Calculation of Invariants Assertions

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

***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.**

***Keywords*—Invariant Assertions; Calculus; Formal Program Verification; GCL; Induction.**

I. Introducción

Todos los algoritmos planteados en este trabajo serán escritos en pseudolenguaje GCL (Guarded Command Language) [1], que es un pseudolenguaje definido por Dijkstra, que admite la escritura de algoritmos no determinísticos y su diseño, admite una lógica de Hoare y fórmulas para precondiciones más débiles, relativamente simples, que facilitan la actividad de corrección de un programa. Todas las aserciones en este trabajo se asumirán que son escritas en el lenguaje de las aserciones de [2].

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

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

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

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

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

$$H_k(Post) \equiv$$

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

para $k \geq 1$

A. Contribución

En la lógica de Hoare [3] para hacer la corrección parcial de un ciclo Do con postcondición $Post$ se debe aplicar la regla del invariante:

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

en combinación con la regla

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

Adicionalmente para demostrar terminación y hacer una corrección total, se debe usar una función de cota $f(\vec{x})$ para el ciclo (donde $\vec{x}$ es un estado del programa) y demostrar las siguientes dos obligaciones de prueba.

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

y

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

En total se necesitan demostrar cinco teoremas u obligaciones de pruebas, para verificar que el invariante Inv propuesto es correcto, lo cual es un trabajo grande incluso para ciclos muy simples, y por otro lado las reglas no explican como se construye dicho predicado Inv.

Por otro lado se sabe que los predicados $H_k(Post)$ de la definición de wp de Dijkstra para ciclos Do son invariantes correctos, que por definición tienen asociados una condición

978-1-5386-3057-0/17/$31.00

de terminación (condición que de satisfacerla un estado al inicio de las iteraciones, hace que el ciclo no pueda iterar más de k veces). Aprender a calcular $H_k(Post)$ representa una alternativa para conseguir invariantes, sin necesidad de usar las reglas de inferencia de Hoare y sin necesidad de encontrar funciones de cota, igualmente la misma observación es válida para la precondición más débil del ciclo, dado una postcondición $Post$. En este trabajo se evidencia que para la corrección de ciclos, es mucho más simple calcular o identificar a $H_k(Post)$ o la precondición más débil, en lugar de conjeturar un invariante aplicando las reglas de Hoare.

Concretamente si tenemos un algoritmo de la forma Do y una postcondición $Post$, entonces en el desarrollo del trabajo se responden las siguientes preguntas:

1) Dada una aserción Inv, ¿Es Inv el predicado $H_k(Post)$ del ciclo para algún k?
2) Dada una aserción Inv, ¿Es Inv la precondición más débil del ciclo con postcondición $Post$?
3) ¿Cómo calculo el predicado $H_k(Post)$ del ciclo Do para algún k?
4) ¿Cómo calculo la precondición más débil del ciclo Do y postcondición $Post$?
5) ¿Cómo calculo la condición de terminación de un $H_k(Post)$?

Las preguntas 1 y 2 se responden en general mediante el uso del teorema 1 y el Corolario 1 para ciclos con un cuerpo potencialmente no determinístico, pero que se comporta determinísticamente sobre las variables que ocurren en la guardia del ciclo. Las preguntas 3 y 4 se responden usando una técnica basada en inducción matemática, pero sólo para ciclos de la forma

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

o

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

donde S_0 no modifica la variable i y no modifica las variables de la expresión C (generalmente llamada centinela) y el operador $\wedge$ es con cortocircuito. Para la pregunta 5 se establece un criterio general en el teorema 1 para construir un predicado T_k que corresponde a la condición de terminación de $H_k(Post)$ para el Do, por otro lado se establece un criterio para los dos tipos de ciclos anteriores que determina las condiciones suficientes para que la condición de terminación sea el predicado $a \leq i \leq N$.

B. Trabajos Relacionados

Originalmente en [1] la definición recursiva de wp que se expuso al inicio no incluía la función sintáctica $domain$ en sus reglas, esto fue corregido en [2], donde lo incorpora a la regla de wp de la asignación, pero no en las demás reglas como se definió al inicio de la introducción. Una justificación de la incorporación de $domain$ en las reglas de wp del IF y Do, se encuentra en [4], donde se hace una revisión de la semántica denotacional de GCL incluyendo el estado $abort$. La función sintáctica $domain$, aplica sobre expresiones, pero su incorporación en las reglas de construcción de wp del IF y Do, traen dificultades adicionales que no se tenían en [2], para manejar estas dificultades, en [5] se definió la función sintáctica $support$, que viene siendo el análogo a $domain$, pero aplica sobre instrucciones en lugar de expresiones, en este trabajo se muestran algunas propiedades nuevas de $support$.

