Available online at [www.sciencedirect.com](http://www.sciencedirect.com/)

[Electronic Notes in Theoretical Computer Science 339 (2018) 63–83](https://doi.org/10.1016/j.entcs.2018.06.005)

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Calculation of Invariants Assertions

Federico Flaviani[1](#_bookmark0)

*Universidad Simo´n Bol´ıvar 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 *Hk*(*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.](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 Com- mand Language) [[1](#_bookmark15)], 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 correc- tion activity of a program. All assertions in this paper shall be assumed to be written in the language of the assertions of [[2](#_bookmark16)].

Dijkstra’s logic [[1](#_bookmark15)] 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 post- condition *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 calculat- ing weakest preconditions between instruction and instruction, from the end of the program to the beginning.

1 Email: [fflaviani@usb.ve](mailto:fflaviani@usb.ve)

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

1571-0661/© 2018 The Author(s). Published by Elsevier B.V.

This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

Dijkstra in [[1](#_bookmark15)] established the rules that define the function of syntactic trans- formation *wp* according to the following paragraph:

If *B, B*0*,..., Bn* and *S, S*0*,..., Sn* are Boolean expressions and GCL’s statements respectively, if *IF* and *Do* are abreviations of statements *if B*0 *→ S*0[] *...* []*Bn → Sn fi* and *do B → S od* respectively and if *domain*(*B*0*,..., Bn*) denoted a predicate that if is satisfied in a state, none of the expressions *Bi*, when these evaluated in that state, these incur an illegal operation (such as dividing by 0), then:

* *wp*(*SKIP, Post*) := *Post*
* *wp*(*yi*1 *,..., yik* := *Exp*1*,..., Expk, Post*) := *domain*(*Exp*1*,..., Expk*) *∧*

*Post*[*yi*1 *,..., yik* := *Exp*1*,..., Expk*]

* *wp*(*S*0; *S*1*, Post*) := *wp*(*S*0*, wp*(*S*1*, Post*))
* *wp*(*IF, Post*) := *domain*(*B*0*,..., Bn*)*∧*(*B*0 *∨· · ·∨Bn*)*∧*(*B*0 *⇒ wp*(*S*0*, Post*))*∧...*

*∧* (*Bn ⇒ wp*(*Sn, Post*))

* *wp*(*Do, Post*) := (*∃k|k ≥* 0: *Hk*(*Post*))

where *Hk*(*Post*) is a predicate that satisfies the equations:

*H*0(*Post*) *≡ domain*(*B*) *∧ ¬B ∧ Post*

*Hk*(*Post*) *≡*

*H*0(*Post*) *∨* (*domain*(*B*) *∧ B ∧ wp*(*S, Hk−*1(*Post*)))

for *k ≥* 1

* 1. *Contribution*

In Hoare’s logic [[3](#_bookmark17)] to make the partial correction of a loop *Do* with postcondition

*Post*, the invariant rule must be applied:

*Inv ⇒ domain*(*B*) *{Inv ∧ B}S{Inv}*

*{Inv}do B → S od{Inv ∧ ¬B},*

in combination with rule

*{Inv}do B → S od{Inv ∧ ¬B} Inv ∧ ¬B ⇒ Post*

*{Inv}do B → S od{Post}.*

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

and

*Inv ∧ B ⇒ f* (***x***) *≥* 0

*{Inv ∧ B ∧ f*0 = *f* (***x***)*}S{f*0 *> f* (***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 *Hk*(*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 calcu- late *Hk*(*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 calcu- late or identify *Hk*(*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:

* + 1. Given an assertion *Inv*, is *Inv* the predicate *Hk*(*Post*) of the loop for some *k*?
    2. Given an assertion *Inv*, is *Inv* the weakest precondition of the loop with post- condition *Post*?
    3. How do I calculate the predicate *Hk*(*Post*) of the loop *Do* for some *k*?
    4. How do I calculate the weakest precondition of loop *Do* and postcondition

*Post*?

* + 1. How do I calculate the termination condition of a *Hk*(*Post*)?

Questions (i) and (ii) are answered by using Theorem [3.1](#_bookmark8) and Corollary [3.3](#_bookmark9) 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 /*= *N → S*0; *i* := *i* +1 *od*

or

*do i /*= *N Λ C → S*0; *i* := *i* +1 *od*

where *S*0 does not modify variable *i* and operator *Λ* is short-circuited. For question

(v) a general criterion is established in Theorem [3.1](#_bookmark8) to construct a predicate *Tk* that corresponds to the termination condition of *Hk*(*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 ≤ i ≤ N* .

* 1. *Related Jobs*

Originally in [[1](#_bookmark15)] 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](#_bookmark16)], 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](#_bookmark18)], 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](#_bookmark16)], to handle these difficulties, in [[5](#_bookmark19)] 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](#_bookmark20)].

On the other hand, in [[7](#_bookmark21)]- [[10](#_bookmark23)] also responds to questions (iv) and (v), using a semantic type method called “calculation of invariant relations” [[11](#_bookmark24)], which basi- cally 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 as- sertions must be a set theory language. This work is created as a continuation of [[5](#_bookmark19)] and is the syntactic counterpart of [[11](#_bookmark24)] [[8](#_bookmark22)] 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](#_bookmark24)] [[8](#_bookmark22)] 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](#_bookmark24)] [[8](#_bookmark22)], in the classical assertion’s language of book [[2](#_bookmark16)] 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](#_bookmark25)].

