Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 339 (2018) 63–83

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Calculation of Invariants Assertions

Federico Flaviani[1]

Universidad Simón Bolívar
Caracas, Venezuela

Abstract

In this paper we present a series of theorems that allow to establish strategies for the calculation of invariant assertions, such as the Dijkstra's $H_k(Post)$, or the weakest precondition of the loop. A criterion is also shown for calculating the termination condition of a loop. As in the integrals calculus, the strategies proposed here to perform the calculation of an invariant, will depend on the shape of the loop with which it is working, particularly will work with for-type loops with or without early termination due to a sentry.
http://www.elsevier.com/locate/entcs.

Keywords: Invariant Assertions, Calculus, Formal Program Verification, GCL, Induction.

1 Introduction

All the algorithms proposed in this paper will be written in GCL (Guarded Command Language) [1], which is a pseudolanguage defined by Dijkstra, which supports the writing of non-deterministic algorithms and their design, supports Hoare logic and formulas for weakest precondition, relatively simple, that facilitate the correction activity of a program. All assertions in this paper shall be assumed to be written in the language of the assertions of [2].

Dijkstra's logic [1] for program correction is based on the predicate transformer wp (weakest precondition), which is basically a syntactic two-variable function that symbolically returns the weakest precondition of a statement st given a postcondition $Post$ (using the classic notation of two-variable functions, the notation $wp(st, Post)$ refers to the result of applying to the function wp, the st and $Post$ arguments, this result is the weakest precondition, symbolically speaking, of the statement st with postcondition $Post$). The successive use of wp allows calculating weakest preconditions between instruction and instruction, from the end of the program to the beginning.

[1] Email: fflaviani@usb.ve

https://doi.org/10.1016/j.entcs.2018.06.005

Dijkstra in [1] established the rules that define the function of syntactic transformation wp according to the following paragraph:

If $B, B_0, \ldots, B_n$ and $S, S_0, \ldots, S_n$ are Boolean expressions and GCL's statements respectively, if IF and Do are abreviations of statements $if\ B_0 \to S_0[] \ldots []B_n \to S_n\ fi$ and $do\ B \to S\ od$ respectively and if $domain(B_0, \ldots, B_n)$ denoted a predicate that if is satisfied in a state, none of the expressions B_i, when these evaluated in that state, these incur an illegal operation (such as dividing by 0), then:

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

  where $H_k(Post)$ is a predicate that satisfies the equations:

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

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

for $k \geq 1$

1.1 Contribution

In Hoare's logic [3] to make the partial correction of a loop Do with postcondition $Post$, the invariant rule must be applied:

$$\frac{Inv \Rightarrow domain(B) \qquad \{Inv \wedge B\}S\{Inv\}}{\{Inv\}do\ B \to S\ od\{Inv \wedge \neg B\},}$$

in combination with rule

$$\frac{\{Inv\}do\ B \to S\ od\{Inv \wedge \neg B\} \qquad Inv \wedge \neg B \Rightarrow Post}{\{Inv\}do\ B \to S\ od\{Post\}.}$$

Furthermore to demonstrate termination and make a full correction, a bound function $f(\boldsymbol{x})$ for the loop (where $\boldsymbol{x}$ is a program state) must be used and the following two test obligations must be demonstrated.

$$Inv \wedge B \Rightarrow f(\boldsymbol{x}) \geq 0$$

and

$$\{Inv \wedge B \wedge f_0 = f(\boldsymbol{x})\}S\{f_0 > f(\boldsymbol{x})\}$$

In total it is necessary to prove five theorems or test obligations to verify that the invariant Inv proposed is correct, which is a great work even for very simple loops, and on the other hand the rules do not explain how the predicate Inv is constructed.

On the other hand, it is known that the predicates $H_k(Post)$ of the Dijkstra's wp definition for loops Do are correct invariants, which by definition have associated a termination condition (condition that when a state satisfying at the beginning of the iterations, cause than loop cant iterate more than k times). Learning to calculate $H_k(Post)$ represents an alternative to get invariants, without using the Hoare inference rules and without having to find bound functions, the same observation also holds for the weakest precondition of the loop, given a postcondition $Post$. In this work shows that for prooving loops correctness, it is much simpler to calculate or identify $H_k(Post)$ or the weakest precondition, rather than to conjecture an invariant by applying the Hoare rules.

Concretely if we have an algorithm of form Do and a postcondition $Post$, then in the development of the work the following questions are answered:

(i) Given an assertion Inv, is Inv the predicate $H_k(Post)$ of the loop for some k?

(ii) Given an assertion Inv, is Inv the weakest precondition of the loop with postcondition $Post$?

(iii) How do I calculate the predicate $H_k(Post)$ of the loop Do for some k?

(iv) How do I calculate the weakest precondition of loop Do and postcondition $Post$?

(v) How do I calculate the termination condition of a $H_k(Post)$?

Questions (i) and (ii) are answered by using Theorem 3.1 and Corollary 3.3 for loops with a potentially non-deterministic body, but which behaves deterministically on the variables that occur in the loop guard. Questions (iii) and (iv) are answered using a technique based on mathematical induction, but only for loops of form

$$do\ i \neq N \rightarrow S_0; i := i + 1\ od$$

or

$$do\ i \neq N \wedge C \rightarrow S_0; i := i + 1\ od$$

where S_0 does not modify variable i and operator $\wedge$ is short-circuited. For question (v) a general criterion is established in Theorem 3.1 to construct a predicate T_k that corresponds to the termination condition of $H_k(Post)$ for Do, on the other hand a criterion is established for the two types of previous loops that determines the sufficient conditions so that the termination condition is predicate $a \leq i \leq N$.

1.2 Related Jobs

Originally in [1] the recursive definition of wp that was exposed at the beginning did not include the syntactic function *domain* in its rules, this was corrected in [2], where it incorporates it to the rule of wp of the assignment, but not in the other rules as defined at the beginning of the introduction. A justification for the incorporation of *domain* in the rules of wp of IF and Do, is in [4], where a revision of the denotational semantics of GCL including state *abort* is made. The syntactic function *domain* applies on expressions, but its incorporation in the construction rules of wp of IF

and *Do*, bring additional difficulties that were not in [2], to handle these difficulties, in [5] is defined the syntactic function *support*, which is being the analogous to *domain*, but applies on instructions instead of expressions. This work use some new properties of *support* demostrated in [6].