Por otro lado en [6]-[9] también responden a las preguntas 4 y 5, usando un método de tipo semántico llamado "cálculo de relaciones invariantes" [10], que básicamente consiste en obtener la relación que resulta de la interpretación bajo semántica denotacional, del cuerpo del ciclo Do y calcular la clausura reflexiva-transitiva de dicha relación. Esta técnica presupone que el lenguaje para las aserciones debe ser el de la teoría de conjuntos. Este trabajo se crea como continuación de [5] y es la contraparte sintáctica usando wp de [10][7], básicamente la técnica que aquí se usa consiste en calcular una fórmula general que exprese el resultado de aplicar sucesivamente wp al cuerpo del Do, esto tiene su contraparte semántica en [10][7] donde se compone sucesivamente la relación resultante de la interpretación del cuerpo del ciclo. El enfoque aquí presentado muestra que se puede responder fácilmente a las preguntas 1,2,3,4 y 5 sin necesidad de ir al mundo semántico de [10][7], en el lenguaje clásico de aserciones del libro [2] y usando GCL con todo el poder de expresión del no determinismo. Para entender la relación que existe entre la técnica semántica de relaciones invariantes y la técnica sintáctica de la aserción invariante se puede revisar [11].

En el área de derivación automática de invariantes ha habido un interés reciente en los últimos años [12]-[21], adicionalmente existen aplicaciones como [22][23] que pueden calcular invariantes para ciclos donde las expresiones de las asignaciones del cuerpo del ciclo son todas lineales o traducibles a sistemas de transición lineales, de igual forma en [24] se encuentra otra técnica que es aplicable sólo a ciclos donde el cuerpo es traducible a una transformación afín de espacios vectoriales. Aplicaciones basadas en lógica de Hoare y separación tenemos a [25]-[27] y basadas en wp se encuentra [28], sólo que funciona para programas no estructurados.

El desarrollo de las técnicas aquí presentadas, tiene como objetivo a largo plazo, construir un cálculo de invariantes lo suficientemente eficiente y limpio como para que pueda ser implementado al igual como las aplicaciones Mapple y Mathematica implementaron el cálculo integral. Una aplicación existente que puede calcular precondiciones más débiles, en base a cómputos simbólicos, pidiendo menos condiciones que las del párrafo anterior, se describe en [29], que es una aplicación basada en relaciones invariantes, hecha con los paquetes de clausuras reflexiva-transitivas de Mathematica (Wolfram Research). Por ser este trabajo la contraparte sintáctica de [10][7], se espera posible implementar una aplicación de cálculo de $H_k(Post)$ en base a los teoremas de este trabajo y [5], similar a [29].

C. Estructura del artículo

A continuación se presentan cinco secciones de las cuales, en la primera de ellas se estudia el comportamiento del transformador de predicados wp sobre los operadores $\wedge$ y $\vee$ y su relación con la función sintáctica $support$. Luego se caracterizan las fórmulas de precondición más débil y $H_k(Post)$ en base a un predicado que llamaremos inv. En la sección siguiente, se dan ejemplos del uso de los teoremas para

responder las preguntas 1,2 y 5 definidas en la introducción. Seguidamente se demuestran teoremas que establecen una relación entre el predicado inv y el cálculo sucesivo de wp. De último se muestran ejemplos del uso de los teoremas para responder a las preguntas 3 y 4 de la introducción.

II. No determinismo y propiedades de wp y $support$

Para demostrar los teoremas de las próximas secciones, se usaran las siguientes propiedades del transformador de predicados wp, que se demuestran por inducción estructural sobre el tamaño de la instrucción.

Lema 1. $wp(S, P \wedge Q) \equiv wp(S, P) \wedge wp(S, Q)$

Lema 2. *Sea S una instrucción (determinística o no) tal que se comporta de forma determinística sobre las variables del predicado $P \vee Q$, entonces*

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

Lema 3. *Sea S una instrucción (determinística o no) tal que se comporta de forma determinística sobre las variables del predicado P y no modifica los valores de las variables ϵ y k, entonces para $k \geq 0$ se tiene que*

$$wp(S, (\exists \epsilon | 0 \leq \epsilon < k : P)) \equiv (\exists \epsilon | 0 \leq \epsilon < k : wp(S, P))$$

Demostración: Primero se observa que como S no modifica los valores de las variables ϵ y k, entonces las instrucciones $S; \epsilon := k - 1$ y $\epsilon := k - 1; S$ son denotacionamente iguales, por lo tanto

$wp(S, P[\epsilon := k - 1]) \equiv wp(S; \epsilon := k - 1, P) \equiv$

$wp(\epsilon := k - 1; S, P) \equiv wp(\epsilon := k - 1, wp(S, P)) \equiv$

$wp(S, P)[\epsilon := k - 1]$

La demostración sigue por inducción natural sobre k. ■

Lema 4. *Sea S una instrucción que no modifica los valores de las variables ϵ y k, entonces para $k \geq 0$ se tiene que*

$$wp(S, (\forall \epsilon | 0 \leq \epsilon \leq k : P)) \equiv (\forall \epsilon | 0 \leq \epsilon \leq k : wp(S, P))$$

