Cálculo de Precondiciones más Débiles

Federico Flaviani
Universidad Simón Bolívar
Caracas, Venezuela
federico.flaviani@gmail.com

***Resumen*—El cálculo comenzó a desarrollarse en el siglo XVII con la intención de conseguir técnicas estándar para resolver problemas referentes a derivadas, integrales y ecuaciones diferenciales. Gracias al desarrollo de esta ciencia y la estandarización de las técnicas de cálculo integral, hoy en día es posible que un estudiante de primer año de universidad pueda aprender a resolver una cantidad enorme de integrales en poco tiempo y aunque el estudiante tenga la sensación de que esta usando el sentido común para hacer dichos cálculos, realmente puede hacerlo porque esta siguiendo un algoritmo de resolución, muy complejo, pero algoritmo en fin. Este algoritmo es el que hace posible que hoy en día, aplicaciones como Mapple o Mathematica puedan resolver simbólicamente integrales y ecuaciones diferenciales. Es decir, el mayor porcentaje del crédito por el éxito de estas aplicaciones, se debe a la existencia de los algoritmos de resolución del cálculo, que han sido desarrollados durante cuatro siglos. Por otro lado en el mundo de la demostración formal de programas, también existen software como en [1], [2], [3], que intentan hacer cómputos simbólicos para la corrección de programas. Estas aplicaciones para la corrección de programas, no usan un algoritmo de cuatro siglo de desarrollo que estandarizan técnicas del cálculo de los invariantes y obligaciones de prueba, por lo que tienen una desventaja natural con respecto a las aplicaciones de resolución simbólica para el cálculo integral o diferencial.**

Por lo argumentado anteriormente se propone que para lograr un avance significativo en las aplicaciones para la corrección de programas, es conveniente crear ahora, en la ciencia de la matemática, un cálculo (al igual como en su tiempo se creó el cálculo integral), para estandarizar los métodos que permiten computar invariantes, obligaciones de prueba y precondiciones más débiles. Este articulo muestra algunos teoremas que tienen la intención de dar un pequeño paso en el desarrollo de este cálculo pero para computar precondiciones más débiles.

Palabras clave—precondición más débil, cálculo, corrección formal de programas, GCL, inducción.

I. Introducción

Todos los algoritmos planteados en este trabajo serán escritos en pseudolenguaje GCL (Guarded Command Language) [5], 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.

Si $B_0, \ldots, B_n$ son expresiones booleanas del lenguaje GCL y definiendo DG como una abreviación de $domain(B_0, \ldots, B_n)$ (predicado que expresa que todas las expresiones están bien definidas [6]), tenemos que la lógica de Hore [4] se basa en las siguientes reglas de inferencia:

$$\{A\}SKIP\{A\}$$

$$\{domain(\overline{E}) \wedge B[\overline{x} := \overline{E}]\}\overline{x} := \overline{E}\{B\}$$

$$\frac{\{A\}S_0\{B\} \qquad \{B\}S_1\{C\}}{\{A\}S_0; S_1\{C\}}$$

$$\frac{A \Rightarrow DG \wedge (B_0 \vee .. \vee B_n) \qquad \{A \wedge B_0\}S_0\{B\} .. \qquad ..\{A \wedge B_n\}S_n\{B\}}{\{A\}if\ B_0 \rightarrow S_0 [] .. [] B_n \rightarrow S_n fi\{B\}}$$

$$\frac{I \Rightarrow domain(B_0) \qquad \{I \wedge B_0\}S_0\{I\}}{\{I\}do\ B_0 \rightarrow S_0\{I \wedge \neg B_0\}}$$

$$\frac{A \Rightarrow A' \qquad \{A'\}S_0\{B'\} \qquad B' \Rightarrow B}{\{A\}S_0\{B\}}$$

Donde $S_0, \ldots, S_n$ son instrucciones A, A', B, B' y C son predicados, I es un predicado que lleva el nombre de "invariante del ciclo" y $\overline{E}$ es una lista de expresiones, que una a una, son del mismo tipo que la lista de variables $\overline{x}$ (la barra superior tanto en E como en x, es una notación que indica que se tiene una lista de expresiones y variables respectivamente).

De las reglas de inferencia de arriba se concluye que una derivación en esta lógica tiene forma de árbol. Adicionalmente una demostración de que un algoritmo es correcto usando la lógica de Hoare, no garantiza la terminación del programa, por lo que el árbol de derivación usando las reglas de Hoare, debe ir acompañado de un argumento de función de cota en cada ciclo Do, si se quiere tener una demostración completa de que el algoritmo es correcto con respecto a la especificación.

Por otro lado la lógica de Dijkstra [5] 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, simbolicamente hablando, de la instrucción $inst$ con la instrucción $Post$). El uso sucesivo de wp permite ir calculando precondiciones más débiles entre instrucción e instrucción, desde el final del programa hasta el inicio como se muestra en la siguiente figura.

$\{Pre\}$	$\{Pre\}$
$instrucción1$	$instrucción1$
$instrucción2$	$instrucción2$
$instrucción3$	$\{A1 : wp(instrucción3, Post)\}$
$\{Post\}$	$instrucción3$
Paso 0	Paso 1
$\{Pre\}$	$\{Pre\}$
$instrucción1$	$\{A3 : wp(instrucción1, A2)\}$
$\{A2 : wp(inst2, A1)\}$	$instrucción1$
$instrucción2$	$instrucción2$
$instrucción3$	$instrucción3$
Paso 2	Paso 3

Fig 1: uso sucesivo de wp

Usando el transformador wp de Dijkstra sucesivamente, se puede calcular la precondición más débil de todo el programa, de modo que para demostrar la corrección de un algoritmo, basta con demostrar que la precondición Pre implica la precondicción más débil calculada (En el caso del ejemplo de la figura se debe demostrar $Pre \Rightarrow A3$)

La corrección de programas usando la lógica de Dijkstra garantiza terminación y no necesita el argumento de función de cota en cada iteración, por lo que induce un método limpio para decidir la corrección de un programa.

