domain$ and the calculation of $wp(Do, P)$, makes it an imperative to study in detail, the solid arguments that guarantee what is the final rule of $wp(Do, P)$, using $domain$.

On the other hand in the classic literature of denotational semantics for programming languages such as [16], written assertions are used in the language of Arithmetic to derive the formulas of wp. However, with the recent interest in the method of invariant relations, for which it is fundamental to write assertions in the language of relations theory, it is relevant to develop a theory of wp, directly in the language of set theory, even though there are already general foundations for the correction of programs such as [17], since the latter is a logical framework and not semantic like this. This work is a continuation of [18], where a general denotational semantics (based on set theory) is established, to support the theory of wp and demonstrate algebraic properties of this transformer, for any programming language and not specifically GCL.

II. General Denotational Semantics

Notation. *The minimum and maximum elements of a Boolean algebra $\{\hat{0}, \hat{1}\}$, are considered objects within ZFS (sets). We denote $TRUE$ and $FALSE$ as true and false of the ZFS metatheory, that is, the adjectives that can be attributed to a ZFS sentence, given an ideal model (if it exists) of ZFS. We denote $true$ and $false$ as predicates that can only be interpreted as $TRUE$ and $FALSE$ respectively.*

Definition (Space of States of an Algorithm)**.** *Let the algorithm*

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

Where S is the statements of the algorithm, $\overline{x}$ is the list of constants and $\overline{T}$ is the list of types of each $\overline{x}$ (where each type is a set), $\overline{y}$ is the list of variables and $\overline{T'}$ is the list of types of each $\overline{y}$.

If $\overline{T} = T_1, T_2, \ldots, T_n$ and $\overline{T'} = T_{n+1}, T_{n+2}, \ldots, T_{n'}$, then the space of states of the previous algorithm is defined as

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

Notation. *In an algorithm with space of states Esp, $\vec{x}$ denotes the list of constants and variables that are in the order in*

which they were declared. That is, $\vec{x} = \overline{x}||\overline{y}$, where $\overline{x}$ and $\overline{y}$ are the lists of constants and variables in the space of states definition, and $||$ is the operator of concatenation of lists.

Notation. *To simplify the notation, the vector $(\vec{x}) = (\overline{x}, \overline{y})$ is simply denoted as $\vec{x}$, so the notation $\vec{x}$ can be understood according to context as a list of syntactic variables $\overline{x}||\overline{y}$, or as a tuple $(\overline{x}, \overline{y})$.*

For example, in the formula $Post(\vec{x}, \overline{Y})$, the notation $\vec{x}$ is interpreted as a list, meaning $Post(\overline{x}, \overline{y}, \overline{Y})$, in contrast to a formula like $\vec{x} \in Esp$, the notation $\vec{x}$ is interpreted as tuple, meaning $(\overline{x}, \overline{y}) \in Esp$.

Definition. *Let an algorithm with space of states Esp and take an element $abort \notin Esp$, then define the space of states extended to the abort as $Esp' := Esp \cup \{abort\}$*

An algorithm in addition to the description of the space of states, consists of phrases of the language that are called statements, and within the statements there are other phrases called expressions of type T (where T is a set). These phrases are denotationally interpreted with the interpretation function $\mathcal{E}$. This interpretation function takes an expression Exp syntactically speaking and returns a function with range in T, which is denoted $\mathcal{E}[\![Exp]\!]$.

Definition. *In an algorithm with space of states Esp, an expression Exp of type T, is a phrase of the language that is interpreted as a function*

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

Notation. *From now on, in order to abbreviate, the interpretation of the expression Exp evaluated in $\vec{x}$ will be written as $Exp(\vec{x})$, instead of $\mathcal{E}[\![Exp]\!](\vec{x})$ and we will denote the domain of the expression for $Dom(Exp)$ instead of $Dom(\mathcal{E}[\![Exp]\!])$.*

Remark. *Do not confuse a Boolean expression with a ZFS predicate, the first one returns an element in the Boolean algebra $\{\hat{0}, \hat{1}\}$, and the second is a phrase interpreted as either $TRUE$ or $FALSE$ in the ZFS metatheory.*

Remark. *The semantic definitions of this section are general, for this reason any possibility of syntax of the expressions and value of its interpretation function is left open.*

Notation. *Given an expression Exp inside an algorithm with space of states Esp, the $domain(Exp)$ is defined by a formula with free variables $\vec{x}$, such that*

$$Dom(\mathcal{E}[\![Exp]\!]) = \{\vec{x} \in Esp | domain(Exp)\}$$

Analogously if B is a predicate, $domain(B)$ is defined as the conjunction of the $domain$ of all the expressions that occur in B. It will define,

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

and if $Exp_1, \ldots, Exp_n$ are expressions or predicates, it will define

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

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

The concept of program statement and interpretation is defined below. To give denotational semantics to the statements, we will use an interpretation function $\mathcal{C}$, that receives an S statement, syntactically speaking, and returns an interpretation $\mathcal{C}[\![S]\!]$, Which is nothing more than a set theoretical relation.

Definition. *In an algorithm with a space of states Esp, an statement S is a phrase of the language that is interpreted as a relation*

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

such that the domain of $\mathcal{C}[\![S]\!]$ is all Esp',

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

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

Executing this instruction for initial values $\vec{x_0}$ is understood as evaluating the previous relation in the values $\vec{x_0}$, where the output of the execution could be any image of the point $\vec{x_0}$.

Additionally, support $supp(\mathcal{C}[\![S]\!])$ is defined, as the set of states that do not result in an $abort$ when executing the instruction, that is,

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

Remark. *Note that since the domain of $\mathcal{C}[\![S]\!]$ is all Esp' then, $\mathcal{C}[\![S]\!](\{\vec{x}\}) \neq \emptyset$ for all $\vec{x} \in Esp'$.*

Definition. *If $\overline{x}$ and $\overline{y}$ are the list of constants and variables of an algorithm A in the order in which they were declared, $\overline{Y}$ a list of variables other than $\overline{x}$, $\overline{y}$ and $Post(\overline{x}, \overline{y}, \overline{Y})$ a predicate that only has as free variables $\overline{x}$, $\overline{y}$ and $\overline{Y}$, then it will define the family of sets*

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

Remark. *Since $Rgo_{\overline{Y}} := \{\vec{x} \in Esp | Post(\vec{x}, \overline{Y})\}$, then to avoid redundancy in the predicates that serve as a postcondition of a statement, it will agree on the assertions of GCL, that the predicate $\vec{x} \in Esp$ does not appear, and this predicate will always be taken as an assumed hypothesis.*

Definition. *Let R be a relation of $A \times B$ and a subset $Rgo \subseteq B$, then*

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

is referred to as "maximum domain of R with range Rgo".

Lemma 1. *Given an algorithm with space of states Esp, a statement S and a predicate $Post(\vec{x}, \overline{Y})$, with $\vec{x} \in Esp$ and $\overline{Y_0}$ list of values of the same type as $\overline{Y}$, then the maximum domain of $\mathcal{C}[\![S]\!]$ and $Rgo_{\overline{Y_0}}$ is equal to*

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

Where $Post(y, \overline{Y_0})$ is a simplified notation that means

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

and $\mathcal{C}[\![S]\!] \restriction_{supp(\mathcal{C}[\![S]\!])}$ is the relation $\mathcal{C}[\![S]\!]$ restricted to domain $supp(\mathcal{C}[\![S]\!])$.