On the other hand, in [7]- [10] also responds to questions (iv) and (v), using a semantic type method called "calculation of invariant relations" [11], which basically consists of obtaining the relation that results from the denotational semantic interpretation of the body of loop *Do*, and then calculate the reflexive-transitive closure of that relationship. This technique presupposes that the language for assertions must be a set theory language. This work is created as a continuation of [5] and is the syntactic counterpart of [11] [8] using *wp*. Basically the technique used here is to calculate a general formula that expresses the result of successively applying *wp* to the body of *Do*, this has its semantic counterpart in [11] [8] where the relation resulting from the interpretation of the body of the loop is successively composed. The approach presented here shows that one can easily answer questions (i),(ii),(iii),(iv) and (v) without having to go to the semantic world of [11] [8], in the classical assertion's language of book [2] and using GCL with all the expression power of non-determinism. To understand the relationship between the semantic technique of invariant relations and the syntactic technique of invariant assertion can be reviewed [12].

In the area of automatic derivation of invariants there has been a recent interest in recent years [13]- [22], furthermore there are applications like [23] [24] that can calculate invariants for loops where the expressions of the assignments of the loop body are all linear or translatable to linear transition systems, in the same way in [25] there is another technique that is applicable only to loops where the body is translatable to a affine transformation of vector spaces. Exists applications based on separation logic and Hoare logic [26]- [28], and on the other hand [29] is a application based in *wp*, but works only for unstructured programs.

The development of the techniques presented here, has as a long-term objective, to construct an invariant assertion calculus sufficiently efficient and clean so that it can be implemented just as the applications Mapple and Mathematica implemented integral calculus. An existing application that can calculate weakest preconditions, based on symbolic computations, requesting less conditions than the previous paragraph, is described in [30], which is an application based on invariant relations, made with reflexive-transitive closure packages of Mathematica's (Wolfram Research). Because this work is the syntactic counterpart of [11] [8], it is expected to implement a calculation application of $H_k(Post)$ based on the theorems of this work and [5], similar to [30].

2 No Determinism and Properties of *wp* and *support*

To demonstrate the theorems of the next sections, the following properties in [6] of the predicate transformer *wp* will be used.

Lemma 2.1 $wp(S, P \wedge Q) \equiv wp(S, P) \wedge wp(S, Q)$

Lemma 2.2 *Let P and R be a predicates, S a statement that behaves deterministically on the values of the variables of P and does not modify the values of the variables of R, and ϵ is a variable not declared in the program, then*

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

Lemma 2.3 *Let P and R be predicates, S a statement that does not modify the values of the variables of R, and ϵ is a variable not declared in the program. If $(\exists\epsilon| : R) \equiv true$, then*

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

Lemma 2.4 *Let P be a predicate and S a statement that does not modify the values of the variables of P, then*

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

where $support(S)$ is a predicate that depends on the constants and variables declared in the program, such that a state satisfies it if and only if the instruction S does not abort when executed in that state.

For example, $true$ is a predicate that for any S, holds that S does not modify its variables, so one way to calculate $support(S)$ is to calculate $wp(S, true) \equiv support(S) \wedge true \equiv support(S)$. For example if S is instruction $if\ a > -3 \rightarrow b := b/a\ []\ a \leq -3 \rightarrow b := 2\ fi$, then

$wp(S, true)$

$\equiv$

$(a > -3 \Rightarrow domain(b/a) \wedge true[b := b/a]) \wedge (a \leq -3 \Rightarrow true[b := 2])$

$\equiv$

$(a > -3 \Rightarrow a \neq 0) \wedge true$

therefore $support(S) \equiv a > -3 \Rightarrow a \neq 0$.

Lemma 2.5 *Let S be a statement, then*

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

Proof.

$support(S; i := i + 1) \equiv wp(S; i := i + 1, true) \equiv$
$wp(S, wp(i := i + 1, true)) \equiv wp(S, true) \equiv support(S)$ □

Lemma 2.6 *Let P and Q be predicates and S a statement. If S does not modify the values of the variables of P, then*

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

Lemma 2.7 *Let P and Q be predicates and S be a statement that behaves deterministically on the values of the variables of P, then*

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

Lemma 2.8 *Let S be a statement, P a predicate and ϵ a variable not declared in the program, then*

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

3 Weakest Precondition and $H_k(Post)$ of Instruction Do

In order to calculate the weakest precondition or $H_k(Post)$ of a loop, the following Theorem and Corollaries are presented.

Theorem 3.1 *Let k be an expression and Do an instruction of the form*

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

and k', ϵ, ϵ' variables not declared in the program (that is, they do not occur in Do) and do not occur in $Post$, S is an instruction (deterministic or non-deterministic).

The predicate $domBG$ is defined recursively such that:

- *In $domBG$ only occur ϵ' and the constants and variables of a program*
- $domBG[\epsilon' := 0] \equiv domain(B)$
- $domBG \equiv wp(S, domBG[\epsilon' := \epsilon' - 1])$ *when* $0 < \epsilon' \leq k \wedge domain(B) \wedge B \wedge support(S)$

Predicate NBG is defined recursively such that:

- *In NBG only occur ϵ and the constants and variables of a program*
- $NBG[\epsilon := 0] \equiv \neg B$
- $NBG \equiv wp(S, NBG[\epsilon := \epsilon - 1])$ *when* $0 < \epsilon \leq k \wedge domain(B) \wedge B \wedge support(S)$

Predicate $TI_{k'}$ is defined as:

$$domain(B) \wedge B \wedge (\exists \epsilon | 1 \leq \epsilon \leq k' : (\forall \epsilon' | 1 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG)$$

Predicate $T_{k'}$ is defined as:

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

Then, if S acts deterministically on the variables of $domBG$ and NBG, it holds that:

(i) *If there is a predicate inv such that:*

- $domain(B) \wedge \neg B \wedge Post \equiv domain(B) \wedge \neg B \wedge inv$
- $TI_{k'} \Rightarrow (wp(S, inv) \equiv inv)$

then

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

for all k' such that $0 \le k' \le k$.