In the area of automatic derivation of invariants there has been a recent interest in recent years [[13](#_bookmark26)]- [[22](#_bookmark27)], furthermore there are applications like [[23](#_bookmark28)] [[24](#_bookmark29)] 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](#_bookmark30)] 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](#_bookmark31)]- [[28](#_bookmark32)], and on the other hand [[29](#_bookmark33)] 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 para- graph, is described in [[30](#_bookmark34)], which is an application based on invariant relations, made with reflexive-transitive closure packages of Mathematica’s (Wolfram Research). Be- cause this work is the syntactic counterpart of [[11](#_bookmark24)] [[8](#_bookmark22)], it is expected to implement a calculation application of *Hk*(*Post*) based on the theorems of this work and [[5](#_bookmark19)], similar to [[30](#_bookmark34)].

1. **No Determinism and Properties of** *wp* **and** *support*

To demonstrate the theorems of the next sections, the following properties in [[6](#_bookmark20)] of the predicate transformer *wp* will be used.

**Lemma 2.1** *wp*(*S, P Λ Q*) *≡ wp*(*S, P* ) *Λ wp*(*S, Q*)

**Lemma 2.2** *Let P and R be a predicates, S a statement that behaves determin- istically 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,* (*∃ϵ|R* : *P* )) *≡* (*∃ϵ|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* (*∃ϵ|* : *R*) *≡ true, then*

*wp*(*S,* (*∀ϵ|R* : *P* )) *≡* (*∀ϵ|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* ) *≡ support*(*S*) *Λ 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 mod- ify its variables, so one way to calculate *support*(*S*) is to calculate *wp*(*S, true*) *≡ support*(*S*) *Λ true ≡ support*(*S*). For example if *S* is instruction *if a > −*3 *→ b* := *b/a* [] *a ≤ −*3 *→ b* := 2 *fi*, then

*wp*(*S, true*)

*≡*

(*a > −*3 *⇒ domain*(*b/a*) *Λ true*[*b* := *b/a*]) *Λ* (*a ≤ −*3 *⇒ true*[*b* := 2])

*≡*

(*a > −*3 *⇒ a /*= 0) *Λ true*

therefore *support*(*S*) *≡ a > −*3 *⇒ a /*= 0.

**Lemma 2.5** *Let S be a statement, then*

*support*(*S*; *i* := *i* + 1) *≡ support*(*S*)

**Proof.**

*support*(*S*; *i* := *i* + 1) *≡ wp*(*S*; *i* := *i* + 1*, true*) *≡*

*wp*(*S, wp*(*i* := *i* + 1*, true*)) *≡ wp*(*S, true*) *≡ support*(*S*) *2*

**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 Λ Q*) *≡ P Λ wp*(*S, Q*)

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

*wp*(*S, P ⇒ Q*) *≡ support*(*S*) *Λ* (*wp*(*S, P* ) *⇒ 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,* (*∀ϵ|* : *P* )) *≡* (*∀ϵ|* : *wp*(*S, P* ))

1. **Weakest Precondition and** *Hk*(*Post*) **of Instruction** *Do*

In order to calculate the weakest precondition or *Hk*(*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 →*

*S*

*od*

*{Post}*

*and kj, ϵ, ϵj variables not declared in the program (that is, they do not oc- cur 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 ϵj and the constants and variables of a program*
* *domBG*[*ϵj* := 0] *≡ domain*(*B*)
* *domBG ≡ wp*(*S, domBG*[*ϵj* := *ϵj −* 1]) *when* 0 *< ϵj ≤ k Λ domain*(*B*) *Λ B Λ*

*support*(*S*)

*Predicate NBG is deﬁned recursively such that:*

* *In NBG only occur ϵ and the constants and variables of a program*
* *NBG*[*ϵ* := 0] *≡ ¬B*
* *NBG ≡ wp*(*S, NBG*[*ϵ* := *ϵ −* 1]) *when* 0 *< ϵ ≤ k Λ domain*(*B*) *Λ B Λ support*(*S*)

*Predicate T Ik′ is deﬁned as:*

*domain*(*B*) *Λ B Λ* (*∃ϵ|*1 *≤ ϵ ≤ kj* : (*∀ϵj|*1 *≤ ϵj ≤ ϵ* : *domBG*) *Λ NBG*)

*Predicate Tk′ is deﬁned as:*

(*∃ϵ|*0 *≤ ϵ ≤ kj* : (*∀ϵj|*0 *≤ ϵj ≤ ϵ* : *domBG*) *Λ NBG*)

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

1. *If there is a predicate inv such that:*
   * *domain*(*B*) *Λ ¬B Λ Post ≡ domain*(*B*) *Λ ¬B Λ inv*
   * *TIk′ ⇒* (*wp*(*S, inv*) *≡ inv*)

*then*

*Hk′* (*Post*) *≡ Tk′ Λ inv*

*for all kj such that* 0 *≤ kj ≤ k.*

1. *In addition to the hypotheses of (i), if the recurrence deﬁning domBG and*

*NBG are deﬁned up to ϵ, ϵj* = *k* + 1*, then*

*B Λ wp*(*S, inv*) *Λ* (*∀ϵj|*0 *≤ ϵj ≤ k* +1 : *domBG*) *Λ NBG*[*ϵ* := *k* + 1] *⇒ Tk* (*∗*)

*If and only if*

*Hk*+1(*Post*) *≡ Hk*(*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 *Tk′ Λ 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 Tk′ will be called “termination condition”, since it de- scribes the weakest condition that causes the loop to iterate at most kj times. Like- wise, predicate TIk′ will be called “termination condition in the iteration” because it describes the condition that causes the loop to iterate at the most kj, starting from a state that is within the iteration.*

*The formula of the ﬁrst item of (i) of the previous theorem will be called “ter- mination 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 *Hk′* (*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](#_bookmark8) 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 pre- sented. The rules of inference that are used in this work are those of the calculative logic (original of [[31](#_bookmark35)]) presented in the book of Gries [[32](#_bookmark36)].

