Fecha de recepción 16/01/2019
Fecha de aceptación: 25/ 03 /2020
Pp 65 – 80

Revisita a la Técnica de Derivación de Invariantes "Reemplazo de Constante por Variable"

Federico Flaviani[1], Soraya Carrasquel[1], David Coronado[1]
fflaviani@usb.ve, scarrasquel@usb.ve, dcoronado@usb.ve.
Departamento de Computación y Tecnología de la Información
Universidad Simón Bolívar, Caracas, Venezuela

Resumen: Dijkstra definió el transformador de predicados wp (*weakest precondition*), y en dicha definición establece para la instrucción de iteración Do un predicado H_k(Post) que describe los estados iniciales que hacen que el ciclo itere a lo sumo k veces, satisfaciendo la postcondición Post. En trabajos recientes se han determinado técnicas para calcular explícitamente H_k(Post), lo cual representa una alternativa a la regla de Hoare del invariante. En este trabajo se demuestra que la técnica de derivación de algoritmos de Dijkstra denominada "reemplazo de constante por variable", genera invariantes que son equivalentes a H_k(Post) para algún k, cuando Post satisface ciertas condiciones.

Palabras Clave: Corrección formal de programas, derivación de algoritmos, invariantes.

Abstract: Dijkstra defined the predicate transformer called wp (weakest precondition), and in that definition it establishes for the iteration statement Do, the predicate H_k(Post), that describes the initial states that cause the loop to iterate at most k times, with a final state which satisfy the post-condition Post. In recent works, techniques have been determined to calculate explicitly H_k(Post), which represents an alternative to the Hoare rule of the invariant. In this work it is demonstrated that the invariants derivation technique, that Dijkstra called "replacement of constant by variable", generates invariants that are equivalent to H_k(Post) for some k, when Post satisfies certain conditions.

Keywords: Formal Program Verification, Derivation of Algorithms, Invariants.

I. Introducción

Al analizar algoritmos que contienen una instrucción de iteración, es común la aparición de invariantes. En **[1]** se presenta la técnica de reemplazo de constante por variable para calcular dicha invariante. En este trabajo se realiza un nuevo análisis de esta técnica relacionado con los predicados H_k(Post) de **[2]**.

Para unificar la presentación de los algoritmos en este trabajo, todos los algoritmos serán escritos en pseudolenguaje *GCL* (Guarded Command Language) **[2]**, que es un pseudolenguaje definido por Dijkstra, donde se pueden escribir algoritmos no determinísticos. *GCL* admite una Lógica de Hoare **[3]** y fórmulas para precondiciones más débiles, relativamente simples, que facilitan la actividad de corrección de un programa.

Si Pre y Post son predicados para ser usados como precondición y postcondición de una instrucción S, entonces la Lógica de Hoare **[3]** se basa en la noción de tripletas {Pre} S {Post}, que se entienden como un predicado que es verdadero si y sólo si la instrucción manda los estados del programa que satisfacen Pre a estados que satisfacen Post. Para hacer derivaciones lógicas sobre estas tripletas, se definen para el lenguaje *GCL*, las siguientes reglas de inferencia, en donde $B_0, \dots, B_n$ son expresiones booleanas del lenguaje *GCL* y *DG* es una abreviación de domain$(B_0, \dots, B_n)$ (predicado que

describe los estados en los cuales las expresiones están bien definidas [4]):

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

$$\{\text{domain}(\overline{\text{Exp}}) \wedge B[\overline{y} := \overline{\text{Exp}}]\}\overline{y} := \overline{\text{Exp}}\{B\}$$

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

$$\frac{\begin{array}{lll} A \Rightarrow DG & \{A \wedge B_0\} & \ldots \{A \wedge B_n\} \\ \wedge (B_0 \vee \cdots \vee B_n) & S_0 & S_n \\ & \{B\} \ldots & \{B\} \end{array}}{\{A\} \text{ if } B_0 \to S_0[\ ] \cdots [\ ] B_n \to S_n \text{ fi } \{B\}}$$

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

$$\frac{\text{Inv} \Rightarrow \text{domain}(B_0) \qquad \{\text{Inv} \wedge B_0\}\, S_0\, \{\text{Inv}\}}{\{\text{Inv}\}\ \text{do}\ B_0 \to S_0\ \text{od}\ \{\text{Inv} \wedge \neg B_0\}}$$

El predicado Inv de la última regla de inferencia se conoce como invariante, y éste se usa para verificar la correctitud de una Tripleta de Hoare que involucre un ciclo.

En este trabajo si se abrevian a S_i con $0 \leq i \leq n$ como instrucciones, entonces se denota a IF como la instrucción de selección

if $B_0 \to$
$\quad S_0$
$[\ ]\ B_1 \to$
$\quad S_1$
$\quad \vdots$
$[\ ]\ B_n \to$
$\quad S_n$
fi

Y la abreviación DO como la instrucción de iteración con múltiples guardias

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

Se denotará como Do a la instrucción de iteración con una sola guardia **do** $B_0 \to S_0$ **od**.

Dijkstra adicionalmente en **[2]** definió el transformador de predicados wp para el lenguaje *GCL*, que es una función que recibe una instrucción y una postcodición sintácticamente hablando y devuelve la precondición más débil de la instrucción y la postcondición. Si BB es una abreviación del predicado $B_0 \vee \ldots \vee B_n$, entonces las reglas que definen wp son las siguientes:

- $\text{wp}(\text{SKIP}, \text{Post}) := \text{Post}$
- $\text{wp}(y_{i_1}, \ldots, y_{i_k} := \text{Exp}_1, \ldots, \text{Exp}_k, \text{Post}) :=$ $\text{domain}(\text{Exp}_1, \ldots, \text{Exp}_k) \wedge$ $\text{Post}[y_{i_1}, \ldots, y_{i_k} := \text{Exp}_1, \ldots, \text{Exp}_k]$
- $\text{wp}(S_0; S_1, \text{Post}) := \text{wp}(S_0, \text{wp}(S_1, \text{Post}))$
- $\text{wp}(\text{IF}, \text{Post}) := \text{domain}(B_0, \ldots, B_n) \wedge$ $(B_0 \vee \cdots \vee B_n) \wedge (B_0 \Rightarrow \text{wp}(S_0, \text{Post})) \wedge \cdots$ $\wedge (B_n \Rightarrow \text{wp}(S_n, \text{Post}))$
- $\text{wp}(DO, \text{Post}) := (\exists k \,|\, k \geq 0 : H_k(\text{Post}))$ en donde $H_k(\text{Post})$ es un predicado que satisface las ecuaciones

  $$H_0(\text{Post}) \equiv \text{domain}(BB) \wedge \neg(BB) \wedge \text{Post}$$

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

  Para $k \geq 1$.

El predicado $H_k(\text{Post})$ en la definición de $\text{wp}(DO, \text{Post})$ anterior, describe el conjunto de estados que hacen que el ciclo DO itere a lo sumo k veces satisfaciendo la postcondición Post al final de la ejecución.

Trabajar con $H_k(\text{Post})$ como precondición de un ciclo, tiene la ventaja de que la expresión k determina el número máximo de iteraciones que el ciclo puede realizar, por lo tanto, se puede estimar la complejidad en tiempo de ejecución. Por otro lado intentar usar $H_k(\text{Post})$ en lugar de cualquier otro invariante puede ser conveniente ya que en los trabajos **[5] [6]**, se ha venido planteando, que en algunos casos es muy fácil calcular $H_k(\text{Post})$ formalmente, sin necesidad de conjeturar un invariante y usar las reglas de Hoare para validar.

Por otro lado, Dijkstra **[1]**, estableció varias técnicas para derivar algoritmos, con el objetivo de que la corrección del algoritmo con respecto a cierta especificación, se realizara durante el proceso de construcción del algoritmo y no al final. Una de estas técnicas se llama "reemplazo de constante por variable" y sirve para construir instrucciones de iteración Do con un invariante

adecuado como precondición, para que satisfaga determinada postcondición.

Por ejemplo, en un programa donde están declaradas A, N, como constantes enteras y r, i, como variables enteras, si se toman como postcondición $r = A^N$, entonces esta técnica consiste en construir un invariante reemplazando una de las constantes de la postcondición por una de las variables del programa. Si se tiene el caso en el cual se reemplaza la constante N por i, entonces el invariante conjeturado sería $r = A^i$, el cual, si se usa para un ciclo de tipo for, con contador i, debe venir en conjunción con el predicado $0 \leq i \leq N$, para asegurar su terminación, concluyendo que el ciclo es de la forma:

$\{\text{Inv} : 0 \leq i \leq N \wedge r = A^i\}$

do $i \neq N \rightarrow$

S_0 ;

$i := i + 1$

od

$\{\text{Post} : r = A^N\}$

Para alguna instrucción S_0 que se determina posteriormente durante el proceso de derivación del algoritmo.

A. Contribución

En el presente trabajo se da una demostración, de que los invariantes obtenidos por la técnica de "reemplazo de constante por variable", son $H_k(\text{Post})$ para algún $k \geq 0$, cuando la postcondición cumple ciertas condiciones con respecto a sus variables de especificación. Adicionalmente, apoyándose del uso de instrucciones de especificación de Morgan **[7]**, se muestra un esquema general de como plantear un ciclo de tipo for y un invariante $H_k(\text{Post})$ con esta técnica.

Se revisita la instrucción de especificación, cuando el uso de las variables de especificación no tiene restricciones y se propone un teorema sobre el comportamiento de $\text{wp}([\text{Pdef}, \text{Qdef}], \text{Post})$, cuando Post es equivalente a Qdef.

Adicionalmente, para mostrar la utilidad de esta técnica con la forma general planteada, se muestra un ejemplo de aplicación para derivar el Algoritmo de eliminación gaussiana, para la triangulación superior de una matriz de números reales.

B. Trabajos Relacionados

Originalmente Dijkstra en **[2]** definió el transformador de predicados wp con las reglas recursivas anteriores, salvo que no usó el predicado domain. Posteriormente en **[4]**, buscando que las aserciones de *GCL* sean totalmente evaluables, es decir que no existan estados donde el valor de verdad de las aserciones sea indefinido, se definió por primera vez el predicado domain, pero sólo en la definición de wp de la asignación. Luego en trabajos recientes como **[8] [9]**, se usa domain también en la regla de definición de wp para el IF, y en **[10]** se hace una demostración basada en semántica denotacional, la cual justifica que para que las aserciones de *GCL* sean totalmente evaluables, se debe usar domain también en la regla del wp del DO al definir $H_0(\text{Post})$. De modo que la definición de wp que se usa y se presentó en este trabajo es la de **[10]**.

La técnica de reemplazo de constante por variable fue planteada originalmente por Dijkstra en **[1]**. Posteriormente, en la misma línea de usar procesos de diseño de algoritmos que aseguren correctitud por construcción, Morgan **[7]** define el concepto de refinamiento, instrucción de especificación y como calcular wp de esta instrucción bajo ciertas condiciones sintácticas sobre las variables de especificación de las aserciones. Posteriormente en **[11],** se revisita la definición de la instrucción de especificación desde un punto de vista denotacional, interpretándola como una relación de estados del programa y con eso se estableció una fórmula general para calcular wp de estas instrucciones.