Proof. A demo style will be used as in [19]

$$\{\vec{x} \in Esp' | \mathcal{C}[\![S]\!](\{\vec{x}\}) \neq \emptyset \wedge \mathcal{C}[\![S]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y_0}}\}$$
$$\overset{\mathcal{C}[\![S]\!] \text{ is defined in all } Esp'}{=}$$
$$\{\vec{x} \in Esp' | \mathcal{C}[\![S]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y_0}}\}$$
$$=$$
$$\{\vec{x} \in Esp' | (\forall y| : y \in \mathcal{C}[\![S]\!](\{\vec{x}\}) \Rightarrow y \in Rgo_{\overline{Y_0}})\}$$
$$=$$
$$\{\vec{x} \in Esp' | (\forall y| : y \in \mathcal{C}[\![S]\!](\{\vec{x}\}) \Rightarrow y \in Esp \wedge Post(y, \overline{Y_0}))\}$$
$$=$$
$$\{\vec{x} \in Esp' | (\forall y| : (y \in \mathcal{C}[\![S]\!](\{\vec{x}\}) \Rightarrow y \in Esp) \wedge (y \in \mathcal{C}[\![S]\!](\{\vec{x}\}) \Rightarrow Post(y, \overline{Y_0})))\}$$
$$=$$
$$\{\vec{x} \in Esp' | (\forall y| : y \in \mathcal{C}[\![S]\!](\{\vec{x}\}) \Rightarrow y \in Esp) \wedge (\forall y| : y \in \mathcal{C}[\![S]\!](\{\vec{x}\}) \Rightarrow Post(y, \overline{Y_0}))\}$$
$$=$$
$$\{\vec{x} \in Esp' | \mathcal{C}[\![S]\!](\{\vec{x}\}) \subseteq Esp \wedge (\forall y| : y \in \mathcal{C}[\![S]\!](\{\vec{x}\}) \Rightarrow Post(y, \overline{Y_0}))\}$$
$$=$$
$$\{\vec{x} \in Esp' | abort \notin \mathcal{C}[\![S]\!](\{\vec{x}\}) \wedge (\forall y| : y \in \mathcal{C}[\![S]\!](\{\vec{x}\}) \Rightarrow Post(y, \overline{Y_0}))\}$$
$$\overset{def}{=}$$
$$\{\vec{x} \in Esp' | \vec{x} \in Esp \wedge \vec{x} \in Esp \wedge abort \notin \mathcal{C}[\![S]\!](\{\vec{x}\}) \wedge (\forall y| : y \in \mathcal{C}[\![S]\!](\{\vec{x}\}) \Rightarrow Post(y, \overline{Y_0}))\}$$
$$=$$
$$\{\vec{x} \in Esp' | \vec{x} \in Esp \wedge \vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge (\forall y| : y \in \mathcal{C}[\![S]\!](\{\vec{x}\}) \Rightarrow Post(y, \overline{Y_0}))\}$$
$$=$$
$$\{\vec{x} \in Esp | \vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge (\forall y| : y \in \mathcal{C}[\![S]\!] \restriction_{supp(\mathcal{C}[\![S]\!])} (\{\vec{x}\}) \Rightarrow Post(y, \overline{Y_0}))\} \qquad \square$$

Remark. *In the demonstration of the lemma it is observed that the maximum domain of $\mathcal{C}[\![S]\!]$ and $Rgo_{\overline{Y_0}}$ is equal to*

$$\{\vec{x} \in Esp' | \mathcal{C}[\![S]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y_0}}\}$$

which in turn is equal to
$\{\vec{x} \in Esp | \vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge (\forall y| : y \in \mathcal{C}[\![S]\!] \restriction_{supp(\mathcal{C}[\![S]\!])} (\{\vec{x}\}) \Rightarrow Post(y, \overline{Y_0}))\}$,
this means that

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

and in particular $\vec{x} \in Esp' \wedge \mathcal{C}[\![S]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y_0}} \Rightarrow \vec{x} \in Esp$, so that

$$\vec{x} \in Esp' \wedge \mathcal{C}[\![S]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y_0}}$$
$$\Longleftrightarrow$$
$$\vec{x} \in Esp' \wedge \vec{x} \in Esp \wedge \mathcal{C}[\![S]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y_0}}$$
$$\Longleftrightarrow$$
$$\vec{x} \in Esp \wedge \mathcal{C}[\![S]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y_0}},$$

that is, the maximum domain of $\mathcal{C}[\![S]\!]$ and $Rgo_{\overline{Y_0}}$ is $\{\vec{x} \in Esp | \mathcal{C}[\![S]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y_0}}\}$

Definition. *Given an algorithm whose space of states is Esp, with list of constants and variables equal to $\vec{x}$ and $Post$ is a predicate that depends on the variables $\vec{x}$ and $\overline{Y}$, we say that Pre is a weakest precondition of S and $Post$ if and only if $\{\vec{x} \in Esp | Pre(\vec{x}, \overline{Y_0})\}$ is the maximum domain of the relation $\mathcal{C}[\![S]\!]$ with range $Rgo_{\overline{Y_0}}$ for any value $\overline{Y_0}$ of the free variables $\overline{Y}$.*

Remark. *According to the lemma 1 we have that a weakest precondition for a statement S and $Post(\vec{x}, \overline{Y})$ is*

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

and by the previous remark is fulfilled that $\mathcal{C}[\![S]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}$ is also a weakest precondition of S and $Post(\vec{x}, \overline{Y})$.

Remark. *Since $\vec{x} \in Esp$ is an assumed hypothesis, then all weakest preconditions are equivalent.*

III. Denotational Semantics of GCL

In this section it will formalize the concept of the statement of an algorithm in GCL, for that we begin with the assignment instruction.

Definition. *If an algorithm has the space of states Esp, then an assignment statement S is a phrase of the form $y_{i_1}, \ldots, y_{i_k} := Exp_1, \ldots, Exp_k$, where $i_j \neq i_{j'}$ for all j and j' with $1 \leq j < j' \leq k$, y_{i_j} is the variable in the position i_j of the variables vector $\overline{y}$ of the definition of space of states, and y_{i_j} is of the same type of expression Exp_{i_j}, it is define the function of several variables*

$$R'_S : \bigcap_{i=1}^{k} Dom(Exp_i) \subseteq Esp \to Esp$$

$$R'_S(\overline{x}, \overline{y}) =$$

$$(\overline{x}, y_1, \ldots, y_{i_1 - 1}, Exp_1(\overline{x}, \overline{y}), \ldots, y_{i_k - 1}, Exp_k(\overline{x}, \overline{y}), \ldots, y_{n'}),$$

so, this sentence is interpreted as

$$\mathcal{C}[\![S]\!] := R'_S \cup ((Dom(R'_S))^c \times \{abort\})$$

where $(Dom(R'_S))^c = Esp' \setminus Dom(R'_S)$.

Example. *For the following algorithm*

$[Const\ n : Integer;$
$Var\ x : Real;$
$\quad y : Real;$
$\quad z : Real;$

$\quad x, z := x + z, (x + n)/z$
$],$

the result of an execution of said algorithm for the initial values n_0, x_0, y_0, z_0, is the result of evaluating the function

$$f : \mathbb{Z} \times \mathbb{R} \times \mathbb{R} \times (\mathbb{R} \setminus \{0\}) \to \mathbb{Z} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R}$$

$$f(n, x, y, z) = (n, x + z, y, \frac{x + n}{z})$$

in the values n_0, x_0, y_0, z_0 when $z_0 \neq 0$, or abort if $z_0 = 0$.