It will be shown by induction on *kj* assuming that *kj ≤ k* and that *kj* is a variable that does not occur in *inv*, *NBG*, *domBG*, *S* and *Post*.

Case 1 *kj* =0

*Hk′* (*Post*)

*≡< kj* =0 *> H*0(*Post*)

*≡*

*domain*(*B*) *Λ ¬B Λ Post*

*≡*

*domain*(*B*) *Λ ¬B Λ inv*

*≡*

*domBG*[*ϵj* := 0] *Λ NBG*[*ϵ* := 0] *Λ inv*

*≡*

(*∀ϵj|*0 *≤ ϵj ≤* 0: *domBG*) *Λ NBG*[*ϵ* := 0] *Λ inv*

*≡*

(*∃ϵ|*0 *≤ ϵ ≤* 0: (*∀ϵj|*0 *≤ ϵj ≤ ϵ* : *domBG*) *Λ NBG*) *Λ inv*

*≡< kj* =0 *>*

(*∃ϵ|*0 *≤ ϵ ≤ kj* : (*∀ϵj|*0 *≤ ϵj ≤ ϵ* : *domBG*) *Λ NBG*) *Λ inv*

It is now assumed that the theorem is true for *kj −* 1 and will be proved for *kj Hk′* (*Post*)

*≡*

*H*0(*Post*) *∨* (*domain*(*B*) *Λ B Λ wp*(*S, Hk′—*1(*Post*)))

*≡<*inductive hypothesis*>*

*H*0(*Post*)*∨*(*domain*(*B*)*ΛBΛwp*(*S,* (*∃ϵ|*0 *≤ ϵ ≤ kj−*1: (*∀ϵj|*0 *≤ ϵj ≤ ϵ* : *domBG*)*Λ*

*NBG*) *Λ inv*))

*≡< S* is deterministic in *domBG* and *NBG*, it does not modify *ϵj, ϵ, kj*, it exists

*ϵj* that 0 *≤ ϵj ≤ ϵ* (because 0 *≤ ϵ*) and Lemas [2.1](#_bookmark1),[2.2](#_bookmark2),[2.3](#_bookmark3)*>*

*H*0(*Post*) *∨* (*domain*(*B*) *Λ B Λ* (*∃ϵ|*0 *≤ ϵ ≤ kj −* 1: (*∀ϵj|*0 *≤ ϵj ≤ ϵ* :

*wp*(*S, domBG*))*Λwp*(*S, NBG*))*Λwp*(*S, inv*))

*≡< wp*(*S, P* ) *⇒ support*(*S*) for any *P >*

*H*0(*Post*) *∨* (*domain*(*B*) *Λ B Λ* (*∃ϵ|*0 *≤ ϵ ≤ kj −* 1:

(*∀ϵj|*0 *≤ ϵj ≤ ϵ* : *wp*(*S, domBG*)) *Λ wp*(*S, NBG*)) *Λ wp*(*S, inv*) *Λ support*(*S*))

*≡<*Definition of *domBG* and *NBG >*

*H*0(*Post*) *∨* (*domain*(*B*) *Λ B Λ* (*∃ϵ|*0 *≤ ϵ ≤ kj −* 1:

(*∀ϵj|*0 *≤ ϵj ≤ ϵ* : *domBG*[*ϵj* := *ϵj*+1])*ΛNBG*[*ϵ* := *ϵ*+1])*Λwp*(*S, inv*)*Λsupport*(*S*))

*≡< wp*(*S, P* ) *⇒ support*(*S*) for any *P >*

*H*0(*Post*) *∨* (*domain*(*B*) *Λ B Λ* (*∃ϵ|*0 *≤ ϵ ≤ kj −* 1:

(*∀ϵj|*0 *≤ ϵj ≤ ϵ* : *domBG*[*ϵj* := *ϵj* + 1]) *Λ NBG*[*ϵ* := *ϵ* + 1]) *Λ wp*(*S, inv*))

*≡*

*H*0(*Post*) *∨* (*domain*(*B*) *Λ B Λ* (*∃ϵ|*0 *≤ ϵ ≤ kj −* 1:

(*∀ϵj|*1 *≤ ϵj ≤ ϵ* +1 : *domBG*) *Λ NBG*[*ϵ* := *ϵ* + 1]) *Λ wp*(*S, inv*))

*≡*

*H*0(*Post*) *∨* (*domain*(*B*) *Λ B Λ* (*∃ϵ|*1 *≤ ϵ ≤ kj* :

(*∀ϵj|*1 *≤ ϵj ≤ ϵ* : *domBG*) *Λ NBG*) *Λ wp*(*S, inv*))

*≡<*Definition of *inv >*

*H*0(*Post*) *∨* (*domain*(*B*) *Λ B Λ* (*∃ϵ|*1 *≤ ϵ ≤ kj* :

(*∀ϵj|*1 *≤ ϵj ≤ ϵ* : *domBG*) *Λ NBG*) *Λ inv*)

*≡<*Definition of *H*0(*Post*) *>*

(*domain*(*B*) *Λ ¬B Λ Post*) *∨* (*domain*(*B*) *Λ B Λ* (*∃ϵ|*1 *≤ ϵ ≤ kj* : (*∀ϵj|*1 *≤ ϵj ≤ ϵ* : *domBG*) *Λ NBG*) *Λ inv*)

*≡<*Definition of *inv >*

(*domain*(*B*) *Λ ¬B Λ inv*) *∨* (*domain*(*B*) *Λ B Λ* (*∃ϵ|*1 *≤ ϵ ≤ kj* : (*∀ϵj|*1 *≤ ϵj ≤ ϵ* : *domBG*) *Λ NBG*) *Λ inv*)