En **[12]** se muestra una técnica semántica llamada relaciones invariantes, para calcular invariantes en el lenguaje de la teoría de relaciones y wp de un ciclo. Análogamente, pero con técnicas sintácticas, en **[5]** y posteriormente en **[6]**, se muestran teoremas que sirven para calcular $H_k(\text{Post})$ explícitamente y por lo tanto para calcular wp de Do con postcondición Post. Estos teoremas de **[6]**, como se muestra en el presente trabajo, traen como corolario el hecho de que el invariante obtenido por la técnica de reemplazo de constante por variable es un $H_k(\text{Post})$ para algún k. Esta prueba se hace en general para cualquier postcondición que cumplan ciertas condiciones sobre sus variables de especificación, sin embargo, ya existe un precedente en **[6]**, que mostró este hecho para un ejemplo en que se construye un algoritmo de ordenamiento con una postcondición sin variables de especificación.

II. Preliminares

En esta sección se enuncian las definiciones y teoremas más recientes, según la bibliografía de este trabajo, relacionados al cálculo de $H_k(\text{Post})$ de un ciclo.

Notación. *Para abreviar* $\text{wp}(S,\text{true})$ *se usará la notación* $\text{support}(S)$*, donde S es una instrucción.*

Así tenemos que $\text{support}(S)$ define el conjunto de estados iniciales más grande en el cual la instrucción S no aborta. En el siguiente lema se muestra un caso, en donde es útil usar support para calcular wp.

Lema 1. *Sea P un predicado y S una instrucción que no modifica los valores de las variables de P, entonces*

$$\mathrm{wp}(S, P) \equiv \mathrm{support}(S) \wedge P.$$

Definición 1. Sea K una expresión y Do una instrucción de la forma

do $B \rightarrow$
$\quad S$
od

{Post}

Sean además k', ϵ, ϵ' variables no declaradas en el programa (es decir no ocurren en Do) y no ocurren en Post*, donde S es una instrucción (determinística o no determinística). Se definen:*

1. *El predicado $domBG$ como un predicado que satisface:*
   - *En $domBG$ ocurren sólo ϵ' y las variables del programa*
   - $domBG[\epsilon' := 0] \equiv \mathrm{domain}(B)$
   - $domBG \equiv \mathrm{wp}(S, domBG[\epsilon' := \epsilon' - 1])$ *suponiendo que* $0 < \epsilon' \leq K \wedge \mathrm{domain}(B) \wedge B \wedge \mathrm{support}(S)$.
2. *El predicado NBG como un predicado que satisface:*
   - *En NBG ocurren sólo ϵ y las variables del programa*
   - $NBG[\epsilon := 0] \equiv \neg B$
   - $NBG \equiv \mathrm{wp}(S, NBG[\epsilon := \epsilon - 1])$ *suponiendo que* $0 < \epsilon \leq K \wedge \mathrm{domain}(B) \wedge B \wedge support(S)$.
3. *El predicado $T_{k'}$ como:*

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

4. *El predicado $TI_{k'}$ como: $T_{k'} \wedge B$.*

En el siguiente teorema se establecen las condiciones suficientes para determinar si un predicado es un $H_{k'}(\mathrm{Post})$, además establece una relación entre $H_{K+1}(\mathrm{Post})$ y $H_K(\mathrm{Post})$.

Teorema 1. *Sean K una expresión, Do una instrucción como en la Definición* 1 *y k', ϵ, ϵ' variables como en la Definición* 1*.*

Entonces, si S actúa de forma determinística sobre las variables de $domBG$ y NBG, se satisfacen las siguientes condiciones:

1. *Si existe un predicado* inv *tal que:*
   a) $\mathrm{domain}(B) \wedge \neg B \wedge \mathrm{Post} \equiv \mathrm{domain}(B) \wedge \neg B \wedge \mathrm{inv}$
   b) $\left(\forall k' \middle| 1 \leq k' \leq K : TI_{k'} \Rightarrow (\mathrm{wp}(S, \mathrm{inv}) \equiv \mathrm{inv})\right)$

   Entonces

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

   para todo k' tal que $0 \leq k' \leq K$.
2. *Adicionalmente a las hipótesis de 1, si la recurrencia que define a $domBG$ y NBG están definidas hasta $\epsilon, \epsilon' = K + 1$, entonces*

$$B \wedge \mathrm{wp}(S, \mathrm{inv}) \wedge (\forall \epsilon' | 0 \leq \epsilon' \leq K + 1 : domBG) \wedge NBG[\epsilon := K + 1] \Rightarrow T_K \quad (1)$$

   Sí y sólo si

$$H_{K+1}(\mathrm{Post}) \equiv H_K(\mathrm{Post}).$$

La demostración de este teorema se encuentra en **[6]**.

El predicado en 1.a) del teorema anterior se le llamará *Obligación de Prueba de Terminación* y al predicado en 1.b), *Obligación de Prueba de Iteración.*

El siguiente corolario muestra como calcular $\mathrm{wp}(\mathrm{Do}, \mathrm{Post})$ y $H_K(\mathrm{Post})$ para un caso particular.

Corolario 1. *Sea un ciclo Do como en la Definición* 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 i ni N. Entonces:*

1. *Si $C \equiv true$, entonces*

$$T_{k'} \equiv N - k' \leq i \leq N \text{ y } TI_{k'} \equiv N - k' \leq i < N.$$

2. *Si* $\mathrm{domain}(C) \equiv a \leq i < N$ *ó si* $\mathrm{domain}(C) \equiv a \leq i \leq N$ *con a constante, entonces*

$$T_{N-a} \equiv a \leq i \leq N \text{ y } TI_{N-a} \equiv a \leq i < N \wedge C$$

$$y\ \mathrm{wp}(\mathrm{Do}, \mathrm{Post}) \equiv H_{N-a}(\mathrm{Post}).$$

III. INSTRUCCIÓN DE ESPECIFICACIÓN

Las especificaciones al estilo Tripletas de Hoare, pueden ser usadas para hacer abstracción de código.

Definición 2. *Si la lista de identificadores declarados en un algoritmo es $\overline{x}$, $\overline{y}$, entonces la nomenclatura $\overline{y}$: $\left[\mathrm{Pdef}(\overline{x}, \overline{y}, \overline{Z}),\ \mathrm{Qdef}(\overline{x}, \overline{y}, \overline{Z})\right]$ se considera como una abstracción de código, que corresponde a la instrucción más general que satisface una especificación que consiste en; un par de predicados* Pdef *y* Qdef*, llamados precondición y postcondición, un conjunto de variables de especificación $\overline{Z}$ (no declaradas en el programa), un conjunto de identificadores $\overline{y}$, considerados variables en el trozo de código que se está abstrayendo, y el resto de los identificadores $\overline{x}$, se consideran constantes dentro del trozo de programa abstraído. Estas abstracciones de código se denominan, instrucciones de especificación.*

Se puede calcular wp de una instrucción de especificación usando los siguientes teoremas.

Teorema 2. *Sea un algoritmo donde $\overline{x},\overline{y}$ es la lista de identificadores declarados. Si* $\mathrm{Pdef}(\overline{x},\overline{y},\overline{Z})$, $\mathrm{Qdef}(\overline{x},\overline{y},\overline{Z})$, $\mathrm{Post}(\overline{x},\overline{y},\overline{X})$ *son predicados, $\overline{T'}$ es la lista de tipos de las variables $\overline{y}$ y $\overline{T''}$ es la lista de tipos de las variables $\overline{Z}$ y $\overline{X}$, entonces una precondición más débil para la instrucción de especificación* $\overline{y}:[\mathrm{Pdef}(\overline{x},\overline{y},\overline{Z}),\mathrm{Qdef}(\overline{x},\overline{y},\overline{Z})]$ *y* $\mathrm{Post}(\overline{x},\overline{y},\overline{X})$ *es*

$$\Big(\exists\overline{y'},\overline{Z}\Big|\overline{y'}\in\overline{T'}\wedge\overline{Z}\in\overline{T''}:\mathrm{Pdef}(\overline{x},\overline{y},\overline{Z})\wedge\mathrm{Qdef}(\overline{x},\overline{y'},\overline{Z})\Big)\wedge\Big(\forall\overline{y'}\,|\,\overline{y'}\in\overline{T'}:\Big(\exists\overline{Z}\Big|\overline{Z}\in\overline{T''}:\mathrm{Pdef}(\overline{x},\overline{y},\overline{Z})\wedge\mathrm{Qdef}(\overline{x},\overline{y'},\overline{Z})\Big)\Rightarrow\mathrm{Post}(\overline{x},\overline{y'},\overline{X})\Big).$$

Corolario 2. *Sean las listas $\overline{x},\overline{y},\overline{Z}$ y predicados* $\mathrm{Pdef}(\overline{x},\overline{y},\overline{Z})$, $\mathrm{Qdef}(\overline{x},\overline{y},\overline{Z})$ *como en la Definición 2, tales que para todo $\overline{x},\overline{y}$ en el que se satisface* $\mathrm{Pdef}(\overline{x},\overline{y},\overline{y})$ *existe $\overline{y'}$ tal que* $\mathrm{Qdef}(\overline{x},\overline{y'},\overline{y})$*. Si las variables libres $\overline{Z}$ son variables de especificación tales que existe un predicado* $\mathrm{Pdef}'(\overline{x},\overline{y})$ *en el que no ocurre $\overline{Z}$, y* $\mathrm{Pdef}(\overline{x},\overline{y},\overline{Z})$ *es de la forma* $\mathrm{Pdef}'(\overline{x},\overline{y})\wedge\overline{y}=\overline{Z}$*, entonces una precondición más débil para la instrucción de especificación*

$$\overline{y}:[\mathrm{Pdef}(\overline{x},\overline{y},\overline{Z}),\mathrm{Qdef}(\overline{x},\overline{y},\overline{Z})]\ \text{y}\ \mathrm{Post}(\overline{x},\overline{y},\overline{X})$$

Es:

$$\mathrm{Pdef}(\overline{x},\overline{y},\overline{y})\wedge\Big(\forall\overline{y'}\,|\,y'\in\overline{T'}:\mathrm{Qdef}(\overline{x},\overline{y'},\overline{y})\Rightarrow\mathrm{Post}(\overline{x},\overline{y'},\overline{X})\Big).$$

Las demostraciones del teorema y corolarios anteriores se encuentran en **[11]**.