Demostración: Demostración análoga a la del Lema 3. ■

Lema 5. *Sea P un predicado y S una instrucción que no modifica los valores de las variables de P, entonces*

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

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

Por ejemplo $true$ es un predicado que para cualquier S, se tiene que S no modifica sus variables, por lo que una forma de calcular $support(S)$ es calculando $wp(S, true) \equiv support(S) \wedge true \equiv support(S)$. Por ejemplo si S es la instrucción $if\ a > -3 \to b := b/a\ []\ a \leq -3 \to b := 2\ fi$, entonces

$wp(S, true)$

$\equiv$

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

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

$\equiv$

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

Con lo que $support(S) \equiv a > -3 \Rightarrow a \neq 0$.

Lema 6. *Sea S una instrucción, entonces*

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

Demostración:

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

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

Lema 7. *Sean P y Q predicados y S una instrucción que se comporta de forma determinística sobre los valores de las variables de P, entonces*

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

por otro lado, si S no modifica los valores de las variables de P, entonces

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

y

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

Lema 8. *Sea P un predicado y S una instrucción que se comporta determinísticamente sobre los valores de las variables de P, entonces*

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

Lema 9. *Sea S una instrucción, P un predicado y ϵ una variable no declarada en el programa, entonces*

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

III. Precondición más débil y $H_k(Post)$ de la instrucción Do

Para poder calcular la precondición más débil o $H_k(Post)$ de un ciclo, se presenta el siguiente Teorema y Corolarios.

Teorema 1. *Sea k una expresión y Do una instrucción de la forma*

do $B \to$

S

od

$\{Post\}$

y k', ϵ, ϵ' variables no declaradas en el programa (es decir no ocurren en Do) y no ocurren en $Post$, S es una instrucción (determinística o no determinística).

Se define el predicado $domBG$ recursivamente tal que:

- *En $domBG$ ocurren sólo ϵ' y las variables del programa*
- $domBG[\epsilon' := 0] \equiv domain(B)$
- $domBG \equiv wp(S, domBG[\epsilon' := \epsilon' - 1])$ *cuando* $0 < \epsilon' \leq k \wedge domain(B) \wedge B \wedge support(S)$

Se define el predicado NBG recursivamente tal que:

- *En NBG ocurren sólo ϵ y las variables del programa*
- $NBG[\epsilon := 0] \equiv \neg B$
- $NBG \equiv wp(S, NBG[\epsilon := \epsilon - 1])$ *cuando* $0 < \epsilon \leq k \wedge domain(B) \wedge B \wedge support(S)$

Se define el predicado $TI_{k'}$ como:

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

Se define el predicado $T_{k'}$ como:

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

Entonces, si S actúa de forma determinística sobre las variables de $domBG$ y NBG, se tiene que:

1) *Si $domBG$ y NBG son satisfacibles en las condiciones de la recurrencia para todo $0 \leq \epsilon, \epsilon' \leq k$, y existe un predicado inv tal que:*
   - $domain(B) \wedge \neg B \wedge Post \equiv domain(B) \wedge \neg B \wedge inv$
   - $TI_{k'} \Rightarrow (wp(S, inv) \equiv inv)$

   entonces
   $$H_{k'}(Post) \equiv T_{k'} \wedge inv$$
   para todo k' tal que $0 \leq k' \leq k$.
2) *Adicionalmente si la recurrencia que definen a $domBG$ y NBG están definidas hasta $\epsilon, \epsilon' = k + 1$ y si $B \wedge wp(S, inv) \wedge (\forall \epsilon' | 0 \leq \epsilon' \leq k + 1 : domBG) \wedge NBG[\epsilon := k + 1] \equiv false$, entonces*
   $$H_{k+1}(Post) \equiv H_k(Post)$$

El predicado inv del teorema no debe confundirse con un invariante, éste más bien, es una subfórmula de un invariante Inv que es de la forma $T_{k'} \wedge inv$.

Note que decir que S actúa de forma determinística sobre las variables de $domBG$ y NBG, significa que en cada iteración, S actua de forma determinística sobre las variables de B.

Notación. *El predicado $T_{k'}$ se llamará "condición de terminación", ya que se puede demostrar que $T_{k'} \equiv H_{k'}(true)$, tomando inv como $true$ y verificando que todas las hipótesis del teorema anterior se cumplen. $H_{k'}(true)$ describe la condición más débil que hace que el ciclo itere a lo sumo k' veces sin solicitar ninguna condición en el estado final de la ejecución.*

A la fórmula del primer item del apartado 1 del teorema anterior se llamará "obligación de prueba de terminación" y a la fórmula del segundo item del apartado 1 del teorema anterior se llamará "obligación de prueba de iteración".