*≡<*Distributivity of *Λ* over *∨ >*

(*domain*(*B*) *Λ inv*) *Λ* (*¬B ∨* (*B Λ* (*∃ϵ|*1 *≤ ϵ ≤ kj* :

(*∀ϵj|*1 *≤ ϵj ≤ ϵ* : *domBG*) *Λ NBG*)))

*≡<*Absorption*>*

(*domain*(*B*) *Λ inv*) *Λ* (*¬B ∨* (*∃ϵ|*1 *≤ ϵ ≤ kj* : (*∀ϵj|*1 *≤ ϵj ≤ ϵ* : *domBG*) *Λ NBG*))

*≡<*Distributivity of *Λ* over *∨ >*

*inv Λ* ((*domain*(*B*) *Λ ¬B*) *∨* (*domain*(*B*) *Λ* (*∃ϵ|*1 *≤ ϵ ≤ kj* :

(*∀ϵj|*1 *≤ ϵj ≤ ϵ* : *domBG*) *Λ NBG*)))

*≡*

*inv Λ* ((*domain*(*B*) *Λ ¬B*) *∨* (*∃ϵ|*1 *≤ ϵ ≤ kj* : *domain*(*B*)*Λ*

(*∀ϵj|*1 *≤ ϵj ≤ ϵ* : *domBG*) *Λ NBG*))

*≡<*Definition of *domBG >*

*inv Λ* ((*domBG*[*ϵj* := 0] *Λ ¬B*) *∨* (*∃ϵ|*1 *≤ ϵ ≤ kj* : *domBG*[*ϵj* := 0]*Λ*

(*∀ϵj|*1 *≤ ϵj ≤ ϵ* : *domBG*) *Λ NBG*))

*≡*

((*domBG*[*ϵj* := 0]*Λ¬B*)*∨*(*∃ϵ|*1 *≤ ϵ ≤ kj* : (*∀ϵj|*0 *≤ ϵj ≤ ϵ* : *domBG*)*ΛNBG*))*Λinv*

*≡< domBG*[*ϵj* := 0] *≡* (*∀ϵj|*0 *≤ ϵj ≤* 0: *domBG*) and definition of *NBG >*

(*∃ϵ|*0 *≤ ϵ ≤ kj* : (*∀ϵj|*0 *≤ ϵj ≤ ϵ* : *domBG*) *Λ NBG*) *Λ inv*

On the other hand, to demonstrate (ii) of the theorem, it will be shown that *Hk*+1(*Post*) is equivalent to a formula of form *p ∨ Hk*(*Post*) with *p* of form *wp*(*S, inv*) *Λq*, since in this way we have *Hk*+1(*Post*) *≡ Hk*(*Post*) iff *p ⇒ Hk*(*Post*). But how *Hk*(*Post*) *≡ Tk Λ inv ≡ Tk Λ wp*(*S, inv*), then *p ⇒ Hk*(*Post*) iff *p ⇒ Tk*.

It is assumed that 0 *< kj ≤ k* + 1 then the same first 8 steps are made

like before but instead of inductive hypothesis in step 2 it applies directly *Hk′—*1(*Post*) *≡ Tk′—*1 *Λ inv*, since in (i) it was shown that this is true for 0 *< kj ≤ k* + 1. With this one has to *Hk′* (*Post*) is equivalent to:

*H*0(*Post*)*∨*(*domain*(*B*)*ΛBΛ*(*∃ϵ|*1 *≤ ϵ ≤ kj* : (*∀ϵj|*1 *≤ ϵj ≤ ϵ* : *domBG*)*ΛNBG*)*Λ*

*wp*(*S, inv*))

Instantiating *kj* := *k* + 1, you have to *Hk*+1(*Post*) is equivalent to

*H*0(*Post*) *∨* (*domain*(*B*) *Λ B Λ wp*(*S, inv*)*Λ*

(*∃ϵ|*1 *≤ ϵ ≤ k* +1 : (*∀ϵj|*1 *≤ ϵj ≤ ϵ* : *domBG*) *Λ NBG*))

*≡*

(*domain*(*B*) *Λ B Λ wp*(*S, inv*) *Λ* (*∀ϵj|*1 *≤ ϵj ≤ k* +1 : *domBG*) *Λ NBG*[*ϵ* := *k* + 1])

*∨*

*H*0(*Post*) *∨* (*domain*(*B*) *Λ B Λ wp*(*S, inv*)*Λ*

(*∃ϵ|*1 *≤ ϵ ≤ k* : (*∀ϵj|*1 *≤ ϵj ≤ ϵ* : *domBG*) *Λ NBG*))

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 *Hk*(*Post*), therefore the previous formula It is equivalent to.

(*domain*(*B*) *Λ B Λ wp*(*S, inv*) *Λ* (*∀ϵj|*1 *≤ ϵj ≤ k* +1 : *domBG*) *Λ NBG*[*ϵ* := *k* + 1])

*∨*

*Hk*(*Post*) *2*

**Corollary 3.3** *If a predicate inv satisﬁes the hypotheses of (i) of Theorem* [*3.1*](#_bookmark8) *for all k and does not satisﬁes formula* (*∗*) *of (ii) of the same theorem for any k, then deﬁning T∞ as*

(*∃ϵ|*0 *≤ ϵ* : (*∀ϵj|*0 *≤ ϵj ≤ ϵ* : *domBG*) *Λ NBG*)

*is fulﬁlled that*

*wp*(*Do, Post*) *≡ T∞ Λ inv*

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

*wp*(*Do, Post*) *≡ Hk*(*Post*) *≡ Tk Λ inv*

**Proof.** Immediate consequence of Theorem [3.1](#_bookmark8) and definition *wp*(*Do, Post*) *≡*

(*∃kj|kj ≥* 0: *Hk′* (*Post*)) *2*

**Corollary 3.4** *Let Do be a loop as in Theorem* [*3.1*](#_bookmark8) *with guard i /*= *N Λ C (with short-circuited Λ) 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*](#_bookmark8) *but substituting B for C then:*