Calcular una precondición más débil usando el transformador wp para cualquier postcondición e instrucción es fundamental, porque permite ir consiguiendo aserciones que permitan subir entre instrucción e instrucción como en la figura. Para lograr esto Dijkstra en [5] definió las reglas que definen la función de transformación sintáctica wp de la siguiente manera:

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

  en donde $H_k(Post)$ es un predicado que satisface las ecuaciones

  $$H_0(Post) \equiv domain(BB) \wedge \neg(B_0 \vee \cdots \vee B_n) \wedge Post$$

  $$H_k(Post) \equiv H_0(Post) \vee wp(IF, H_{k-1}(Post))$$

  para $k \geq 1$

Lamentablemente la fórmula que define a wp en general para ciclos DO, está escrita en lógica de segundo orden y por lo tanto no es aplicable directamente. Tampoco es útil tener aserciones escritas en lógica de segundo orden, ya que si una aserción de segundo orden fuese la postcondición de una instrucción de asignación $y_{i_1}, \ldots, y_{i_k} := Exp_1, \ldots, Exp_k$, entonces no se pudiera aplicar la segunda de las reglas listadas anteriormente, porque el operador de sustitución $[y_{i_1}, \ldots, y_{i_k} := Exp_1, \ldots, Exp_k]$ no esta definido para fórmulas escritas en segundo orden.

Esta limitante de la descripción de wp de Dijkstra lo convierte, en el caso de programas que contienen ciclos DO, en una herramienta teórica sin pragmática alguna. Sin embargo, esta demostrado en [7] que para cada algoritmo en particular, con una postcondición particular, siempre existe un predicado escrito en lógica de primer orden equivalente a la definición de wp para el caso en cuestión. El único problema es que la demostración que se encuentra en [7] no es constructiva y no induce un algoritmo para computar dicho predicado.

La idea que se propone en este artículo es la de dar técnicas de cálculo que permitan construir una fórmula escrita en lógica de primer orden, que sea equivalente a la fórmula de Dijkstra para los ciclos DO, de modo tal de rescatar la idea original del transformador de predicado wp y convertirla en una herramienta práctica para poder decidir si un programa es correcto. Al igual como en el cálculo integral, en donde cada técnica de integración depende del tipo de la integral (si es polinómica, trigonométrica, exponencial, o multiplicación y división de las combinaciones de las tres categorías anteriores) las técnicas que se proponen dependerán del tipo de ciclo.

El resultado será el de caracterizar algunos tipos de ciclos en los que se pueda obtener una plantilla de predicado equivalente a wp en lógica de primer orden, de manera que la corrección del algoritmo consistirá simplemente en la aplicación de la plantilla instanciada al caso particular que se encuentre (sumándole inevitablemente algún toque mínimo del sentido común humano).

A continuación se presentan cuatro secciones de las cuales, en la primera de ellas se hace un repaso de las fórmulas de segundo orden que propuso Dijkstra para definir wp en ciclos DO en general. Luego se caracterizan que fórmulas de primer orden se deben usar para calcular wp en ciclos for. Seguidamente se presentan ejemplos del uso de los teoremas presentados y por último se da una sección de teoremas complementarios para el cálculo de precondiciones más débiles.

II. wp PARA LA INSTRUCCIÓN Do

El lenguaje GCL de Dijkstra fue diseñado para poder hacer algoritmos no determinísticos. Tanto la instrucción condicional como la de iteración tienen versiones no determísticas en GCL, la versión no determinística de la instrucción de iteración se denota DO, mientras la versión determinística se denota Do. La definición de wp que se listó anteriormente para la instrucción de iteración corresponde a la versión no determinística DO de la misma. Se puede deducir una fórmula de wp para la instrucción Do a partir de la anterior, sin embargo en [8] se encuentra dicha fórmula y es:

$$(\exists k | k \geq 0 : H_k(Post))$$

donde $H_k(Post)$ es un predicado definido recursivamente como

$$H_0(Post) := domain(B_0) \wedge \neg B_0 \wedge Post$$

$$H_k(Post) :=$$
$$H_0(Post) \vee (domain(B_0) \wedge B_0 \wedge wp(S_0, H_{k-1}(Post)))$$
$$para\ k \geq 1$$

La fórmula anterior está escrita en lógica de segundo orden, pero es conocido (ver por ejemplo [7]) que si restringimos el lenguaje de las aserciones a cierto tipo de lenguaje de primer orden, es siempre posible que para cada instancia particular de S_0, B_0 y $Post$, se puede escribir la fórmula de wp del párrafo anterior, de una forma equivalente usando estos lenguajes de primer orden.

Este resultado sigue siendo cierto si el lenguaje que se usa para escribir las aserciones es el de la teoría de conjuntos de Zermelo-Fraenkel-Skolem. A continuación se muestra una demostración.

Teorema 1. *Si el lenguaje que se usa para escribir los predicados de las aserciones en GCL, es el lenguaje de primer orden de la teoría de conjuntos de Zermelo-Frankel-Skolem, entonces por cada caso particular de predicado $Post$, de expresión B_0 e instrucción S_0, existe una fórmula escrita en el lenguaje de primer orden de la teoría de conjuntos de Zermelo-Framkel-Skolem, que es equivalente a $wp(do\ B_0 \to S_0\ od, Post)$*

Demostración: Se considerará la semántica denotacional de GCL que se propuso en [8].

Sea S_0 una instrucción de GCL, entonces se denota como $interpre[\![S_0]\!]$ a la relación del espacio de estados al espacio de estados del programa, que corresponde a la interpretación con respecto a la semántica denotacional de [8], de la instrucción S_0 (en [8] se usó la notación R_{S_0} en lugar de $interpre[\![S_0]\!]$).

Se definen recursivamente las siguientes instrucciones

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

Como la interpretación de una secuenciación de instrucciones es la composición de las interpretaciones, entonces la interpretación de la instrucción Do_k satisface la siguiente recurrencia:

$$interpre[\![Do_0]\!] := interpre[\![if \neg B_0 \to SKIP fi]\!]$$

$$interpre[\![Do_{k+1}]\!] := interpre[\![Do_k]\!] \circ interpre[\![If]\!]$$

Por otro lado la fórmula definida por

$$\varphi(k, F, y) :=$$
$$(k = 0 \wedge y = interpre[\![Do_0]\!]) \vee$$
$$(k \neq 0 \wedge k \in \omega \wedge esfuncion(F) \wedge y = F(k-1) \circ interpre[\![If]\!]) \vee$$
$$(\neg(k = 0 \vee (k \neq 0 \wedge k \in \omega \wedge esFuncion(F))) \wedge y = F)$$

satisface que

$$(\forall k, F| : (\exists! y| : \varphi(k, F, y))).$$

Si se define a $G(k, F)$ como el único y que satisface $\varphi(k, F, y)$, entonces por el teorema de recursión transfinita [9] aplicado a los ordinales finitos ω, se tiene que existe una formula ψ que satisface que:

1) $(\forall k| : (\exists! y| : \psi(k, y)))$, es decir ψ define una función F tal que $\psi(k, F(k))$
2) $(\forall k | k \in \omega| : F(k) = G(k, F \restriction_{k-1}))$