Note que el teorema anterior dice, que para demostrar que una aserción es un $H_{k'}(Post)$ de un ciclo dado, entonces basta con demostrar las dos obligaciones de prueba anteriores, lo cual es mucho más simple que demostrar las cinco obligaciones de prueba que define la lógica de Hoare para los invariantes. Seguidamente se demostrará el teorema 1.

Demostración: Por ser este un teorema sobre una fórmula cuyas instancias son fórmulas, entonces se usará un sistema de derivación formal de predicados para asegurar un resultado correcto. El lector debe entender la siguiente demostración como una familia de demostraciones (una por cada instancia del predicado inv), que resulta de aplicar cada una de las derivaciones siguientes en el orden que se presentan. Las reglas de inferencia que se usan en este trabajo son las de la lógica calculativa (original de [30]) presentada en el libro de Gries [31].

Se demostrará por inducción sobre k' suponiendo que $k' \leq k$ y que k' es una variable que no ocurren en inv, NBG, $domBG$, S y $Post$.

Caso 1 $k' = 0$

$H_{k'}(Post)$

$\equiv < k' = 0 >$

$H_0(Post)$

$\equiv$

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

$\equiv$

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

$\equiv$

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

$\equiv$

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

$\equiv$

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

$\equiv < k' = 0 >$

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

Se supone ahora que el teorema es cierto para $k' - 1$ y se demostrará para k'

$H_{k'}(Post)$

$\equiv$

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

$\equiv <$hipótesis inductiva$>$

$H_0(Post) \vee (domain(B) \wedge B \wedge wp(S,$

$(\exists \epsilon | 0 \leq \epsilon \leq k' - 1 : (\forall \epsilon' | 0 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG) \wedge$

$inv))$

$\equiv$

$H_0(Post) \vee (domain(B) \wedge B \wedge wp(S,$

$(\exists \epsilon | 0 \leq \epsilon \leq k' - 1 : (\forall \epsilon' | 0 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG)) \wedge$

$wp(S, inv))$

$\equiv < S$ es determinística en $domBG$ y NBG y Lema 3$>$

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

$wp(S, (\forall \epsilon' | 0 \leq \epsilon' \leq \epsilon : domBG) \wedge NBG)) \wedge wp(S, inv))$

$\equiv$

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

$\equiv$<Lema 4>

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

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

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

$\equiv$<Definición de $domBG$ y $NBG >$

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

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

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

$\equiv$

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

$\equiv$

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

$\equiv$<Definición de $inv >$

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

$\equiv$<Definición de $H_0(Post) >$

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

$\equiv$<Definición de $inv >$

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

$\equiv$<Distributividad del $\wedge$ sobre el $\vee >$

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

$\equiv$<Absorción>

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

$\equiv$<Distributividad del $\wedge$ sobre el $\vee >$

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

$\equiv$

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

$\equiv$<Definición de $domBG >$

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

$\equiv$

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

$\equiv$<Definición de $domBG$ y $NBG >$

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

$\equiv$

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

Por otro lado

Si asumimos las hipótesis del apartado 2) del teorema, suponiendo $0 < k' \leq k+1$ se tiene que después de los mismos primeros 11 pasos de la derivación anterior, $H_{k'}(Post)$ es equivalente a:

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

Instanciando $k' := k + 1$, se tiene que $H_{k+1}(Post)$ es equivalente a:

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

$\equiv$

$H_0(Post) \vee ((domain(B) \wedge B \wedge wp(S, inv) \wedge$

$(\forall \epsilon' | 1 \leq \epsilon' \leq k+1 : domBG) \wedge NBG[\epsilon := k+1])$

$\vee$

$(domain(B) \wedge B \wedge wp(S, inv) \wedge$

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

$\equiv$<Hipótesis>

$H_0(Post) \vee (false$

$\vee$

$(domain(B) \wedge B \wedge wp(S, inv) \wedge$

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

$\equiv$

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

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

La fórmula anterior es la misma que se obtuvo anteriormente justo antes de la equivalencia que se etiquetó con el comentario "Definición de inv", y ya se demostró que dicha fórmula es equivalente a $H_k(Post)$. ■

Corolario 1. *Si un predicado inv cumple con las hipótesis del apartado 1 del Teorema 1 para todo k y no cumple con la hipótesis del apartado 2 del mismo teorema, entonces definiendo T_∞ como*

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

se tiene que

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

por otro lado, si inv cumple con las hipótesis de los apartados 1 y 2, entonces

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

Demostración: Consecuencia inmediata del teorema 1 y la definición $wp(Do, Post) \equiv (\exists k' | k' \geq 0 : H_{k'}(Post))$ ■

Corolario 2. *Sea un ciclo Do como en el teorema 1 con guardia $i \neq N \wedge C$ ($\wedge$ con corto circuito) y cuerpo $S_0; i := i+1$ donde S_0 es una instrucción que no modifica ni i y ni N. Definiendo $domCG$ y NCG igual que los predicados $domBG$ y NBG del teorema 1 pero sustituyendo B por C entonces:*