Theorem 1. *Let the statement $y_{i_1}, \ldots, y_{i_k} := Exp_1, \ldots, Exp_k$ be with variables within the space of states Esp and a postcondition $Post$ for said instruction. A weakest precondition for this statement and $Post$ is:*

$$domain(Exp_1, \ldots, Exp_k) \wedge$$
$$Post[y_{i_1}, \ldots y_{i_k} := Exp_1, \ldots, Exp_k]$$

Proof. The asignement statement will be called as S and will be consider that the variables $\overline{Y}$ are fixed, by Lemma 1 we have that the maximum domain of $\mathcal{C}[\![S]\!]$ and $Rgo_{\overline{Y}}$ is equal to:

$$\begin{aligned}
&\{\vec{x} \in Esp | \vec{x} \in supp(\mathcal{C}[\![S]\!]) \wedge \\
&\qquad (\forall y | : y \in R'_S(\{\vec{x}\}) \Rightarrow Post(y, \overline{Y}))\} \\
&= \\
&\{\vec{x} \in Esp | \vec{x} \in \bigcap_{i=1}^{k} Dom(Exp_i) \wedge \\
&\qquad (\forall y | : y \in R'_S(\{\vec{x}\}) \Rightarrow Post(y, \overline{Y}))\} \\
&= \\
&\{\vec{x} \in Esp | (\forall i | 1 \leq i \leq k : \vec{x} \in Dom(Exp_i)) \wedge \\
&\qquad (\forall y | : y \in R'_S(\{\vec{x}\}) \Rightarrow Post(y, \overline{Y}))\} \\
&= \\
&\{\vec{x} \in Esp | (\forall i | 1 \leq i \leq k : domain(Exp_i)) \wedge \\
&\qquad (\forall y | : y \in R'_S(\{\vec{x}\}) \Rightarrow Post(y, \overline{Y}))\} \\
&= \\
&\{\vec{x} \in Esp | domain(Exp_1, \ldots, Exp_k) \wedge \\
&\qquad (\forall y | : y \in R'_S(\{\vec{x}\}) \Rightarrow Post(y, \overline{Y}))\} \\
&= \\
&\{\vec{x} \in Esp | domain(Exp_1, \ldots, Exp_k) \wedge \\
&\qquad (\forall y | y \in R'_S(\{\vec{x}\}) : Post(y, \overline{Y}))\} \\
&\overset{\mathcal{C}[\![S]\!] \text{ is a function}}{=} \\
&\{\vec{x} \in Esp | domain(Exp_1, \ldots, Exp_k) \wedge \\
&\qquad (\forall y | y = R'_S(\vec{x}) : Post(y, \overline{Y}))\} \\
&= \\
&\{\vec{x} \in Esp | domain(Exp_1, \ldots, Exp_k) \wedge Post(R'_S(\vec{x}), \overline{Y})\} \\
&= \\
&\{(\overline{x}, \overline{y}) \in Esp | domain(Exp_1, \ldots, Exp_k) \wedge \\
&\qquad Post(R'_S(\overline{x}, \overline{y}), \overline{Y})\} \\
&= \\
&\{(\overline{x}, \overline{y}) \in Esp | domain(Exp_1, \ldots, Exp_k) \wedge \\
&Post(\overline{x}, \ldots, y_{i_1 - 1}, Exp_1(\overline{x}, \overline{y}), \ldots, y_{i_k - 1}, Exp_k(\overline{x}, \overline{y}), \ldots, \overline{Y})\} \\
&= \\
&\{(\overline{x}, \overline{y}) \in Esp | domain(Exp_1, \ldots, Exp_k) \wedge \\
&\qquad Post[y_{i_1}, \ldots y_{i_k} := Exp_1, \ldots, Exp_k]\}
\end{aligned}$$

□

Definition. *If an algorithm has a space of states Esp, then phrase $SKIP$ is a statement whose interpretation $\mathcal{C}[\![SKIP]\!]$ is function $id_{Esp'} : Esp' \to Esp'$.*

With this definition the following theorem is trivial.

Theorem 2. *A weakest precondition for a statement $SKIP$ and $Post$ is $Post$.*

Definition. *If an algorithm has as space of states Esp, and let S_0 and S_1 be sentences, then the phrase $S_0; S_1$ is a statement whose interpretation is the relation $\mathcal{C}[\![S_0; S_1]\!] := \mathcal{C}[\![S_1]\!] \circ \mathcal{C}[\![S_0]\!]$, where the operation $'\circ'$ represents the composition of relations, that is, if R_0 and R_1 are relations in Esp', you have to $R_1 \circ R_0 := \{(x, y) \in Esp' \times Esp' | (\exists z | : (x, z) \in R_0 \wedge (z, y) \in R_1)\}$*

Theorem 3. *If a weakest precondition of S_1 and $Post$ is Pre' and Pre is a weakest precondition of S_0 and Pre', then a weakest precondition of $S_0; S_1$ and $Post$ is Pre.*

Proof. The variables $\overline{Y}$ of $Post$ are considered fixed.

Since all weakest preconditions are equivalent and $\mathcal{C}[\![S_1]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}$ is a weakest precondition of S_1 and $Post$, then Pre' is equivalent to $\mathcal{C}[\![S_1]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}$ and for the same reason $\mathcal{C}[\![S_0]\!](\{\vec{x}\}) \subseteq \{z \in Esp | Pre'\}$ is equivalent to Pre.

On the other hand, the maximum domain of $\mathcal{C}[\![S_0; S_1]\!]$ and $Rgo_{\overline{Y}}$ is equal to

$$\{\vec{x} \in Esp | \mathcal{C}[\![S_0; S_1]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}\}$$

By making basic manipulations of the set theory and relationships it can be shown that the previous set is equal to:

$$\begin{aligned}
&\{\vec{x} \in Esp | \\
&\qquad \mathcal{C}[\![S_0]\!](\{\vec{x}\}) \subseteq \{z \in Esp | \mathcal{C}[\![S_1]\!](\{z\}) \subseteq Rgo_{\overline{Y}}\}\} \\
&= \\
&\{\vec{x} \in Esp | \mathcal{C}[\![S_0]\!](\{\vec{x}\}) \subseteq \{z \in Esp | Pre'\}\} \\
&= \\
&\{\vec{x} \in Esp | Pre\}
\end{aligned}$$

□

Definition. *Let $\vec{x}$ be the list of declared variables, a conditional statement IF is a phrase of the form*

$$\begin{aligned}
&if\ B_0 \to \\
&\quad S_0 \\
&[]\ B_1 \to \\
&\quad S_1 \\
&\vdots \\
&[]\ B_n \to \\
&\quad S_n \\
&fi
\end{aligned}$$

where B_i are predicates that only depend on $\vec{x}$ and S_i are statements. Let be the sets $T_i := \{\vec{x} \in Dom(B_i) | B_i\}$ and $B'_i := id_{T_i} \cup (Dom(B_i)^c \times \{abort\})$, where $Dom(B_i)^c = Esp' \setminus Dom(B_i)$. In this way defining

$$R'_{IF} := \bigcup_{i=0}^{n} \mathcal{C}[\![S_i]\!] \circ B'_i$$

the interpretation of the IF instruction is the relation

$$\mathcal{C}[\![IF]\!] := R'_{IF} \cup (Dom(R'_{IF})^c \times \{abort\})$$

where $Dom(R'_{IF})^c = Esp' \setminus Dom(R'_{IF})$

Remark. *To keep the theory in first order, in the union of the definition of R'_{IF}, it is assumed that an indexing function of sets $i \mapsto \mathcal{C}[\![S_i]\!]$ and $i \mapsto B'_i$ is being used (which clearly exist*

by having a finite number of ordered pairs) and not that is indexing neither the statements or the predicates.