Teorema 3. *Sean* $\mathrm{Pdef}(\overline{x},\overline{y},\overline{Y})$ *y* $\mathrm{Qdef}(\overline{x},\overline{y},\overline{X})$ *predicados tal que* $\mathrm{Pdef}(\overline{x},\overline{y},\overline{Y})\Rightarrow P$ *donde P es un predicado en donde no ocurren las variables $\overline{y}$, $\overline{Y}$. Entonces la instrucción de especificación* $\overline{y}:[\mathrm{Pdef}(\overline{x},\overline{y},\overline{Y}),\mathrm{Qdef}(\overline{x},\overline{y},\overline{Y})]$ *(*) es la misma que la instrucción* $\overline{y}:[\mathrm{Pdef}(\overline{x},\overline{y},\overline{Y}),P\wedge\mathrm{Qdef}(\overline{x},\overline{y},\overline{Y})]$ *(**).*

Demostración: La demostración se hace verificando que las dos instrucciones de especificación (*) y (**), verifican que $\mathrm{wp}\big((*),\mathrm{Post}(\overline{x},\overline{y},\overline{X})\big)\equiv\mathrm{wp}\big((**),\mathrm{Post}(\overline{x},\overline{y},\overline{X})\big)$, para cualquier postcondición arbitraria $\mathrm{Post}(\overline{x},\overline{y},\overline{X})$. ■

Teorema 4. *Sean los predicados* $\mathrm{Pdef}(\overline{x},\overline{y},\overline{Z})$ *y* $\mathrm{Qdef}(\overline{x},\overline{y},\overline{Z})$*. Si se denota*

$$Dom_{\overline{x},\overline{W}}\coloneqq\{\overline{y}\in\overline{T'}\,|\,\mathrm{Pdef}(\overline{x},\overline{y},\overline{W})\}$$

$$Rg_{\overline{x},\overline{W}}\coloneqq\{\overline{y}\in\overline{T'}\,|\,\mathrm{Qdef}(\overline{x},\overline{y},\overline{W})\}$$

se tiene que

$$\Big(\forall\overline{x}\,|\,\overline{x}\in\overline{T}:\big(\forall\overline{Z}\,|\,\overline{Z}\in\overline{T''}\wedge Dom_{\overline{x},\overline{Z}}\neq\emptyset:Rg_{\overline{x},\overline{Z}}\neq\emptyset\big)\wedge\Big(\forall\overline{Z_0}\,|\,\overline{Z_0}\in\overline{T''}\wedge Dom_{\overline{x},\overline{Z_0}}=\emptyset\wedge Rg_{\overline{x},\overline{Z_0}}\neq\emptyset:\big(\forall\overline{Z_1}\,|\,\overline{Z_1}\in\overline{T''}\wedge Dom_{\overline{x},\overline{Z_1}}\neq\emptyset:Rg_{\overline{x},\overline{Z_1}}\not\subseteq Rg_{\overline{x},\overline{Z_0}}\big)\Big)\wedge\Bigg(\forall\overline{Z_0},\overline{Z_1}\,|\,\overline{Z_0},\overline{Z_1}\in\overline{T''}\wedge Dom_{\overline{x},\overline{Z_0}}\neq\emptyset\wedge Dom_{\overline{x},\overline{Z_1}}\neq\emptyset:\begin{pmatrix}Dom_{\overline{x},\overline{Z_0}}\cap Dom_{\overline{x},\overline{Z_1}}=\emptyset\\\wedge\\Rg_{\overline{x},\overline{Z_0}}\not\subseteq Rg_{\overline{x},\overline{Z_1}}\wedge Rg_{\overline{x},\overline{Z_1}}\not\subseteq Rg_{\overline{x},\overline{Z_0}}\end{pmatrix}\vee\Big(Dom_{\overline{x},\overline{Z_0}}=Dom_{\overline{x},\overline{Z_1}}\wedge Rg_{\overline{x},\overline{Z_0}}=Rg_{\overline{x},\overline{Z_1}}\Big)\Bigg)$$

),(*)

Sí y sólo si

$$\mathrm{wp}\Big(\overline{y}:[\mathrm{Pdef},\ \mathrm{Qdef}],\mathrm{Qdef}(\overline{x},\overline{y},\overline{X})\Big)\equiv\mathrm{Pdef}(\overline{x},\overline{y},\overline{X}).$$

Observación 1. Si existe instrucción S que satisface la tripleta $\{\mathrm{Pdef}\}\,S\,\{\mathrm{Qdef}\}$, entonces es cierto en (*), la hipótesis $\Big(\forall\overline{x}\,|\,\overline{x}\in\overline{T}:\big(\forall\overline{Z}\,|\,\overline{Z}\in\overline{T''}\wedge Dom_{\overline{x},\overline{Z}}\neq\emptyset:Rg_{\overline{x},\overline{Z}}\neq\emptyset\big)\Big)$. Si en Pdef y Qdef, las variables $\overline{\mathrm{Z}}$ ocurren solamente en un predicado $\overline{y'}\approx f(\overline{Z})$, con ≈ relación de equivalencia, $\overline{y'}$ sublista de $\overline{y}$ y f función que admite inversa, entonces se cumplen el resto de las hipótesis en (*).

IV. Técnica de Reemplazo de Constante por Variable

El Corolario 1 sugiere una justificación de la técnica clásica de derivación de invariantes, denominada *reemplazo de constantes por variable*. Esta técnica consiste en sustituir una constante N de la postcondición por una variable fresca i y usar este nuevo predicado como invariante de un ciclo Do con guardia $i\neq N$ e incremento de i de uno en uno.

Este hecho se generaliza en el siguiente teorema el cuál es el pilar fundamental del trabajo.

Teorema 5. *Sea un predicado* Post *que depende de una constante declarada N y K es una expresión que depende de las variables del programa, si P es el predicado* Post *en el que se reemplazan algunas de las ocurrencias de N por i. Si los predicados $TI_K\wedge P, P[i:=i+1]$, y la lista de variables $\overline{y}$ cumplen con las hipótesis del Teorema 4, entonces la siguiente Tripleta de Hoare es verdadera e* Inv *es equivalente a* $H_K(\mathrm{Post})$*:*

$\{\text{Inv} : T_K \wedge P\}$

do $i \neq N \rightarrow$

$\quad \overline{y} : [TI_K \wedge P, P[i := i+1]];$

$\quad i := i+1$

od

$\{\text{Post}\}$

Donde en la lista $\overline{y}$ no se encuentran ni i ni N y los predicados T_K y TI_K se calculan según el Corolario 1 *como $N-K \leq i \leq N$ y $N-K \leq i < N$ respectivamente.*

Demostración: En este trabajo, si un predicado es redundante, se indicará tachando el mismo con una línea transversal.

La demostración consiste en tomar inv como P, y mostrar que se cumplen las obligaciones de prueba de terminación e iteración del Teorema 1. Se empieza por la obligación de prueba de terminación, para esto se considera que z es una variable fresca y que Post' es el predicado Post en el que se reemplazan por z las mismas ocurrencias de N en Post que fueron reemplazadas por i en P, es decir que $\text{Post}'[z := N]$ es Post y $\text{Post}'[z := i]$ es P.

$\cancel{\text{domain}(B)} \wedge \neg B \wedge \text{Post}$

$\equiv$

$\text{i} = \text{N} \wedge \text{Post}'[\text{z} := \text{N}]$

$\equiv$ $\quad \langle$reemplazo de iguales con $\text{i} = \text{N}\rangle$

$\text{i} = \text{N} \wedge \text{Post}'[\text{z} = \text{i}]$

$\equiv$

$true \wedge i = N \wedge P$

$\equiv$

$\text{domain}(\text{B}) \wedge \neg \text{B} \wedge \text{P}$

Para demostrar la obligación de prueba de iteración se supone como hipótesis $TI_{k'}$ con $1 \leq \text{k}' \leq \text{N}$ y se hace lo siguiente:

$$\text{wp}\left(\overline{y} : \begin{bmatrix} TI_K \\ \wedge \\ P \end{bmatrix}, \; P[i := i+1]\right]; \; i = i+1, P\right)$$

$\equiv$

$$\text{wp}\left(\overline{y} : \begin{bmatrix} TI_K \\ \wedge \\ P \end{bmatrix}, \; P[i := i+1]\right], \text{wp}(i = i+1, P)\right)$$

$\equiv$

$$\text{wp}\left(\overline{y} : \begin{bmatrix} TI_K \\ \wedge \\ P \end{bmatrix}, \; P[i := i+1]\right], P[i = i+1]\right)$$

$\equiv$ $\quad \langle$Teorema 4$\rangle$

$TI_K \wedge P$

$\equiv$ $\quad \langle$hipótesis $TI_{k'} \wedge 1 \leq k' \leq K \Rightarrow TI_K\rangle$

P

Como en la lista $\overline{y}$ no se encuentran i y N, entonces la instrucción $\overline{y} : [TI_K \wedge P, P[i := i+1]]$ no modifica ni a i ni a N y por lo tanto se cumplen las hipótesis del Corolario 1 y los predicados T_K y TI_K son $N-K \leq i \leq N$ y $N-K \leq i < N$ respectivamente. ■

V. Preliminares del Álgebra Lineal

Las siguientes definiciones y teoremas serán utilizados en el siguiente capítulo, para simplificar la nomenclatura de las especificaciones y las pruebas de corrección de los algoritmos.

Definición 3. *Dado un entero positivo N y un matriz A de tamaño $N \times N$, para cada $0 \leq i,j \leq N$ se escribirá como $TS_{i,j}(A)$ el predicado*

$$\left(\forall i', j' \mid 0 \leq j' < i \wedge j' < i' < \#filas(A) : A[i', j'] = 0\right)$$
$$\wedge \left(\forall j' \mid i < j' < j : A[j', i] = 0\right)$$

Esta situación se muestra en la Figura 1*, en la cual, la matriz está representada por el cuadrado, x en una posición de la columna i, significa que esa posición contiene, posiblemente, un valor distinto de cero.*

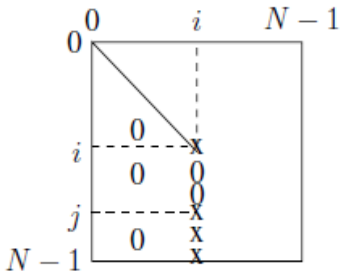

Figura 1. Matriz de tamaño $N \times N$ *que satisface* $TS_{i,j}(A)$. **Fuente:** Elaboración propia

Lema 2. *Dado un entero positivo N y un matriz A de tamaño $N \times N$, si i es un entero tal que $0 \leq i,j < N$, entonces se cumple*

1. $TS_{0,0}(A) \equiv true.$
2. $TS_{i,N}(A) \equiv TS_{i+1,i+1}(A).$

3. $TS_{i,i+1}(A) \equiv TS_{i,i}(A)$.
4. *Si* $i < j$ *entonces* $TS_{i,j+1}(A) \equiv TS_{i,j}(A) \wedge A[j,i] = 0$.

Demostración: Como $0 \leq i, j < N$, entonces los predicados TS de todos los items del lema estan bien definidos. Ahora se demuestra cada ítem.

1. Como $0 \leq j' < 0$ y $0 < j' < 0$ son falsos, entonces los rangos de los cuantificadores ∀ en $TS_{0,0}(A)$ son falsos y por lo tanto todo el predicado $TS_{0,0}(A)$ es verdadero.
2. El segundo ∀ en $TS_{i,N}(A)$ es $(\forall j' | i < j' < \#filas(A) : A[j', i] = 0)$ y en conjunción con el ∀ primero, $TS_{i,N}(A)$ sería equivalente a $(\forall i', j' | 0 \leq j' < i + 1 \wedge j' < i' < \#filas(A) : A[i', j'] = 0)$ que es equivalente a $TS_{i+1,i+1}(A)$.
3. Como no existe j' tal que $i < j' < i + 1$ y tampoco tal que $i < j' < i$, entonces los predicados
$$(\forall j' | i < j' < i + 1 : A[j', i] = 0)$$
$$(\forall j' | i < j' < i : A[j', i] = 0)$$
son ambos equivalentes a $true$ y por lo tanto $TS_{i,i+1}(A)$ y $TS_{i,i}(A)$ son equivalentes.
4. Como $0 \leq i < j < N$ se concluye que existe j' tal que $i < j' < j + 1$, por lo tanto
$$(\forall j' | i < j' < j + 1 : A[j', i] = 0)$$
$$\equiv$$
$$(\forall j' | i < j' < j : A[j', i] = 0) \wedge A[j, i] = 0$$
demostrando así el enunciado. ■

Con el fin de usar notación compacta en los invariantes, en las dos siguientes definiciones se introduce una nomenclatura para indicar intercambios y combinaciones lineales de las filas de una matriz, pero sólo entre las columnas i y k.

