MateFun: Functional Programming and Math with adolescents

Alejandra Carboni, alejandra.carboni@psico.edu.uy[1,2], Victor Koleszar, vkoleszar@psico.edu.uy[1,2], Gonzalo Tejera, gtejera@fing.edu.uy[1,3], Marcos Viera, mviera@fing.edu.uy[1,3], and Javier Wagner, javier.wagner@gmail.com[4]

[1]Centro Interdisciplinario en Cognición para la Enseñanza y el Aprendizaje - Universidad de la República
[2]Centro de investigación básica en psicología - Facultad de Psicología - Universidad de la República
[3]Instituto de Computación - Facultad de Ingeniería - Universidad de la República
[4]Colegio Nacional José Pedro Varela

***Abstract*—The MateFun project arises with the intention of approaching disciplines that intersect in the field of education, based on the transversality of information and communication technologies with respect to Engineering, Communications, Psychology, Teaching and Pedagogy. In Uruguay, computer courses are increasingly being integrated into curricular content, since learning to program seems to be part of the skills needed for today's people. However, programming in general is absent in mathematics teaching, contrary to the inherent relationship of both. In turn, according to 2015 data from the Educational Monitor of the National Administration of Public Education of Uruguay, Mathematics is the least approved subject of the basic secondary cycle. MateFun is a functional programming language, accessible from a web application, especially aimed to math functions learning. It is intended that through MateFun the learning of programming strengthens the appropriation of the concept of mathematical function, at the same through this project we seek to generate scientific evidence of the transfer or the contributions of programming learning to math. Preliminary data show that the adolescents who experimented with MateFun had similar learning to the control group in mathematical functions, but they also acquired basic knowledge of functional programming.**

***Keywords*—Educational Math, Functional Programming, Educational Aplications**

I. Introducción

Las Tecnologías de la Información y la Comunicación son cada vez más accesibles. En Uruguay a partir de la implementación del Plan Ceibal[1], la disponibilidad de este tipo de recursos en escuelas y liceos del país es muy amplia. Al mismo tiempo los cursos sobre las Ciencias de la Computación se integran cada vez con mayor frecuencia a los contenidos curriculares, ya que aprender, entender, usar algoritmos y programar parecen ser parte de las habilidades tecnológicas necesarias de la actualidad y el futuro.

Por otro lado, la enseñanza y aprendizaje de la matemática es tema de preocupación en la actualidad. Según datos de 2015 del Monitor Educativo Liceal[2] de ANEP (Administración Nacional de Educación Pública), Matemáticas es la materia menos aprobada del Ciclo Básico, y a nivel internacional los resultados de las pruebas PISA no revelan avances en los resultados en esta área [1]. Algunos estudios hacen énfasis en el punto clave de la transición de la aritmética al álgebra. Estos muestran que en la transición se presentan grandes desafíos cognitivos para los estudiantes que deben alcanzar las nociones propias del álgebra simbólica [2], [3]. Nuestros sistemas educativos prevén la enseñanza del álgebra a partir de la secundaria, por lo que la transición parece ser un punto clave en el aprendizaje de la misma.

Varios autores mencionan que los estudiantes tienen dificultades con la estructura matemática en lo relativo a aspectos simbólicos (álgebra) y esto refleja las dificultades que los mismos ya tienen en el sistema de numeración [4]. Butto y Rojano [5] afirman que para lograr un buen desarrollo del pensamiento algebraico, es imprescindible que los estudiantes puedan pensar y entender la simbología y las operaciones aritméticas. Por tanto, será sobre un consolidado pensamiento aritmético que podrán construirse las nociones básicas del álgebra. Otras investigaciones también han descrito las dificultades o errores que cometen los estudiantes al iniciarse o trabajar en el álgebra [6], por ejemplo relacionados al aprendizaje del símbolo de igual cómo una señal para presentar el resultado de una operación [7].

En general, la programación está ausente en la enseñanza de la matemática, contrariamente a la relación inherente de ambas (e.g. las nociones de algoritmo o función son claves tanto en matemática como computación). A pesar de que las primeras experiencias de uso educativo integrado de ambas son de fines de los 60, con el uso de lenguajes como el LOGO [8], esto no ha perdurado en el tiempo como práctica educativa. En este trabajo se propone retomar estas ideas, particularmente se busca que el aprendizaje de la programación fortalezca la apropiación del concepto de función matemática y viceversa. Para esto introducimos MateFun, un lenguaje funcional puro diseñado para el aprendizaje de funciones, que tiene y utiliza una terminología y sintaxis similar a las de la matemática.