Lemma 2. *The following propositions are true*

1) $\vec{x} \in supp(\mathcal{C}[\![IF]\!]) \equiv abort \notin R'_{IF}(\{\vec{x}\}) \wedge \vec{x} \notin Dom(R'_{IF})^c$
2) $y \in \mathcal{C}[\![IF]\!] \restriction_{supp(\mathcal{C}[\![IF]\!])} (\{\vec{x}\}) \equiv \vec{x} \in supp(\mathcal{C}[\![IF]\!]) \wedge y \in R'_{IF}(\{\vec{x}\})$
3) $y \in R'_{IF}(\{\vec{x}\}) \equiv (\exists i | 0 \leq i \leq n : y \in \mathcal{C}[\![S_i]\!] \restriction_{T_i} (\{\vec{x}\}) \vee (\vec{x} \in Dom(B_i)^c \wedge y = abort))$
4) $abort \notin R'_{IF}(\{\vec{x}\}) \equiv (\forall i | 0 \leq i \leq n : \vec{x} \in Dom(B_i) \wedge (\vec{x} \in T_i \Rightarrow \vec{x} \in supp(\mathcal{C}[\![S_i]\!])))$
5) $\vec{x} \in Dom(R'_{IF}) \equiv (\exists i | 0 \leq i \leq n : \vec{x} \in T_i \vee \vec{x} \in Dom(B_i)^c)$

Proof. It is proved first that

$y \in \mathcal{C}[\![IF]\!](\{\vec{x}\}) \equiv y \in R'_{IF}(\{\vec{x}\}) \vee (\vec{x} \in Dom(R'_{IF})^c \wedge y = abort)$ $(*)$,

with which it is easily demonstrated 1. Then to proof 2) it is done the following

$y \in \mathcal{C}[\![IF]\!] \restriction_{supp(\mathcal{C}[\![IF]\!])} (\{\vec{x}\})$
$\iff$
$\vec{x} \in supp(\mathcal{C}[\![IF]\!]) \wedge y \in \mathcal{C}[\![IF]\!](\{\vec{x}\})$
$\overset{using\ (*)}{\iff}$
$\vec{x} \in supp(\mathcal{C}[\![IF]\!]) \wedge (y \in R'_{IF}(\{\vec{x}\}) \vee (\vec{x} \in Dom(R'_{IF})^c \wedge y = abort))$
$\iff$
$(\vec{x} \in supp(\mathcal{C}[\![IF]\!]) \wedge y \in R'_{IF}(\{\vec{x}\})) \vee (\vec{x} \in supp(\mathcal{C}[\![IF]\!]) \wedge \vec{x} \in Dom(R'_{IF})^c \wedge y = abort)$
$\overset{using\ 1)}{\iff}$
$(\vec{x} \in supp(\mathcal{C}[\![IF]\!]) \wedge y \in R'_{IF}(\{\vec{x}\})) \vee (abort \notin R'_{IF}(\{\vec{x}\}) \wedge \vec{x} \notin Dom(R'_{IF})^c \wedge \vec{x} \in Dom(R'_{IF})^c \wedge y = abort)$
$\iff$
$\vec{x} \in supp(\mathcal{C}[\![IF]\!]) \wedge y \in R'_{IF}(\{\vec{x}\})$

To proof 3) it is done the following

$y \in R'_{IF}(\{\vec{x}\})$
$\iff$
$y \in (\overset{n}{\underset{i=0}{\cup}} \mathcal{C}[\![S_i]\!] \circ B'_i)(\{\vec{x}\})$
$\iff$
$y \in \overset{n}{\underset{i=0}{\cup}} (\mathcal{C}[\![S_i]\!] \circ B'_i)(\{\vec{x}\})$
$\iff$
$(\exists i | 0 \leq i \leq n : (\mathcal{C}[\![S_i]\!] \circ B'_i)(\{\vec{x}\}))$
$\iff$
$(\exists i | 0 \leq i \leq n : y \in (\mathcal{C}[\![S_i]\!] \circ (id_{T_i} \cup (Dom(B_i)^c \times \{abort\})))(\{\vec{x}\}))$
$\iff$
$(\exists i | 0 \leq i \leq n : y \in (\mathcal{C}[\![S_i]\!] \circ id_{T_i} \cup \mathcal{C}[\![S_i]\!] \circ (Dom(B_i)^c \times \{abort\}))(\{\vec{x}\}))$
$\iff$
$(\exists i | 0 \leq i \leq n : y \in \mathcal{C}[\![S_i]\!] \circ id_{T_i}(\{\vec{x}\}) \cup \mathcal{C}[\![S_i]\!] \circ (Dom(B_i)^c \times \{abort\})(\{\vec{x}\}))$
$\iff$
$(\exists i | 0 \leq i \leq n : y \in \mathcal{C}[\![S_i]\!] \restriction_{T_i} (\{\vec{x}\}) \vee y \in (Dom(B_i)^c \times \{abort\})(\{\vec{x}\}))$
$\iff$
$(\exists i | 0 \leq i \leq n : y \in \mathcal{C}[\![S_i]\!] \restriction_{T_i} (\{\vec{x}\}) \vee (\vec{x} \in Dom(B_i)^c \wedge y = abort))$

To proof 4) it is done the following

$abort \notin R'_{IF}(\{\vec{x}\})$
$\iff$
$\neg(abort \in R'_{IF}(\{\vec{x}\}))$
$\iff$
$\neg(\exists i | 0 \leq i \leq n : abort \in \mathcal{C}[\![S_i]\!] \restriction_{T_i} (\{\vec{x}\}) \vee \vec{x} \in Dom(B_i)^c)$
$\iff$
$(\forall i | 0 \leq i \leq n : abort \notin \mathcal{C}[\![S_i]\!] \restriction_{T_i} (\{\vec{x}\}) \wedge \vec{x} \in Dom(B_i))$
$\iff$
$(\forall i | 0 \leq i \leq n : \vec{x} \in Dom(B_i) \wedge \neg(\vec{x} \in T_i \wedge abort \in \mathcal{C}[\![S_i]\!](\{\vec{x}\})))$
$\iff$
$(\forall i | 0 \leq i \leq n : \vec{x} \in Dom(B_i) \wedge (\vec{x} \notin T_i \vee abort \notin \mathcal{C}[\![S_i]\!](\{\vec{x}\})))$
$\iff$
$(\forall i | 0 \leq i \leq n : \vec{x} \in Dom(B_i) \wedge (\vec{x} \in T_i \Rightarrow \vec{x} \in supp(\mathcal{C}[\![S_i]\!])))$