*TIk′ ≡ i /*= *N Λ CΛ*

((*N − kj ≤ i < N Λ* (*∀ϵj|*0 *≤ ϵj < N − i* : *domCG*))*∨*

(*∃ϵ|*1 *≤ ϵ ≤ kj* : (*∀ϵj|*0 *≤ ϵj ≤ ϵ* : *domCG*) *Λ NCG*))

*and*

*Tk′ ≡* (*N − kj ≤ i ≤ N Λ* (*∀ϵj|*0 *≤ ϵj < N − i* : *domCG*))*∨*

(*∃ϵ|*0 *≤ ϵ ≤ kj* : (*∀ϵj|*0 *≤ ϵj ≤ ϵ* : *domCG*) *Λ NCG*)

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

1. *If C ≡ true, then*

*Tk′ ≡ N − kj ≤ i ≤ N and TIk′ ≡ N − kj ≤ i < N*

1. *If domain*(*C*) *≡ a ≤ i < N or domain*(*C*) *≡ a ≤ i ≤ N, then*

*TN—a ≡ a ≤ i ≤ N and T IN—a ≡ a ≤ i < N Λ C and wp*(*Do, Post*) *≡*

*HN—a*(*Post*)

# Examples of Algorithm Correctness Using *Hk*(*Post*)

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

do *i /*= *N Λ A*[*i*] */*=0 *→*

*i* := *i* +1

od

*{Post* : (*∀k|*0 *≤ k < i* : *A*[*k*] */*= 0)*}*

It is fulfilled that *domain*(*A*[*i*] */*= 0) *≡* 0 *≤ i < N* and if we take as hypothesis

*TIN* , that in this case by Corollary [3.4](#_bookmark10) is 0 *≤ i < N Λ A*[*i*] */*= 0, then:

*wp*(*i* := *i* + 1*,* (*∀k|*0 *≤ k < i* : *A*[*k*] */*= 0))

*≡*

(*∀k|*0 *≤ k < i* +1 : *A*[*k*] */*= 0)

*≡*

(*∀k|*0 *≤ k < i* : *A*[*k*] */*= 0) *Λ A*[*i*] */*=0

*≡< A*[*i*] */*=0 *≡ true* by hypothesis*>* (*∀k|*0 *≤ k < i* : *A*[*k*] */*= 0)

Taking *inv* as (*∀k|*0 *≤ k < i* : *A*[*k*] */*= 0), the iteration test obligation is fulfilled and as *inv ≡ Post*, then the termination obligation test is trivially met, concluding that

*HN* (*Post*) *≡* 0 *≤ i ≤ N Λ* (*∀k|*0 *≤ k < i* : *A*[*k*] */*= 0)

that according to (ii) of Corollary [3.4](#_bookmark10), 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](#_bookmark10) suggests a justification for the classical tech- nique of derivation of invariants, called “replacement of constants by variable”. This technique consists of substituting a constant *N* of the postcondition for a fresh vari- able *i* and using this new predicate as invariant of a loop *Do* with guard *i /*= *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 /*= *N →*

*S*0;

*i* := *i* +1

od

*{Post* : *Sorted*(*A, N* )*}*

Where the invariant is obtained by calculating *Hk*(*Post*) *≡ N −k ≤ i ≤ N Λinv* according to (i) of Corollary [3.4](#_bookmark10) and Theorem [3.1](#_bookmark8). To get *inv* we take the postcon- dition by substituting the constant *N* for *i*, this new predicate *inv ≡ Sorted*(*A, i*) satisfies that *inv*[*i* := *N* ] *≡ Post* and therefore the obligation test of termination.

According to Theorem [3.1](#_bookmark8) instruction *S*0; *i* := *i* + 1 must satisfy that if *TIk* is true, then

*wp*(*S*0; *i* := *i* + 1*, inv*) *≡ wp*(*S*0*, Sorted*(*A, i* + 1)) *≡ inv*

so instruction *S*0 (using specification statements of [[33](#_bookmark37)]) must be

[*TIk Λ 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.

# 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,..., wp*(*S, Post*)*,...* )) a number *ϵ* of times.

**Lemma 5.1** *Let S*0 *be a statement that does not modify the value of variables i and if . Let k be an expression and let ϵ be a variable not declared in the program. Let PG be a predicate such that PG ≡ wp*(*S*0; *i* := *i* + 1*,PG*[*ϵ* := *ϵ −* 1]) *when* 0 *< ϵ ≤ k Λ domain*(*B*) *Λ B Λ support*(*S*0; *i* := *i* + 1)*, then assuming that if − kj ≤ i < if ,*

*you have that for* 0 *< kj ≤ k:*

*TIk′ Λ support*(*S*0) *⇒ wp*(*S*0; *i* := *i* + 1*,PG*[*ϵ* := *if − i*]) *≡ P G*[*ϵ* := *if − i*]

**Proof.** To make this demonstration, it is assumed to be true *if − kj ≤ i < if* and *TIk′* . But how *if − kj ≤ i < if ⇒ if − k ≤ i < if* and *TIk′ ⇒ domain*(*B*) *Λ B*, you can assume *if −k ≤ i < if* and *domain*(*B*)*ΛB* as well. It also *support*(*S*0; *i* := *i*+1) is assumed since by Lema [2.5](#_bookmark4) *support*(*S*0) *≡ support*(*S*0; *i* := *i* + 1).