Como $G(k, F)$ en notación de llaves es la función a trozos

$$G(k, F) = \begin{cases} interpre[\![Do_0]\!] & si \quad k = 0 \\ F(k-1) \circ interpre[\![If]\!] & si \quad k \in \omega\ y\ esFuncion(F) \\ F & sino \end{cases}$$

entonces la fórmula

$$(\forall k | k \in \omega| : F(k) = G(k, F \restriction_{k-1}))$$

es equivalente a la recurrencia que define a $interpre[\![Do_k]\!]$ arriba y por lo tanto, $F(k)$ debe ser igual a $interpre[\![Do_k]\!]$. Por esta razón, como se tiene que F es una función tal que $F(k)$ es el único valor en el que $\psi(k, F(k))$ es verdad, entonces cuando el predicado

$$\psi(k, R)$$

sea cierto, debe ocurrir que $R = interpre[\![Do_k]\!]$.

Por otro lado usando las notaciones de [8], las cuales son: $\vec{x}$ para indicar el vector de constantes y variables declaradas en un algoritmo, Esp para indicar el espacio de estados del algoritmo y $Rgo_{\overline{Y}}$ para referirse al conjunto $\{\vec{x} \in Esp | Post(\vec{x}, \overline{Y})\}$, se demostró que

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

es un predicado equivalente a $wp(do\ B_0 \to S_0\ od, Post)$. Sin embargo, usando el predicado ψ se puede reescribir la fórmula anterior como

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

la cual está escrita en el lenguaje de primer orden de la teoría de conjuntos. ■

Según el teorema anterior, si se usa el lenguaje de la teoría de conjuntos de Zermelo-Frankel-Skolem para escribir las aserciones, entonces se tiene garantía de que toda precondición más débil se puede escribir dentro del lenguaje. El siguiente corolario habla sobre algunos tipos de lenguajes, de la lógica de primer orden, que son convenientes para escribir las aserciones de los algoritmos.

Corolario 1. *Sea L un lenguaje de la lógica de primer orden para escribir predicados en las aserciones de GCL. Si L es un lenguaje tal que, por cada fórmula en L, existe una fórmula equivalente en el lenguaje de la teoría de conjuntos de Zermelo-Frankel-Skolem y por cada fórmula en la teoría de conjuntos de Zermelo-Frankel-Skolem, existe una fórmula equivalente en L, entonces por cada caso particular de predicado $Post$, de expresión B_0 e instrucción S_0, existe una fórmula escrita en L, que es equivalente a $wp(do\ B_0 \to S_0\ od, Post)$ y por lo tanto el transformador de predicados wp esta bien definido en todo L*

El corolario y teorema anterior sugieren que con el lenguaje adecuado, siempre es posible encontrar un predicado de primer orden para la precondición más débil de un ciclo, sin embargo como la demostración no es constructiva, no se tiene un procedimiento para obtener dicho predicado. El objetivo de este trabajo, es el de mostrar un método que permite conseguir la precondición más débil (escrita en lógica de primer orden) de cierto tipos de ciclos.

En las siguientes secciones usaremos las siguientes propiedades del transformador de predicado wp:

- $wp(S_0, False) \equiv False$
- $wp(S_0, Post_1 \wedge Post_2) \equiv wp(S_0, Post_1) \wedge wp(S_0, Post_2)$
- Si $Post_1 \Rightarrow Post_2$ entonces $wp(S_0, Post_1) \Rightarrow wp(S_0, Post_2)$
- Si S_0 es una instrucción determinística, entonces $wp(S_0, Post_1 \vee Post_2) \equiv wp(S_0, Post_1) \vee wp(S_0, Post_2)$

III. Instrucción for

Una instrucción for es equivalente a un ciclo Do de la siguiente forma:

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

Donde S_0 es un bloque de instrucciones que no modifica las variable i y N. Para poder calcular la precondición más débil de un ciclo de este estilo, presentamos el siguiente teorema y Corolario.

Teorema 2. *Sea un algoritmo con la estructura que se presentó anteriormente y k, k' son variable que no ocurren en S_0 y $Post$, entonces:*

1) *Si existe un predicado inv tal que $inv[i := N] \equiv Post$ y $(\forall i | N - k \leq i < N : wp(S_0; i := i+1, inv) \equiv inv)$, donde las variables k y k' no ocurren en inv, entonces*

$$H_{k'}(Post) \equiv N - k' \leq i \leq N \wedge inv$$

   para todo k' tal que $0 \leq k' \leq k$.
2) *Adicionalmente si $wp(S_0; i := i + 1, inv) \equiv False$ cuando $i = N - (k + 1)$, entonces*

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

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 ecuacional presentada en el libro de Gries [10].

Se demostrará por inducción sobre k' suponiendo que $k' \leq k$ y que k' y k son variables que no ocurren en inv, S_0 y $Post$.

Caso 1 $k' = 0$

$H_{k'}(Post)$
$\equiv< k' = 0 >$
$H_0(Post)$
$\equiv$
$i = N \wedge Post$
$\equiv$
$i = N \wedge inv[i := N]$
$\equiv$
$i = N \wedge inv[i := i]$
$\equiv$
$N \leq i \leq N \wedge inv$
$\equiv< k' = 0 >$
$N - k' \leq i \leq N \wedge inv$

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

$H_{k'}(Post)$
$\equiv$
$H_0(Post) \vee (i \neq N \wedge wp(S_0; i := i + 1, H_{k'-1}(Post)))$

$\equiv$<hipótesis inductiva>

$H_0(Post) \vee (i \neq N \wedge wp(S_0; i := i + 1,$
$N - (k' - 1) \leq i \leq N \wedge inv))$

$\equiv$<hipótesis>

$H_0(Post) \vee (i \neq N \wedge wp(S_0; i := i + 1,$
$N - (k' - 1) \leq i \leq N) \wedge wp(S_0; i := i + 1, inv))$

$\equiv< i$ y N no se modifican en S_0 y k' no ocurre en $S_0 >$

$H_0(Post) \vee (i \neq N \wedge N - (k' - 1) \leq i + 1 \leq N \wedge$
$wp(S_0; i := i + 1, inv))$
$\equiv$
$H_0(Post) \vee (i \neq N \wedge N - k' \leq i \leq N - 1 \wedge$
$wp(S_0; i := i + 1, inv))$