(ii) *In addition to the hypotheses of (i), if the recurrence defining $domBG$ and NBG are defined up to $\epsilon, \epsilon' = k+1$, then*

$B \wedge wp(S, inv) \wedge (\forall \epsilon' | 0 \le \epsilon' \le k+1 : domBG) \wedge NBG[\epsilon := k+1] \Rightarrow T_k \ (*)$

If and only if

$$H_{k+1}(Post) \equiv H_k(Post)$$

The predicate inv of the theorem should not be confused with an invariant, this rather, is a sub-formula of an invariant Inv that is of the form $T_{k'} \wedge inv$.

Note that saying that S acts deterministically on the variables of $domBG$ and NBG, means that in each iteration, S acts deterministically on the variables of B.

Remark 3.2 *The predicate $T_{k'}$ will be called "termination condition", since it describes the weakest condition that causes the loop to iterate at most k' times. Likewise, predicate $TI_{k'}$ will be called "termination condition in the iteration" because it describes the condition that causes the loop to iterate at the most k', starting from a state that is within the iteration.*

The formula of the first item of (i) of the previous theorem will be called "termination test obligation" and the formula of the second item of (i) of the previous theorem will be called "iteration test obligation".

Note that the previous theorem says, that to prove that an assertion is a $H_{k'}(Post)$ of a given loop, then it is sufficient to demonstrate the two previous test obligations, which is much simpler than demonstrating the five test obligations that define the Hoare's logic for the invariant assertion. Next, Theorem 3.1 will be demonstrated.

Proof. Because this is a theorem about a formula whose instances are formulas, then a system of formal derivation of predicates will be used to ensure a correct result. The reader should understand the following demonstration as a family of demonstrations (one for each instance of predicates inv, NBG, and $domBG$), which results from applying each of the following derivations in the order they are presented. The rules of inference that are used in this work are those of the calculative logic (original of [31]) presented in the book of Gries [32].

It will be shown by induction on k' assuming that $k' \le k$ and that k' is a variable that does not occur in inv, NBG, $domBG$, S and $Post$.

Case 1 $k' = 0$

$H_{k'}(Post)$

$\equiv< k' = 0 >$

$H_0(Post)$

$\equiv$

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

$\equiv$

$domain(B) \wedge \neg B \wedge inv$

$\equiv$

$domBG[\epsilon' := 0] \wedge NBG[\epsilon := 0] \wedge inv$

$\equiv$

$(\forall \epsilon' | 0 \leq \epsilon' \leq 0 : domBG) \wedge NBG[\epsilon := 0] \wedge inv$

$\equiv$

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

$\equiv< k' = 0 >$

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

It is now assumed that the theorem is true for $k' - 1$ and will be proved for k'

$H_{k'}(Post)$

$\equiv$

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

$\equiv$<inductive hypothesis>

$H_0(Post) \vee (domain(B) \wedge B \wedge wp(S, (\exists \epsilon | 0 \leq \epsilon \leq k'-1 : (\forall \epsilon' | 0 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG) \wedge inv))$

$\equiv<$ S is deterministic in $domBG$ and NBG, it does not modify ϵ', ϵ, k', it exists ϵ' that $0 \leq \epsilon' \leq \epsilon$ (because $0 \leq \epsilon$) and Lemas 2.1,2.2,2.3>

$H_0(Post) \vee (domain(B) \wedge B \wedge (\exists \epsilon | 0 \leq \epsilon \leq k' - 1 : (\forall \epsilon' | 0 \leq \epsilon' \leq \epsilon : wp(S, domBG)) \wedge wp(S, NBG)) \wedge wp(S, inv))$

$\equiv< wp(S, P) \Rightarrow support(S)$ for any $P >$

$H_0(Post) \vee (domain(B) \wedge B \wedge (\exists \epsilon | 0 \leq \epsilon \leq k' - 1 :$
$(\forall \epsilon' | 0 \leq \epsilon' \leq \epsilon : wp(S, domBG)) \wedge wp(S, NBG)) \wedge wp(S, inv) \wedge support(S))$

$\equiv$<Definition of $domBG$ and NBG >

$H_0(Post) \vee (domain(B) \wedge B \wedge (\exists \epsilon | 0 \leq \epsilon \leq k' - 1 :$
$(\forall \epsilon' | 0 \leq \epsilon' \leq \epsilon : domBG[\epsilon' := \epsilon'+1]) \wedge NBG[\epsilon := \epsilon+1]) \wedge wp(S, inv) \wedge support(S))$

$\equiv< wp(S, P) \Rightarrow support(S)$ for any $P >$

$H_0(Post) \vee (domain(B) \wedge B \wedge (\exists \epsilon | 0 \leq \epsilon \leq k' - 1 :$
$(\forall \epsilon' | 0 \leq \epsilon' \leq \epsilon : domBG[\epsilon' := \epsilon' + 1]) \wedge NBG[\epsilon := \epsilon + 1]) \wedge wp(S, inv))$

$\equiv$

$H_0(Post) \vee (domain(B) \wedge B \wedge (\exists \epsilon | 0 \leq \epsilon \leq k' - 1 :$

$(\forall \epsilon' | 1 \leq \epsilon' \leq \epsilon + 1 : domBG) \wedge NBG[\epsilon := \epsilon + 1]) \wedge wp(S, inv))$

$\equiv$

$H_0(Post) \vee (domain(B) \wedge B \wedge (\exists \epsilon | 1 \leq \epsilon \leq k' :$
$(\forall \epsilon' | 1 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG) \wedge wp(S, inv))$

$\equiv$<Definition of inv >

$H_0(Post) \vee (domain(B) \wedge B \wedge (\exists \epsilon | 1 \leq \epsilon \leq k' :$
$(\forall \epsilon' | 1 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG) \wedge inv)$

$\equiv$<Definition of $H_0(Post)$ >

$(domain(B) \wedge \neg B \wedge Post) \vee (domain(B) \wedge B \wedge (\exists \epsilon | 1 \leq \epsilon \leq k' :$
$(\forall \epsilon' | 1 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG) \wedge inv)$

$\equiv$<Definition of inv >

$(domain(B) \wedge \neg B \wedge inv) \vee (domain(B) \wedge B \wedge (\exists \epsilon | 1 \leq \epsilon \leq k' :$
$(\forall \epsilon' | 1 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG) \wedge inv)$

$\equiv$<Distributivity of $\wedge$ over $\vee$ >

$(domain(B) \wedge inv) \wedge (\neg B \vee (B \wedge (\exists \epsilon | 1 \leq \epsilon \leq k' :$
$(\forall \epsilon' | 1 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG)))$