*wp*(*S*0; *i* := *i* + 1*,PG*[*ϵ* := *if − i*])

*≡*

*wp*(*S*0; *i* := *i* + 1*,* (*∀ϵ|ϵ* = *if − i* : *P G*))

*≡*

*wp*(*S*0*,* (*∀ϵ|ϵ* = *if −* (*i* + 1) : *P G*[*i* := *i* + 1]))

*≡< S*0 does not modify *ϵ, if , i*, there is *ϵ* such that *ϵ* = *if −* (*i* + 1) and Lema [2.3](#_bookmark3)*>*

(*∀ϵ|ϵ* = *if −* (*i* + 1) : *wp*(*S*0*,PG*[*i* := *i* + 1]))

*≡*

(*∀ϵ|ϵ* = *if −* (*i* + 1) : *wp*(*S*0; *i* := *i* + 1*,PG*))

*≡<* Hypothesis *if − k ≤ i < if* implies 0 *≤ if −* (*i* + 1) *< k >*

(*∀ϵ|ϵ* = *if −* (*i* + 1) *Λ* 0 *≤ if −* (*i* + 1) *< k* : *wp*(*S*0; *i* := *i* + 1*,PG*))

*≡*

(*∀ϵ|ϵ* = *if −* (*i* + 1) *Λ* 0 *≤ ϵ < k* : *wp*(*S*0; *i* := *i* + 1*,PG*))

*≡<*Hypotheses are satisfied to apply definition of *PG >*

(*∀ϵ|ϵ* = *if −* (*i* + 1) *Λ* 0 *≤ ϵ < k* : *P G*[*ϵ* := *ϵ* + 1])

*≡<* 0 *≤ ϵ < k* is redundant*>*

(*∀ϵ|ϵ* = *if −* (*i* + 1) : *P G*[*ϵ* := *ϵ* + 1])

*≡*

*P G*[*ϵ* := *ϵ* + 1][*ϵ* := *if −* (*i* + 1)]

*≡*

*P G*[*ϵ* := *if − i*] *2*

**Lemma 5.2** *Let R be a predicate and S a statement that behaves deterministically on the values of the variables of R. If if 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 if |R* : *if* )) *≡*

*support*(*S*) *Λ Exp* = (*min if |wp*(*S, R*): *if* )

**Proof.** Analogous to Lema 17 of [[6]](#_bookmark20) *2*

**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*](#_bookmark8)*, taking S as S*0; *i* := *i*+1*. Let k be an expression*

*and let ϵ, if and kj be variables not declared in the program, then abbreviating m as*

(*min if |i ≤ if ≤ i* + *kj Λ NBG*[*ϵ* := *if − i*]: *if* )*,*

*you have that for* 0 *< kj ≤ k:*

*TIk′ ⇒ wp*(*S*0; *i* := *i* + 1*,ϵ* = *m − i*) *≡ support*(*S*0) *Λ ϵ* = *m − i −* 1

**Proof.** Assuming *TIk′* you have to:

It is true *¬NBG*[*ϵ* := *i − i*], and therefore, to consider that *if* can be equal to *i*

in the calculation of *m*, it is impossible, in this way:

*m*

=

(*min if |i ≤ if ≤ i* + *kj Λ NBG*[*ϵ* := *if − i*]: *if* )

=

(*min if |i < if ≤ i* + *kj Λ NBG*[*ϵ* := *if − i*]: *if* ) (*∗∗*) With this it can be deduced that:

*wp*(*S*0; *i* := *i* + 1*,ϵ* = *m − i*)

*≡*

*wp*(*S*0*,ϵ* = *m*[*i* := *i* + 1] *− i −* 1)

*≡*

*wp*(*S*0*,ϵ* + *i* +1 = (*min if |i < if ≤ i* + *kj* +1 *Λ NBG*[*ϵ* := *if − i*][*i* := *i* + 1] : *if* ))

*≡<*Lemas [5.2](#_bookmark12) and [2.6](#_bookmark5)*>*

*support*(*S*0) *Λ ϵ* + *i* +1 = (*min if |i < if ≤ i* + *kj* + 1*Λ*

*wp*(*S*0*,NBG*[*ϵ* := *if − i*][*i* := *i* + 1]) : *if* )

*≡*

*support*(*S*0) *Λ ϵ* + *i* +1 = (*min if |i < if ≤ i* + *kj* + 1*Λ*

*wp*(*S*0; *i* := *i* + 1*,NBG*[*ϵ* := *if − i*]) : *if* )

*≡<*Abbreviating (*min if |if* = *i* + *kj* +1 *Λ wp*(*S*0; *i* := *i* + 1*,*

*NBG*[*ϵ* := *if − i*]) : *if* ) as *m*1 *> support*(*S*0) *Λ ϵ* = *min*(*m*1*,* (*min if |i < if ≤ i* + *kjΛ*

*wp*(*S*0; *i* := *i* + 1*,N BG*[*ϵ* := *if − i*]) : *if* )) *− i −* 1

*≡<*Lema [5.1](#_bookmark11)*>*

*support*(*S*0) *Λϵ* = *min*(*m*1*,* (*min if |i < if ≤ i*+ *kj ΛNBG*[*ϵ* := *if −i*]: *if* )) *−i−*1

*≡<*observation (*∗∗*) *>*

*support*(*S*0) *Λ ϵ* = *min*(*m*1*, m*) *− i −* 1

*≡< TIk′* implies that exists *m ≤ i* + *kj < m*1 *>*

*support*(*S*0) *Λ ϵ* = *m − i −* 1 *2*

**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, if and kj variables not declared in the program, which does not occur in Post and* 0 *< kj ≤ k, deﬁning the predicate NBG as in Theorem* [*3.1*](#_bookmark8) *and a predicate PostG that satisﬁes the following recursive equations:*