$\equiv$<hipótesis>

$H_0(Post) \vee (i \neq N \wedge N - k' \leq i \leq N - 1 \wedge inv)$
$\equiv$
$H_0(Post) \vee (N - k' \leq i \leq N - 1 \wedge inv)$
$\equiv$
$(i = N \wedge Post) \vee (N - k' \leq i \leq N - 1 \wedge inv)$

$\equiv$<hipótesis>

$(i = N \wedge inv[i := N]) \vee (N - k' \leq i \leq N - 1 \wedge inv)$
$\equiv$
$(i = N \wedge inv[i := i]) \vee (N - k' \leq i \leq N - 1 \wedge inv)$
$\equiv$
$(i = N \wedge inv) \vee (N - k' \leq i \leq N - 1 \wedge inv)$

$\equiv$
$(i = N \vee N - k' \leq i \leq N - 1) \wedge inv$
$\equiv$
$(N - k' \leq i \leq N) \wedge inv$

Por otro lado

Si $wp(S_0; i := i+1, inv) \equiv False$ cuando $i = N-(k+1)$ entonces

$H_{k+1}(Post)$
$\equiv$
$H_0(Post) \vee (i \neq N \wedge wp(S_0; i := i+1, H_k(Post)))$
$\equiv$
$H_0(Post) \vee (i \neq N \wedge wp(S_0; i := i+1,$
$N - k \leq i \leq N \wedge inv))$
$\equiv$
$H_0(Post) \vee (i \neq N \wedge wp(S_0; i := i+1,$
$N - k \leq i \leq N) \wedge wp(S_0; i := i+1, inv))$

$\equiv<$ i y N no se modifican en S_0 y k no ocurre en S_0 $>$

$H_0(Post) \vee (i \neq N \wedge N - k \leq i + 1 \leq N \wedge$
$wp(S_0; i := i+1, inv))$

$\equiv$<hipótesis>

$H_0(Post) \vee (i \neq N \wedge N - k - 1 \leq i \leq N - 1 \wedge$
$wp(S_0; i := i+1, inv))$
$\equiv$
$H_0(Post) \vee (i \neq N \wedge N - (k+1) \leq i \leq N - 1 \wedge$
$wp(S_0; i := i+1, inv))$
$\equiv$
$H_0(Post) \vee (N - (k+1) \leq i \leq N - 1 \wedge$
$wp(S_0; i := i+1, inv))$

$\equiv< k \geq 0 >$

$H_0(Post) \vee ((i = N - (k+1) \vee N - k \leq i \leq N - 1) \wedge$
$wp(S_0; i := i+1, inv))$
$\equiv$
$H_0(Post) \vee (i = N - (k+1) \wedge wp(S_0; i := i+1, inv)) \vee$
$(N - k \leq i \leq N - 1 \wedge wp(S_0; i := i+1, inv))$

$\equiv$<hipótesis>

$H_0(Post) \vee (i = N - (k+1) \wedge False) \vee$
$(N - k \leq i \leq N - 1 \wedge inv)$
$\equiv$
$H_0(Post) \vee False \vee (N - k \leq i \leq N - 1 \wedge inv)$
$\equiv$
$H_0(Post) \vee (N - k \leq i \leq N - 1 \wedge inv)$
$\equiv$
$(i = N \wedge Post) \vee (N - k \leq i \leq N - 1 \wedge inv)$

$\equiv$<hipótesis>

$(i = N \wedge inv[i := N]) \vee (N - k \leq i \leq N - 1 \wedge inv)$
$\equiv$
$(i = N \wedge inv[i := i]) \vee (N - k \leq i \leq N - 1 \wedge inv)$
$\equiv$
$(i = N \wedge inv) \vee (N - k \leq i \leq N - 1 \wedge inv)$
$\equiv$
$(i = N \vee N - k \leq i \leq N - 1) \wedge inv$
$\equiv$
$N - k \leq i \leq N \wedge inv$
$\equiv$
$H_k(Post)$ ■

Corolario 2. *Si un predicado inv cumple con las hipótesis del caso 1 del teorema 2 para todo k y no cumple con la hipótesis del caso 2 del mismo teorema, entonces*

$$wp(do\ B_0 \rightarrow S_0; i := i+1\ od, Post) \equiv i \leq N \wedge inv$$

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

$wp(do\ B_0 \rightarrow S_0; i := i+1\ od, Post) \equiv N - k \leq i \leq N \wedge inv$

Demostración: Si inv cumple con las hipótesis del caso 1 del teorema 2 para todo k y no se cumple la hipótesis del caso 2, entonces

$wp(do\ B_0 \rightarrow S_0; i := i+1\ od, Post)$
$\equiv$
$(\exists k | k \geq 0 : H_k(Post))$
$\equiv$
$(\exists k | k \geq 0 : N - k \leq i \leq N \wedge inv)$
$\equiv$
$(\exists k | k \geq 0 : N - k \leq i \wedge i \leq N \wedge inv)$
$\equiv$
$i \leq N \wedge inv \wedge (\exists k | k \geq 0 : N - k \leq i)$
$\equiv$
$i \leq N \wedge inv \wedge True$
$\equiv$
$i \leq N \wedge inv$

Si se cumplen las hipótesis de los casos 1 y 2 del teorema 2, entonces asumiendo que la variable k del teorema es mayor o igual a cero, se tiene que

$(\exists k'' | k'' \geq 0 : H_{k''}(Post))$
$\equiv$
$(\exists k'' | 0 \leq k'' \leq k : H_{k''}(Post)) \vee$
$(\exists k'' | k < k'' : H_{k''}(Post))$

$\equiv$<caso 1 del teorema 2>

$(\exists k'' | 0 \leq k'' \leq k : N - k'' \leq i \leq N \wedge inv) \vee$
$(\exists k'' | k < k'' : H_k(Post))$
$\equiv$
$(\exists k'' | 0 \leq k'' \leq k : N - k'' \leq i \wedge i \leq N \wedge inv) \vee$
$(\exists k'' | : k < k'' \wedge H_k(Post))$
$\equiv$
$(i \leq N \wedge inv \wedge (\exists k'' | 0 \leq k'' \leq k : N - k'' \leq i)) \vee$
$(H_k(Post) \wedge (\exists k'' | : k < k''))$
$\equiv$

$(i \leq N \wedge inv \wedge N - k \leq i) \vee (H_k(Post) \wedge True)$
$\equiv$
$(N - k \leq i \wedge i \leq N \wedge inv) \vee (H_k(Post))$
$\equiv$
$(N - k \leq i \leq N \wedge inv) \vee (N - k \leq i \leq N \wedge inv)$
$\equiv$
$(N - k \leq i \leq N \wedge inv)$

■

El teorema anterior tiene 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 como el que establece el teorema anterior

Teorema 3. *Sea un algoritmo con la estructura mostrada anteriormente, ϵ una variable distinta de N que no ocurre en $S_0; i := i + 1$ ni en $Post$, i no ocurre en $Post$, S_0 es una instrucción determinista y definiendo el predicado $PostG$ tal que satisfaga las siguientes ecuaciones recursivas:*

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

entonces el predicado $PostG[\epsilon := N - i]$ es un predicado, que satisface las hipótesis del caso 1 del teorema 2.