El paradigma de programación funcional está muy relacionado con la matemática. Sin embargo, los lenguajes funcionales de propósito general, como por ejemplo Haskell [9], contienen muchos conceptos (e.g. *type-classes*, *tipos algebraicos*, *alto orden*, etc.) que dificultan su inmediata apropiación

[1]https://www.ceibal.edu.uy
[2]http://servicios.ces.edu.uy/monitorces/servlet/portada

por parte de los estudiantes. El objetivo de MateFun es ser un lenguaje simple, que permita además que los estudiantes experimenten y desarollen el pensamiento matemático y el pensamiento computacional de forma interactiva a través de una interfaz web. Esto lo consiguen implementando soluciones creativas a problemas matemáticos asociados al concepto de función.

El artículo esta organizado de la siguiente manera. En la Sección II se presenta el marco teórico. En la Sección III se introduce el lenguaje MateFun, mientras que en la Sección IV se describe una intervención en aula que se realizó utilizando el lenguaje con adolescentes de segundo año de secundaria. Finalmente, en la Sección V concluimos y presentamos líneas de trabajo futuro.

II. Marco Teórico

La relación entre Matemática y Programación que se encuentra en la literatura es muy variada. Statz [10] plantea que el aprendizaje de la programación impacta en la habilidad para la resolución de problemas, mientras que Howe [11] ya plantea una relación positiva entre la capacidad de programar y las habilidades matemáticas. Se sostiene también que la programación incrementa la experimentación matemática, disminuye la brecha de abstracción por el entorno que facilita el aprendizaje de conceptos como variable, función, entre otros, y posibilita la noción de recursión que es una generalización de la inducción [12]. Otros autores afirman que el aprendizaje basado en el diseño de videojuegos ayuda a explorar y entender conceptos matemáticos en la interpretación de problemas a través de escenarios y objetos de los juegos [13], [14].

En la literatura podemos ver que desde la aparición y avance de la accesibilidad a medios digitales, se han desarrollado algunas herramientas para facilitar el aprendizaje del álgebra matemático. Desde los años 80 algunos Entornos de Aprendizaje Interactivo (Interactive Learning Enviroments, ILE) han sido diseñados para el álgebra, encontramos en la literatura intervenciones o aplicaciones educativas como el ALGEBRA-TUTOR [15] o el PAT [16] para ecuaciones simples. Alguna de estas herramientas han evolucionado y se han actualizado. Hoy encontramos entornos que se utilizan para el aprendizaje del álgebra, basados en Sistemas Algebraicos Computacionales (Computer Algebra System, CAS), que generalmente se utilizan para la resolución de ecuaciones, en algunos casos permite visualizar las soluciones paso a paso, o también la representación gráfica de funciones. Como ejemplos encontramos el DERIVE, o el APLUSIX [17]. Otra categoría de herramientas relacionados con entornos virtuales han sido los que buscan asociar elementos matemáticos con representaciones físicas. En este caso se busca por ejemplo a través de balanzas virtuales mostrar el significado de las equivalencias, o a través del cálculo de áreas realizar operaciones con polinomios [18]. Dentro del mismo marco, podemos visualizar también aplicaciones de entretenimiento que tienen fines educativos. Recientemente se ha desarrollado el juego/programa DragonBox Algebra [19] que pretende facilitar el aprendizaje de la resolución de ecuaciones lineales.

Todas estas herramientas son ejemplo de aplicaciones de las TIC para trabajar con álgebra pero que no incluyen el aprendizaje de la programación. Para la enseñanza de la programación encontramos distintas tecnologías y lenguajes, como el uso de actividades CS-unplugged, aplicaciones para tablets o celulares, lenguajes imperativos o funcionales, programación en bloques, entre otros. Se llama CS-unplugged a un conjunto de actividades educativas diseñadas para introducir principios fundamentales de la ciencia de la computación (CS) sin usar computadoras [20]. También existen ejemplos directamente basados en la enseñanza de lenguajes de programación.