$\equiv$<Absorption>

$(domain(B) \wedge inv) \wedge (\neg B \vee (\exists \epsilon | 1 \leq \epsilon \leq k' : (\forall \epsilon' | 1 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG))$

$\equiv$<Distributivity of $\wedge$ over $\vee$ >

$inv \wedge ((domain(B) \wedge \neg B) \vee (domain(B) \wedge (\exists \epsilon | 1 \leq \epsilon \leq k' :$
$(\forall \epsilon' | 1 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG)))$

$\equiv$

$inv \wedge ((domain(B) \wedge \neg B) \vee (\exists \epsilon | 1 \leq \epsilon \leq k' : domain(B) \wedge$
$(\forall \epsilon' | 1 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG))$

$\equiv$<Definition of $domBG$ >

$inv \wedge ((domBG[\epsilon' := 0] \wedge \neg B) \vee (\exists \epsilon | 1 \leq \epsilon \leq k' : domBG[\epsilon' := 0] \wedge$
$(\forall \epsilon' | 1 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG))$

$\equiv$

$((domBG[\epsilon' := 0] \wedge \neg B) \vee (\exists \epsilon | 1 \leq \epsilon \leq k' : (\forall \epsilon' | 0 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG)) \wedge inv$

$\equiv$< $domBG[\epsilon' := 0] \equiv (\forall \epsilon' | 0 \leq \epsilon' \leq 0 : domBG)$ and definition of NBG >

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

On the other hand, to demonstrate (ii) of the theorem, it will be shown that $H_{k+1}(Post)$ is equivalent to a formula of form $p \vee H_k(Post)$ with p of form $wp(S, inv) \wedge q$, since in this way we have $H_{k+1}(Post) \equiv H_k(Post)$ iff $p \Rightarrow H_k(Post)$. But how $H_k(Post) \equiv T_k \wedge inv \equiv T_k \wedge wp(S, inv)$, then $p \Rightarrow H_k(Post)$ iff $p \Rightarrow T_k$.

It is assumed that $0 < k' \leq k + 1$ then the same first 8 steps are made like before but instead of inductive hypothesis in step 2 it applies directly $H_{k'-1}(Post) \equiv T_{k'-1} \wedge inv$, since in (i) it was shown that this is true for $0 < k' \leq k+1$. With this one has to $H_{k'}(Post)$ is equivalent to:

$$H_0(Post) \vee (domain(B) \wedge B \wedge (\exists \epsilon | 1 \leq \epsilon \leq k' : (\forall \epsilon' | 1 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG) \wedge wp(S, inv))$$

Instantiating $k' := k+1$, you have to $H_{k+1}(Post)$ is equivalent to

$$\begin{aligned}
&H_0(Post) \vee (domain(B) \wedge B \wedge wp(S, inv) \wedge \\
&\qquad (\exists \epsilon | 1 \leq \epsilon \leq k+1 : (\forall \epsilon' | 1 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG)) \\
&\equiv \\
&(domain(B) \wedge B \wedge wp(S, inv) \wedge (\forall \epsilon' | 1 \leq \epsilon' \leq k+1 : domBG) \wedge NBG[\epsilon := k+1]) \\
&\vee \\
&H_0(Post) \vee (domain(B) \wedge B \wedge wp(S, inv) \wedge \\
&\qquad (\exists \epsilon | 1 \leq \epsilon \leq k : (\forall \epsilon' | 1 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG))
\end{aligned}$$

The last disjunction of the previous formula is the same as that obtained earlier just before the equivalence that was labeled with the comment "Definition of inv", and it has already been shown that this formula is equivalent to $H_k(Post)$, therefore the previous formula It is equivalent to.

$$\begin{aligned}
&(domain(B) \wedge B \wedge wp(S, inv) \wedge (\forall \epsilon' | 1 \leq \epsilon' \leq k+1 : domBG) \wedge NBG[\epsilon := k+1]) \\
&\vee \\
&H_k(Post)
\end{aligned}$$

□

Corollary 3.3 *If a predicate inv satisfies the hypotheses of (i) of Theorem 3.1 for all k and does not satisfies formula (∗) of (ii) of the same theorem for any k, then defining T_∞ as*

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

is fulfilled that

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

on the other hand, if domBG, NBG and inv satisfies the hypotheses of (i) and (ii) and formula (∗), then

$$wp(Do, Post) \equiv H_k(Post) \equiv T_k \wedge inv$$

Proof. Immediate consequence of Theorem 3.1 and definition $wp(Do, Post) \equiv (\exists k' | k' \geq 0 : H_{k'}(Post))$ □

Corollary 3.4 *Let Do be a loop as in Theorem 3.1 with guard $i \neq N \wedge C$ (with short-circuited $\wedge$) and body $S_0; i := i+1$, where S_0 is a statement that does not modify neither i nor N. If domCG and NCG are defined like the predicates domBG and NBG of Theorem 3.1 but substituting B for C then:*

$$TI_{k'} \equiv i \neq N \wedge C \wedge$$
$$((N - k' \leq i < N \wedge (\forall \epsilon' | 0 \leq \epsilon' < N - i : domCG)) \vee$$
$$(\exists \epsilon | 1 \leq \epsilon \leq k' : (\forall \epsilon' | 0 \leq \epsilon' \leq \epsilon : domCG) \wedge NCG))$$

and

$$T_{k'} \equiv (N - k' \leq i \leq N \wedge (\forall \epsilon' | 0 \leq \epsilon' < N - i : domCG)) \vee$$
$$(\exists \epsilon | 0 \leq \epsilon \leq k' : (\forall \epsilon' | 0 \leq \epsilon' \leq \epsilon : domCG) \wedge NCG)$$

Additionally it is fulfilled that if a is a constant, it is holds:

(i) *If* $C \equiv true$, *then*
$T_{k'} \equiv N - k' \leq i \leq N$ *and* $TI_{k'} \equiv N - k' \leq i < N$

(ii) *If* $domain(C) \equiv a \leq i < N$ *or* $domain(C) \equiv a \leq i \leq N$, *then*
$T_{N-a} \equiv a \leq i \leq N$ *and* $TI_{N-a} \equiv a \leq i < N \wedge C$ *and* $wp(Do, Post) \equiv H_{N-a}(Post)$