Demostración: Instanciemos el predicado $PostG[\epsilon := N - i]$ con $i := N$

$PostG[\epsilon := N - i][i := N]$

$\equiv <$ i puede ocurrir en $PostG >$

$PostG[i := N][\epsilon := N - N]$
$\equiv$
$PostG[i := N][\epsilon := 0]$

$\equiv <$ ϵ es una variable fresca$>$

$PostG[\epsilon := 0][i := N]$

$\equiv$<hipótesis>

$Post[i := N]$

$\equiv <$ i no ocurre en $Post >$

$Post$

Por otro lado tenemos que:

$(\forall i | N - k \leq i < N : wp(S_0; i := i+1, PostG[\epsilon := N - i]))$
$\equiv$
$(\forall i | N - k \leq i < N : wp(S_0; i := i + 1,$
$(\forall \epsilon | \epsilon = N - i : PostG)))$
$\equiv$
$(\forall i | N - k \leq i < N : wp(S_0; i := i + 1,$
$(\forall \epsilon | : \epsilon = N - i \Rightarrow PostG)))$
$\equiv$
$(\forall i | N - k \leq i < N : (\forall \epsilon | : wp(S_0; i := i + 1,$
$\epsilon = N - i \Rightarrow PostG)))$
$\equiv$
$(\forall i | N - k \leq i < N : (\forall \epsilon | : wp(S_0; i := i + 1,$
$\epsilon \neq N - i \vee PostG)))$

$\equiv <$ $S_0; i := i + 1$ es una instrucción determinista$>$

$(\forall i | N - k \leq i < N : (\forall \epsilon | : wp(S_0; i := i + 1, \epsilon \neq N - i) \vee$
$wp(S_0; i := i + 1, PostG)))$

$\equiv <$ i y N no se modifican en S_0 y ϵ no ocurre en $S_0 >$

$(\forall i | N - k \leq i < N : (\forall \epsilon | : \epsilon \neq N - (i + 1) \vee$
$wp(S_0; i := i + 1, PostG)))$
$\equiv$
$(\forall i | N - k \leq i < N :$
$(\forall \epsilon | : \epsilon = N - (i + 1) \Rightarrow wp(S_0; i := i + 1, PostG)))$
$\equiv$
$(\forall i | N - k \leq i < N :$
$(\forall \epsilon | \epsilon = N - (i + 1) : wp(S_0; i := i + 1, PostG)))$
$\equiv$
$(\forall i | N - k \leq i < N :$
$(\forall \epsilon | i = N - (\epsilon + 1) : wp(S_0; i := i + 1, PostG)))$
$\equiv$
$(\forall i | N - k \leq i < N : (\forall \epsilon | i = N - (\epsilon + 1) \wedge N - k \leq i \wedge$
$i < N : wp(S_0; i := i + 1, PostG)))$
$\equiv$
$(\forall i | N - k \leq i < N : (\forall \epsilon | i = N - (\epsilon + 1) \wedge$
$N - k \leq N - (\epsilon + 1) \wedge N - (\epsilon + 1) < N :$
$wp(S_0; i := i + 1, PostG)))$
$\equiv$
$(\forall i | N - k \leq i < N : (\forall \epsilon | i = N - (\epsilon + 1) \wedge \epsilon \leq k - 1 \wedge$
$-1 < \epsilon : wp(S_0; i := i + 1, PostG)))$
$\equiv$
$(\forall i | N - k \leq i < N : (\forall \epsilon | i = N - (\epsilon + 1) \wedge 0 \leq \epsilon < k :$
$wp(S_0; i := i + 1, PostG)))$

$\equiv$<hipótesis>

$(\forall i | N - k \leq i < N : (\forall \epsilon | i = N - (\epsilon + 1) \wedge 0 \leq \epsilon < k :$
$PostG[\epsilon := \epsilon + 1]))$
$\equiv$
$(\forall i | N - k \leq i < N : (\forall \epsilon | i = N - (\epsilon + 1) :$
$PostG[\epsilon := \epsilon + 1]))$
$\equiv$
$(\forall i | N - k \leq i < N : (\forall \epsilon | \epsilon = N - (i + 1) :$
$PostG[\epsilon := \epsilon + 1]))$
$\equiv$
$(\forall i | N - k \leq i < N : PostG[\epsilon := \epsilon + 1][\epsilon := N - (i + 1)])$
$\equiv$
$(\forall i | N - k \leq i < N : PostG[\epsilon := N - (i + 1) + 1])$
$\equiv$
$(\forall i | N - k \leq i < N : PostG[\epsilon := N - i])$ ■

IV. EJEMPLOS

El teorema anterior sugiere un método para calcular la precondición más débil de un ciclo del tipo que se presentó anteriormente. 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:

A. Factorial

do $i \neq N \rightarrow$
$fact := fact * i;$
$i := i + 1$
od
$\{Post : N > 0 \wedge fact = (N-1)!\}$

La instrucción de asignación en paralelo $fact, i := fact * i, i+1$ 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:

$wp(fact, i := fact * i, i+1, N > 0 \wedge fact = (N-1)!)$
$\equiv$
$(N > 0 \wedge fact = (N-1)!)[fact, i := fact * i, i+1]$
$\equiv$
$N > 0 \wedge fact * i = (N-1)!$

Ahora al resultado anterior se le vuelve aplicar wp obteniendo

$wp(fact, i := fact * i, i+1, N > 0 \wedge fact * i = (N-1)!)$
$\equiv$
$(N > 0 \wedge fact * i = (N-1)!)[fact, i := fact * i, i+1]$
$\equiv$
$N > 0 \wedge fact * i * (i+1) = (N-1)!$

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

$wp(fact, i := fact * i, i+1, N > 0 \wedge fact * i * (i+1) = (N-1)!)$
$\equiv$
$(N > 0 \wedge fact * i * (i+1) = (N-1)!)[fact, i := fact * i, i+1]$
$\equiv$
$N > 0 \wedge fact * i * (i+1) * (i+2) = (N-1)!$

y si se define

$$i^{\overline{\epsilon}} := \begin{cases} i(i+1)(i+2)\cdots(i+\epsilon-1) & si \quad \epsilon > 0 \\ 1 & si \quad \epsilon = 0 \end{cases}$$

entonces es fácil demostrar por inducción, que el resultado de aplicar wp al cuerpo de éste ciclo y a la postcondición $N > 0 \wedge fact = (N-1)!$ un número de ϵ de veces es igual a

$$N > 0 \wedge fact * i^{\overline{\epsilon}} = (N-1)!$$

Por lo tanto este predicado satisface la recurrencia que define a $PostG$ en el último teorema de la sección anterior para $0 \leq \epsilon \leq N-1$.