To proof 5) it is done the following

$\vec{x} \in Dom(R'_{IF})$
$\iff$
$(\exists y | : y \in R'_{IF}(\{\vec{x}\}))$
$\overset{Item\ 3)\ of\ Lema\ 2}{\iff}$
$(\exists y | : (\exists i | 0 \leq i \leq n : y \in \mathcal{C}[\![S_i]\!] \restriction_{T_i} (\{\vec{x}\}) \vee (\vec{x} \in Dom(B_i)^c \wedge y = abort)))$
$\iff$
$(\exists i | 0 \leq i \leq n : (\exists y | : y \in \mathcal{C}[\![S_i]\!] \restriction_{T_i} (\{\vec{x}\}) \vee (\vec{x} \in Dom(B_i)^c \wedge y = abort)))$
$\iff$
$(\exists i | 0 \leq i \leq n : (\exists y | : y \in \mathcal{C}[\![S_i]\!] \restriction_{T_i} (\{\vec{x}\})) \vee (\exists y | y = abort : \vec{x} \in Dom(B_i)^c))$
$\iff$
$(\exists i | 0 \leq i \leq n : \vec{x} \in Dom(\mathcal{C}[\![S_i]\!] \restriction_{T_i}) \vee \vec{x} \in Dom(B_i)^c)$
$\iff$
$(\exists i | 0 \leq i \leq n : \vec{x} \in T_i \vee \vec{x} \in Dom(B_i)^c)$ □

Theorem 4. *If Pre_i is a weakest precondition of S_i and $Post$, then*

$$domain(B_0, \ldots, B_n) \wedge (B_0 \vee \cdots \vee B_n) \wedge (B_0 \Rightarrow Pre_0) \wedge \cdots \wedge (B_n \Rightarrow Pre_n)$$

is a weakest precondition of IF and $Post$

Proof. Suppose that $\overline{Y}$ are fixed and assuming the hypothesis that $\vec{x} \in Esp$, by Lema 1 the weakest precondition of IF and $Post$ is as follows:

$\vec{x} \in supp(\mathcal{C}[\![IF]\!]) \wedge (\forall y | : y \in \mathcal{C}[\![IF]\!] \restriction_{supp(\mathcal{C}[\![IF]\!])} (\{\vec{x}\}) \Rightarrow Post(y, \overline{Y}))$
$\overset{Item\ 2)\ of\ Lema\ 2}{\iff}$
$\vec{x} \in supp(\mathcal{C}[\![IF]\!]) \wedge (\forall y | : y \in R'_{IF}(\{\vec{x}\}) \Rightarrow Post(y, \overline{Y}))$
$\overset{Item\ 3)\ of\ Lema\ 2}{\iff}$
$\vec{x} \in supp(\mathcal{C}[\![IF]\!]) \wedge (\forall y | : (\exists i | 0 \leq i \leq n : y \in \mathcal{C}[\![S_i]\!] \restriction_{T_i} (\{\vec{x}\}) \vee (\vec{x} \in Dom(B_i)^c \wedge y = abort)) \Rightarrow Post(y, \overline{Y}))$
$\iff$

$$\vec{x} \in supp(\mathcal{C}[\![IF]\!])\wedge$$
$$(\forall y| : (\forall i|0 \leq i \leq n : y \in \mathcal{C}[\![S_i]\!] \restriction_{T_i} (\{\vec{x}\})\vee$$
$$(\vec{x} \in Dom(B_i)^c \wedge y = abort) \Rightarrow Post(y, \overline{Y})))$$
$$\iff$$
$$\vec{x} \in supp(\mathcal{C}[\![IF]\!])\wedge$$
$$(\forall i|0 \leq i \leq n : (\forall y| : y \in \mathcal{C}[\![S_i]\!] \restriction_{T_i} (\{\vec{x}\})\vee$$
$$(\vec{x} \in Dom(B_i)^c \wedge y = abort) \Rightarrow Post(y, \overline{Y})))$$
$$\overset{Item\ 1)\ and\ 4)\ of\ Lema\ 2}{\iff}$$
$$\vec{x} \in Dom(R'_{IF}) \wedge (\forall i|0 \leq i \leq n : \vec{x} \in Dom(B_i)\wedge$$
$$(\vec{x} \in T_i \Rightarrow \vec{x} \in supp(\mathcal{C}[\![S_i]\!]))\wedge(\forall y| : y \in \mathcal{C}[\![S_i]\!] \restriction_{T_i} (\{\vec{x}\})\vee$$
$$(\vec{x} \in \cancel{Dom(B_i)^c}^{\,false} \wedge y = abort) \Rightarrow Post(y, \overline{Y})))$$
$$\iff$$
$$\vec{x} \in Dom(R'_{IF}) \wedge (\forall i|0 \leq i \leq n :$$
$$\vec{x} \in Dom(B_i) \wedge (\vec{x} \in T_i \Rightarrow \vec{x} \in supp(\mathcal{C}[\![S_i]\!]))\wedge$$
$$(\forall y| : \vec{x} \in T_i \wedge y \in \mathcal{C}[\![S_i]\!](\{\vec{x}\}) \Rightarrow Post(y, \overline{Y})))$$
$$\iff$$
$$\vec{x} \in Dom(R'_{IF}) \wedge (\forall i|0 \leq i \leq n :$$
$$\vec{x} \in Dom(B_i) \wedge (\vec{x} \in T_i \Rightarrow \vec{x} \in supp(\mathcal{C}[\![S_i]\!]))\wedge$$
$$(\vec{x} \in T_i \Rightarrow (\forall y| : y \in \mathcal{C}[\![S_i]\!](\{\vec{x}\}) \Rightarrow Post(y, \overline{Y}))))$$
$$\iff$$
$$\vec{x} \in Dom(R'_{IF}) \wedge (\forall i|0 \leq i \leq n : \vec{x} \in Dom(B_i))\wedge$$
$$(\forall i|0 \leq i \leq n : \vec{x} \in T_i \Rightarrow$$
$$\vec{x} \in supp(\mathcal{C}[\![S_i]\!]) \wedge (\forall y| : y \in \mathcal{C}[\![S_i]\!](\{\vec{x}\}) \Rightarrow Post(y, \overline{Y})))$$
$$\overset{definition\ and\ Lema\ 1}{\iff}$$
$$\vec{x} \in Dom(R'_{IF}) \wedge (\forall i|0 \leq i \leq n : \vec{x} \in Dom(B_i))\wedge$$
$$(B_0 \Rightarrow Pre_0) \wedge \cdots \wedge (B_n \Rightarrow Pre_n)$$
$$\overset{Item\ 5)\ of\ Lema\ 2}{\iff}$$
$$(\exists i|0 \leq i \leq n : \vec{x} \in T_i \vee \vec{x} \in \cancel{Dom(B_i)^c}^{\,false}) \wedge (B_0 \Rightarrow Pre_0)$$
$$\wedge \cdots \wedge (B_n \Rightarrow Pre_n) \wedge (\forall i|0 \leq i \leq n : \vec{x} \in Dom(B_i))$$
$$\overset{definition}{\iff}$$
$$(B_0 \vee \cdots \vee B_n) \wedge (B_0 \Rightarrow Pre_0) \wedge \cdots \wedge (B_n \Rightarrow Pre_n)\wedge$$
$$domain(B_0, \ldots, B_n) \qquad \square$$

Remark. *If n in the definition of the relation R'_{IF}, were infinite, then the formula of second last step of the previous proof, is still valid to define by comprehension the maximum domain of the relation and $Rgo_{\overline{Y}}$.*

Remark. *According to [20], a propositional variable in the language of [19], is a predicate without variables, however, an algorithm does not have declared propositional variables, but Booleans, since otherwise there would be $\vec{x} \in Esp$, with coordinates equal to $TRUE$ or $FALSE$, and These are objects of the metatheory. To simulate the use of propositional variables with booleans, it is agreed that, if P is a boolean variable then the predicate $P = \hat{1}$ can be abbreviated as P. For example, predicates like $P \wedge Exp$ are considered an abbreviation of $P = \hat{1} \wedge Exp$.*

A. Iteration statement

It is not straightforward from the language and axiomatization of ZFS set theory, that any recursively defined set exists, however any set that wants to be defined recursively, can be defined with a formula in the ZFS language, that is equivalent to the initial recursion. The construction that shows the existence of this formula in the first-order language of ZFS, which defines equivalently the set that was initially defined recursively and allows demonstrating the existence of the set, is known as "transfinite recurrence metatheorem". Given a definition of a set made recursively, this metatheorem shows constructively, what is the formula within the language of ZFS, which defines the same set.