Definición 4. *Dados una matriz A de dimensiones* $N \times N$ *y enteros tales que* $0 \leq i, j < N$ *y* $0 \leq k \leq N$, *se define* $\text{swap}_{i,j}^{k}(A)$ *como la única matriz B que satisface* $(\forall k' | i \leq k' < k : B[i, k'] = A[j, k'] \wedge A[i, k'] = B[j, k'])$

$\wedge$

$$(\forall i', k' | 0 \leq k', i' < \# filas(A) \wedge i' \neq i \wedge i' \neq j : B[i', k'] = A[i', k'])$$

$\wedge$

$$(\forall k' | 0 \leq k' < i \vee k \leq k' < \# filas(A) : B[i, k'] = A[i, k'] \wedge B[j, k'] = A[j, k'])$$

$\wedge$

$$\dim(B) = \dim(A)$$

Un ejemplo de matriz swap se muestra en la Figura 2, se han intercambiado los valores en x de la matriz A por sus valores en *, dejando los demás sin modificar.

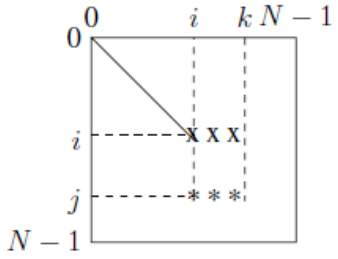

Figura 2: Matriz $swap_{i,j}^{k}(A)$. **Fuente**: Elaboración propia

Lema 3. *Dado un entero positivo* N *y una matriz* A *de tamaño* $N \times N$, *si* $0 \leq i, j, i', j' < N$ *y* $0 \leq k \leq N$, *entonces se cumple:*

1. *Si* $i < k$ *entonces* $\text{swap}_{i,j}^{k}(A)[j, i] = A[i, i]$.
2. *Si* $i < j' < j$ *entonces* $\text{swap}_{i,j}^{k}(A)[j', i] = A[j', i]$.
3. *Si* $j' < i$ *entonces* $\text{swap}_{i,j}^{k}(A)[i', j'] = A[i', j']$.
4. $\text{swap}_{i,j}^{i}(A) = A$.
5. *Si* $i < j \wedge i < k$ *entonces*
$$TS_{i,j+1}\left(\text{swap}_{i,j}^{k}(A)\right) \equiv TS_{i,j}(A) \wedge A[i, i] = 0.$$

Demostración: Como $0 \leq i, j, i', j' < N$ y $0 \leq k \leq N$, entonces los predicados TS y funcionales swap de todos los ítems del lema están bien definidos.

Las demostraciones de los ítems 1 a 4 son triviales.

5.

$$TS_{i,j+1}\left(\text{swap}_{i,j}^{k}(A)\right)$$
$$\equiv \quad \langle \text{item 4 del Lema 2} \rangle$$
$$TS_{i,j}\left(\text{swap}_{i,j}^{k}(A)\right) \wedge \text{swap}_{i,j}^{k}(A)[j, i] = 0$$
$$\equiv \quad \langle \text{item 1 de Lema 3} \rangle$$
$$TS_{i,j}\left(\text{swap}_{i,j}^{k}(A)\right) \wedge A[i, i] = 0$$
$$\equiv$$
$$(\forall i', j' | 0 \leq j' < i \wedge j' < i' < \# filas(A) : \text{swap}_{i,j}^{k}(A)[i', j'] = 0) \wedge (\forall j' | i < j' < j : \text{swap}_{i,j}^{k}(A)[j', i] = 0) \wedge A[i, i] = 0$$
$$\equiv \quad \langle \text{item 2 y 3 de Lema 3} \rangle$$

$$(\forall i', j' \mid 0 \le j' < i \wedge j' < i' < \#filas(A) : A[i', j'] = 0)$$
$$\wedge\ (\forall j' \mid i < j' < j :\ A[j', i] = 0) \wedge\ A[i, i] = 0$$

$\equiv$

$$TS_{i,j}(A) \wedge A[i, i] = 0 \quad \blacksquare$$

Definición 5. *Dados una matriz A de dimensiones $N \times N$ y enteros tales que $0 \le i, j < N$ y $0 \le k \le N$, entonces se define* $\mathrm{lcom}_{i,j}^{k}(A)$ *como la única matriz B que satisface*

$$\left(\forall k' \middle| i \le k' < k : B[j, k'] = -\frac{A[j, i]}{A[i, i]} A[i, k'] + A[j, k']\right)$$

$\wedge$

$$(\forall i', k' | 0 \le k', i' < \#filas(A) \wedge i' \ne j : B[i', k'] = A[i', k'])$$

$\wedge$

$$(\forall k' | 0 \le k' < i \vee k \le k' < \#filas(A) : B[j, k'] = A[j, k'])$$

$\wedge$

$$\dim(B) = \dim(A).$$

Un esquema de cómo luce una matriz lcom se muestra en la Figura 3, se han sustituidos los valores en Γ y en * por $-\frac{\Gamma}{\Delta}$x $+*$, dejando los demás sin modificar, es decir, se comenzó la operación *sumar a la fila j, la fila* i *multiplicada por* $-\frac{\Gamma}{\Delta'}$, *y se ha realizado desde la columna* i *hasta la columna* $k-1$.

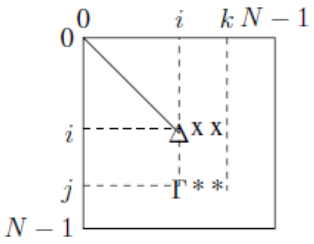

Figura 3: Matriz $\mathrm{lcom}_{i,j}^{k}(A)$. **Fuente**: Elaboración propia

Lema 4. *Dado un entero positivo N y una matriz A de tamaño $N \times N$, si $0 \le i, j, i', j' < N$ se cumple*

1. $\mathrm{lcom}_{i,j}^{N}(A)[j, i] = 0.$
2. *Si* $\mathrm{i} < \mathrm{j}' < \mathrm{j}$, *entonces* $\mathrm{lcom}_{i,j}^{N}(A)[j', i] = A[j', i].$
3. *Si $j' < i$, entonces* $\mathrm{lcom}_{i,j}^{N}(A)[i', j'] = A[i', j'].$
4. $\mathrm{lcom}_{i,j}^{i}(A) = A.$
5. *Si $i < j$ entonces* $TS_{i,j+1}\left(\mathrm{lcom}_{i,j}^{N}(A)\right) \equiv TS_{i,j}(A).$

Demostración: Como $0 \le i, j, i', j' < N$, entonces los predicados TS y funcionales lcom de todos los ítems del lema están bien definidos.

1. Tomando $k = N$ e instanciando la primera fórmula cuantificada universalmente de la definición de lcom con $k' := i$ se tiene que por ser cierto $i \le i < N$ se cumple

$$\overset{true}{\cancel{i \le i < N}} \Rightarrow$$
$$\mathrm{lcom}_{i,j}^{N}(A)[j, i] = -\frac{A[j, i]}{\cancel{A[i, i]}} \cancel{A[i, i]} + A[j, i]$$

$\equiv$

$$\mathrm{lcom}_{i,j}^{N}(A)[j, i] = 0$$

La demostración de los ítems 2, 3, 4 es trivial.

5.

$$TS_{i,j+1}\left(\mathrm{lcom}_{i,j}^{N}(A)\right)$$
$\equiv$ ⟨item 4 de Lema 2⟩
$$\mathrm{TS}_{i,j}\left(\mathrm{lcom}_{i,j}^{N}(\mathrm{A})\right) \wedge \mathrm{lcom}_{i,j}^{N}(\mathrm{A})[\mathrm{j}, \mathrm{i}] = 0$$
$\equiv$ ⟨ item 1 de Lema 4 ⟩
$$\mathrm{TS}_{i,j}\left(\mathrm{lcom}_{i,j}^{N}(\mathrm{A})\right) \wedge 0 = 0$$

$\equiv$

$$(\forall i', j' | 0 \le j' < i \wedge j' < i' < \#\,\mathrm{filas}(A):\ \mathrm{lcom}_{i,j}^{N}(A)[i', j'] = 0) \wedge$$
$$\left(\forall \mathrm{j}' \middle| \mathrm{i} < \mathrm{j}' < \mathrm{j} :\ \mathrm{lcom}_{i,j}^{N}(\mathrm{A})[\mathrm{j}', \mathrm{i}] = 0\right)$$

$\equiv$ ⟨item 2 y 3 de Lema 4⟩

$$(\forall \mathrm{i}', \mathrm{j}' | 0 \le \mathrm{j}' < \mathrm{i} \wedge \mathrm{j}' < \mathrm{i}' < \#\,\mathrm{filas}(\mathrm{A}) :\ \mathrm{A}[\mathrm{i}', \mathrm{j}'] = 0)$$
$$\wedge\ (\forall \mathrm{j}' | \mathrm{i} < \mathrm{j}' < \mathrm{j} : \mathrm{A}[\mathrm{j}', \mathrm{i}] = 0)$$

$\equiv$

$$\mathrm{TS}_{\mathrm{i,j}}(A) \quad \blacksquare$$

Definición 6. *Sea A una matriz y b un arreglo de tipos T. Si $\overline{[i, j]}$ es una lista de coordenadas y $\overline{j}$ es una lista de índices válidos dentro de las dimensiones de la matriz y del arreglo respectivamente y $\overline{Exp}$ es una lista de expresiones de tipo T y de longitud igual a longitud de $\overline{[i, j]}$ y $\overline{j}$, entonces se definen: $A_{\overline{Exp}}^{\overline{[i,j]}}$ y $b_{\overline{Exp}}^{\overline{j}}$ como una matriz y un arreglo idénticos a A y b respectivamente, salvo en las posiciones $\overline{[i, j]}$ para la matriz y las posiciones $\overline{j}$ para el arreglo, cuyo valores son $\overline{Exp}$.*

Lema 5. *Sean* A *y* A_0 *matrices, b y b_0 arreglos. Si* i *y* j *son índices válidos dentro de las dimensiones de A, A_0, b y b_0, entonces es cierto que:*

1. $b^{i,j}_{b[j],b[i]} = b_0{}^{i,j}_{b_0[j],b_0[i]} \equiv b = b_0.$
2. *Si* $0 \le i < j < N$ *y si se denota* Exp *como* $-\frac{b[i]}{A[i,i]}A[j,i] + b[j]$, *entonces*

$$b^j_{Exp} = b^j_{0\,Exp[b:=b_0]}$$

$$\equiv$$

$$b = b_0.$$

3. *Si* $i \le k < N$ *y* $0 \le i < j < N$ *entonces*

$$A^{[i,k],[j,k]}_{A[j,k],A[i,k]} = \mathrm{swap}^{k+1}_{i,j}(A_0)$$

$$\equiv$$

$$A = \mathrm{swap}^{k}_{i,j}(A_0)$$

4. *Si se denota* Exp_1 *como* $\frac{-A_0[j,i]}{A[i,i]}A[i,k] + A[j,k]$ *y* $i \le k < N$ *y* $0 \le i < j < N$ *entonces*

$$A^{[j,k]}_{Exp_1} = \mathrm{lcom}^{k+1}_{i,j}(A_0) \equiv A = \mathrm{lcom}^{k}_{i,j}(A_0).$$