$TI_{k'} \equiv i \neq N \wedge C \wedge$

$((N - k' \leq i < N \wedge (\forall \epsilon' | 0 \leq \epsilon' < N - i : domCG)) \vee$

$(\exists \epsilon | 1 \leq \epsilon \leq k' : (\forall \epsilon' | 0 \leq \epsilon' \leq \epsilon : domCG) \wedge NCG))$

y

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

$(\exists \epsilon | 0 \leq \epsilon \leq k' : (\forall \epsilon' | 0 \leq \epsilon' \leq \epsilon : domCG) \wedge NCG)$

Adicionalmente se cumple que si a es una constante se tiene:

1) *Si $C \equiv true$, entonces*
   $T_{k'} \equiv N - k' \leq i \leq N$ *y* $TI_{k'} \equiv N - k' \leq i < N$
2) *Si $domain(C) \equiv a \leq i < N$ o $domain(C) \equiv a \leq i \leq N$, entonces*
   $T_{N-a} \equiv a \leq i \leq N$ *y* $TI_{N-a} \equiv a \leq i < N \wedge C$ *y* $wp(Do, Post) \equiv H_{N-a}(Post)$

IV. Ejemplos de corrección de algoritmos usando $H_k(Post)$

A continuación a partir de la conjetura de inv se llevará a cabo la corrección del siguiente algoritmo

do $i \neq N \wedge A[i] \neq 0 \rightarrow$

$i := i + 1$

od

$\{Post : (\forall k | 0 \leq k < i : A[k] \neq 0)\}$

Se tiene que $domain(A[i] \neq 0) \equiv 0 \leq i < N$ y si tomamos como hipóstes TI_N, que en este caso por Corolario 2 es $0 \leq i < N \wedge A[i] \neq 0$, entonces:

$wp(i := i + 1, (\forall k | 0 \leq k < i : A[k] \neq 0))$

$\equiv$

$(\forall k | 0 \leq k < i + 1 : A[k] \neq 0)$

$\equiv$

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

$\equiv < A[i] \neq 0 \equiv true$ por hipótesis>

$(\forall k | 0 \leq k < i : A[k] \neq 0)$

Tomando inv como $(\forall k | 0 \leq k < i : A[k] \neq 0)$, se cumple la obligación de prueba de iteración y como $inv \equiv Post$, entonces se cumple trivialmente la obligación de prueba de terminación, concluyendo que

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

que según el apartado 2 del Corolario 2, es la precondición más débil del algoritmo.

Se puede apreciar claramente que lo anterior es mucho más simple, que demostrar las cinco obligaciones de prueba que establece la lógica de Hoare, para corregir el ciclo anterior.

V. Teoremas de computo para el predicado inv

El Teorema y Corolarios anteriores, tienen la misma limitante que el Teorema de la Invariancia, que pretende que se busque un predicado invariante inv sin ningún método o heurística particular. A continuación se dará un Teorema, que sugiere un método que permite conseguir un predicado inv como el de los Teoremas de la sección anterior, basado en el cálculo de $wp(S, wp(S, \ldots, wp(S, Post), \ldots))$ un número ϵ de veces.

Lema 10. *Sea S_0 una instrucción que no modifica el valor de las variables i y i_f. Sea ϵ una variable no declarada en el programa. Sea PG un predicado tal que $PG \equiv wp(S_0; i := i + 1, PG[\epsilon := \epsilon - 1])$ cuando $0 < \epsilon \leq k$ y TI_k, entonces suponiendo que $i_f - k' \leq i < i_f$ tenemos que para $0 < k' \leq k$:*

$$TI_{k'}$$

$$\Rightarrow$$

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

Demostración: Para hacer esta demostración, se supone que es cierto $i_f - k' \leq i < i_f$ y $TI_{k'}$. Pero como $i_f - k' \leq i < i_f \Rightarrow i_f - k \leq i < i_f$ y $TI_{k'} \Rightarrow TI_k$, se puede asumir $i_f - k \leq i < i_f$ y T_k también.

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

$\equiv$

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

$\equiv$

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

$\equiv$<Lema 9>

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

$\equiv$

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

$\equiv <$ S_0 no modifica a i, i_f y ϵ y lema 7$>$

$(\forall \epsilon | : support(S_0) \wedge$

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

$\equiv$

$support(S_0) \wedge$

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

$\equiv$

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

$\equiv$<Existe ϵ tal que $\epsilon = i_f - (i+1)$ >

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

$\equiv <$ $wp(S_0, P) \Rightarrow support(S_0)$ para cualquier $P >$

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

$\equiv$

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

$\equiv$ La hipótesis $i_f - k \leq i < i_f$ implica $0 \leq i_f - (i+1) < k$

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

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

$\equiv$

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

$\equiv$<Se cumplen hipótesis para aplicar definición de $PG >$

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

$\equiv <$ $0 \leq \epsilon < k$ es redundante$>$

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

$\equiv$

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

$\equiv$

$PG[\epsilon := i_f - i]$ ■

Lema 11. *Sea R un predicado y S una instrucción que se comporta determinísticamente sobre los valores de las variables de R y no modifica las variables i, i_f, k' y ϵ, si denotamos $cond$ como $i < i_f \leq i + k' + 1$, entonces*

$$wp(S, \epsilon \neq (min\ i_f | cond \wedge R : i_f) - i - 1) \equiv$$

$$support(S) \wedge \epsilon \neq (min\ i_f | cond \wedge wp(S, R) : i_f) - i - 1$$