4 Examples of Algorithm Correctness Using $H_k(Post)$

Then from the conjecture of *inv* the correctness of the following algorithm will be carried out

do $i \neq N \wedge A[i] \neq 0 \rightarrow$
$i := i + 1$
od
$\{Post : (\forall k | 0 \leq k < i : A[k] \neq 0)\}$

It is fulfilled that $domain(A[i] \neq 0) \equiv 0 \leq i < N$ and if we take as hypothesis TI_N, that in this case by Corollary 3.4 is $0 \leq i < N \wedge A[i] \neq 0$, then:

$$wp(i := i + 1, (\forall k | 0 \leq k < i : A[k] \neq 0))$$
$$\equiv$$
$$(\forall k | 0 \leq k < i + 1 : A[k] \neq 0)$$
$$\equiv$$
$$(\forall k | 0 \leq k < i : A[k] \neq 0) \wedge A[i] \neq 0$$
$$\equiv < A[i] \neq 0 \equiv true \text{ by hypothesis} >$$
$$(\forall k | 0 \leq k < i : A[k] \neq 0)$$

Taking *inv* as $(\forall k | 0 \leq k < i : A[k] \neq 0)$, the iteration test obligation is fulfilled and as $inv \equiv Post$, then the termination obligation test is trivially met, concluding that

$$H_N(Post) \equiv 0 \leq i \leq N \wedge (\forall k | 0 \leq k < i : A[k] \neq 0)$$

that according to (ii) of Corollary 3.4, is the weakest precondition of the algorithm.

It can be clearly seen that the foregoing is much simpler, than demonstrating the five test obligations established by the Hoare's logic, to prove the correctness of the previous loop.

On the other hand, Corollary 3.4 suggests a justification for the classical technique of derivation of invariants, called "replacement of constants by variable". This technique consists of substituting a constant N of the postcondition for a fresh variable i and using this new predicate as invariant of a loop Do with guard $i \neq N$ and increment of i of one in one.

For example, for a sort algorithm for an array A of length N with postcondition $Sorted(A, N)$, then the algorithm can be constructed based on a loop of the form:

do $i \neq N \rightarrow$
 S_0;
 $i := i + 1$
od
$\{Post : Sorted(A, N)\}$

Where the invariant is obtained by calculating $H_k(Post) \equiv N - k \leq i \leq N \wedge inv$ according to (i) of Corollary 3.4 and Theorem 3.1. To get inv we take the postcondition by substituting the constant N for i, this new predicate $inv \equiv Sorted(A, i)$ satisfies that $inv[i := N] \equiv Post$ and therefore the obligation test of termination.

According to Theorem 3.1 instruction $S_0; i := i + 1$ must satisfy that if TI_k is true, then

$$wp(S_0; i := i + 1, inv) \equiv wp(S_0, Sorted(A, i + 1)) \equiv inv$$

so instruction S_0 (using specification statements of [33]) must be

$$[TI_k \wedge Sorted(A, i), Sorted(A, i + 1)].$$

The above specification instruction is the most general of all that we can use, but any instruction that is a refinement of it, is an instruction that guarantees a correct sort algorithm. The internal loop of the Bubblesort or Insertsort algorithm are examples of refinements of the previous specification instruction.

5 Computation theorems for predicate inv

The previous Theorem and Corollaries have the same limitation as the Invariance Theorem, which pretends that an invariant predicate inv be searched without any particular method or heuristic. Next, a Theorem will be given, which suggests a method that allows to obtain a predicate inv like the one in the Theorems of the previous section, based on the calculation of $wp(S, wp(S, \ldots, wp(S, Post), \ldots))$ a number ϵ of times.

Lemma 5.1 *Let S_0 be a statement that does not modify the value of variables i and i_f. Let k be an expression and let ϵ be a variable not declared in the program. Let PG be a predicate such that $PG \equiv wp(S_0; i := i + 1, PG[\epsilon := \epsilon - 1])$ when $0 < \epsilon \leq k \wedge domain(B) \wedge B \wedge support(S_0; i := i + 1)$, then assuming that $i_f - k' \leq i < i_f$,* *you have that for* $0 < k' \leq k$*:*

$$TI_{k'} \wedge support(S_0) \quad \Rightarrow \quad wp(S_0; i := i+1, PG[\epsilon := i_f - i]) \equiv PG[\epsilon := i_f - i]$$

Proof. To make this demonstration, it is assumed to be true $i_f - k' \leq i < i_f$ and $TI_{k'}$. But how $i_f - k' \leq i < i_f \Rightarrow i_f - k \leq i < i_f$ and $TI_{k'} \Rightarrow domain(B) \wedge B$, you can assume $i_f - k \leq i < i_f$ and $domain(B) \wedge B$ as well. It also $support(S_0; i := i+1)$ is assumed since by Lema 2.5 $support(S_0) \equiv support(S_0; i := i+1)$.

$wp(S_0; i := i+1, PG[\epsilon := i_f - i])$

$\equiv$

$wp(S_0; i := i+1, (\forall \epsilon | \epsilon = i_f - i : PG))$

$\equiv$

$wp(S_0, (\forall \epsilon | \epsilon = i_f - (i+1) : PG[i := i+1]))$

$\equiv$< S_0 does not modify ϵ, i_f, i, there is ϵ such that $\epsilon = i_f - (i+1)$ and Lema 2.3>

$(\forall \epsilon | \epsilon = i_f - (i+1) : wp(S_0, PG[i := i+1]))$

$\equiv$

$(\forall \epsilon | \epsilon = i_f - (i+1) : wp(S_0; i := i+1, PG))$

$\equiv$< Hypothesis $i_f - k \leq i < i_f$ implies $0 \leq i_f - (i+1) < k$ >

$(\forall \epsilon | \epsilon = i_f - (i+1) \wedge 0 \leq i_f - (i+1) < k : wp(S_0; i := i+1, PG))$

$\equiv$

$(\forall \epsilon | \epsilon = i_f - (i+1) \wedge 0 \leq \epsilon < k : wp(S_0; i := i+1, PG))$

$\equiv$<Hypotheses are satisfied to apply definition of PG >

$(\forall \epsilon | \epsilon = i_f - (i+1) \wedge 0 \leq \epsilon < k : PG[\epsilon := \epsilon + 1])$