De esta forma usando el teorema recién referenciado, tenemos que

$(N > 0 \wedge fact * i^{\overline{\epsilon}} = (N-1)!)[\epsilon := N - i]$
$\equiv$
$N > 0 \wedge fact * i^{\overline{N-i}} = (N-1)!$
$\equiv$
$N > 0 \wedge fact * i^{\overline{N-i}} = i^{\overline{N-i}} * (i-1)!$
$\equiv <$ *regla de cancelación* $>$
$N > 0 \wedge fact = (i-1)!$

Es un predicado que satisface las hipótesis del caso 1 del teorema 2, es decir

$(N > 0 \wedge fact = (i-1)!)[i := N] \equiv Post$

y

$(\forall i | N - (N-1) \leq i < N :$
$wp(fact, i := fact * i, i+1, fact = (i-1)!) \equiv$
$fact = (i-1)!).$

Adicionalmente si $i = 0 = N - ((N-1)+1)$, entonces

$wp(fact, i := fact * i, i+1, N > 0 \wedge fact = (i-1)!)$
$\equiv$
$N > 0 \wedge fact * i = ((i+1)-1)!$
$\equiv$
$N > 0 \wedge fact * i = i!$
$\equiv$
$N > 0 \wedge fact * 0 = 0!$
$\equiv$
$N > 0 \wedge 0 = 1$
$\equiv$
$N > 0 \wedge False$
$\equiv$
$False$

Por lo que se cumple la hipótesis del caso 2 del teorema 2 y por lo tanto la precondición más débil del ciclo de éste ejemplo, es la siguiente:

$$N - (N-1) \leq i \leq N \wedge N > 0 \wedge fact = (i-1)!$$

que es equivalente a

$$1 \leq i \leq N \wedge N > 0 \wedge fact = (i-1)!$$

B. Sumatoria de los primeros N enteros al cuadrado

Para ejemplificar el uso de los teoremas anteriores, se considerará ahora el siguiente algoritmo para calcular la sumatoria de los primeros N enteros positivos al cuadrado.

do $i \neq N \rightarrow$

$suma := suma + i * i;$
$i := i + 1$
od
$\{Post : N \geq 0 \wedge suma = \sum_{j=0}^{N-1} j * j\}$

La instrucción de asignación en paralelo $suma, i := suma + i * i, i + 1$, 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.

Se Aplica el transformador wp al cuerpo del ciclo y la postcondición:

$wp(suma, i := suma+i*i, i+1, N \geq 0 \wedge suma = \sum_{j=0}^{N-1} j^2)$
$\equiv$
$(N \geq 0 \wedge suma = \sum_{j=0}^{N-1} j^2)[suma, i := suma + i * i, i + 1]$
$\equiv$
$N \geq 0 \wedge suma + i * i = \sum_{j=0}^{N-1} j^2$

Se aplica ahora el transformador wp al cuerpo del ciclo y al resultado anterior

$wp(suma, i := suma + i * i, i + 1, N \geq 0 \wedge suma + i * i = \sum_{j=0}^{N-1} j^2)$
$\equiv$
$(N \geq 0 \wedge suma+i^2 = \sum_{j=0}^{N-1} j^2)[suma, i := suma+i*i, i+1]$
$\equiv$
$N \geq 0 \wedge suma + i * i + (i + 1)^2 = \sum_{j=0}^{N-1} j^2$
$\equiv$
$N \geq 0 \wedge suma + \sum_{j=i}^{i+1} j^2 = \sum_{j=0}^{N-1} j^2$

Por inducción se puede demostrar que aplicar un número ϵ de veces, el transformador wp al cuerpo del ciclo y la postcondición se obtiene la siguiente fórmula:

$$N \geq 0 \wedge suma + \sum_{j=i}^{i+\epsilon-1} j^2 = \sum_{j=0}^{N-1} j^2$$

Es facil demostrar despejando la variable $suma$, que la fórmula es siempre satisfacible para cualquier valor de ϵ, por lo tanto el predicado

$(N \geq 0 \wedge suma + \sum_{j=i}^{i+\epsilon-1} j^2 = \sum_{j=0}^{N-1} j^2)[\epsilon := N - i]$
$\equiv$
$N \geq 0 \wedge suma + \sum_{j=i}^{i+N-i-1} j^2 = \sum_{j=0}^{N-1} j^2$
$\equiv$
$N \geq 0 \wedge suma + \sum_{j=i}^{N-1} j^2 = \sum_{j=0}^{N-1} j^2$

Satisface las hipótesis del caso 1 del teorema 2 para cualquier valor de k y por lo tanto la precondición más débil de este ejemplo es $i \leq N \wedge N \geq 0 \wedge suma + \sum_{j=i}^{N-1} j^2 = \sum_{j=0}^{N-1} j^2$

Esta precondición puede refinarse aún más haciendo lo siguiente:

$i \leq N \wedge N \geq 0 \wedge suma + \sum_{j=i}^{N-1} j^2 = \sum_{j=0}^{N-1} j^2$
$\equiv$
$i \leq N \wedge N \geq 0 \wedge suma = \sum_{j=0}^{N-1} j^2 - \sum_{j=i}^{N-1} j^2$
$\equiv$
$i \leq N \wedge N \geq 0 \wedge (i \geq 0 \Rightarrow suma = \sum_{j=0}^{i-1} j^2)$
$\wedge (i < 0 \Rightarrow suma = -\sum_{j=i}^{0} j^2)$
$\equiv$
$i \leq N \wedge N \geq 0 \wedge suma = \sum_{j=0}^{i-1} j^2 - \sum_{j=i}^{0} j^2$

Esta precondición dice que el ciclo puede empezar desde un índice i con valor negativo, pero para eso la variable sum debe ser inicializada en el ciclo con el valor negativo

$$-\sum_{j=i}^{0} j^2,$$

para que en cada iteración le sea sumada una cantidad positiva, hasta que su valor llegue a ser positivo e igual al valor que indica la postcondición.