Demostración: $\epsilon \neq (min\ i_f | cond \wedge R : i_f) - i - 1$ es equivalente a

$((\exists if | cond : R) \vee \epsilon + i + 1 \neq +\infty) \wedge$

$(\neg(\exists if | cond : R) \vee \neg(0 \leq \epsilon \leq k') \vee \neg(R[i_f := \epsilon + i + 1]) \vee$

$(\exists i_f | cond : R \wedge i_f < \epsilon + i + 1))$.

Luego se aplica Lemas 2, 3 y 7 a la fómula anterior. ■

Teorema 2. *Sea un ciclo Do donde S es la instrucción $S_0; i := i + 1$ con S_0 una instrucción que no modifica el valor de la variable i. Sean ϵ, i_f variables no declaradas en el programa, que no ocurre en $Post$, definiendo el predicado NBG como en el Teorema 1 y un predicado $PostG$ que satisfaga las siguientes ecuaciones recursivas:*

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

entonces abreviando $\underline{m}$ como

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

se tiene que:

1) *Si $PostG$ es satisfacible dentro de las condiciones de la recurrencia para todo $0 \leq \epsilon \leq k$, entoces el predicado $PostG[\epsilon := \underline{m} - i]$ es un predicado, que satisface las hipótesis del predicado inv en el apartado 1 del Teorema 1.*
2) *Adicionalmente si la recursión que define a $PostG$ está definida hasta $k + 1$ y $PostG[\epsilon := k + 1] \equiv False$, entonces se satisfacen las hipótesis del apartado 2 del Teorema 1.*

Demostración: Definamos inv como $PostG[\epsilon := \underline{m} - i]$ y se demostrará que inv cumple con las ecuaciones del Teorema 1

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

$\equiv$

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

$\equiv$

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

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

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

$\equiv$

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

Por otro lado suponiendo $TI_{k'}$ tenemos que:

Es cierto $\neg NBG[\epsilon := i - i]$, y por lo tanto considerar que i pueda ser igual a i_f en el cálculo de $\underline{m}$, es imposible, de esta forma:

$\underline{m}$

$=$

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

$=$

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

Con esto se puede deducir que:

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

$\equiv$<mismos primeros 4 pasos de la prueba del Lema 10>

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

$\equiv$

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

$\equiv$

$(\forall \epsilon | : wp(S_0, \epsilon \neq (min\ i_f | i_f - k' - 1 \leq i < i_f \wedge$
$(NBG[\epsilon := i_f - i])[i := i + 1] : i_f) - i - 1 \vee$
$PostG[i := i + 1]))$

Como S_0 no modifica los valores de las variables i, i_f, k', ϵ y se comporta determinísticamente sobre las variables de NBG, entonces S_0 se comporta determinísticamente sobre los valores de las variables del lado izquierdo de la disyunción y por Lema 7 la fórmula anterior es equivalente a:

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

$\equiv$<Lema 11>

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

$\equiv$

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

$\equiv$

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

$\equiv$<Lema 10>

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

$\equiv$

$(\forall \epsilon | : support(S_0) \wedge$
$(\epsilon \neq min(i + k' + 1, \underline{m}) - i - 1 \vee$
$wp(S_0, PostG[i := i + 1])))$

$\equiv <\ TI_{k'}$ implica que existe $\underline{m}$ y es menor o igual a $i + k' >$

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

$\equiv$

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

$\equiv$

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

$\equiv <\ TI_{k'}$ implica que existe $\underline{m}$ y por lo tanto existe
$\epsilon = \underline{m} - i - 1 >$

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

$\equiv$

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

$\equiv$

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

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

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

$\equiv$

$PostG[\epsilon := \epsilon + 1][\epsilon = \underline{m} - i - 1]$

$\equiv$

$PostG[\epsilon := \underline{m} - i]$

$\equiv$

inv ■

VI. Ejemplos de Cálculo de Invariantes

El Teorema 2 sugiere un método para calcular $H_k(Post)$ de un ciclo. La técnica consiste en aplicar el transformador de predicados al cuerpo del ciclo y la postcondición ϵ veces hasta deducir el predicado $PostG$. Se muestra a continuación un ejemplo del uso del Teorema 2:

A. Fibonacci

do $i \neq N \rightarrow$

$x, z := z, x + z;$

$i := i + 1$

od

$\{Post : z = fib(N + 1)\}$

La instrucción de asignación en paralelo $x, z, i := z, x + z, i+1$ es equivalente a las dos instrucciones del bloque interno del ciclo, por lo que para resumir se va a usar la instrucción de asignación en paralelo en los cálculos de este ejemplo.

