UNA PROPUESTA PARA PROMOVER LA ARTICULACIÓN ENTRE TEMAS BÁSICOS DE ALGEBRA Y COMPUTACIÓN

Mauricio A. Martel – María Elena Markiewicz

Universidad Nacional de Río Cuarto

Ruta 36 Km. 601

E-mail: mmartel@exa.unrc.edu.ar

Categoría del trabajo: Relato de experiencia

Nivel Educativo: Educación superior

Palabras claves: álgebra –computación – articulación – integración de contenidos

Resumen

Este trabajo se basa en un problema didáctico que hemos detectado en el ámbito de las carreras de Analista en Computación, Profesorado y Licenciatura en Ciencias de la Computación en el ámbito de nuestra universidad, vinculado a la falta de integración entre los contenidos que se trabajan en las asignaturas básicas de matemática que se imparten en el primer año de estas carreras (en particular, en la asignatura Introducción al Algebra) con cuestiones específicas referidas a la Computación. El objetivo de este trabajo es mostrar una propuesta de cómo relacionar dichas cuestiones, de manera que los alumnos puedan, desde el comienzo de su carrera, vincular contenidos y tener una visión más clara de las importantes aplicaciones que tienen ciertos temas claves de Algebra en las Ciencias de la Computación. Esta propuesta apunta a que los alumnos puedan construir un significado diferente de los objetos estudiados, enriquecido con nuevas situaciones estrechamente vinculadas a la computación, promoviendo así una relación diferente con los saberes que les van a ser de utilidad en su futura práctica profesional.

1. Introducción

Muchas veces, en los primeros años de las carreras de Computación, durante el cursado de alguna asignatura de Matemática, muchos alumnos se preguntan: ¿y esto para que me sirve? Es claro que, cuando recorren los tramos más avanzados de la carrera comienzan a comprender la importancia de los tópicos que se han visto en los primeros años. Pero muchas

veces les resulta dificultoso relacionar y aplicar los temas trabajados en Algebra en problemáticas que surgen desde contextos computacionales.

Creemos que una de las posibles causas de este hecho radica en que, en el caso de nuestra universidad, durante muchos años, los contenidos de Algebra se han dado aislados y pocas veces relacionados con computación. Según Chevallard (2004) la fragmentación, el encierro, incluso la agresividad disciplinar son rasgos que caracterizan la enseñanza.

En este sentido es que, desde la asignatura Introducción al Algebra, correspondiente al 1er. año de las carreras de Analista en Computación, Profesorado y Licenciatura en Ciencias de la Computación, estamos trabajando conjuntamente profesores de matemática y estudiantes avanzados de la Licenciatura en Computación, a fin de tratar de contrarrestar, en parte al menos, esta fragmentación de contenidos y lograr la vinculación de los mismos, más específicamente, lograr que los alumnos puedan tomar contacto con ciertas situaciones extraídas de contextos computacionales, en las cuales el Algebra juega un papel fundamental.

El objetivo de este trabajo es mostrar una propuesta de cómo es posible trabajar algunos temas fundamentales de la asignatura Introducción al Algebra para relacionarlos con cuestiones referidas a la computación. Para ello, vamos a mostrar primero cómo se trabajó en este sentido con dos temas fundamentales de la asignatura, como son el **Principio de Inducción** y las **relaciones de equivalencia**, y luego haremos una síntesis de las vinculaciones de otros tópicos estudiados en la materia y su relación con computación.

2. Desarrollo

Uno de los temas fundamentales que abordamos en la asignatura Introducción al Algebra es el conjunto de los números naturales, su presentación axiomática, la definición y propiedades de sus operaciones y relaciones fundamentales y, en especial, el **Principio de inducción matemática**. Usualmente, en la materia, el uso de este Principio se restringía a la demostración de propiedades en el conjunto de los números naturales IN. Sin embargo, los alcances del mismo en computación exceden este uso. Es así que, hemos planteado a nuestros alumnos, por un lado, situaciones donde el uso de este principio se conecta con la verificación y derivación de programas y, por otro lado, hemos mostrado cómo el principio de inducción puede ser extendido para demostrar propiedades referidas a otros conjuntos inductivos como árboles, expresiones, etc (inducción estructural).

Vamos a profundizar estas cuestiones:

Por una parte, el Principio de Inducción nos permite demostrar propiedades de funciones ya definidas recursivamente, pero también derivar funciones a partir de otras ya definidas para obtener programas que cumplan con requisitos determinados.