Los lenguajes de programación permiten comunicar a un computador tareas a realizar. De acuerdo al enfoque que se utilice para diseñar soluciones, que se determina en los conceptos y formas de abstracción, los lenguajes de programación se pueden clasificar en distintos paradigmas, de los cuales se destacan el imperativo y declarativo. El paradigma imperativo es el más antiguo, dado que es el más cercano a la arquitectura del computador, y el más utilizado. Este paradigma establece que un programa consiste de un estado (variables del programa) y una serie de instrucciones que realizan cambios al mismo. Por otro lado, en el paradigma declarativo, en lugar de describir paso a paso la solución a un problema, el foco se encuentra en describir el problema, declarando ciertos aspectos que lleven a su solución.

La programación funcional es un enfoque de programación declarativa, vinculado al cálculo lambda, que utiliza el concepto de función matemática. El cálculo lambda fue desarrollado por Alonzo Church definiendo la noción de función computable. La programación funcional se basa en el llamado a funciones como elemento central [21].

El uso de lenguajes de programación funcionales ofrece la posibilidad de una asociación directa con el álgebra, dada la similaridad de conceptos y sintaxis [22]. La programación en lenguajes funcionales se basa en cómo las funciones, variables y valores funcionan en matemáticas [23]. Es decir que las funciones son una asignación a partir de valores de entrada a valores de salida [24]. En los lenguajes imperativos los programas también manipulan variables y funciones, pero las variables en estos lenguajes no representan un valor, sino que son un lugar para almacenar valores, y dicho contenido puede variar en cualquier momento. Por lo tanto las funciones en lenguajes imperativos son conceptualmente muy distintas a las algebraicas [25]. Desde que se empezaron a utilizar este tipo de herramientas que vinculan la programación y las matemáticas, el enfoque mayoritario era que el estudiante logre transferir los conceptos de un campo al otro para poder mejorar la comprensión de un concepto particular ya sea en matemática o en programación [25]. Si bien este enfoque ha tenido resultados en algunas áreas de la matemática, como la geometría por ejemplo, en lo relativo al álgebra las investigaciones se han centrado más en las herramientas de análisis y representación de funciones, no tanto en la programación.

Como antecedente de este trabajo encontramos lo realizado por Schanzer [25], basado en la herramienta Bootstrap, que busca que los estudiantes apliquen conceptos de álgebra para programar un videojuego. Si bien el diseño experimental no es totalmente controlado, el autor afirma que los estudiantes y docentes mejoran su desempeño en los test de ansiedad matemática luego del uso de esta herramienta.

```
conj Mes = { Enero , Febrero , Marzo
           , Abril , Mayo , Junio
           , Julio , Agosto , Setiembre
           , Octubre , Noviembre , Diciembre }

conj Rno0 = { x en R | x /= 0 }
```

Figura 1: Ejemplos de conjuntos definidos por el usuario

III. MateFun

En esta sección introducimos el lenguaje MateFun, un lenguaje de programación funcional pura (i.e. las funciones no tienen efectos laterales) dirigido al aprendizaje de funciones matemáticas. MateFun puede ser accedido a través de un entorno de programación integrado web[3] que permite programar, gestionar y ejecutar programas, y visualizar gráficas y figuras. Cuando los jóvenes se inician en el concepto de función se espera que puedan comprender los conceptos de dominio, codominio, evaluación y definición de funciones. Es así que los programas en MateFun requieren de la determinación del dominio y codominio de la función, y por último la definición de la función. Luego, a través del intérprete del lenguaje pueden evaluar funciones y graficar algunas de ellas.

III-A. El Lenguaje

La sintaxis del lenguaje fue diseñada con el objetivo de ser minimal y cercana a la notación utilizada en matemáticas. Se busca que el lenguaje se pueda asimilar rápidamente y que su relación con los conceptos matemáticos subyacentes pueda ser reconocida por los usuarios. Considerando que el público objetivo está compuesto por estudiantes de secundaria de habla hispana, las palabras clave se definieron en español.

La sintaxis del lenguaje se describe en EBNF (Extended Backus–Naur form) [26] omitiendo los espacios.

Un programa es una lista de definiciones de conjuntos y funciones.

```
prg  = {cdef | fdef};
```

III-A1. Conjuntos: Se utilizan para determinar el dominio y codominio de las funciones, es decir que corresponden a los tipos de los lenguajes de programación.Utilizamos esta terminología (conjuntos en lugar de tipos) para acercarnos más al concepto de función matemática. En la Figura 1 se muestran algunos ejemplos de definiciones de conjuntos. Un conjunto se define usando la palabra reservada **conj**, se le asigna un nombre y los elementos que contiene.

```
cdef = "conj",cname,"=","{",(enum | comp),"}";
```

El nombre de un conjunto (cname) es una cadena de caracteres alfanuméricos que debe iniciar con un caracter alfabético en mayúscula.