$\equiv$< $0 \leq \epsilon < k$ is redundant>

$(\forall \epsilon | \epsilon = i_f - (i+1) : PG[\epsilon := \epsilon + 1])$

$\equiv$

$PG[\epsilon := \epsilon + 1][\epsilon := i_f - (i+1)]$

$\equiv$

$PG[\epsilon := i_f - i]$ □

Lemma 5.2 *Let R be a predicate and S a statement that behaves deterministically on the values of the variables of R. If i_f is a variable not declared in the program and Exp is an expression in which S does not modify the value of its variables, then*

$$wp(S, Exp = (min\ i_f | R : i_f)) \equiv$$

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

Proof. Analogous to Lema 17 of [6] □

Lemma 5.3 *Let S_0 be a statement that does not modify the value of variable i. It is defined NBG as in Theorem 3.1, taking S as $S_0; i := i+1$. Let k be an expression* *and let* ϵ, i_f *and* k' *be variables not declared in the program, then abbreviating* $\underline{m}$ *as*

$$(min\ i_f | i \leq i_f \leq i + k' \wedge NBG[\epsilon := i_f - i] : i_f),$$

you have that for $0 < k' \leq k$*:*

$$TI_{k'} \qquad \Rightarrow \qquad wp(S_0; i := i + 1, \epsilon = \underline{m} - i) \equiv support(S_0) \wedge \epsilon = \underline{m} - i - 1$$

Proof. Assuming $TI_{k'}$ you have to:

It is true $\neg NBG[\epsilon := i - i]$, and therefore, to consider that i_f can be equal to i in the calculation of $\underline{m}$, it is impossible, in this way:

$\underline{m}$
$=$
$(min\ i_f | i \leq i_f \leq i + k' \wedge NBG[\epsilon := i_f - i] : i_f)$
$=$
$(min\ i_f | i < i_f \leq i + k' \wedge NBG[\epsilon := i_f - i] : i_f) \qquad (**)$

With this it can be deduced that:

$wp(S_0; i := i + 1, \epsilon = \underline{m} - i)$
$\equiv$
$wp(S_0, \epsilon = \underline{m}[i := i + 1] - i - 1)$
$\equiv$
$wp(S_0, \epsilon + i + 1 = (min\ i_f | i < i_f \leq i + k' + 1 \wedge NBG[\epsilon := i_f - i][i := i + 1] : i_f))$

$\equiv$<Lemas 5.2 and 2.6>

$support(S_0) \wedge \epsilon + i + 1 = (min\ i_f | i < i_f \leq i + k' + 1 \wedge$
$wp(S_0, NBG[\epsilon := i_f - i][i := i + 1]) : i_f)$
$\equiv$
$support(S_0) \wedge \epsilon + i + 1 = (min\ i_f | i < i_f \leq i + k' + 1 \wedge$
$wp(S_0; i := i + 1, NBG[\epsilon := i_f - i]) : i_f)$

$\equiv$<Abbreviating $(min\ i_f | i_f = i + k' + 1 \wedge wp(S_0; i := i + 1,$
$NBG[\epsilon := i_f - i]) : i_f)$ as m_1 >
$support(S_0) \wedge \epsilon = min(m_1, (min\ i_f | i < i_f \leq i + k' \wedge$
$wp(S_0; i := i + 1, NBG[\epsilon := i_f - i]) : i_f)) - i - 1$
$\equiv$<Lema 5.1>

$support(S_0) \wedge \epsilon = min(m_1, (min\ i_f | i < i_f \leq i + k' \wedge NBG[\epsilon := i_f - i] : i_f)) - i - 1$

$\equiv$<observation $(**)$ >

$support(S_0) \wedge \epsilon = min(m_1, \underline{m}) - i - 1$

$\equiv$< $TI_{k'}$ implies that exists $\underline{m} \leq i + k' < m_1$ >

$support(S_0) \wedge \epsilon = \underline{m} - i - 1$ □

Theorem 5.4 *Let Do be a loop where S is the statement $S_0; i := i + 1$ with S_0 a statement that does not modify the value of the variable i. Let k be an expression and be ϵ, i_f and k' variables not declared in the program, which does not occur in Post and $0 < k' \leq k$, defining the predicate NBG as in Theorem 3.1 and a predicate PostG that satisfies the following recursive equations:*

- $PostG[\epsilon := 0] \equiv Post$
- $PostG \equiv wp(S_0; i := i + 1, PostG[\epsilon := \epsilon - 1])$ *when* $0 < \epsilon \leq k$ *and* TI_k

then abbreviating $\underline{m}$ as

$$(min\ i_f | i \leq i_f \leq i + k' \wedge NBG[\epsilon := i_f - i] : i_f),$$

you have to:

(i) *The predicate $PostG[\epsilon := \underline{m} - i]$ is a predicate, which satisfies the hypotheses of the predicate inv in (i) of Theorem 3.1.*

(ii) *Additionally if the recursion that defines PostG, domBG and NBG are defined up to k+1, then $TI_{k+1} \wedge PostG[\epsilon := k+1] \Rightarrow TI_k$ is equivalent to $H_{k+1}(Post) \equiv H_k(Post)$.*

Proof. *inv* is defined as $PostG[\epsilon := \underline{m} - i]$ and it will be shown that *inv* complies with the equations of the Theorem 3.1

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

$\equiv$

$domain(B) \wedge NBG[\epsilon := 0] \wedge PostG[\epsilon := 0]$

$\equiv$

$domain(B) \wedge NBG[\epsilon := i - i] \wedge PostG[\epsilon := i - i]$

$\equiv$<Since $NBG[\epsilon := i - i] \equiv \neg B \equiv true$ then $\underline{m} = i$

$domain(B) \wedge NBG[\epsilon := i - i] \wedge PostG[\epsilon := \underline{m} - i]$

$\equiv$

$domain(B) \wedge \neg B \wedge inv$

On the other hand assuming $TI_{k'}$ you have to

$wp(S_0; i := i + 1, inv)$

$\equiv$

$wp(S_0; i := i + 1, PostG[\epsilon := \underline{m} - i])$

$\equiv$

$wp(S_0; i := i + 1, (\forall \epsilon | : \epsilon = \underline{m} - i \Rightarrow PostG))$