Veamos un ejemplo de esto:

Si tenemos una función definida recursivamente por:

$f(0) = 0$

$f(n+1) = f(n) + 2n + 1$

Podemos demostrar que f satisface la propiedad: $\forall n \in IN :\ f(n) = n^2$., usando el Principio de Inducción Matemática, del siguiente modo:

Caso base: f(0)	Etapa inductiva: Supongamos $f(n) = n^2$ y demostremos que $f(n+1) = (n+1)^2$
= {definición de f}	f(n+1)
0	= {definición de f}
= {algebra: $0 = 0^2$}	f(n) + 2.n + 1
0^2	= {hipótesis inductiva}
	n^2+ 2.n + 1
	= {cuadrado de un binomio}
	$(n+1)^2$ **(1)**

En contextos computacionales, lo que se ha hecho se llama **verificación de programas**, ya que estamos comprobando que un programa dado (una definición recursiva) satisface su especificación.. (Blanco- Smith- Barsotti, 2008, pág 165)

Si bien la técnica de verificación es muy útil, dado que permite tener una certeza mucho mayor de que el programa construido es correcto, el problema está en cómo se encuentra este programa. En realidad, lo que se quiere construir es, además del programa, una demostración de que efectivamente el programa satisface dicha especificación. Veamos un ejemplo de esto:

Supongamos ahora que tenemos una función f que satisface la especificación:

$$\forall n \in IN :\ f(n) = n^2$$

y queremos hallar una función recursiva (un programa) que satisfaga la especificación dada.

Esto lo podemos realizar de la siguiente forma:

Caso base	Etapa recursiva
f(0) = {especificación o def. de f} 0^2 = {algebra: $0^2 = 0$} 0	f(n+1) = {especificación o def. de f} $(n+1)^2$ = {cuadrado de um binomio} n^2+ 2.n + 1 = {especificación o def. de f} f(n) + 2.n + 1

Por lo tanto, podemos definir a f recursivamente como sigue:

f(0) = 0

f(n+1) = f(n) + 2.n + 1

y esta función satisface la propiedad requerida.

Lo que se acaba de hacer corresponde a lo que se llama **derivación de programas**, ya que se ha construido un programa que satisface la especificación y la demostración de que lo hace. Esto último podemos verlo notando que es inmediato cómo recuperar la demostración por inducción de que la función satisface la propiedad requerida realizada anteriormente en **(1)**.

Otro de los aspectos de la inducción que hemos trabajado es la **inducción estructural**, que puede ser usada para probar propiedades sobre conjuntos definidos inductivamente (no precisamente IN). Este tipo de definiciones consisten en dar para un conjunto S dos tipos de condiciones:

- Reglas base: que afirman que algún elemento simple o estructura básica pertenece a S.
- Reglas inductivas: que afirman que un elemento compuesto pertenece a S siempre que sus partes pertenezcan a S. Permite construir un elemento a partir de elementos ya construidos.

En computación existen varias estructuras definidas de esta forma, acerca de las cuales se necesitan probar ciertas propiedades. Las *expresiones* y los *árboles* son dos ejemplos importantes de este tipo de estructuras. Cuando se tiene un conjunto inductivo se pueden probar propiedades en estos conjuntos utilizando el llamado **Principio de Inducción Estructural*:***

Sea *S* un conjunto inductivo y sea *P* una propiedad sobre los elementos de *S*.

Si se cumple que:

1. *Caso Base*: para cada elemento $x \in S$, tal que x es una estructura básica, entonces $P(x)$ es verdadero; y
2. *Caso Inductivo*: para cada elemento $x \in S$ construido por una regla inductiva utilizando los elementos $y_1, y_2, \ldots, y_n$: si $P(y_1), P(y_2), \ldots, P(y_n)$ son verdaderos entonces $P(x)$ lo es,

entonces: $P(x)$ se cumple para todos los $x \in S$

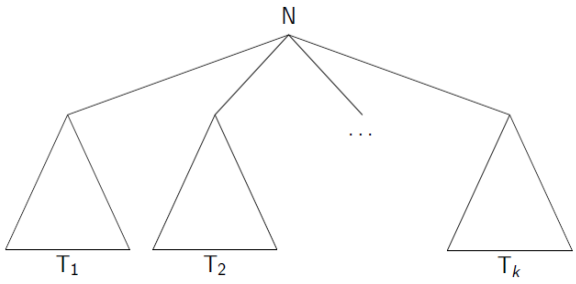
Por ejemplo, el conjunto de los árboles es un conjunto inductivo, ya que se puede definir del siguiente modo:

1. Regla base: Un solo nodo es un árbol, y ese nodo es la raíz del árbol.

2. Regla inductiva: Si $T_1, T_2, \ldots, T_k$ son árboles, entonces se puede formar un nuevo árbol de la siguiente forma:

• Se crea un nuevo nodo *N*, el cual será la raíz del árbol.