A detailed demonstration of the transfinite recursion metatheorem is found in [21], however it is more general than what is needed for this section, since it is valid for recursion on any ordinal. In this section we will use a version of this theorem restricted to ω, whose statement is:

Theorem 5. *If you have a predicate φ such that it satisfies $(\forall k, F| : (\exists! y| : \varphi(k, F, y)))$. Defining $G(k, F)$ as the only one y such that $\varphi(k, F, y)$. Then you can write a formula ψ where the following is demonstrable:*

1) *$(\forall k| : (\exists! y| : \psi(k, y)))$, that is, ψ defines a function F such that $\psi(k, F(k))$*
2) *$(\forall k|k \in \omega| : F(k) = G(k, F \restriction_{k-1}))$*

An informal explanation of the previous theorem would be, that the expression $F(k) = G(k, F \restriction_{k-1})$ is a recursive definition of the set $F(k)$ and the formula $\psi(k, y)$ is a version in the ZFS language, which defines by comprehension a set y that has been equal to $F(k)$.

Definition. *An iteration statement Do is a phrase of the form*

do $B_0 \rightarrow$
S_0
od

where B_0 is a predicate that only depends on $\vec{x}$ and S_0 is an statement. Let's recursively define the following instructions

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

Let be the sets $T_i := \{\vec{x} \in Esp|\vec{x} \in supp(\mathcal{C}[\![Do_i]\!])\}$ and $B'_i := id_{T_i} \cup (\{abort\} \times \{abort\})$. In this way defining

$$R'_{Do} := \bigcup_{i=0}^{\infty} \mathcal{C}[\![Do_i]\!] \circ B'_i$$

the interpretation of statement Do is the relation

$$\mathcal{C}[\![Do]\!] := R'_{Do} \cup (Dom(R'_{Do})^c \times \{abort\})$$

where $Dom(R'_{Do})^c = Esp' \setminus Dom(R'_{Do})$

Remark. *Since the interpretation of a sequence of statements is the composition of the interpretations, then the interpretation of instruction Do satisfies the following recurrence:*

$$\mathcal{C}[\![Do_0]\!] := \mathcal{C}[\![if \neg B_0 \rightarrow SKIP fi]\!]$$
$$\mathcal{C}[\![Do_{k+1}]\!] := \mathcal{C}[\![Do_k]\!] \circ \mathcal{C}[\![If]\!]$$

By the metatheorem of the transfinite recursion, there exists predicate $\psi(k, R)$, which is true only when $R = \mathcal{C}[\![Do_k]\!]$. In

this way, within the language of the set theory of ZFS, predicates of the form $P(\dots, \mathcal{C}[\![Do_k]\!], \dots)$ can be written as an abbreviation of $(\forall R|\psi(k,R) : P(\dots, R, \dots))$ and that is how it will be understood in the following definitions and theorems. Additionally, since the set $\{\langle k, R\rangle \in \omega \times 2^{Esp' \times Esp'} | \psi(k,R)\}$ exists by the comprehension axiom, then the function that indexes $k \mapsto \mathcal{C}[\![Do_k]\!]$ exists, therefore, keeping the theory in first order, every time that an expression containing $C[\![Do_k]\!]$ is written, it will be understood that a set is being indexed, and not an statement.

Remark. *The states that do not result in an abort when executing the instruction Do_k of the previous definition are those states of the machine that of running Do starting in that state, cause that the cycle is executed at the most k times before the guard is false.*

Remark. *GCL does not have an IF instruction with infinite guards, but in the supposition that an instruction of this type is allowed, a mnemonic for remember the previous definition, it is to think that instruction D would be equivalent to an instruction in the following way:*

$$
\begin{array}{l}
if\ x \in supp(\mathcal{C}[\![Do_0]\!]) \rightarrow\ Do_0 \\
[]\ x \in supp(\mathcal{C}[\![Do_1]\!]) \rightarrow\ Do_1 \\
\vdots \\
[]\ x \in supp(\mathcal{C}[\![Do_n]\!]) \rightarrow\ Do_n \\
\vdots \\
fi
\end{array}
$$

Lemma 3. *Let A be an algorithm with space of states Esp, $\vec{x} \in Esp$ and $k \leq k'$. If $abort \neq y$ and $y \in \mathcal{C}[\![Do_k]\!](\{\vec{x}\})$, then $y \in \mathcal{C}[\![Do_{k'}]\!](\{\vec{x}\})$ and if $y \in \mathcal{C}[\![Do_{k'}]\!](\{\vec{x}\})$ and $y \notin \mathcal{C}[\![Do_k]\!](\{\vec{x}\})$, then $abort \in \mathcal{C}[\![Do_k]\!](\{\vec{x}\})$*

Corollary 1. *Let A be an algorithm with space of states Esp and $\vec{x} \in Esp$. If $k \leq k'$ and $abort \notin \mathcal{C}[\![Do_k]\!](\{\vec{x}\})$, then $\mathcal{C}[\![Do_k]\!](\{\vec{x}\}) = \mathcal{C}[\![Do_{k'}]\!](\{\vec{x}\})$*

Proof. Since $abort \notin \mathcal{C}[\![Do_k]\!](\{\vec{x}\})$, then all $y \in \mathcal{C}[\![Do_k]\!](\{\vec{x}\})$ complies with the hypotheses of lemma 3 and therefore $y \in \mathcal{C}[\![Do_{k'}]\!](\{\vec{x}\})$, so that $\mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \subseteq \mathcal{C}[\![Do_{k'}]\!](\{\vec{x}\})$. On the other hand, there can not exist y such that $y \in \mathcal{C}[\![Do_{k'}]\!](\{\vec{x}\})$ and $y \notin \mathcal{C}[\![Do_k]\!](\{\vec{x}\})$, since by lemma 3, $abort \in \mathcal{C}[\![Do_k]\!](\{\vec{x}\})$ would have to be contradicted by the hypothesis of corollary. Thus $\mathcal{C}[\![Do_k]\!](\{\vec{x}\}) = \mathcal{C}[\![Do_{k'}]\!](\{\vec{x}\})$ □

Lemma 4. *Let A be an algorithm with a space of states Esp and a postcondition $Post(\vec{x}, \overline{Y})$, then the maximum domain of $\mathcal{C}[\![Do]\!]$ and $Rgo_{\overline{Y}}$ is*

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

Proof. Suppose that $\overline{Y}$ are fixed values. Since $\mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}$ is a weakest precondition of Do_k and $Post(\vec{x}, \overline{Y})$, then by Theorem 4 and the remark immediately following this Theorem, the maximum domain of $\mathcal{C}[\![Do]\!]$ and $Rgo_{\overline{Y}}$ is

$$\{\vec{x} \in Esp | domain(\vec{x} \in supp(Do_0), \vec{x} \in supp(Do_1), \dots) \wedge$$

$$(\exists k | k \geq 0 : \vec{x} \in supp(Do_k)) \wedge$$

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

but since $\vec{x} \in supp(Do_k)$ is an expression defined for all $\vec{x} \in Esp$, then $domain(\vec{x} \in supp(Do_0), \vec{x} \in supp(Do_1), \dots) \equiv true$ and therefore the maximum domain is simply

$$\{\vec{x} \in Esp | (\exists k | k \geq 0 : \vec{x} \in supp(Do_k)) \wedge$$

$$(\forall k | k \geq 0 : \vec{x} \in supp(Do_k) \Rightarrow \mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}})\}.$$

Let's prove now that the formula that defines by comprehension the previous set is equivalent to $(\exists k | k \geq 0 : \mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}})$

Let's show the equivalence in the $\Rightarrow$ direction.