$\equiv$<Lema 2.8>

$(\forall \epsilon| : wp(S_0; i := i+1, \epsilon = \underline{m} - i \Rightarrow PostG))$

$\equiv <$ $S_0; i := i+1$ acts deterministically on the variables
of NBG and the variables i_f, i, k', ϵ and Lemas 2.7 and 2.5$>$

$(\forall \epsilon| : support(S_0) \wedge (wp(S_0; i := i+1, \epsilon = \underline{m} - i) \Rightarrow wp(S_0; i := i+1, PostG)))$

$\equiv <$Lema 5.3$>$

$(\forall \epsilon| : support(S_0) \wedge (\epsilon = \underline{m} - i - 1 \Rightarrow wp(S_0; i := i+1, PostG)))$
$\equiv$
$support(S_0) \wedge (\forall \epsilon| : \epsilon = \underline{m} - i - 1 \Rightarrow wp(S_0; i := i+1, PostG))$
$\equiv$
$support(S_0) \wedge (\forall \epsilon|\epsilon = \underline{m} - i - 1 : wp(S_0; i := i+1, PostG))$

$\equiv < TI_{k'}$ implies that exists $\underline{m}$ and therefore exists ϵ such that $\epsilon = \underline{m} - i - 1 >$

$(\forall \epsilon|\epsilon = \underline{m} - i - 1 : support(S_0) \wedge wp(S_0; i := i+1, PostG))$

$\equiv < wp(S, P) \Rightarrow support(S)$ for any P and Lema 2.5$>$

$(\forall \epsilon|\epsilon = \underline{m} - i - 1 : wp(S_0; i := i+1, PostG))$

$\equiv < TI_{k'} \Rightarrow 0 \leq \underline{m} - i - 1 < k'$ and def of $PostG >$

$(\forall \epsilon|\epsilon = \underline{m} - i - 1 : PostG[\epsilon := \epsilon + 1])$
$\equiv$
$PostG[\epsilon := \epsilon + 1][\epsilon := \underline{m} - i - 1]$
$\equiv$
$PostG[\epsilon := \underline{m} - i]$
$\equiv$
inv

On the other hand, assuming the hypotheses of (ii) of the theorem, the previous proof is valid taking k as $k+1$ and therefore $TI_{k+1} \Rightarrow (wp(S, inv) \equiv inv)$, then demonstrating by cases we have to that if $T_k \equiv false$ then

$B \wedge wp(S, inv) \wedge (\forall \epsilon'|0 \leq \epsilon' \leq k+1 : domBG) \wedge NBG[\epsilon := k+1] \Rightarrow T_k$
$\equiv < T_k \equiv false >$
$B \wedge wp(S, inv) \wedge (T_k \vee ((\forall \epsilon'|0 \leq \epsilon' \leq k+1 : domBG) \wedge NBG[\epsilon := k+1])) \Rightarrow T_k$
$\equiv$
$B \wedge wp(S, inv) \wedge T_{k+1} \Rightarrow T_k$
$\equiv < B \wedge T_{k+1} \equiv TI_{k+1}$ and $TI_{k+1} \Rightarrow (wp(S, inv) \equiv inv) >$
$TI_{k+1} \wedge inv \Rightarrow T_k$
$\equiv$
$TI_{k+1} \wedge PostG[\epsilon := \underline{m} - i] \Rightarrow T_k$

Since T_k is false then $\underline{m} - i > k$, but in conjunction with TI_{k+1} which implies $NBG[\epsilon := k+1]$, It is fulfilled $\underline{m} - i = k+1$ and therefore the previous formula is equivalent to

$$TI_{k+1} \wedge PostG[\epsilon := k+1] \Rightarrow T_k$$

For the case in which T_k is true, trivially the last and first implication are equivalent, with which by (ii) of Theorem 3.1, we have the proof. □

6 Examples of Calculation of Invariants Assertions

Theorem 5.4 suggests a method to calculate $H_k(Post)$ of a loop. The technique consists of applying the predicate transformer to the body of the loop and the postcondition ϵ times until the predicate $PostG$ is deduced. An example of the use of Theorem 5.4 is shown below:

6.1 Fibonacci

do $i \neq N \rightarrow$
    $x, z := z, x+z;$
    $i := i+1$
od
$\{Post : z = fib(N+1)\}$

The parallel assignment instruction $x, z, i := z, x+z, i+1$ is equivalent to the two instructions of the internal block of the loop, so in order to summarize, the parallel assignment instruction will be use in the calculations of this example.

wp is applied once to the body of the loop and to the postcondition:
$wp(x, z, i := z, x+z, i+1, z = fib(N+1))$
$\equiv$
$x + z = fib(N+1)$

Now to the previous result is applied again wp
$wp(x, z, i := z, x+z, i+1, x+z = fib(N+1))$
$\equiv$
$z + (x+z) = fib(N+1)$
$\equiv$
$x + 2z = fib(N+1)$

If to the previous result it is apply again wp, it is obtained
$wp(x, z, i := z, x+z, i+1, x+2z = fib(N+1))$
$\equiv$
$z + 2(x+z) = fib(N+1)$
$\equiv$

$2x + 3z = fib(N + 1)$

If to the previous result it is apply again wp, it is obtained $3x+5z = fib(N+1)$, so it is observed that the coefficients that accompany the x and y are the numbers of the Fibonacci sequence, so it is easy to show by induction, that the result of applying wp to the body of this loop and to postcondition $z = fib(N+1)$ a number of ϵ times is equal to

$$fib(\epsilon)x + fib(\epsilon + 1)z = fib(N + 1)$$

We will call this predicate $PostG$, which for $\epsilon = N$ is satisfiable (taking $x, z := 0, 1$), so $PostG$ is not false when $\epsilon = N$, for this reason $PostG$ is not false for any $\epsilon \leq N$. This is because if $PostG \equiv false$ for some $\epsilon = e < N$, then $PostG$ would be false for all $\epsilon > e$ (including $\epsilon = N$), since $wp(S, false) \equiv false$ and $PostG$ was obtained from applying wp successively. Therefore, you can not use (ii) of the Theorem 5.4 for any $k < N$.