Demostración: La demostración de 1 y 2 en el sentido $\Leftarrow$ es trivial porque como $b = b_0$, entonces b y b_0 son intercambiables. Para demostrar el sentido $\Rightarrow$ de 1 se muestra que $b[i] = b_0[i]$ y $b[j] = b_0[j]$, y por ende $b = b^{i,j}_{b[i],b[j]} \overset{\text{hip}}{\cong} b_0{}^{i,j}_{b[i],b[j]} = b_0{}^{i,j}_{b_0[i],b_0[j]} = b_0$. Para demostrar $\Rightarrow$ de 2, se muestra que Exp' definida como $\frac{b[i]}{A[i,i]}A[j,i]$, cumple con $b[j] = Exp + Exp'$ y $b_0[j] = (Exp + Exp')[b := b_0]$ y que como $b[i] = b_0[i]$ para $i \neq j$, entonces $Exp' = Exp'[b := b_0]$, por lo tanto

$$\begin{aligned} b = b^j_{Exp+Exp'} &= b^j_{b^j_{Exp}[j]+Exp'} \overset{\text{hip}}{\cong} b_0{}^j_{b^j_{Exp}[j]+Exp'} \\ &= b_0{}^j_{b_0{}^j_{Exp[b:=b_0]}[j]+Exp'[b:=b_0]} \\ &= b_0{}^j_{Exp[b:=b_0]+Exp'[b:=b_0]} = b_0. \end{aligned}$$

Para demostrar 3 y 4 se verifica que $\mathrm{swap}^{k+1}_{i,j}(A_0) = \mathrm{swap}^{k}_{i,j}(A_0)^{[i,k],[j,k]}_{A_0[j,k],A_0[i,k]}$ y $\mathrm{lcom}^{k}_{i,j}(A_0)^{[j,k]}_{Exp_1[A:=A_0]} = \mathrm{lcom}^{k+1}_{i,j}(A_0)$. Luego la demostración es análoga a 1 y 2. ■

Definición 7. *Dado un entero positivo* N, *dos matrices* A, A_0, *de dimensiones* $N \times N$ *y dos vectores* b, b_0 *de* N *coordenadas, se dice que los sistemas de ecuaciones* $A\vec{x} = b$ *y* $A_0\vec{x} = b_0$, *tienen las mismas soluciones, si y sólo si es cierto el predicado* $A, b \approx A_0, b_0$, *que es una abreviación de:*

$$\left\{(x_0, \dots, x_{N-1}) \in R^N \middle| \left(\forall i | 0 \le i < N : \sum_{j=0}^{N-1} A[i,j]x_j\right) = b[i]\right\}$$

=

$$\left\{(x_0, \dots, x_{N-1}) \in R^N \middle| \left(\forall i | 0 \le i < N : \sum_{j=0}^{N-1} A_0[i,j]x_j\right) = b_0[i]\right\}$$

Un resultado clásico del Álgebra Lineal establece que, si realizas operaciones elementales por filas en una matriz, el sistema de ecuaciones asociado mantiene el mismo conjunto de soluciones. Las operaciones aquí definidas no son elementales ya que no modifican la fila completa, sin embargo, mantienen el conjunto de soluciones como se demuestra a continuación.

Lema 6. *Dado un entero positivo* N, *dos matrices* A, A_0, *de dimensiones* $N \times N$ *y dos vectores* b, b_0 *de* N *coordenadas, entonces*

1. *Si* $TS_{i,j}(A)$ *con* $0 \le i, j < N$, *entonces si se cumple que*

$$\mathrm{swap}^N_{i,j}(A), b^{i,j}_{b[j],b[i]} \approx A_0, b_0$$

$$\equiv$$

$$A, b \approx A_0, b_0.$$

2. *Si* $TS_{i,j}(A)$ *con* $0 \le i, j < N$, *entonces si se denota* Exp *como* $-\frac{b[i]}{A[i,i]}A[j,i] + b[j]$, *se cumple que*

$$\mathrm{lcom}^N_{i,j}(A), b^j_{Exp} \approx A_0, b_0$$

$$\equiv$$

$$A, b \approx A_0, b_0.$$

Demostración: Como $\approx$ es una relación de equivalencia entonces, para demostrar 1, basta con demostrar que

$$\mathrm{swap}^N_{i,j}(A), b^{i,j}_{b[j],b[i]} \approx A, b$$

Y para demostrar 2

$$\mathrm{lcom}^N_{i,j}(A), b^j_{Exp} \approx A, b.$$

1. Las soluciones del sistema están definidas mediante

$$\left(\forall i' \middle| 0 \le i' < N : \textstyle\sum_{j'=0}^{N-1} \mathrm{swap}^N_{i,j}(A)[i', j']x_{j'} = b^{i,j}_{b[j],b[i]}[i']\right)$$

Cuando $i' \notin \{i,j\}$ se tiene que $\text{swap}^{\text{N}}_{\text{i,j}}(\text{A})[\text{i}',\text{j}'] = \text{A}[\text{i}',\text{j}']$ y $b^{i,j}_{b[j],b[i]}[i'] = b[i']$, por lo tanto, lo anterior es equivalente a

$$\left(\forall i' \middle| 0 \le i' < N \wedge i' \notin \{i,j\} : \sum_{j'=0}^{N-1} A[i',j']x_{j'} = b[i']\right)$$

$$\wedge$$

$$\sum_{\text{j}'=0}^{\text{i}-1} \cancel{A[i,j']}^{0} x_{j'} + \sum_{\text{j}'=\text{i}}^{\text{N}-1} \text{swap}^{N}_{i,j}(A)[i,j']x_{j'} = b[j]$$

$$\wedge$$

$$\sum_{\text{j}'=0}^{\text{i}-1} \cancel{A[j,j']}^{0} x_{j'} + \sum_{j'=\text{i}}^{N-1} \text{swap}^{N}_{i,j}(A)[j,j']x_{j'} = b[i]$$

Como por hipótesis $TS_{i,j}(A)$, se tiene que $A[i,j'] = A[j,j'] = 0$ cuando $0 \le j' < i$ y aplicando la definición de $\text{swap}^{N}_{i,j}(A)$ entonces, equivalentemente se tiene que

$$\left(\forall i' \middle| 0 \le i' < N \wedge i' \notin \{i,j\} : \sum_{j'=0}^{N-1} A[i',j']x_{j'} = b[i']\right)$$

$$\wedge$$

$$\sum_{\text{j}'=0}^{\text{i}-1} \cancel{A[j,j']}^{0} x_{j'} + \sum_{\text{j}'=\text{i}}^{\text{N}-1} \text{A[j,j']x}_{\text{j}'} = \text{b[j]}$$

$$\wedge$$

$$\sum_{\text{j}'=0}^{\text{i}-1} \cancel{A[i,j']}^{0} x_{j'} + \sum_{\text{j}'=\text{i}}^{\text{N}-1} \text{A[i,j']x}_{\text{j}'} = \text{b[i]}$$

Lo cual es equivalente a

$$\left(\forall i' \middle| 0 \le i' < N : \sum_{j'=0}^{N-1} A[i',j']x_{j'} = b[i']\right).$$

2. Esta demostración es análoga a la anterior, separando del rango del ∀ el caso $i' = j$. ■

VI. Algoritmo de Eliminación Gaussiana

En esta sección se derivará el Algoritmo de eliminación gaussiana usando la técnica de *reemplazo de constante por variable* recursivamente. Cada predicado Pdef y Qdef que se usará, satisfacen las condiciones de la Observación 1, de modo que al encontrar una instrucción S que satisfaga $\{\text{Pdef}\}\ S\ \{\text{Qdef}\}$, se tiene garantía que las hipótesis del Teorema 4 se cumplieron durante todo el proceso.

La postcondición de este algoritmo debe indicar que el estado final de la matriz debe ser diagonal superior, esto se expresa formalmente con el predicado $TS_{N,N}(A)$. Adicionalmente la postcondición debe decir que la solución del sistema $A\vec{x} = b$, debe ser la misma al principio que al final de la ejecución del programa. Denotando A_0, b_0 como las variables de especificación, que indican el estado inicial de la matriz A y el vector b, entonces formalmente se puede expresar esta condición con el predicado $A, b \approx A_0, b_0$. Por lo dicho anteriormente la postcondición de este algoritmo debe ser:

$$\{\text{Post} : TS_{N,N}(\text{A}) \wedge A, b \approx A_0, b_0\}.$$

La construcción de este algoritmo se realizará en cinco partes.

A. *Primera Parte*

Usando la técnica de reemplazo de constante por variable, al reemplazar N por i en Post, aparece $TS_{i,i}(A)$, pero como en $TS_{i,i}(A)$ el índice i no puede superar las dimensiones de la matriz y aumenta de uno en uno dentro del ciclo, entonces el programa no podrá iterar más de N veces, por lo tanto, se toma K con el valor N, obteniendo así la Tripleta 1.

Tripleta 1: Aplicación de reemplazo de constante por variable a Post

$\{Inv_1 : T_N \wedge TS_{i,i}(A) \wedge A, b \approx A_0, b_0\}$
do i ≠ N →
  A, b, j, k, a : [Pdef_1, Qdef_1];
  i :=i+1
od
$\{\text{Post} : TS_{N,N}(A) \wedge A, b \approx A_0, b_0\}$

Donde los predicados Pdef_1 y Qdef_1 están definidos como:

$$\text{Pdef}_1 : TI_N \wedge TS_{i,i}(A) \wedge A, b \approx A_0, b_0$$

$$\text{Qdef}_1 : TS_{i+1,i+1}(A) \wedge A, b \approx A_0, b_0$$

Y, además

$$T_N \equiv N - N \le i \le N \equiv 0 \le i \le N \text{ y}$$

$$TI_N \equiv N - N \le i < N \equiv 0 \le i < N.$$

Por otro lado, inicializando el ciclo en $i := 0$ se puede calcular la precondición de todo el programa haciendo:

$$\text{wp}(i := 0, Inv_1)$$

$$\equiv$$

$$Inv_1[i := 0]$$

$$\equiv$$

$$0 \le 0 \le N \wedge TS_{0,0}(A) \wedge A, b \approx A_0, b_0$$

$$\equiv \quad \langle \text{Lema 2} \rangle$$

$$0 \le N \wedge true \wedge A, b \approx A_0, b_0.$$

Como $A = A_0 \wedge b = b_0 \Rightarrow A, b \approx A_0, b_0$, la precondición anterior puede ser fortalecida según las reglas de

Hoare, de modo que una precondición válida para todo el programa sería

$$\{\text{Pre}: 0 \leq \text{N} \wedge \text{A} = \text{A}_0 \wedge \text{b} = \text{b}_0\}.$$

Nótese que $\text{Pdef}_1 \Rightarrow 0 \leq \text{i} < \text{N}$ y en $0 \leq i < N$ no ocurren variables de especificación ni las variables $A,\ b,\ j,\ k,\ a$, que son las que pueden ser modificadas por la instrucción de especificación $A,\ b,\ j,\ k,\ a : [\text{Pdef}_1, \text{Qdef}_1]$. Usando el Teorema 3 se determina que la instrucción de especificación anterior es idéntica a la instrucción

$$A, b, j, k, a : \left[\text{Pdef}_1, 0 \leq \text{i} < \text{N} \wedge \text{Qdef}_1\right].$$

Para refinar la instrucción de especificación en un trozo de código de *GCL*, será conveniente usar la última versión porque es de utilidad contar con el predicado $0 \leq i < N$ en conjunción con Qdef_1.

El algoritmo final se muestra en el Algoritmo 1.