Taking witness k' to $(\exists k | k \geq 0 : \vec{x} \in supp(Do_k))$ is fulfilled that:

$$
\begin{array}{l}
k' \geq 0 \wedge \vec{x} \in supp(Do_{k'}) \wedge (\forall k | k \geq 0 : \vec{x} \in supp(Do_k) \Rightarrow \\
\mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}) \\
\overset{instantiate\ k:=k'}{\Rightarrow} \\
k' \geq 0 \wedge \vec{x} \in supp(Do_{k'}) \wedge (k' \geq 0 \wedge \vec{x} \in supp(Do_{k'}) \Rightarrow \\
\mathcal{C}[\![Do_{k'}]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}) \\
\overset{Modus\ Ponens}{\Rightarrow} \\
k' \geq 0 \wedge \mathcal{C}[\![Do_{k'}]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}} \\
\Rightarrow \\
(\exists k | k \geq 0 : \mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}})
\end{array}
$$

Let's now show the equivalence in the $\Leftarrow$ direction.

Let's show that $(\exists k | k \geq 0 : \mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}) \Rightarrow (\exists k | k \geq 0 : \vec{x} \in supp(\mathcal{C}[\![Do_k]\!]))$

$$
\begin{array}{l}
(\exists k | k \geq 0 : \mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}) \\
\iff \\
(\exists k | k \geq 0 : (\forall y | : y \in \mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \Rightarrow y \in Rgo_{\overline{Y}})) \\
\iff \\
(\exists k | k \geq 0 : \\
\qquad (\forall y | : y \in \mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \Rightarrow y \in Esp \wedge y \in Rgo_{\overline{Y}})) \\
\iff \\
(\exists k | k \geq 0 : (\forall y | : y \in \mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \Rightarrow y \in Esp) \wedge \\
\qquad (\forall y | : y \in \mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \Rightarrow y \in Rgo_{\overline{Y}}))) \\
\iff \\
(\exists k | k \geq 0 : \vec{x} \in supp(\mathcal{C}[\![Do_k]\!]) \wedge \\
\qquad (\forall y | : y \in \mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \Rightarrow y \in Rgo_{\overline{Y}})) \\
\Rightarrow \\
(\exists k | k \geq 0 : \vec{x} \in supp(\mathcal{C}[\![Do_k]\!]))
\end{array}
$$

Assuming $(\exists k | k \geq 0 : \mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}})$ let's show now that $(\forall k | k \geq 0 : \vec{x} \in supp(\mathcal{C}[\![Do_k]\!]) \Rightarrow \mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}})$.

Since $(\exists k | k \geq 0 : \mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}})$, then by the principle of well ordering there is a first k_m that meets

$\mathcal{C}[\![Do_{k_m}]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}$, let's prove by cases:

Case 1 $k < k_m$:

In this case $\mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \nsubseteq Rgo_{\overline{Y}}$, that is, there exists $y \in \mathcal{C}[\![Do_k]\!](\{\vec{x}\})$ such that $y \notin Rgo_{\overline{Y}}$. If $y \neq abort$, then by lemma 3 is fulfilled that $y \in \mathcal{C}[\![Do_{k_m}]\!](\{\vec{x}\})$ which contradicts $\mathcal{C}[\![Do_{k_m}]\!] \subseteq Rgo_{\overline{Y}}$ since $y \notin Rgo_{\overline{Y}}$, of this form $y = abort$ and therefore $\vec{x} \notin supp(\mathcal{C}[\![Do_k]\!])$ and therefore it is fulfilled that $\vec{x} \in supp(\mathcal{C}[\![Do_k]\!]) \Rightarrow \mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}$.

Case 2 $k \geq k_m$:

Since $\mathcal{C}[\![Do_{k_m}]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}$ is true and $abort \notin Rgo_{\overline{Y}}$, then $abort \notin \mathcal{C}[\![Do_{k_m}]\!](\{\vec{x}\})$ and by corollary 3 $\mathcal{C}[\![Do_k]\!](\{\vec{x}\}) = \mathcal{C}[\![Do_{k_m}]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}$ and therefore it is true that $\vec{x} \in supp(\mathcal{C}[\![Do_k]\!]) \Rightarrow \mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}$. □

Remark. *Using the formula ψ that is extracted from the transfinite recursion metatheorem, the weakest precondition of the previous theorem*

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

can understanding as an abbreviation of

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

which is written in the first-order language of set theory, showing that there is always a weakest precondition for Do and $Post$ written in the language of ZF.

Lemma 5. *Let B, B_0, W, P propositions, then.*

$$(B \wedge \neg B_0 \wedge P) \vee (B \wedge B_0 \wedge W)$$

$$\iff$$

$$B \wedge (B_0 \Rightarrow W) \wedge (\neg B_0 \Rightarrow B \wedge \neg B_0 \wedge P)$$

Lemma 6.

$$B \wedge (\neg B_0 \vee W) \wedge (B_0 \vee B) \wedge (B_0 \vee P)$$

$$\iff$$

$$(B \wedge \neg B_0 \wedge P) \vee (B \wedge B_0 \wedge W)$$

Theorem 6. *Let $Post$ be a predicate, there is a predicate $H(k, Post)$ in the ZFS language, which satisfies the formulas*

$$H(0, Post) := domain(B_0) \wedge \neg B_0 \wedge Post$$

$$H(k, Post) :=$$

$$H(0, Post) \vee (domain(B_0) \wedge B_0 \wedge Pre_{k-1}) \ para \ k \geq 1,$$

where Pre_{k-1} is a weakest precondition of S_0 and $H(k-1, Post)$, in addition the maximum domain of $\mathcal{C}[\![Do]\!]$ and $Rgo_{\overline{Y}}$ is

$$\{\vec{x} \in Esp | (\exists k | k \geq 0 : H_k(Post))\}$$

Proof. It suffices to show by induction that $\mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}$ is equivalent to $H(k, Post)$.

If $k = 0$ you have to

$\mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}$

$\iff$

$\mathcal{C}[\![if \neg B_0 \rightarrow SKIP fi]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}$

$\overset{weakest\ precondition\ of\ Do_0\ and\ Post\ and\ Theorem\ 4}{\iff}$

$domain(B_0) \wedge \neg B_0 \wedge (\neg B_0 \Rightarrow \mathcal{C}[\![SKIP]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}})$

$\overset{Modus\ Pones\ and\ Theorem\ 2}{\iff}$

$domain(B_0) \wedge \neg B_0(\vec{x}) \wedge Post(\vec{x}, \overline{Y})$

$\iff$

$H(0, Post)$

If $k \geq 1$ then:

$\mathcal{C}[\![Do_{k-1}]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}$ is the weakest precondition of Do_{k-1} and $Post$ which by inductive hypothesis is equivalent to $H(k-1, Post)$. By theorem 3 we have a weakest precondition of $Do_k = If; Do_{k-1}$ and $Post$ is equal to a weakest precondition of If and $H(k-1, Post)$, which is equal by Theorem 4 to

$$domain(B_0) \wedge (B_0 \Rightarrow Pre_{k-1}) \wedge$$

$$(\neg B_0 \Rightarrow H(k-1, Post)) \quad (**).$$

On the other hand, since $\mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}$ is the weakest precondition of $Do_k = If; Do_{k-1}$ and $Post$, then $\mathcal{C}[\![Do_k]\!](\{\vec{x}\}) \subseteq Rgo_{\overline{Y}}$ must be equivalent to (**).