* *PostG*[*ϵ* := 0] *≡ Post*
* *PostG ≡ wp*(*S*0; *i* := *i* + 1*, PostG*[*ϵ* := *ϵ −* 1]) *when* 0 *< ϵ ≤ k and T Ik*

*then abbreviating m as*

(*min if |i ≤ if ≤ i* + *kj Λ NBG*[*ϵ* := *if − i*]: *if* )*,*

*you have to:*

1. *The predicate PostG*[*ϵ* := *m − i*] *is a predicate, which satisﬁes the hypotheses of the predicate inv in (i) of Theorem* [*3.1*](#_bookmark8)*.*
2. *Additionally if the recursion that deﬁnes PostG, domBG and NBG are deﬁned up to k*+1*, then T Ik*+1*ΛPostG*[*ϵ* := *k*+1] *⇒ T Ik is equivalent to Hk*+1(*Post*) *≡ Hk*(*Post*)*.*

**Proof.** *inv* is defined as *PostG*[*ϵ* := *m − i*] and it will be shown that *inv* complies with the equations of the Theorem [3.1](#_bookmark8)

*domain*(*B*) *Λ ¬B Λ Post*

*≡*

*domain*(*B*) *Λ NBG*[*ϵ* := 0] *Λ PostG*[*ϵ* := 0]

*≡*

*domain*(*B*) *Λ NBG*[*ϵ* := *i − i*] *Λ PostG*[*ϵ* := *i − i*]

*≡<*Since *NBG*[*ϵ* := *i − i*] *≡ ¬B ≡ true* then *m* = *i domain*(*B*) *Λ NBG*[*ϵ* := *i − i*] *Λ PostG*[*ϵ* := *m − i*]

*≡*

*domain*(*B*) *Λ ¬B Λ inv*

On the other hand assuming *TIk′* you have to

*wp*(*S*0; *i* := *i* + 1*, inv*)

*≡*

*wp*(*S*0; *i* := *i* + 1*, PostG*[*ϵ* := *m − i*])

*≡*

*wp*(*S*0; *i* := *i* + 1*,* (*∀ϵ|* : *ϵ* = *m − i ⇒ PostG*))

*≡<*Lema [2.8](#_bookmark7)*>*

(*∀ϵ|* : *wp*(*S*0; *i* := *i* + 1*,ϵ* = *m − i ⇒ PostG*))

*≡< S*0; *i* := *i* + 1 acts deterministically on the variables

of NBG and the variables *if , i, kj,ϵ* and Lemas [2.7](#_bookmark6) and [2.5](#_bookmark4)*>*

(*∀ϵ|* : *support*(*S*0) *Λ* (*wp*(*S*0; *i* := *i* + 1*,ϵ* = *m − i*) *⇒ wp*(*S*0; *i* := *i* + 1*, PostG*)))

*≡<*Lema [5.3](#_bookmark13)*>*

(*∀ϵ|* : *support*(*S*0) *Λ* (*ϵ* = *m − i −* 1 *⇒ wp*(*S*0; *i* := *i* + 1*, PostG*)))

*≡*

*support*(*S*0) *Λ* (*∀ϵ|* : *ϵ* = *m − i −* 1 *⇒ wp*(*S*0; *i* := *i* + 1*, PostG*))

*≡*

*support*(*S*0) *Λ* (*∀ϵ|ϵ* = *m − i −* 1: *wp*(*S*0; *i* := *i* + 1*, PostG*))

*≡< TIk′* implies that exists *m* and therefore exists *ϵ* such that *ϵ* = *m − i −* 1 *>*

(*∀ϵ|ϵ* = *m − i −* 1: *support*(*S*0) *Λ wp*(*S*0; *i* := *i* + 1*, PostG*))

*≡< wp*(*S, P* ) *⇒ support*(*S*) for any *P* and Lema [2.5](#_bookmark4)*>* (*∀ϵ|ϵ* = *m − i −* 1: *wp*(*S*0; *i* := *i* + 1*, PostG*))

*≡< TIk′ ⇒* 0 *≤ m − i −* 1 *< kj* and def of *PostG >*

(*∀ϵ|ϵ* = *m − i −* 1: *PostG*[*ϵ* := *ϵ* + 1])

*≡*

*PostG*[*ϵ* := *ϵ* + 1][*ϵ* := *m − i −* 1]

*≡*

*PostG*[*ϵ* := *m − i*]

*≡*

*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 *T Ik*+1 *⇒* (*wp*(*S, inv*) *≡ inv*), then demonstrating by cases we have to that if *Tk ≡ false* then

*B Λ wp*(*S, inv*) *Λ* (*∀ϵj|*0 *≤ ϵj ≤ k* +1 : *domBG*) *Λ NBG*[*ϵ* := *k* + 1] *⇒ Tk*

*≡< Tk ≡ false >*

*B Λ wp*(*S, inv*) *Λ* (*Tk ∨* ((*∀ϵj|*0 *≤ ϵj ≤ k* +1 : *domBG*) *Λ NBG*[*ϵ* := *k* + 1])) *⇒ Tk*

*≡*

*B Λ wp*(*S, inv*) *Λ Tk*+1 *⇒ Tk*

*≡< B Λ Tk*+1 *≡ TIk*+1 and *TIk*+1 *⇒* (*wp*(*S, inv*) *≡ inv*) *> T Ik*+1 *Λ inv ⇒ Tk*

*≡*

*TIk*+1 *Λ PostG*[*ϵ* := *m − i*] *⇒ Tk*

Since *Tk* is false then *m − i > k*, but in conjunction with *TIk*+1 which implies *NBG*[*ϵ* := *k* + 1], It is fulfilled *m − i* = *k* + 1 and therefore the previous formula is equivalent to

*TIk*+1 *Λ PostG*[*ϵ* := *k* + 1] *⇒ Tk*

For the case in which *Tk* is true, trivially the last and first implication are equivalent, with which by (ii) of Theorem [3.1](#_bookmark8), we have the proof. *2*