Los elementos se pueden expresar por enumeración o por comprensión. Para definir un conjunto por enumeración se deben enumerar, separados por coma, los elementos (cadenas alfanuméricas que comienzan con mayúscula) que lo integran.

```
enum = {ename, ","}, ename ;
```

Por ejemplo, Mes en la Figura 1 es un conjunto definido por enumeración que contiene los elementos Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio , Agosto, Setiembre, Octubre, Noviembre y Diciembre.

Para definir un conjunto por comprensión se debe establecer un conjunto de partida y la condición que deben cumplir los elementos de dicho conjunto para pertenecer al conjunto que se está definiendo. Por lo tanto el nuevo conjunto será subconjunto del conjunto de partida.

```
enum = {ename, ","}, ename ;
```

Los conjuntos pueden ser elementales o compuestos. Los conjuntos elementales son los primitivos (**R** para los reales, **Fig** para las figuras y **Color** para los colores) y los enumerados definidos por el usuario. Notar que la cantidad de conjuntos primitivos es menor de lo usual en los lenguajes de programación (e.g. no hay tipos para los caracteres, booleanos, punto flotante, etc.), esto se debe a la decisión desde el diseño de que sea minimal, de manera que la cantidad de conceptos a aprender para poder utilizar el lenguaje sea la menor posible. También son elementales los conjuntos definidos por comprensión que tienen como conjunto de partida un conjunto elemental. Los conjuntos compuestos son las n-tuplas de conjuntos y las secuencias de un conjunto.

```
conj = "R" | "Fig" | "Color" | cname
     | {conj,"X"},conj,"X",conj | conj,"*" ;
```

La n-tupla es la generalización del producto cartesiano. Por ejemplo la tupla **R** X **R** contiene pares de reales, que se escriben entre paréntesis y separados por coma; un elemento de este conjunto es (2.3,4.2) . La secuencia es lo que en los lenguajes de programación se suele llamar lista[4]; i.e. el conjunto de las secuencias de elementos de un conjunto dado. Una posible secuencia de reales (**R**$*$) es 1.8**:**9.3**:**2.4**:[]**.

Una condición puede ser una relación binaria entre dos expresiones (exp) o una lista de estas relaciones, entre paréntesis y separadas por comas. Las relaciones binarias son las usuales de igualdad y orden. En el caso de tener una lista de relaciones, el resultado de su evaluación es la conjunción lógica ($\wedge$) entre todas ellas. No se incluyen operadores lógicos debido a que agregan complejidad y son conceptos que en general se abordan sólo lateralmente en secundaria.

```
cond = rel | "(",{rel,","},rel,")"
rel  = exp,rop,exp
rop  = "==" | "/=" | "<" | ">" | "<=" | ">="
```

Más adelante mostraremos como se definen las expresiones, pero sin necesidad de entrar en detalle podemos ver que el conjunto RNo0 de la Figura 1 representa al conjunto de los reales (**R**) distintos de 0. Notar que las variables, a diferencia de los nombres y elementos de conjuntos, comienzan con minúsculas.

III-A2. Funciones: Para definir una función se debe indicar su signatura y la ecuación que la define.

[3]http://www.cicea.ei.udelar.edu.uy/investigacion/linea-de-investigacion-tics/

[4]De nuevo se toma la terminología matemática.

```
cuad :: R -> R
cuad(x) = x * x

inverso :: RNo0 -> R
inverso(x) = 1 / x

areaTria :: R X R -> R
areaTria(base,alt) = (base * alt) / 2
```

Figura 2: Ejemplos de funciones definidas por el usuario

```
fdef = sig,ecu
```

La signatura se compone del nombre de la función, el conjunto dominio y el codominio.

```
sig  = fname,"::",conj,"->",conj
```

A diferencia de otros lenguajes funcionales similares, no se cuenta con inferencia de tipos; dado que consideramos importante que el estudiante exprese explícitamente los conjuntos dominio y codominio de cada función. Por ejemplo, la signatura de la función cuad de la Figura 2 es cuad **:: R ->R**, indicando que la función cuad va de reales a reales. En la signatura de la función areaTria podemos ver que para representar funciones de múltiples variables (en este caso dos reales) utilizamos tuplas como dominio. Notar que en inverso utilizamos como dominio el conjunto RNo0 definido en la Figura 1.