Let's show now for cases that if $k \geq 1$ then the predicate (**) is equivalent to

$$H(0, Post) \vee (domain(B_0) \wedge B_0 \wedge Pre_{k-1})$$

Case 1 $k = 1$:

It is abbreviated $domain(B_0)$ by B, Pre_{k-1} by W, Pre_{k-2} by W' and $Post$ by P

$H(0, Post) \vee (domain(B_0) \wedge B_0 \wedge Pre_{k-1})$

$\overset{notation}{\iff}$

$H(0, Post) \vee (B \wedge B_0 \wedge W)$

$\iff$

$(B \wedge \neg B_0 \wedge P) \vee (B \wedge B_0 \wedge W)$

$\overset{lemma\ 5}{\iff}$

$B \wedge (B_0 \Rightarrow W) \wedge (\neg B_0 \Rightarrow B \wedge \neg B_0 \wedge P)$

$\iff$

$B \wedge (B_0 \Rightarrow W) \wedge (\neg B_0 \Rightarrow H(0, Post))$

$\overset{notation}{\iff}$

$domain(B_0) \wedge (B_0 \Rightarrow Pre_{k-1}) \wedge (\neg B_0 \Rightarrow H(k-1, Post))$

Case 2 $k > 1$:

$domain(B_0) \wedge (B_0 \Rightarrow Pre_{k-1}) \wedge (\neg B_0 \Rightarrow H(k-1, Post))$

$\overset{notation}{\iff}$

$B \wedge (B_0 \Rightarrow W) \wedge (\neg B_0 \Rightarrow H(k-1, Post))$

$\iff$

$B \wedge (B_0 \Rightarrow W) \wedge (\neg B_0 \Rightarrow (H(0, Post) \vee (B \wedge B_0 \wedge W')))$

$\iff$

$B \wedge (\neg B_0 \vee W) \wedge (B_0 \vee H(0, Post) \vee (B \wedge B_0 \wedge W'))$

$\iff$

$B \wedge (\neg B_0 \vee W) \wedge (B_0 \vee (B \wedge \neg B_0 \wedge P) \vee (B \wedge B_0 \wedge W'))$

$\overset{absorption}{\iff}$

$B \wedge (\neg B_0 \vee W) \wedge (B_0 \vee (B \wedge P))$

$\iff$

$B \wedge (\neg B_0 \vee W) \wedge (B_0 \vee B) \wedge (B_0 \vee P)$

$\overset{lemma\ 6}{\iff}$

$(B \wedge \neg B_0 \wedge P) \vee (B \wedge B_0 \wedge W)$

$\iff$

$H(0, Post) \vee (B \wedge B_0 \wedge W)$

$\overset{notation}{\iff}$

$H(0, Post) \vee (domain(B_0) \wedge B_0 \wedge Pre_{k-1})$ □

Let's now define the non-deterministic iteration statement

Definition. *The non-deterministic iterative instruction DO is defined as a phrase of the form*

$do\ B_0 \to S_0$
$[]\ B_1 \to S_1$
$\vdots$
$[]\ B_n \to S_n$
od

Where B_i are expressions of Boolean type and S_i are instructions. The interpretation of said instruction is the same as the interpretation of instruction do $BB \to IF$ od, where BB denotes $B_0 \vee \cdots \vee B_n$

Remark. *According to Theorem 6 the predicates $H(k, Post)$ with $k \geq 1$ for this statement DO are equal to*

$$H(0, Post) \vee (domain(B_0, \ldots, B_n) \wedge BB \wedge Pre_{k-1}),$$

, where Pre_{k-1} is the weakest precondition of IF and $H_{k-1}(Post)$, which by theorem 4 is of the form $domain(B_0, \ldots, B_n) \wedge BB \wedge (B_0 \Rightarrow pre_0) \wedge \cdots \wedge (B_n \Rightarrow pre_n)$, thus by the idempotency of the $\wedge$, you have that $H(k, Post)$ with $k \geq 1$ equals simply

$$H(0, Post) \vee Pre_{k-1}$$

IV. Conclusion

The results of the previous theorems justify the following recursive definition of the predicate transformer wp:

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

where $H(k, Post)$ is a first-order predicate that satisfies the equations

$$H(0, Post) \equiv domain(BB) \wedge \neg BB \wedge Post$$

$$H(k, Post) \equiv$$

$$H(0, Post) \vee wp(IF, H(k-1, Post))\ for\ k \geq 1$$

These rules to calculate wp are valid on any algorithm, with any type of data from the universe of set theory. Since the Hoare logic can be derived from the definition of wp, then the Hoare logic is also true of any algorithm with any type of data in the universe of set theory. In particular, the invariant rule would be valid over algorithms with any type of data such as ultrafilters, $\sigma-$algebras, etc.

References

[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] C. A. R. Hoare. *An axiomatic basis for computer programming*. Communications of the ACM, vol. 12, no. 10, pp. 576580, 1969.

[3] D. Gries and F. B. Schneider. *A logical approach to discrete math*. New York, New York: Springer, 1993.

[4] E. W. Dijkstra. *A Discipline of Programming*. Englewood Cliffs , New Jersey: Prentice-Hall, 1976.

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

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

[7] G. O' Regan. *Giants of Computing*. London, Springer-Verlag, 2013.

[8] G. O' Regan. *Concise Guide to Formal Methods*. Mallow, Co. Cork, Ireland, Springer, 2017.

[9] G. O' Regan. *Mathematical Approaches to Software Quality*. Mallow, Co. Cork, Ireland, Springer, 2006.

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

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

[12] O. Mraihi, W. Ghardallou, A. Louhichi, L.L. Jilani, K. Bsaies, A. Mili, *Computing preconditions and postconditions of while loops*, in Proc. Int. Colloq. on Theoretical Aspects of Computing, Johannesburg, SA, 2011.

[13] L.L. Jilani, O. Mraihi, A. Louhichi, W. Ghardallou, K. Bsaies, A. Mili, *Invariant functions and invariant relations: An alternative to invariant assertions*, J. Symbolic Comput., vol. 48, pp. 1-36, 2013.

[14] A. Louhichi, W. Ghardallou, K. Bsaies, *Verifying While Loops with Invariant Relations*, Int. J. Critical Computer-Based Syst., vol. 5, no. 1-2, 2013.

[15] W. Ghardallou, O. Mraihi, A. Louhichi, L.L. Jilani, K. Bsaies, A. Mili, *A versatile concept for the analysis of loops*, J. Log. Algebr. Program., vol 81, no. 5, pp. 606-622, 2012.

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

[17] J. A. Bohórquez. *An elementary and unified approach to program correctness*. Formal Asp. Comput, vol. 22, No 5, pp. 611-627, 2010.

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

[19] E. W. Dijkstra, Scholten, S. Carel, *Predicate calculus and program semantics*, New York, Texts and Monographs in Computer Science, Springer-Verlag, 1990.

[20] C. Rocha, *The Formal System of Dijkstra and Scholten*.Logic, Rewriting, and Concurrency, vol. 9200, pp. 580-597, 2015.

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