# Examples of Calculation of Invariants Assertions

Theorem [5.4](#_bookmark14) suggests a method to calculate *Hk*(*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](#_bookmark14) is shown below:

* 1. *Fibonacci*

do *i /*= *N →*

*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))

*≡*

*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))

*≡*

*z* + (*x* + *z*)= *fib*(*N* + 1)

*≡*

*x* + 2*z* = *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* + 2*z* = *fib*(*N* + 1))

*≡*

*z* + 2(*x* + *z*)= *fib*(*N* + 1)

*≡*

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

If to the previous result it is apply again *wp*, it is obtained 3*x* + 5*z* = *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*(*ϵ*)*x* + *fib*(*ϵ* + 1)*z* = *fib*(*N* + 1)

We will call this predicate *PostG*, which for *ϵ* = *N* is satisfiable (taking *x, z* := 0*,* 1), so *PostG* is not false when *ϵ* = *N* , for this reason *PostG* is not false for any *ϵ ≤ N* . This is because if *PostG ≡ false* for some *ϵ* = *e < N* , then *PostG* would be false for all *ϵ > e* (including *ϵ* = *N* ), since *wp*(*S, false*) *≡ false* and *PostG* was obtained from applying *wp* successively. Therefore, you can not use (ii) of the Theorem [5.4](#_bookmark14) for any *k < N* .

Therefore, predicate *PostG* satisfies the recurrence of Theorem [5.4](#_bookmark14) by taking *k* as *N* . Since *m* = *N* , it is concluded that *PostG*[*N − i*] satisfies the hypotheses of (i) of Theorem [3.1](#_bookmark8) and by the previous paragraph, does not comply with (ii) of Theorem [3.1](#_bookmark8). Therefore, it is obtained:

*HN* (*Post*)

*≡*

*TN Λ* (*fib*(*ϵ*)*x* + *fib*(*ϵ* + 1)*z* = *fib*(*N* + 1))[*ϵ* := *N − i*]

*≡*

0 *≤ i ≤ N Λ fib*(*N − i*)*x* + *fib*(*N − i* + 1)*z* = *fib*(*N* + 1) Which is a valid invariant assertion for the loop.

* 1. *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](#_bookmark14). do *i /*= *N Λ A*[*i −* 1] = *A*[*N − i*] *→*

*pal* := *A*[*i*]= *A*[*N −* 1 *− i*];

*i* := *i* +1

od

*{Post* : *pal ≡ palind*(*A,* 0*,N* )*}*

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

(*∀k|i ≤ 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*]) *≡* 1 *≤ i ≤ N* , then Corollary [3.4](#_bookmark10) says that

*T IN—*1 *≡* 1 *≤ i < NΛ A*[*i −* 1] = *A*[*N − i*] and *TN—*1 *≡* 1 *≤ i ≤ N*

Assuming *T IN—*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 ≡ palind*(*A,* 0*,N* ))

*≡*

0 *≤ i < N Λ* (*A*[*i*]= *A*[*N −* 1 *− i*] *≡ palind*(*A,* 0*,N* ))

*≡<* 0 *≤ i < N ≡ true* by hypothesis *T IN—*1 *> A*[*i*]= *A*[*N −* 1 *− i*] *≡ palind*(*A,* 0*,N* )

Assuming *T IN—*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*] *≡ palind*(*A,* 0*,N* ))

*≡*

0 *≤ i < N Λ* (*A*[*i* + 1] = *A*[*N −* 2 *− i*] *≡ palind*(*A,* 0*,N* ))

*≡<* 0 *≤ i < N ≡ true* by hypothesis *T IN—*1 *> A*[*i* + 1] = *A*[*N −* 2 *− i*] *≡ palind*(*A,* 0*,N* )

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

(*ϵ* = 0?*pal* : *A*[*i* + *ϵ −* 1] = *A*[*N − ϵ − i*]) *≡ 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* (*≤ N −* 1) number of times, the satisfiable formula *i* + *ϵ* = *N ∨ A*[*i* + *ϵ −* 1] */*= *A*[*N − i − ϵ*] is obtained, that when replacing *ϵ* := *if − i* it result

*if* = *N ∨ A*[*if −* 1] */*= *A*[*N − if* ]*.*

Thus, the weakest precondition of the loop is:

1 *≤ i ≤ N*

*Λ*

((*m* = *i*?*pal* : *A*[*m −* 1] = *A*[*N − m*]) *≡ palind*(*A,* 0*,N* )) where *m* is an abbreviation of

(*min if |i ≤ if ≤ i* + *N −* 1 *Λ* (*if* = *N ∨ A*[*if −* 1] */*= *A*[*N − if* ]) : *if* )

# Conclusions

The theorems presented here are a small contribution to the development of a prag- matic 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 *Hk*(*Post*) of certain instructions *Do*, in a fast and formal way.

Since invariant relationships and invariant assertions are related [[12](#_bookmark25)], for future research it is proposed to extract from the implementation of [[30](#_bookmark34)], the aspects that allow the implementation of the invariant calculation technique described in this paper. This implementation will allow not only calculating the invariants *Hk*(*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 *Hk*(*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 Teor´ıa Axiom´atica del Lenguaje GCL de Dijkstra*. Proc. CoNCISa 2015, Valencia, Venezuela, Noviembre 2015, pp. 153-164.
5. F. Flaviani. *Calculo de Precondiciones M´as D´ebiles*. ReVeCom, Diciembre 2016, Vol. 3, No. 2, pp. 68-80.
6. F. Flaviani. *Propiedades Algebraicas y Decidibilidad del Transformador de Predicados wp sobre la Teor´ıa 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.