V. Teoremas Complementarios

A continuación se presenta un teorema y corolario cuyas demostraciones son análogas a las que se presentaron anteriormente.

Sea un algoritmo con la estructura

do $i \neq N \rightarrow$
$S_0;$
$i := i - 1$
od
$\{Post\},$

donde las variables i y N no son modificadas en S_0.

Teorema 4. *Sea un algoritmo con la estructura que se presentó arriba y k, k' variables que no ocurren en S_0 y $Post$, entonces:*
si existe un predicado inv tal que $inv[i := N] \equiv Post$ y $(\forall i | N < i \leq N + k : wp(S_0; i := i - 1, inv) \equiv inv)$, donde las variables k, k' no ocurren en inv, entonces

$$H_{k'}(Post) \equiv N \leq i \leq N + k' \wedge inv$$

para todo k' tal que $0 \leq k' \leq k$.

Adicionalmente si $wp(S_0; i := i-1, inv) \equiv False$ *cuando* $i = N + (k+1)$*, entonces*

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

Corolario 3. *Si un predicado* inv *cumple con las hipótesis del caso 1 del teorema 4 para todo* k *y no cumple la hipótesis del caso 2 del mismo teorema, entonces*

$$wp(do\ B_0 \to S_0; i := i-1\ od, Post) \equiv N \leq i \wedge inv$$

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

$$wp(do\ B_0 \to S_0; i := i+1\ od, Post) \equiv N \leq i \leq N+k \wedge inv$$

Definición. *La función rampa se define de la siguiente manera:* $r^{(N)}(i) = \begin{cases} i & si \quad i \leq N \\ N & si \quad i > N \end{cases}$

La guardia de todos los ciclos que se han estudiado hasta ahora es $i \neq N$, si la intercambiamos por $i < N$, la precondición más débil es ligeramente distinta. El siguiente teorema proporciona un resultado al respecto.

Teorema 5. *Sea el algoritmo con la estructura*

do $i < N \to$
  S_0;
  $i := i + 1$
od
$\{Post\}$,

donde la variable i *y* N *no son modificadas en* S_0.
Si k, k' *son variables que no ocurren en* S_0 *y* $Post$*, entonces:*

1) *si existe un predicado* inv *tal que* $inv[i := N] \equiv Post$ *y* $(\forall i | N - k \leq i < N : wp(S_0; i := i+1, inv) \equiv inv)$*, donde las variables* k, k' *no ocurren en* inv*, entonces*

$$H_{k'}(Post) \equiv N - k' \leq i \wedge inv[i := r^{(N)}(i)]$$

   para todo k' *tal que* $0 \leq k' \leq k$.
2) *Adicionalmente si* $wp(S_0; i := i+1, inv) \equiv False$ *cuando* $i = N - (k+1)$*, entonces*

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

Demostración: Se demuestra por inducción sobre k' suponiendo que $k' \leq k$ y que k' y k son variables que no ocurren en inv, S_0 y $Post$.

Caso 1 $k' = 0$

$H_{k'}(Post)$
$\equiv< k' = 0 >$
$H_0(Post)$
$\equiv$
$i \geq N \wedge Post$
$\equiv$
$(i = N \vee i > N) \wedge Post$
$\equiv$
$(i = N \wedge Post) \vee (i > N \wedge Post)$

$\equiv$<hipótesis>

$(N \leq i \leq N \wedge inv[i := N]) \vee (i > N \wedge Post)$

$\equiv$<hipótesis>

$(N \leq i \leq N \wedge inv[i := N]) \vee (i > N \wedge inv[i := N])$

$\equiv< r^{(N)}(i) = N$ por definición>

$(N \leq i \leq N \wedge inv[i := r^{(N)}(i)]) \vee$
$(i > N \wedge inv[i := r^{(N)}(i)])$
$\equiv< k' = 0 >$
$(N - k' \leq i \leq N \wedge inv[i := r^{(N)}(i)]) \vee$
$(i > N \wedge inv[i := r^{(N)}(i)])$
$\equiv$
$(N - k' \leq i \leq N \vee i > N) \wedge inv[i := r^{(N)}(i)]$
$\equiv$
$N - k' \leq i \wedge inv[i := r^{(N)}(i)]$

A continuación se supone que el teorema es cierto para $k' - 1$ y se demuestra para k'

$H_{k'}(Post)$
$\equiv$
$H_0(Post) \vee (i < N \wedge wp(S_0; i := i+1, H_{k'-1}(Post)))$

$\equiv$<hipótesis inductiva>

$H_0(Post) \vee (i < N \wedge wp(S_0; i := i+1,$
$N - (k'-1) \leq i \wedge inv[i := r^{(N)}(i)]))$
$\equiv$
$H_0(Post) \vee (i < N \wedge wp(S_0; i := i+1,$
$N - (k'-1) \leq i) \wedge wp(S_0; i := i+1, inv[i := r^{(N)}(i)]))$

$\equiv< i$ y N no se modifican en S_0 y k' no ocurre en $S_0 >$

$H_0(Post) \vee (i < N \wedge N - (k'-1) \leq i+1 \wedge$
$wp(S_0; i := i+1, inv[i := r^{(N)}(i)]))$
$\equiv$
$H_0(Post) \vee (i < N \wedge N - k' \leq i \wedge$
$wp(S_0; i := i+1, inv[i := r^{(N)}(i)]))$

$\equiv< r^{(N)}(i) = i$ por definición>

$H_0(Post) \vee (i < N \wedge N - k' \leq i \wedge wp(S_0; i := i+1, inv))$

$\equiv$<hipótesis>

$H_0(Post) \vee (i < N \wedge N - k' \leq i \wedge inv)$

$\equiv< r^{(N)}(i) = i$ por definición>

$H_0(Post) \vee (i < N \wedge N - k' \leq i \wedge inv[i := r^{(N)}(i)])$
$\equiv$

$(i \geq N \wedge Post) \vee (i < N \wedge N - k' \leq i \wedge inv[i := r^{(N)}(i)])$

$\equiv$<hipótesis>

$(i \geq N \wedge inv[i := N]) \vee (i < N \wedge N - k' \leq i \wedge$
$inv[i := r^{(N)}(i)])$

$\equiv< r^{(N)}(i) = N$ cuando $i \geq N >$

$(i \geq N \wedge inv[i := r^{(N)}(i)]) \vee (i < N \wedge N - k' \leq i \wedge$
$inv[i := r^{(N)}(i)])$
$\equiv$
$(i \geq N \vee (i < N \wedge N - k' \leq i)) \wedge inv[i := r^{(N)}(i)]$
$\equiv$
$(N - k' \leq i) \wedge inv[i := r^{(N)}(i)]$