• Se agrega una arista desde el nodo *N* hasta las raíces de cada uno de los árboles $T_1, T_2, \ldots, T_k$.



Dada la definición inductiva de árbol, se quiere probar, por inducción estructural, que todo árbol tiene un nodo más que la cantidad de aristas. Es decir, se quiere probar que para todo árbol, se cumple la siguiente propiedad:

P(T): Si T es un árbol, y T tiene n nodos y a aristas, entonces n=a+1.

1.C*aso Base*: $P(N)$, donde *N* es un nodo. En este caso el árbol tienen un solo nodo (**n=1**) y ninguna arista (**a=0**). Luego, **n=a+1** se cumple (1=0+1).

2. *Caso Inductivo*: Sea *T* un árbol construido inductivamente, con un nodo raíz *N* y *k* árboles $T_1, T_2, \ldots, T_k$. Suponemos que vale $P(T_1), P(T_2), \ldots, P(T_k)$ (hipótesis inductiva) y tenemos que demostrar que vale $P(T)$ (tesis inductiva).

En general**,** $P(T_i)$ nos dice que el árbol T_i tiene un nodo más que la cantidad de aristas, es decir, que T_i tiene $n_i = a_i + 1$ nodos.

Pero, por definición inductiva de T, sabemos que T tiene un nodo raíz *N* y todos los nodos de los árboles T_i (i=1,…,k), o sea que tiene $1 + n_1 + n_2 + \ldots + n_k$ nodos.

También sabemos que T tiene k aristas que se agregaron explícitamente en la definición inductiva, más las aristas de los árboles T_j (i=1,…,k), o sea que tiene $k + a_1 + a_2 + \ldots + a_k$ aristas.

Tenemos que probar que vale $P(T)$, o sea que T tiene un nodo más que la cantidad de aristas, es decir, tenemos que probar: $1 + n_1 + n_2 + \ldots + n_k = (k + a_1 + a_2 + \ldots + a_k) + 1$

Pero: $1 + n_1 + n_2 + \ldots + n_k$

= {por hipótesis inductiva}

$1 + (a_1 + 1) + (a_2 + 1) + \ldots + (a_k + 1)$

= {por propiedad asociativa y conmutativa de la suma en IN}

$1 + a_1 + a_2 + a_k + 1 + 1 + \ldots + 1$

= {sumamos k veces el número 1}

$1 + a_1 + a_2 + \ldots + a_k + k$

= {por propiedad asociativa y conmutativa de la suma en IN }

$k + a_1 + a_2 + \ldots + a_k + 1$

Luego, hemos demostrado por **inducción estructural** que todo árbol T tiene un nodo más que la cantidad de aristas.

Otro de los temas fundamentales abordados en la asignatura son las **relaciones de equivalencia.** En este trabajo queremos mostrar, en particular, una de las aplicaciones de las relaciones de equivalencias en computación que se trabajó con los alumnos de Introducción al Algebra: las llamadas “Pruebas de software”.

La prueba de software consiste en determinar cómo operan los componentes de un sistema en situaciones representativas y verificar si su comportamiento es el esperado, es el proceso de ejecución de un programa con la intención de descubrir algún error. Como no es posible hacer una prueba exhaustiva, se realizan pruebas de unidades. Un tipo especial de prueba es la prueba de clases de equivalencia El aspecto importante de las clases de equivalencia es que forman una partición del conjunto. Esto tiene dos implicaciones importantes para la prueba de programas: el hecho de que se representa el conjunto entero provee una forma de completitud, y el hecho de tener conjuntos disjuntos ayuda a evitar la redundancia. La idea de la **prueba de clases de equivalencia** es identificar los casos de prueba usando un representante de cada clase. Si las clases se eligen sabiamente, esto reduce ampliamente la redundancia entre los

casos de prueba. Por ejemplo, dado el problema de determinar si un triángulo es equilátero, isósceles o escaleno, seguro se tiene un caso de prueba para verificar si un triángulo es equilátero, ya que se puede elegir como entrada para un caso de prueba la tupla (5, 5, 5), por ejemplo. Si se elige este caso, no se esperaría aprender mucho de un caso de prueba como (6, 6, 6) o (10, 10, 10). Estos casos se tratarían de la misma forma que el primero, y así son redundantes.