Algoritmo 1: Algoritmo inicial de eliminación Gaussiana

```
|[ Const N : Integer;
Var i, j, k : Integer;
Var a : Real;
Var A : Matrix N× N of Real;
Var b : Array of length N of Real;
{Pre: 0 ≤ N ∧ A = A0 ∧ b = b0}
i=0;
{Inv1: 0 ≤ i ≤ N ∧ TS_{i,i}(A) ∧ A, b ≈ A0, b0}
do i≠ N →
   A, b, j, k, a : [Pdef1, 0 ≤ i < N ∧ Qdef1];
   i := i+1
od
{Post: TS_{N,N}(A) ∧ A, b ≈ A0, b0}
]|
```

B. Segunda Parte

A continuación se refina la instrucción de especificación $A,\ b,\ j,\ k,\ a : \left[\text{Pdef}_1, 0 \leq \text{i} < N \wedge \text{Qdef}_1\right]$ en una instrucción S de *GCL* que sólo modifique las variables $A,\ b,\ j,\ k,\ a$ y que satisfaga la tripleta $\{\text{Pdef}_1\}\ \text{S}\ \{\text{Qdef}_1' : 0 \leq \text{i} < \text{N} \wedge \text{Qdef}_1\}$.

La postcondición de la tripleta anterior puede escribirse de forma equivalente haciendo uso del Lema 2 de la siguiente manera

$$\begin{aligned} & 0 \leq i < N \wedge \text{Qdef}_1 \\ \equiv & \\ & 0 \leq i < N \wedge TS_{i+1,i+1}(A) \wedge A, b \approx A_0, b_0 \\ \equiv & \quad \langle \text{Lema } \mathbf{2} \rangle \\ & 0 \leq i < N \wedge TS_{i,N}(A) \wedge A, b \approx A_0, b_0 \end{aligned}$$

Usando la técnica de reemplazo de constante por variable, al reemplazar N por j en Qdef_1, aparece $\text{TS}_{\text{i},j}(\text{A})$, pero como el índice j en el predicado $TS_{i,j}(A)$ debe ser mayor que i, pero no mayor que las dimensiones de la matriz y j se incrementa de uno en uno, en cada iteración, entonces el algoritmo sólo puede iterar a lo sumo $N - i - 1$ iteraciones, por lo tanto se toma K como $N - i - 1$. Obteniéndose la Tripleta 2.

Tripleta 2: Aplicación de reemplazo de constante por variable a Qdef_1'

$$\begin{aligned} & \{Inv_2 : T_{N-i-1} \wedge 0 \leq \text{i} < N \wedge \text{TS}_{\text{i,j}}(\text{A}) \wedge A, b \approx A_0, b_0\} \\ & \textbf{do } \text{j} \neq \text{N} \rightarrow \\ & \quad \text{A, b, k, a} : [\text{Pdef}_2, \text{Qdef}_2]; \\ & \quad \text{j} := \text{j+1} \\ & \textbf{od} \\ & \{\text{Qdef}_1' : 0 \leq \text{i} < \text{N} \wedge \text{TS}_{\text{i,N}}(\text{A}) \wedge A, b \approx A_0, b_0\} \end{aligned}$$

Donde los predicados Pdef_2 y Qdef_2 se definen como:

$$\text{Pdef}_2 : TI_{N-i-1} \wedge 0 \leq i < N \wedge TS_{i,j}(A) \wedge A, b \approx A_0, b_0$$

$$\text{Qdef}_2 : 0 \leq i < N \wedge TS_{i,j+1}(A) \wedge A, b \approx A_0, b_0$$

y

$$\begin{aligned} T_{N-i-1} & \equiv N - (N - i - 1) \leq j \leq N \equiv i < j \leq N \\ TI_{N-i-1} & \equiv N - (N - i - 1) \leq j < N \equiv i < j < N \end{aligned}$$

De esta forma como $i < j$, se puede inicializar el ciclo en $j := i + 1$, y entonces se calcula la precondición más débil del algoritmo recién planteado haciendo

$$\begin{aligned} & \text{wp}(j := i + 1, Inv_2) \\ \equiv & \\ & Inv_2[j := i + 1] \\ \equiv & \\ & \cancel{i \leq i + 1 \leq N} \wedge 0 \leq i < N \wedge TS_{i,i+1}(A) \\ & \qquad \wedge A, b \approx A_0, b_0 \\ \equiv & \\ & 0 \leq i < N \wedge TS_{i,i+1}(A) \wedge A, b \approx A_0, b_0 \\ \equiv & \quad \langle \text{Item 3 Lema } \mathbf{2} \rangle \\ & 0 \leq i < N \wedge TS_{i,i}(A) \wedge A, b \approx A_0, b_0 \\ \equiv & \\ & \text{Pdef}_1 \end{aligned}$$

De esta forma se muestra que la precondición más débil de este trozo de código es Pdef_1 como se necesita.

Por otro lado, análogo a la primera parte, se tiene que $\text{Pdef}_2 \Rightarrow 0 \leq \text{i} < \text{j} < \text{N}$ y en $0 \leq i < j < N$ no ocurren variables de especificación ni las variables A, b, k, a, que son las que pueden ser modificadas por la instrucción de especificación $A, b, k, a : [\text{Pdef}_2, \text{Qdef}_2]$. Usando el Teorema 3 se determina que la instrucción de especificación anterior es idéntica a la instrucción

$$\text{A, b, k, a} : [\text{Pdef}_2, 0 \leq \text{i} < \text{j} < \text{N} \wedge \text{Qdef}_2].$$

Será conveniente usar esta versión de la instrucción.

Con esto se concluye que el Algoritmo 2, refina a la primera instrucción de especificación

$A, b, j, k, a : [\text{Pdef}_1, 0 \le i < N \wedge \text{Qdef}_1]$

propuesta en la primera parte.

Algoritmo 2: Refinamiento de $A, b, j, k, a : [\text{Pdef}_1, 0 \le i < N \wedge \text{Qdef}_1]$

$\{\text{Pdef}_1 : 0 \le i < N \wedge TS_{i,i}(A) \wedge A, b \approx A_0, b_0\}$
j := i+1;
$\{Inv_2 : 0 \le i < j \le N \wedge TS_{i,j}(A) \wedge A, b \approx A_0, b_0\}$
do j ≠ N →
  A, b, k, a : $[\text{Pdef}_2, 0 \le i < j < N \wedge \text{Qdef}_2]$;
  j :=j+1
od
$\{\text{Qdef}_1' : 0 \le i < N \wedge TS_{i+1,i+1}(A) \wedge A, b \approx A_0, b_0\}$

C. Tercera Parte

En esta parte se busca refinar la tripleta $A, b, k, a : [\text{Pdef}_2, 0 \le i < j < N \wedge \text{Qdef}_2]$ en un trozo de código S de *GCL* que satisfaga

$$\{\text{Pdef}_2\}\ \text{S}\ \{\text{Qdef}_2' : 0 \le \text{i} < \text{j} < \text{N} \wedge \text{Qdef}_2\}$$

donde S sólo modifica las variables A, b, k, a. Definiendo como Exp a la expresión $-A[j,i] * (b[i]/A[i,i]) + b[j]$, se propone el Algoritmo 3.

Algoritmo 3: Refinamiento de $A, b, k, a : [\text{Pdef}_2, 0 \le i < j < N \wedge \text{Qdef}_2]$

$\{\text{Pdef}_2 : 0 \le i < j < N \wedge TS_{i,j}(A) \wedge A, b \approx A_0, b_0\}$
if A[i, i] = 0 ∧ A[j, i] ≠0 →
  A, b, k :
  $\left[\begin{array}{cc} \text{A} = \text{A}_0 \wedge \text{b} = \text{b}_0 & \text{A} = \text{swap}_{i,j}^N(\text{A}_0) \\ \wedge & \wedge \\ 0 \le \text{i} < \text{j} < \text{N} & , \quad \text{b} = \text{b}_{0\,b_0[j],b_0[i]}^{i,j} \\ & \wedge \\ & 0 \le \text{i} < \text{j} < \text{N} \end{array}\right]$
[ ] A[i, i] ≠ 0 ∧ A[j, i] ≠0 →
  A, b, k, a :
  $\left[\begin{array}{cc} \text{A} = \text{A}_0 \wedge \text{b} = \text{b}_0 & \text{A} = \text{lcom}_{i,j}^N(\text{A}_0) \\ \wedge & \wedge \\ 0 \le \text{i} < \text{j} < \text{N} & , \quad \text{b} = \text{b}_{0\,\text{Exp}[b,A:=b_0,A_0]}^{j} \\ \wedge & \wedge \\ \text{A[i,i]} \ne 0 & 0 \le \text{i} < \text{j} < \text{N} \end{array}\right]$
[ ] A[j, i] = 0 →
  SKIP
fi
$\{\text{Qdef}'_2 : 0 \le i < j < N \wedge TS_{i,j+1}(A) \wedge A, b \approx A_0, b_0\}$

Ahora se debe calcular wp del IF anterior y Qdef_2', pero para esto, primero se calculará wp de ambas instrucciones de especificación $A, b, k : [\text{Pdef}_3, \text{Qdef}_3]$ y $A, b, k, a : [\text{Pdef}_4, \text{Qdef}_4]$, que se encuentran en el cuerpo de la primera y segunda guardia del IF anterior con Qdef_2'.

$\text{wp}(A, b, k : [\text{Pdef}_3, \text{Qdef}_3], \text{Qdef}_2')$

≡ ⟨Teorema 3⟩

$$\text{wp}\left(A, b, k: \left[\begin{array}{cc} \text{Pdef}_3, & A = \text{swap}_{i,j}^N(A_0) \\ & \wedge \\ & b = b_{0\,b_0[j],b_0[i]}^{i,j} \end{array}\right], \text{Qdef}_2'\right)$$

≡ ⟨Corolario 2⟩

$$0 \le i < j < N \wedge (\forall A', b', k' | \begin{array}{c} A' = \text{swap}_{i,j}^N(A) \\ \wedge \\ b' = b_{b[j],b[i]}^{i,j} \end{array} :$$
$$0 \le i < j < N \wedge TS_{i,j+1}(A') \wedge A', b' \approx A_0, b_0)$$

≡ ⟨regla de un punto⟩

$$0 \le i < j < N \wedge TS_{i,j+1}\left(\text{swap}_{i,j}^N(A)\right) \wedge \text{swap}_{i,j}^N(A), b_{b[j],b[i]}^{i,j} \approx A_0, b_0$$

≡ ⟨item 5 del Lema 3⟩

$$0 \le i < j < N \wedge TS_{i,j}(A) \wedge A[i,i] = 0 \wedge \text{swap}_{i,j}^N(A), b_{b[j],b[i]}^{i,j} \approx A_0, b_0$$

≡ ⟨item 1 del Lema 6⟩

$$0 \le i < j < N \wedge TS_{i,j}(A) \wedge A[i,i] = 0 \wedge A, b \approx A_0, b_0$$

Por otro lado, se calcula

$\text{wp}(A, b, k, a : [\text{Pdef}_4, \text{Qdef}_4], \text{Qdef}_2')$

≡ ⟨Teorema 3⟩

$$\text{wp}\ (A, b, k, a : \left[\begin{array}{cc} \text{Pdef}_4, & A = \text{lcom}_{i,j}^N(A_0) \\ & \wedge \\ & b = b_{0\,Exp[b,A:=b_0,A_0]}^{j} \end{array}\right], \text{Qdef}_2')$$

≡ ⟨Corolario 2⟩

$$0 \le i < j < N \wedge A[i,i] \ne 0 \wedge (\forall A', b', k', a' | \begin{array}{c} A' = \text{lcom}_{i,j}^N(A) \\ \wedge \\ b' = b_{Exp}^{j} \end{array} :$$
$$0 \le i < j < N \wedge TS_{i,j+1}(A') \wedge A', b' \approx A_0, b_0)$$