Si $wp(S_0; i := i+1, inv) \equiv False$ cuando $i = N-(k+1)$ entonces

$H_{k+1}(Post)$
$\equiv$
$H_0(Post) \vee (i < N \wedge wp(S_0; i := i+1, H_k(Post)))$

$\equiv$<caso 1 del teorema>

$H_0(Post) \vee (i < N \wedge wp(S_0; i := i+1, N - k \leq i \wedge$
$inv[i := r^{(N)}(i)]))$
$\equiv$
$H_0(Post) \vee (i < N \wedge wp(S_0; i := i+1, N - k \leq i) \wedge$
$wp(S_0; i := i+1, inv[i := r^{(N)}(i)]))$

$\equiv< i$ y N no se modifican en S_0 y k no ocurre en $S_0 >$

$H_0(Post) \vee (i < N \wedge N - k \leq i+1 \wedge$
$wp(S_0; i := i+1, inv[i := r^{(N)}(i)]))$
$\equiv$
$H_0(Post) \vee (i < N \wedge N - k - 1 \leq i \wedge$
$wp(S_0; i := i+1, inv[i := r^{(N)}(i)]))$
$\equiv$
$H_0(Post) \vee (i < N \wedge N - (k+1) \leq i \wedge$
$wp(S_0; i := i+1, inv[i := r^{(N)}(i)]))$
$\equiv$
$H_0(Post) \vee (i < N \wedge (i = N - (k+1) \vee N - k \leq i) \wedge$
$wp(S_0; i := i+1, inv[i := r^{(N)}(i)]))$
$\equiv$
$H_0(Post) \vee (((i < N \wedge i = N - (k+1)) \vee$
$(i < N \wedge N - k \leq i)) \wedge wp(S_0; i := i+1, inv[i := r^{(N)}(i)]))$
$\equiv$
$H_0(Post) \vee (i < N \wedge i = N - (k+1) \wedge$
$wp(S_0; i := i+1, inv[i := r^{(N)}(i)])) \vee$
$(i < N \wedge N - k \leq i \wedge wp(S_0; i := i+1, inv[i := r^{(N)}(i)]))$

$\equiv< r^{(N)}(i) = i$ cuando $i < N >$

$H_0(Post) \vee (i < N \wedge i = N - (k+1) \wedge$
$wp(S_0; i := i+1, inv)) \vee$
$(i < N \wedge N - k \leq i \wedge wp(S_0; i := i+1, inv))$

$\equiv$<hipótesis>

$H_0(Post) \vee (i < N \wedge i = N - (k+1) \wedge False) \vee$
$(i < N \wedge N - k \leq i \wedge inv)$
$\equiv$
$H_0(Post) \vee False \vee (i < N \wedge N - k \leq i \wedge inv)$
$\equiv$
$H_0(Post) \vee (i < N \wedge N - k \leq i \wedge inv)$
$\equiv$
$(i \geq N \wedge Post) \vee (i < N \wedge N - k \leq i \wedge inv)$

$\equiv$<hipótesis>

$(i \geq N \wedge inv[i := N]) \vee (i < N \wedge N - k \leq i \wedge inv)$

$\equiv< r^{(N)}(i) = N$ si $i \geq N$ y $r^{(N)}(i) = i$ si $i < N \quad >$

$(i \geq N \wedge inv[i := r^{(N)}(i)]) \vee$
$(i < N \wedge N - k \leq i \wedge inv[i := r^{(N)}(i)])$
$\equiv$
$(i \geq N \vee (i < N \wedge N - k \leq i)) \wedge inv[i := r^{(N)}(i)]$
$\equiv$
$N - k \leq i \wedge inv[i := r^{(N)}(i)]$
$\equiv$
$H_k(Post)$ ■

La demostración del siguiente corolario es análoga a la del corolario 2.

Corolario 4. *Si un predicado inv cumple con las hipótesis del caso 1 del teorema 5 para todo k y no cumple con la hipótesis del caso 2 del mismo teorema, entonces*

$$wp(do\ B_0 \rightarrow S_0; i := i+1\ od, Post) \equiv$$

$$inv[i := r^{(N)}(i)]$$

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

$$wp(do\ B_0 \rightarrow S_0; i := i+1\ od, Post) \equiv$$

$$N - k \leq i \wedge inv[i := r^{(N)}(i)]$$

Nota. *Como las hipótesis de los casos 1 y 2 del teorema 5 son las mismas que la de los casos 1 y 2 del teorema 2, y para el caso del ejemplo de factorial, el predicado*

$$N > 0 \wedge fact = (i-1)!$$

satisfacía dichas hipótesis, entonces según el corolario 4 se tiene que para el mismo algoritmo pero sustituyendo la guardia por $i < N$, la precondición más débil del ciclo sería

$$1 \leq i \wedge fact = (r^{(N)}(i) - 1)!$$

VI. Conclusiones

Los teoremas aquí presentados son un pequeño aporte para el desarrollo de un cálculo efectivo para la corrección de programas. Desarrollar técnicas de cálculo que simplifiquen el trabajo en el área de corrección de algoritmos, es una tarea difícil. Sin embargo, los ejemplos aquí presentados muestran que usando los teoremas adecuados, es posible conseguir la precondición más débil de ciertas instrucciones Do de una forma rápida y formal. Por esta razón, si los esfuerzos de los investigadores se concentraran en desarrollar teoremas como estos, la corrección de programas pasaría a ser en la práctica, un tema de interés a nivel de ingeniería, que impulsaría a la ciencia de la computación a niveles de exactitud y robustez mucho mayores de los de hoy en día.

Referencias

[1] S. Magill, A. Nanevski, E. Clarke, and P. Lee. *Inferring invariants in separation logic for imperative list-processing programs*. SPACE, 1(1):57, 2006.

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

[3] C. Varming and L. Birkedal. *Higher-order separation logic in isabelle/holcf*. Electronic Notes in Theoretical Computer Science, 218:371389, 2008

[4] C. A. R. Hoare. *An axiomatic basis for computer programming*. Communications of the ACM, 12(10):576580, 1969.

[5] E. W. Dijkstra. *Guarded commands, nondeterminacy and formal derivation of programs*. Communications of the ACM, 18(8):453457, 1975.

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

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

[8] F. Flaviani. *Modelo Relacional de la Teoría Axiomática del Lenguaje GCL de Dijkstra*. CONCISA Valencia, Venezuela, 2015

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

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