La ecuación se define dando el nombre de la función, las variables independientes (i.e. parámetros) y el cuerpo de la función.

```
ecu  = fname,"(",[{vname,","},vname],")",gexp
```

El cuerpo de una función se compone de una lista de expresiones, cada cual se ejecuta al cumplirse una condición (si no se cumple ninguna de las anteriores), seguida de una expresión por defecto, que se ejecuta si no se cumple ninguna de las condiciones.

```
gexp = {exp,"si",cond,"o"},exp
```

Una expresión puede ser un valor, una variable, una tupla (expresiones entre paréntesis separadas por comas) o la aplicación de un operador binario, unario o una función.

```
exp  = val | vname | "(",{exp,","},exp,")"
     | exp,bop,exp | uop,exp
     | fun,"(",{exp,","},exp,")"
```

Notar que no se incluyen expresiones condicionales (del tipo if−then−else) ni *pattern-matching*, la selección se realiza utilizando una notación similar a la usada para definir funciones por casos en matemáticas.

Un valor literal puede ser un elemento de un conjunto definido por enumeración (enum), un número (num), un color (e.g. **Rojo**), una figura vacía (**FigVacia**) o una secuencia vacía ([]).

```
val  = ename | num
     | "Rojo" | "Verde" | "Azul" | "Negro"
```

```
abs :: R -> R
abs(x) =  x si x >= 0
       o -x

max :: R X R -> R
max(x,y) = x si x >= y
         o y

dias :: Mes -> R
dias(m) = 31 si m == Enero
        o 28 si m == Febrero
        o 31 si m == Marzo
        o 30 si m == Abril
        o 31 si m == Mayo
        o 30 si m == Junio
        o 31 si m == Julio
        o 31 si m == Agosto
        o 30 si m == Setiembre
        o 31 si m == Octubre
        o 30 si m == Noviembre
        o 31
```

Figura 3: Ejemplos de funciones definidas por casos

```
    | "Blanco" | "Gris" | "Amarillo"
    | "FigVacia" | "[]"
```

Los operadores son los operadores aritméticos (definidos para **R** y todos sus sub-conjuntos), más la proyección de una tupla (**!**) y el insertar al inicio de una secuencia (**:**).

```
uop  = "-"
bop  = "+" | "-" | "*" | "/" | "^"
     | "!"| ":"
```

Si volvemos entonces a las funciones definidas en la Figura 2, podemos ver que en el caso de la función cuad, la variable independiente es x y el cuerpo la expresión x ∗ x. Por lo que dicha función retorna el cuadrado del número real al que se la aplique.

En la Figura 3 se muestran ejemplos de funciones definidas por casos. Las condiciones de cada caso se analizan en el orden en que están definidos (de “arriba” hacia “abajo”) y una vez que se cumple una de ellas, se evalúa la expresión asociada y no se analizan más casos. Para asegurar que las funciones sean totales en el dominio, el lenguaje impone la existencia de un caso por defecto. Entonces, la función abs calcula el valor absoluto de un número real y la función dias retorna la cantidad de días que (generalmente) tiene un mes[5]. Hay dos aspectos importantes a destacar sobre la función dias que tienen que ver con las decisiones de diseño del lenguaje. El primero es que la función no es numérica, esto es importante porque uno de los objetivos de enseñar funciones mediante la programación es reforzar la idea de que no todas las funciones son numéricas. El otro es la *verbosidad* de la función, que es

[5]Queda como ejercicio para el lector escribir una función que tenga en cuenta a los años bisiestos.

```
areaCirc :: R -> R
areaCirc(r) = 3.14 * cuad(r)

factorial :: R -> R
factorial(x) = x * factorial(x-1) si x > 0
             o 1
```

Figura 4: Ejemplos de funciones con aplicación de funciones

un precio a pagar debido a la simplicidad del lenguaje; en este caso por la falta de expresiones condicionales y operadores booleanos. Esto no nos resulta particularmente preocupante, dado que priorizamos el rápido aprendizaje en lugar de la utilidad del lenguaje para la implementación de aplicaciones de mediano o gran porte.