≡ ⟨regla de un punto⟩

$$0 \le i < j < N \wedge A[i,i] \ne 0 \wedge TS_{i,j+1}\left(\text{lcom}_{i,j}^N(A)\right) \wedge \text{lcom}_{i,j}^N(A), b_{Exp}^{j} \approx A_0, b_0$$

≡ ⟨ítem 5 del Lema 4⟩

$$0 \le i < j < N \wedge A[i,i] \neq 0 \wedge TS_{i,j}(A) \wedge \mathrm{lcom}^N_{i,j}(A), b^j_{Exp} \approx A_0, b_0$$

$\equiv$ ⟨item 2 del Lema 6⟩

$$0 \le i < j < N \wedge A[i,i] \neq 0 \wedge TS_{i,j}(A) \wedge A, b \approx A_0, b_0$$

Usando los cálculos anteriores y abreviando B_0, B_1 y B_2 como las guardias del IF anterior, se calcula wp de IF y Qdef'_1 de la siguiente manera.

$$\mathrm{wp}(IF, \mathrm{Qdef}'_2)$$

$\equiv$

$$\cancel{\mathrm{domain}(B_0, B_1, B_2)^{0\le i,j<N}} \wedge (\cancel{B_0 \vee B_1 \vee B_2}^{true}) \wedge$$
$$(B_0 \Rightarrow \mathrm{wp}(A, b, k : [\mathrm{Pdef}_3, \mathrm{Qdef}_3], \mathrm{Qdef}'_2)) \wedge$$
$$(B_1 \Rightarrow \mathrm{wp}(A, b, k, a : [\mathrm{Pdef}_4, \mathrm{Qdef}_4], \mathrm{Qdef}'_2)) \wedge$$
$$(B_2 \Rightarrow \mathrm{wp}(\mathrm{SKIP}, \mathrm{Qdef}'_2))$$

$\equiv$

$$0 \le i, j < N \wedge$$
$$(A[i,i] = 0 \wedge A[j,i] \neq 0 \Rightarrow 0 \le i < j < N \wedge$$
$$TS_{i,j}(A) \wedge (\cancel{A[i,i] = 0}^{true}) \wedge A, b \approx A_0, b_0) \wedge$$
$$(A[i,i] \neq 0 \wedge A[j,i] \neq 0 \Rightarrow 0 \le i < j < N \wedge$$
$$(\cancel{A[i,i] \neq 0}^{true}) \wedge TS_{i,j}(A) \wedge A, b \approx A_0, b_0) \wedge$$
$$(A[j,i] = 0 \Rightarrow 0 \le i < j < N \wedge A, b \approx A_0, b_0$$
$$\wedge \quad \cancel{TS_{i,j+1}(A)}^{TS_{i,j}(A) \wedge (\cancel{A[j,i]=0}^{true})} \quad)$$

$\equiv$ $\langle (B_0 \Rightarrow P) \wedge (B_1 \Rightarrow P) \wedge (B_2 \Rightarrow P) \equiv P \rangle$

$$\cancel{0 \le i, j < N} \wedge 0 \le i < j < N \wedge TS_{i,j}(A) \wedge$$
$$A, b \approx A_0, b_0$$

$\equiv$

$$\mathrm{Pdef}_2$$

Con esto queda demostrado que la tripleta $\{\mathrm{Pdef}_2\}$ IF $\{\mathrm{Qdef}'_2\}$ es cierta.

D Cuarta Parte

A continuación, se refina la instrucción de especificación

$$\mathrm{A, b, k}: \begin{bmatrix} A = A_0 \wedge b = b_0 & A = \mathrm{swap}^N_{i,j}(A_0) \\ \wedge & \wedge \\ 0 \le i < j < N & b = b_0{}^{i,j}_{b_0[j],b_0[i]} \\ & \wedge \\ & 0 \le i < j < N \end{bmatrix}$$

en una instrucción de *GCL*.

Usando la técnica de reemplazo de constante por variable, al reemplazar N por k en Qdef_3, aparece $A = \mathrm{swap}^k_{i,j}(A_0)$, pero, para que *swap* no se comporte como la función identidad, salvo al inicio de la primera iteración, debe ocurrir que $i \le k \le N$. Como k aumenta de uno en uno en el ciclo, entonces basta con que el ciclo itere a lo sumo $N - i$ veces. Por lo tanto, se toma $K := N - i$ generando la Tripleta 3.

Tripleta 3: Aplicación de reemplazo de constante por variable a Qdef_3

$\{Inv_3 : T_{N-i} \wedge 0 \le i < j < N \wedge A = \mathrm{swap}^k_{i,j}(A_0) \wedge b = b_0{}^{i,j}_{b_0[j],b_0[i]}\}$
do k≠ N →
  A, b : $[\mathrm{Pdef}_5, \mathrm{Qdef}_5]$;
  k := k+1
od
$\{\mathrm{Qdef}_3 : 0 \le i < j < N \wedge A = \mathrm{swap}^N_{i,j}(A_0) \wedge b = b_0{}^{i,j}_{b_0[j],b_0[i]}\}$

Donde los predicados Pdef_5 y Qdef_5 están definidos como:

$$\mathrm{Pdef}_5 : TI_{N-i} \wedge 0 \le i < j < N \wedge A = \mathrm{swap}^k_{i,j}(A_0) \wedge b = b_0{}^{i,j}_{b_0[j],b_0[i]}$$

$$\mathrm{Qdef}_5 : 0 \le i < j < N \wedge A = \mathrm{swap}^{k+1}_{i,j}(A_0) \wedge b = b_0{}^{i,j}_{b_0[j],b_0[i]}.$$

Y, además

$$T_{N-i} \equiv N - (N - i) \le k \le N \equiv i \le k \le N$$
$$TI_{N-i} \equiv N - (N - i) \le k < N \equiv i \le k < N$$

Así, como $i \le k$ y $b = b_0{}^{i,j}_{b_0[j],b_0[i]}$, se puede inicializar el ciclo en $\mathrm{k, b[i], b[j]} := \mathrm{i, b[j], b[i]}$, entonces se calcula la precondición más débil de este algoritmo haciendo

$$\mathrm{wp}(k, b[i], b[j] := i, b[j], b[i], Inv_3)$$

$\equiv$

$$\text{Inv}_3\left[k, b := i, b^{i,j}_{b[j],b[i]}\right]$$

$\equiv$

$$\cancel{0 \le i,j < N} \wedge i \le \cancel{i \le N} \wedge 0 \le i < j < N \wedge A = \text{swap}^{i}_{i,j}(A_0) \wedge b^{i,j}_{b[j],b[i]} = {b_0}^{i,j}_{b_0[j],b_0[i]}$$

$\equiv$ ⟨item 4 y 1 del Lema 3 y Lema 5 ⟩

$$0 \le i < j < N \wedge A = A_0 \wedge b = b_0$$

$\equiv$

$$\text{Pdef}_3$$

Con lo que la precondición más débil de este trozo de código es Pdef_3 como se necesita.

Como $\text{Pdef}_5 \Rightarrow i \le k < N$ y en $i \le k < N$ no ocurren variables de especificación ni A, b, entonces por el Teorema 3 la instrucción de especificación anterior es idéntica a la instrucción $\text{A, b} : [\text{Pdef}_5, \text{i} \le \text{k} < \text{N} \wedge \text{Qdef}_5]$. Esta instrucción se realiza con $A[i,k],\ A[j,k] := A[j,k], A[i,k]$, ya que:

$$\text{wp}(A[i,k], A[j,k] := A[j,k], A[i,k], i \le k < N \wedge \text{Qdef}_5)$$

$\equiv$

$$\cancel{0 \le i,j,k < N} \wedge i \le k < N \wedge \text{Qdef}_5\left[A := A^{[i,k],[j,k]}_{A[j,k],A[i,k]}\right]$$

$\equiv$

$$i \le k < N \wedge 0 \le i < j < N \wedge A^{[i,k],[j,k]}_{A[j,k],A[i,k]} = \text{swap}^{k+1}_{i,j}(A_0) \wedge b = {b_0}^{i,j}_{b_0[j],b_0[i]}$$

$\equiv$ ⟨item 3 del Lema 5⟩

$$i \le k < N \wedge 0 \le i < j < N \wedge A = \text{swap}^{k}_{i,j}(A_0) \wedge b = {b_0}^{i,j}_{b_0[j],b_0[i]}$$

$\equiv$

$$\text{Pdef}_5$$

Con todo esto se concluye que el Algoritmo 4 es un refinamiento de la instrucción $\text{A, b, k} : [\text{Pdef}_3, \text{Qdef}_3]$.

Algoritmo 4: Refinamiento de $\text{A, b, k} : [\text{Pdef}_3, \text{Qdef}_3]$

$\{\text{Pdef}_3 : 0 \le i < j < N \wedge A = A_0 \wedge b = b_0\}$
k, b[i], b[j] := i,b[j], b[i];
$\{Inv_3 : i \le k \le N \wedge 0 \le i < j < N \wedge A = \text{swap}^{k}_{i,j}(A_0) \wedge b = {b_0}^{i,j}_{b_0[j],b_0[i]}\}$
do $\text{k} \neq \text{N} \rightarrow$
  $A[i,k], A[j,k] := A[j,k], A[i,k];$
  k :=k+1
od
$\{\text{Qdef}_3 : 0 \le i < j < N \wedge A = \text{swap}^{N}_{i,j}(A_0) \wedge b = {b_0}^{i,j}_{b_0[j],b_0[i]}\}$

E. Quinta Parte

A continuación, se refina la instrucción de especificación

$$A, b, k, a : \begin{bmatrix} \text{A} = A_0 \wedge b = b_0 \\ \wedge \\ 0 \le i < j < N \\ \wedge \\ A[i,i] \neq 0 \end{bmatrix} , \begin{bmatrix} A = \text{lcom}^{N}_{i,j}(A_0) \\ \wedge \\ b = {b_0}^{j}_{Exp[b,A:=b_0,A_0]} \\ \wedge \\ 0 \le i < j < N \end{bmatrix}$$

en una instrucción de GCL.

Según las Reglas de Hoare es suficiente con conseguir una tripleta verdadera con una postcondición más fuerte que Qdef_4, como por ejemplo $\text{Qdef}_4 \wedge \text{a} = -\text{A}_0[\text{j, i}] \wedge \text{A[i, i]} \neq 0$, la cual se denotará de ahora en adelante como Qdef'_4.

Usando la técnica de reemplazo de constante por variable, al reemplazar N por k en $\text{Qdef}_4{}'$, aparece $A = \text{lcom}^{k}_{i,j}(A_0)$, pero como el predicado lcom es útil cuando $i \le k \le N$ y k aumenta de uno en uno en el ciclo, entonces basta con que el ciclo itere a lo sumo $N - \boldsymbol{i}$ veces. Entonces tomando $K := N - i$ se genera la Tripleta 4.