La función *nextdate,* por ejemplo, ilustra muy bien la elección de la relación de equivalencia subyacente. Esta es una función de tres variables, *día*, *mes* y *año*, que dada una fecha retorna la fecha del día siguiente. El domino de entrada consiste de los conjuntos *D*, *M* y *A*, definidos sobre los siguientes rangos:

$D = \{dia: 1 \leq dia \leq 31\}$; $M = \{mes: 1 \leq mes \leq 12\}$; $A = \{año: 1812 \leq año \leq 2012\}$

Dado un programa que implementa la función *nextdate*, *p*ara verificar que funciona correctamente, se pueden realizar distintos casos de prueba. Se puede particionar el dominio en las siguientes clases de equivalencia:

$D = \{día: 1 \leq día \leq 31\} \cup \{día: día < 1\} \cup \{día: día > 31\}$

$M = \{mes: 1 \leq mes \leq 12\} \cup \{mes: mes < 1\} \cup \{mes: mes > 12\}$

$A = \{año: 1812 \leq año \leq 2012\} \cup \{año: año < 1812\} \cup \{año: año > 2012\}$

Estas clases determinarán ciertos casos de prueba. Pero si se elige más cuidadosamente la relación de equivalencia, las clases serían más útiles. Se podrían mejorar estas clases, concentrándose en un tratamiento más específico. Dada una fecha… ¿Cómo se calcula la siguiente fecha?. Si no es el último día del mes, la función simplemente incrementa el valor del día. Al final del mes, el siguiente día es 1 y el mes se incrementa. Al final del año, el día y el mes se setean en 1, y el año se incrementa. Finalmente, el problema de los años bisiestos hace que sea interesante determinar el último día de un mes. Con todo esto en mente, se pueden construir las siguientes clases de equivalencia:

$$D = D_1 \cup D_2 \cup D_3 \cup D_4$$

$$M = M_1 \cup M_2 \cup M_3$$

$$A = A_1 \cup A_2 \cup A_3$$

$D_{1=}$\{dia: $1 \leq$ dia ≤ 28\} $D_{2=}$\{dia: dia = 29\} $D_{3=}$\{dia: dia = 30\} $D_{4=}$\{dia: dia = 31\}	M_1=\{mes: mes tiene 30 dias\} M_2=\{mes: mes tiene 31 dias\} M_3=\{mes: mes es Febrero\}	A_1=\{año: año = 1900\} A_2=\{año: $1812 \leq$ año ≤ 2012 ^ año=0 mod 4\} A_3=\{año: $1812 \leq$ año ≤ 2012 ^ año $\neq 0$ mod 4\}

Esto no es un conjunto perfecto de clases de equivalencia, pero su uso servirá para revelar errores potenciales. La clave en la prueba de clases de equivalencia es la elección de la relación de equivalencia que determina las clases. Mientras mejores sea las clases, mejor van a ser los casos de prueba que sirvan para detectar errores en los programas.

Si bien en este trabajo hemos profundizado en dos vínculos importantes entre álgebra y computación, debemos destacar que en la asignatura hemos tratado de relacionar otros temas básicos con cuestiones específicas de Computación. Por ejemplo:

- Las **relaciones n-arias**, en general, cumplen un rol muy importante en Base de Datos. Para representar la información en una Base de Datos se utiliza un modelo relacional, que consiste en una colección de tablas (relaciones n-arias) para representar los datos y las relaciones entre esos datos. En este sentido, en la asignatura, se plantearon situaciones donde intervenían relaciones n-arias definidas sobre conjuntos con diferentes tipos de objetos (no necesariamente números) y las **operaciones entre dichas relaciones**.
- Las **relaciones de orden** tienen importancia en diversas áreas de computación, por ejemplo, en los tipos abstractos de datos, cuando se necesita definir un orden entre los elementos: árboles binarios de búsqueda, diccionarios, etc. y también en Ingeniería de Software, en lo que tiene que ver con la planificación temporal de proyectos. El método PERT, por ejemplo, es básicamente un método para analizar las tareas involucradas en completar un proyecto dado, especialmente el tiempo para completar cada tarea, e identificar el tiempo mínimo necesario para completar el proyecto total. En este sentido, es que hemos planteado situaciones que involucran relaciones de orden definidas sobre conjuntos de tareas que se requieren para completar un determinado proyecto

- La **clausura transitiva** de una relación tiene muchas aplicaciones prácticas en Computación, ya que se utiliza para calcular la alcanzabilidad en un grafo. Es por esto que se

hizo especial hincapié en este tema en Algebra, planteando diferentes tipos de situaciones donde surgía la necesidad de determinar la clausura transitiva de una relación dada y su visualización a través del grafo correspondiente a la misma.