Se aplica wp una vez al cuerpo del ciclo y a la postcondición:

$wp(x, z, i := z, x + z, i + 1, z = fib(N + 1))$

$\equiv$

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

Ahora al resultado anterior se le vuelve aplicar wp obteniendo

$wp(x, z, i := z, x + z, i + 1, x + z = fib(N + 1))$

$\equiv$

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

$\equiv$

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

Si al resultado anterior se le vuelve aplicar wp se obtiene:

$wp(x, z, i := z, x + z, i + 1, x + 2z = fib(N + 1))$

$\equiv$

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

$\equiv$

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

Si al resultado anterior se le vuelve aplicar wp se obtiene $3x+5z = fib(N+1)$ por lo que se observa que los coeficientes que acompañan la x y la y son los números de la secuencia de Fibonacci, por lo que es fácil demostrar por inducción, que el resultado de aplicar wp al cuerpo de éste ciclo y a la postcondición $z = fib(N + 1)$ un número de ϵ de veces es igual a

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

A este predicado lo llamaremos $PostG$, el cual para $\epsilon = N$ es satisfacible (tomando $x, z := 0, 1$), por lo que $PostG$ no es falso cuando $\epsilon = N$, por esta razón $PostG$ no es falso para ningún $\epsilon \leq N$. Esto se debe a que si $PostG \equiv false$ para algún $\epsilon = e < N$, entonces $PostG$ sería falso para todos los $\epsilon > e$ (incluyendo $\epsilon = N$), ya que $wp(S, false) \equiv false$ y $PostG$ se obtuvo de aplicar wp sucesivamente.

Por lo tanto el predicado $PostG$ satisface la recurrencia del Teorema 2 tomando k como N, como $\underline{m} = N$ se concluye que $PostG[N - i]$ satisface las hipótesis del apartado 1 del Teorema 1. Por lo tanto se tiene que:

$H_N(Post)$

$\equiv$

$T_N \wedge (fib(\epsilon)x + fib(\epsilon + 1)z = fib(N + 1))[\epsilon := N - i]$

$\equiv$

$0 \leq i \leq N \wedge fib(N-i)x+fib(N-i+1)z = fib(N+1)$

El cual es un invariante válido para el ciclo.

B. Palabras palíndromes

El siguiente algoritmo para la verificación de si un arreglo de caracteres de tamaño N es un String palíndrome, es un ejemplo del uso del Teorema 2.

do $i \neq N \wedge A[i-1] = A[N-i] \rightarrow$

$pal := A[i] = A[N - 1 - i];$

$i := i + 1$

od

$\{Post : pal \equiv palind(A, 0, N)\}$

Donde el predicado $palind(A, i, N)$ se define como

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

La instrucción de asignación en paralelo $pal, i := A[i] = A[N - 1 - i], i + 1$, es equivalente a las dos instrucciones que se encuentran dentro del ciclo anterior, por lo tanto para simplificar los cálculos siguientes, se usará la instrucción de asignación en paralelo en lugar del cuerpo del ciclo anterior.

Como $domain(A[i - 1] = A[N - i]) \equiv 1 \leq i \leq N$, entonces el Corolario 2 dice que $TI_{N-1} \equiv 1 \leq i < N \wedge A[i - 1] = A[N - i]$ y $T_{N-1} \equiv 1 \leq i \leq N$

Se Aplica el transformador wp al cuerpo del ciclo y la postcondición suponiendo TI_{N-1}:

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

$\equiv$

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

$\equiv < 0 \leq i < N \equiv true$ por hipótesis $TI_{N-1} >$

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

Se aplica ahora el transformador wp al cuerpo del ciclo y al resultado anterior suponiendo TI_{N-1}

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

$\equiv$

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

$\equiv < 0 \leq i < N \equiv true$ por hipótesis $TI_{N-1} >$

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

Por inducción se puede demostrar que aplicar un número ϵ ($\leq N-1$) de veces, el transformador wp al cuerpo del ciclo y la postcondición, se obtiene la siguiente fórmula satisfacible:

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

De igual forma al aplicar el transformador wp al cuerpo del ciclo y la negación de la guardia un número ϵ ($\leq N-1$) de veces, se obtiene la fórmula $i+\epsilon = N \vee A[i+\epsilon-1] \neq A[N-i-\epsilon]$ satisfacible, que al sustituir $\epsilon := i_f - i$ se tiene

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

De esta forma la precondición más débil del ciclo es:

$$1 \leq i \leq N$$

$$\wedge$$

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

donde $\underline{m}$ es una abreviación de

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

VII. Conclusiones

Los teoremas aquí presentados son un pequeño aporte para el desarrollo de un cálculo pragmático para la corrección de programas. Los ejemplos aquí presentados muestran que usando los teoremas adecuados, es posible conseguir la precondición más débil o $H_k(Post)$ de ciertas instrucciones Do, de una forma rápida y formal.