Tripleta 4: Aplicación de reemplazo de constante por variable a Qdef'_4

$\{Inv_4 : T_{N-i} \wedge 0 \le i < j < N \wedge A = \text{lcom}^{\text{k}}_{\text{i,j}}(\text{A}_0) \wedge b = {b_0}^{j}_{Exp[b,A:=b_0,A_0]} \wedge a = -A_0[j,i] \wedge A[i,i] \neq 0\}$
do $\text{k} \neq \text{N} \rightarrow$
  A, b, a : $[\text{Pdef}_6, \text{Qdef}_6]$;
  k := k+1
od
$\{\text{Qdef}'_4 : 0 \le i < j < N \wedge A = \text{lcom}^{N}_{i,j}(A_0) \wedge b = {b_0}^{j}_{Exp[b,A:=b_0,A_0]} \wedge a = -A_0[j,i] \wedge A[i,i] \neq 0\}$

Donde los predicados Pdef_6 y Qdef_6 están definidos como:

$$\text{Pdef}_6 : TI_{N-i} \wedge 0 \le i < j < N \wedge A = \text{lcom}_{i,j}^{k}(A_0) \wedge b = {b_0}^{j}_{Exp[b,A:=b_0,A_0]} \wedge a = -A_0[j,i] \wedge A[i,i] \neq 0$$

$$\text{Qdef}_6 : 0 \le i < j < N \wedge A = \text{lcom}_{i,j}^{k+1}(A_0) \wedge b = {b_0}^{j}_{Exp[b,A:=b_0,A_0]} \wedge a = -A_0[j,i] \wedge A[i,i] \neq 0$$

y, además

$$T_{N-i} \equiv N-(N-i) \le k \le N \equiv i \le k \le N$$

$$TI_{N-i} \equiv N-(N-i) \le k < N \equiv i \le k < N$$

De esta forma, como $i \le k$, $b = {b_0}^{j}_{Exp[b,A:=b_0,A_0]}$ y $a = -A_0[j,i]$, se puede inicializar el ciclo en $k, b[j], a := i, Exp, -A[j,i]$, y entonces se calcula la precondición más débil de este algoritmo haciendo

$$\text{wp}(k, b[j], a := i, Exp, -A[j,i], Inv_4)$$

$$\equiv$$

$$A[i,i] \neq 0 \wedge \cancel{0 \le i,j < N} \wedge Inv_4\left[k, b, a := i, b^{j}_{Exp}, -A[j,i]\right]$$

$$\equiv$$

$$i \le i \le N \wedge 0 \le i < j < N \wedge \text{A} = \text{lcom}^{i}_{i,j}(A_0) \wedge -A[j,i] = -A_0[j,i] \wedge A[i,i] \neq 0 \wedge b^{j}_{Exp} = {b_0}^{j}_{Exp[b,A:=b_0,A_0]}$$

$$\equiv \quad \langle \text{item 4 del Lema 4} \rangle$$

$$0 \le i < j < N \wedge A = A_0 \wedge \cancel{-A[j,i] = -A_0[j,i]}^{\,true} \wedge A[i,i] \neq 0 \wedge b^{j}_{Exp} = {b_0}^{j}_{Exp[b:=b_0]}$$

$$\equiv \quad \langle \text{item 2 del Lema 5} \rangle$$

$$0 \le i < j < N \wedge A = A_0 \wedge A[i,i] \neq 0 \wedge b = b_0$$

$$\equiv$$

$$\text{Pdef}_4$$

Con lo que la precondición más débil de este trozo de código es Pdef_4 como se necesita.

Según el Teorema 3, la instrucción $\text{A, b, a} : [\text{Pdef}_6, \text{Qdef}_6]$ es idéntica a $\text{A, b, a} : [\text{Pdef}_6, \text{i} \le \text{k} < \text{N} \wedge \text{Qdef}_6]$, la cual se puede refinar con la instrucción $A[j,k] := a * (A[i,k]/A[i,i]) + A[j,k]$, ya que:

$$\text{wp}\left(A[j,k] := a * \left(\frac{A[i,k]}{A[i,i]}\right) + A[j,k],\ i \le k < N \wedge \text{Qdef}_6\right)$$

$$\equiv$$

$$A[i,i] \neq 0 \wedge 0 \le i,j,k < N \wedge i \le k < N \wedge \text{Qdef}_6\left[A := A^{[j,k]}_{\frac{a}{A[i,i]}A[i,k]+A[j,k]}\right]$$

$$\equiv$$

$$i \le k < N \wedge 0 \le i < j < N \wedge A^{[j,k]}_{\frac{a}{A[i,i]}A[i,k]+A[j,k]} = \text{lcom}^{k+1}_{i,j}(A_0) \wedge a = -A_0[j,i] \wedge A[i,i] \neq 0 \wedge b = {b_0}^{j}_{Exp[b,A:=b_0,A_0]}$$

$$\equiv \quad \langle \text{item 4 del Lema 5} \rangle$$

$$i \le k < N \wedge 0 \le i < j < N \wedge A = \text{lcom}^{k}_{i,j}(A_0) \wedge a = -A_0[j,i] \wedge A[i,i] \neq 0 \wedge b = {b_0}^{j}_{Exp[b,A:=b_0,A_0]}$$

$$\equiv$$

$$\text{Pdef}_6$$

Con todo esto se concluye que el Algoritmo 5 refina la instrucción $\text{A, b, k, a}: [\text{Pdef}_4, \text{Qdef}_4]$.

Algoritmo 5: Refinamiento de $\text{A, b, k, a} : [\text{Pdef}_4, \text{Qdef}_4]$

$\{\text{Pdef}_4 : 0 \le i < j < N \wedge A = A_0 \wedge b = b_0 \wedge A[i,i] \neq 0\}$

k,b[j], a:=i, -A[j,i]*(b[i]/A[i,i])+b[j],-A[j,i];

$\{Inv_4: i \le k \le N \wedge 0 \le i < j < N \wedge b = {b_0}^{j}_{Exp[b,A:=b_0,A_0]} \wedge A = \text{lcom}^{k}_{i,j}(A_0) \wedge a = -A[j,i] \wedge A[i,i] \neq 0\}$

do k ≠ N →
  A[j,k]:=a*(A[i,k]/A[i,i])+A[j,k];
  k :=k+1
od

$\{\text{Qdef}_4: 0 \le i < j < N \wedge A = \text{lcom}^{N}_{i,j}(A_0) \wedge b = {b_0}^{j}_{Exp[b,A:=b_0,A_0]}\}$

Como en los Algoritmos 4 y 5 unas instrucciones concretas S y S' satisfacen $\{\text{Pdef}_3\}\, S\, \{\text{Qdef}_3\}$ y $\{\text{Pdef}_4\}\, S'\, \{\text{Qdef}_4\}$ se satisface la hipótesis $\left(\forall \overline{x} \mid \overline{x} \in \overline{T} : \left(\forall \overline{Z} \mid \overline{Z} \in \overline{T''} \wedge Dom_{\overline{x},\overline{Z}} \neq \emptyset : Rg_{\overline{x},\overline{Z}} \neq \emptyset\right)\right)$ del Teorema 4 para Pdef_3, Qdef_3 y Pdef_4, Qdef_4, se tiene, por la

construcción anidada, que también se satisface la hipótesis para Pdef_2, Qdef_2 y Pdef_1, Qdef_1.

El algoritmo final es el mostrado en el Algoritmo 6.

Algoritmo 6: Algoritmo Final de Eliminación *Gaussiana*

```
|[ Const N : Integer;
Var i, j, k: Integer;
Var a: Real;
Var A: Matrix N× N of Real;
Var b: Array of length N of Real;
{Pre: 0 ≤ N ∧ A = A0 ∧ b = b0}
i=0;
do i ≠ N →
    j := i+1;
   do j ≠ N →
      if A[i,i] = 0 ∧ A[j,i] ≠ 0 →
         k, b[i], b[j] := i, b[j], b[i];
         do k ≠ N →
            A[i,k], A[j,k] := A[j,k], A[i,k];
            k := k+1
         od
      [ ] A[i,i] ≠ 0 ∧ A[j,i] ≠ 0 →
         k, b[j], a := i, −A[j,i]*(b[i]/A[i,i])+b[j],
                                        −A[j,i];
         do k ≠ N →
            A[j,k] := a*(A[i,k]/A[i,i])+A[j,k];
            k := k+1
         od
      [ ] A[j,i] = 0 →
         SKIP
      fi;
      j := j+1
   od;
   i := i+1
od
{Post: TS_{N,N}(A) ∧ A, b ≈ A0, b0}
]|
```

VII. Conclusiones

Se verificó que el Teorema 5 permite desarrollar programas correctos de forma ordenada y rápida. En particular, para el Algoritmo de eliminación gaussiana, se evidencia como aparecen instrucciones de especificación con variables de especificación en Pdef, que están relacionadas con las variables del programa por medio de $\approx$, y no por $=$, que era lo reglamentario según [7]. Esto muestra las limitaciones de la definición de instrucción de especificación de [7], y la necesidad de una generalización, que permita usar las variables de especificación en Pdef y Qdef de manera arbitraria. Los teoremas de la Sección III son un pequeño aporte en vías de dicha generalización.

Referencias

[1] E. W. Dijkstra, A discipline of programming, N. J.: Prentice–Hall, 1976. ISBN: 0132-15871.

[2] E. W. Dijkstra, "Guarded commands, nondeterminacy and formal derivation of programs," *Commun. ACM,* vol. 18, pp. 453-457, 1975. ISSN: 0001-0782. DOI: 10.1145/360933.360975.

[3] C. A. R. Hoare, "An axiomatic basis for computer programming," *Commun. ACM,* vol. 12, pp. 576-580, 1969. ISSN: 0001-0782. DOI: 10.1145/363235.363259.

[4] D. Gries, The Science of Programming. Monographs in Computer Science, NY: Springer, 1989. ISBN: 9780387964805. Avalaible online in: https://books.google.co.ve/books?id=vv5pot-ySsEC.

[5] F. Flaviani, "Cálculo de precondiciones más débiles," *Revista Venezolana de Computación,* vol. 3, nº 2, pp. 68-80, 2016. ISSN: 2244-7040. Avalaible online in: http://saber.ucv.ve/ojs/index.php/rev_vcomp/article/download/11731/11548.

[6] F. Flaviani, "Calculation of invariants assertions," *Electronic Notes in Theoretical Computer Science,* vol. 339, pp. 63-83, 2018. The XLII Latin American Computing Conference. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2018.06.005.

[7] C. Morgan, "The specification statement," *ACM Trans. Program. Lang. Syst.,* vol. 10, pp. 403-419, 1988. ISSN: 0164-0925. DOI: 10.1145/44501.44503.

[8] M. Torodova and D. A. Orozova, "The predicate transformer and its application in introduction to programming courses," *Burgas Free University Yearbook,* vol. 32, nº 1, pp. 194-207, 2015. Avalaible online in: https://pdfs.semanticscholar.org/6ca6/8b13f2e8b259e5be74ba2aa03c59785408e8.pdf.

[9] G. O'Regan, Giants of Computing: A Compendium of Select, Pivotal Pioneers, London: Springer, 2013. ISBN: 9781447153405. Avalible online in: https://books.google.co.ve/books?id=oSq5BAAAQBAJ.

[10] F. Flaviani, "Inference of the definition of the predicate transformer wp with occurrences of the predicate domain based on denotational semantics of GCL on ZF set theory," de *XLII Latin American Computing Conference*, 2018. DOI: 10.1109/CLEI.201800095.

[11] F. Flaviani, "Modelo relacional de la teoría axiomática del lenguaje GCL de Dijkstra," en Memorias de la *Tercera Conferencia Nacional de Computación, Informática y Sistemas*, Valencia, Venezuela, 2015, pp. 153-164.

[12] O. Mraihi, W. Ghardallou, A. Louhichi, L. L. Jilani, K. Bsaies and A. Mili., "Computing preconditions and postconditions of while loops," de *International Colloquium on Theoretical Aspects of Computing*, 2011. ISSN: 0302-9743. DOI: 10.1007/978-3-642-23283-1_13.