Therefore, predicate $PostG$ satisfies the recurrence of Theorem 5.4 by taking k as N. Since $\underline{m} = N$, it is concluded that $PostG[N - i]$ satisfies the hypotheses of (i) of Theorem 3.1 and by the previous paragraph, does not comply with (ii) of Theorem 3.1. Therefore, it is obtained:

$H_N(Post)$
$\equiv$
$T_N \wedge (fib(\epsilon)x + fib(\epsilon + 1)z = fib(N + 1))[\epsilon := N - i]$
$\equiv$
$0 \leq i \leq N \wedge fib(N - i)x + fib(N - i + 1)z = fib(N + 1)$
Which is a valid invariant assertion for the loop.

6.2 Palindrome Words

The following algorithm for the verification of whether an array of characters of size N is a palindrome String is an example of the use of Theorem 5.4.

do $i \neq N \wedge A[i - 1] = A[N - i] \rightarrow$
  $pal := A[i] = A[N - 1 - i]$;
  $i := i + 1$
od
$\{Post : pal \equiv palind(A, 0, N)\}$

Where predicate $palind(A, i, N)$ is defined as

$$(\forall k | i \leq k < N : A[k] = A[N - 1 - k])$$

The parallel assignment instruction $pal, i := A[i] = A[N-1-i], i+1$, is equivalent to the two instructions of the internal block of the previus loop, so in order to summarize calculations, the parallel assignment instruction will be use instead of internal block of the previus loop.

Since $domain(A[i-1] = A[N-i]) \equiv 1 \leq i \leq N$, then Corollary 3.4 says that $TI_{N-1} \equiv 1 \leq i < N \wedge \quad A[i-1] = A[N-i]$ and $T_{N-1} \equiv 1 \leq i \leq N$

Assuming TI_{N-1}, the transformer wp is applied to the body of the loop and the postcondition

$wp(pal, i := A[i] = A[N-1-i], i+1, pal \equiv palind(A, 0, N))$

$\equiv$

$0 \leq i < N \wedge (A[i] = A[N-1-i] \equiv palind(A, 0, N))$

$\equiv < 0 \leq i < N \equiv true$ by hypothesis $TI_{N-1} >$

$A[i] = A[N-1-i] \equiv palind(A, 0, N)$

Assuming TI_{N-1}, transformer wp is now applied to the body of the loop and to the previous result

$wp(pal, i := A[i] = A[N-1-i], i+1, A[i] = A[N-1-i] \equiv palind(A, 0, N))$

$\equiv$

$0 \leq i < N \wedge (A[i+1] = A[N-2-i] \equiv palind(A, 0, N))$

$\equiv < 0 \leq i < N \equiv true$ by hypothesis $TI_{N-1} >$

$A[i+1] = A[N-2-i] \equiv palind(A, 0, N)$

By induction it can be shown that applying a ϵ $(\leq N-1)$ number of times, the wp transformer to the body of the loop and the postcondition, the following satisfiable formula is obtained

$$(\epsilon = 0?pal : A[i+\epsilon-1] = A[N-\epsilon-i]) \equiv palind(A, 0, N)$$

In the same way, when applying the wp transformer to the body of the loop and the denied guard a ϵ $(\leq N-1)$ number of times, the satisfiable formula $i + \epsilon = N \vee A[i+\epsilon-1] \neq A[N-i-\epsilon]$ is obtained, that when replacing $\epsilon := i_f - i$ it result

$$i_f = N \vee A[i_f - 1] \neq A[N - i_f].$$

Thus, the weakest precondition of the loop is:

$$1 \leq i \leq N$$
$$\wedge$$
$$((\underline{m} = i?pal : A[\underline{m}-1] = A[N-\underline{m}]) \equiv palind(A, 0, N))$$

where $\underline{m}$ is an abbreviation of

$$(min\ i_f | i \leq i_f \leq i + N - 1 \wedge (i_f = N \vee A[i_f - 1] \neq A[N - i_f]) : i_f)$$

7 Conclusions

The theorems presented here are a small contribution to the development of a pragmatic calculation for the correction of programs. The examples presented here show that using the appropriate theorems, it is possible to get the weakest precondition or $H_k(Post)$ of certain instructions Do, in a fast and formal way.

Since invariant relationships and invariant assertions are related [12], for future research it is proposed to extract from the implementation of [30], the aspects that allow the implementation of the invariant calculation technique described in this paper. This implementation will allow not only calculating the invariants $H_k(Post)$, but also the complexity of the algorithms, since if f is the complexity function of the body of a loop and it can be verified that the initial conditions of the iteration satisfy $H_k(Post)$, then it is inferred, that the complete algorithm is of complexity $O(kf)$.

References

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

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

[3] C. A. R. Hoare, *An Axiomatic Basis for Computer Programming*, Commun. of the ACM, 12(10):576-580, 1969.

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

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

[6] F. Flaviani. *Propiedades Algebraicas y Decidibilidad del Transformador de Predicados wp sobre la Teoría de Conjuntos*. Proc. CoNCISa 2017, Ciudad Guayana, Venezuela, 2017, pp. 94-105.

[7] 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.

[8] 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.

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

[10] 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.

[11] A. Mili, S. Aharon, C. Nadkarni, L.L. Jilani, A. Louhichi, O. Mraihi, *Reflexive transitive invariant relations: A basis for computing loop functions*, J. Symbolic Comput., vol 45, no. 11, pp. 1114-1143, 2010.

[12] O. Mraihi, A. Louhichi, L.L. Jilani, J. Desharnais, A. Mili, *Invariant Assertions, Invariant Relations, and Invariant Functions*, Science of Computer Programming, Elsevier, vol 78, no. 9, pp. 1212-1239, 2013.

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

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

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

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

[17] 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.

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

[19] T. Jebelean, M. Giese. in *First International Workshop on Invariant Generation*, Research Institute on Symbolic Computation, Hagenberg, Austria, 2007.

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

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

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

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

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

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

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

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

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

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

[30] A. Louhichi, O. Mraihi, W. Ghardallou, L.L. Jilani, K. Bsaies, A. Mili. *Invariant Relations: An Automated Tool to Analyze Loops*, Proc. 5th Int. conference on Verification and Evaluation of Computer and Communication Systems, 2011, pp. 84-95.

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

[32] D. Gries, F. B. Schneider. *A Logical Approach to Discrete Math*, New York, *Springer*, 1993.

[33] C. Morgan. *The specification statement*, ACM Transactions on Programming Languages and Systems, Vol. 10, no. 3,pp. 403-419, 1988.