Como las relaciones invariantes y las aserciones invariantes están relacionadas [11], para futuras investigaciones se propone extraer de la implementación de [29], los aspectos que permitan implementar la técnica de cálculo de invariantes descrita en este trabajo. Dicha implementación permitirá no sólo calcular los invariantes $H_k(Post)$, sino también la complejidad de los algoritmos, ya que si f es la función de complejidad del cuerpo de un ciclo y se puede verificar que las condiciones iniciales de la iteración satisfacen $H_k(Post)$, entonces se infiere, que el algoritmo completo es de complejidad $O(kf)$.

Referencias

[1] E. W. Dijkstra, "Guarded Commands, Nondeterminacy and Formal Derivation of Programs,". *Commun. ACM*, vol. 18, no. 8, pp. 453-457, 1975.

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

[3] C. A. R. Hoare, "An Axiomatic Basis for Computer Programming," *Commun. ACM*, vol. 12, no. 10, pp. 576-580, 1969.

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

[5] F. Flaviani, "Cálculo de Precondiciones Más Débiles," *ReVeCom*, vol.3, no 2, pp. 68-80, Dic. 2016.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] J. Berdine, A. Chawdhary, B. Cook, D. Distefano, P. OHearn, "Variance analyses from invariance analyses," in *Proc. 34th Annu. Symp. of Principles Programming Languages*, Nice, France, 2007.

[13] E. Rodríguez-Carbonnell, D. Kapur, "Program verification using automatic generation of invariants," in *Int. Conf. Theoretical Aspects of Computing*, 2004, vol. 3407, pp. 325-340.

[14] J. Carette, R. Janicki. "Computing properties of numeric iterative programs by symbolic computation," *Fundamentae Informatica*, vol. 80, no. 1-3, pp. 125-146, 2007.

[15] M. A. Colon, S. Sankaranarayana, H. B. Sipma. "Linear invariant generation using non linear constraint solving," *Computer Aided Verification*, vol. 2725, pp. 420-432, 2003.

[16] M. D. Ernst, J. H Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, C. Xiao. "The Daikon system for dynamic detection of likely invariants," *Science of Computer Programming*, 2006.

[17] J.C. Fu, F. B. Bastani, I-L. Yen, "Automated discovery of loop invariants for high assurance programs synthesized using ai planning techniques," in *11th High Assurance Systems Eng. Symp.*, Nanjing, China, 2008, pp. 333-342.

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

[19] L. Kovacs, T. Jebelean. "Automated generation of loop invariants by recurrence solving in theorema," In *D. Proc. 6th Int. Symp. Symbolic and Numeric Algorithms for Scientific Computing*, Timisoara, Romania, 2004, Mirton Publisher, pp. 451-464.

[20] L. Kovacs, T. Jebelean. "An algorithm for automated generation of invariants for loops with conditionals," *7th Int. Symp. Symbolic and Numeric Algorithms for Scientific Computing*, Timisoara, Romania, 2005, pp. 16-19.

[21] S. Sankaranarayana, H. B. Sipma, Z. Manna, "Non linear loop invariant generation using Groebner bases," In *Proc. ACM SIGPLAN Principles of Programming Languages*, 2004, pp. 381-329.

[22] A. Gupta, A. Rybalchenko, "InvGen: An Efficient Invariant Generator,"*Int. Conf. on Computer Aided Verification*, 2009, pp. 634-640.

[23] S. Sankaranarayanan, M. Colon, H. Sipma, Z. Manna. *Stanford Invariant Generator*. (2006, May 3)[Online]. Available: http://theory.stanford.edu/ srirams/Software/sting.html

[24] E. Rodríguez-Carbonell, D. Kapur, "Generating all polynomial invariants in simple loops," *J. Symbolic Comput.*, vol. 42, no. 4, pp. 443-476, 2007.

[25] S. Magill, A. Nanevski, E. Clarke, P. Lee, "Inferring Invariants in Separation Logic for Imperative List-processing Programs," *SPACE*, Vol. 1, no. 1, pp. 5-7, 2006.

[26] J. Berdine, B. Cook, S. Ishtiaq, *SLAyer: Memory Safety for Systems-Level Code*, Gopalakrishnan, Heidelberg, Springer, vol. 6806, pp. 178-183, 2011.

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

[28] M. Barnett, K. Rustan, M. Leino, Microsoft Research. "Weakest-Precondition of Unstructured Programs," in *05 Proc. 6th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, 2006, Vol. 31, no. 1, pp. 82-87.

[29] 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.

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

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