- La **coordinabilidad de conjuntos** se utiliza en la Teoría de Computabilidad. En este sentido, se analizó el hecho de que existen infinitos problemas a resolver e infinitos programas que implementan una solución. Como, además, |Problemas| = |IR| y |Programas| = |IN| y se sabe que |IR| es mayor que |IN| entonces se cumple que |Problemas| es mayor que |Programas|. Esto quiere decir que hay más problemas que programas, y por lo tanto, existen problemas que no se pueden resolver algorítmicamente, y su solución no se puede encontrar automáticamente por medio de una computadora.

- En la asignatura también se aborda en profundidad el estudio del **conjunto de los números enteros Z**, y en particular la **relación de divisibilidad**. Respecto de este tema hemos hecho hincapié en la importancia de esta relación y de la "**primalidad**" en lo referido a la seguridad informática, más precisamente en la criptografía, referente a la transmisión de mensajes codificados para no poder ser descifrados por un ajeno al sistema. La idea fundamental es que los algoritmos que se conocen para saber si un número es o no primo precisan demasiado tiempo, lo cual parece indicar que si se envía un número primo (o compuesto) lo suficientemente grande, un posible receptor indeseado del mensaje no va a ser capaz de saber si el número es primo o extraer sus factores. También se ha trabajado **la relación de congruencia** ligada a la idea de que en computación muchas veces, no se puede trabajar con números grandes y en su defecto se trabaja en Z_2 (clase de congruencia módulo 2) y se realizan las operaciones en este conjunto, en lugar de realizarlas en Z.

3. Conclusiones

A través de este trabajo hemos querido mostrar como es posible abordar ciertos temas de Algebra en estrecha relación con situaciones extraídas de contextos computacionales.

Si bien somos conscientes de que son muchas las conexiones entre el álgebra y la computación y que, en realidad, desde el punto de vista didáctico, lo óptimo sería trabajar de una manera co-disciplinar en la que el abordaje de determinadas situaciones requieran de la necesidad del aporte de las distintas disciplinas para su estudio, creemos haber avanzado un paso respecto de años anteriores. Es claro que no es posible realizar una evaluación inmediata de nuestra propuesta, porque las consecuencias de esta manera de abordar los contenidos sólo

pueden medirse a largo plazo y de una manera cualitativa. Sin embargo estamos convencidos de que es positivo realizar esta vinculación entre contenidos, ya que permite al alumno construir un significado diferente de los objetos involucrados, dado que se incluyen nuevos sistemas de prácticas, con nuevas situaciones estrechamente ligadas a las ciencias de la computación, acompañadas de un lenguaje diferente en algunos casos. Pero también sabemos que es necesario seguir trabajando en este sentido, tanto desde la asignatura que nos compete: Introducción al Algebra, como desde las asignaturas específicas de las carreras de Computación, a fin de que el alumno logre una relación diferente con el saber.

4. **Referencias bibliográficas**

- BLANCO, J., SMITH, S., BARSOTTI, D. 2008 Cálculo de programas, Impreso por la Facultad de Matemática, Astronomía y Física de la Universidad Nacional de Córdoba.

- CHEVALLARD, Y.(2004) Hacia una didáctica de la codisciplinariedad. Notas sobre una nueva epistemología escolar. Traducción de « Vers une didactique de la codisciplinariété. Notes sur une nouvelle épistemologie scolaire » realizada por Marianna Bosch. *IUFM d'Aix-Marseille & UMR ADEF*.

- GRASSMANN, W.K., TREMBLAY, J.P 1998 Matemática Discreta y Lógica, Prentice Hall, Madrid

- HOPCROFT, J.- MOTWANI, R., ULLMAN, J. 2001 Introduction to Automata Theory, Languages, and Computation. Addison Wesley, Second Edition

- JORGENSEN, P. 1995 Software Testing: A Craftsman's Approach. CRC Press,

- ROSEN, Kenneth H. Discrete Mathematics and Its Applications. McGraw-Hill, Sixth Edition