Como vimos en la especificación de exp, una expresión puede ser también la aplicación de una función, que se realiza escribiendo el nombre de la función (fun) y entre paréntesis, separados con comas, sus argumentos; i.e. las funciones no están currificadas. Las funciones pueden ser las declaradas por el usuario o las funciones primitivas, definidas para operar con números, secuencias, colores y figuras.

```
fun  = fname | fnum | fsec | fcol | ffig
```

Por ejemplo, en el cuerpo de la función areaCirc de la Figura 4 se aplica la función cuad a la variable r. Por lo tanto, esta función calcula el área de un triángulo dado su radio r. La función factorial tiene una definición recursiva, es decir que se llama a sí misma.

Además de los operadores aritméticos que ya vimos, el lenguaje provee de funciones primitivas para hallar el seno, coseno y raíz cuadrada de un número.

```
fnum = "sen" | "cos" | "raizcuad"
```

III-B. Secuencias

De manera de poder manipular *colecciones de elementos* de una forma simple, el lenguaje incorpora la noción de secuencia. Las secuencias se definen de manera inductiva, con el valor **[]** de secuencia vacía y el operador **:** que a partir de un elemento y una secuencia de elementos (todos pertenecientes al mismo conjunto), retorna la secuencia resultante de agregar el elemento al principio de la secuencia dada. Por ejemplo, utilizando secuencias podemos definir la función tienen, que dado un número de días, retorna todos los meses que tienen ese número de días (y la secuencia vacía si ningún mes los tiene):

```
tienen :: R -> Mes*
tienen(d) = Abril:Junio:Setiembre:
            Noviembre:[] si    d == 30
          o Enero:Marzo:Mayo:Julio:
            Agosto:Octubre:
            Diciembre:[] si    d == 31
          o Febrero:[]   si ( d == 28
                            , d == 29)
          o []
```

```
suma :: R* -> R
suma(l) = 0 si l == []
        o primero(l) + suma(resto(l))

largo :: R* -> R
largo(l) = 0 si l == []
         o 1 + largo(resto(l))

conj RSeqNV = { l en R* | largo(l) /= 0 }

maximo :: RSeqNV -> R
maximo (l) = primero(l) si resto(l) == []
           o max( primero(l)
                , maximo(resto(l)))
```

Figura 5: Ejemplos de funciones recursivas sobre secuencias

Además de esa forma de construcción de secuencias, la función **rango** permite construir secuencias de racionales a partir de un valor inicial, un valor final y un valor de paso. Por ejemplo **rango**(0,100,10) retorna la secuencia 0**:**10**:**20**:**30**:**40**:**50**:**60**:**70**:**80**:**90**:**100**:[]**.

```
fsec = "rango" | "primero" | "resto"
```

Para poder obtener los elementos de una secuencia existen las funciones *destructoras* **primero** y **resto**, que respectivamente retornan el primer elemento de una secuencia y la secuencia resultante de quitar el primer elemento. Por ejemplo la expresión **primero**(**rango**(0,100,10)) retorna el valor real 0 y **resto**(**rango**(0,100,10)) retorna la secuencia 10**:**20**:**30**:**40**:**50**:**60**:**70**:**80**:**90**:**100**:[]**. Notar que en estos ejemplos hemos realizado *composición de funciones*, dado que a **primero** y **resto** les hemos pasado el resultado de aplicar la función **rango**.

Dada la naturaleza inductiva del conjunto secuencia, es natural que muchas funciones que operan sobre secuencias se definan usando recursión. En la Figura 5 se muestran algunos ejemplos de este tipo de funciones. La función suma, para obtener la suma de una secuencia de reales, suma el primer real de la secuencia con el resultado de la suma del resto (considerando que la secuencia vacía suma 0). De forma similar se puede calcular el largo de una secuencia. Para la función maximo, que calcula el máximo de una secuencia de reales no vacía, definimos un nuevo conjunto RSeqNV, de manera que la misma no sea parcial en su dominio. Notar que dentro de las condiciones de los conjuntos se pueden utilizar funciones definidas en el mismo programa.

III-C. Colores, Figuras y Animaciones

Con el objetivo de que los estudiantes puedan crear aplicaciones visuales, y por lo tanto más motivantes para el adolescente, se agregaron al lenguaje funciones que permiten la creación y manipulación de figuras.

Además de los colores predefinidos, se pueden obtener nuevos colores utilizando la función **rgb**, que a partir de tres

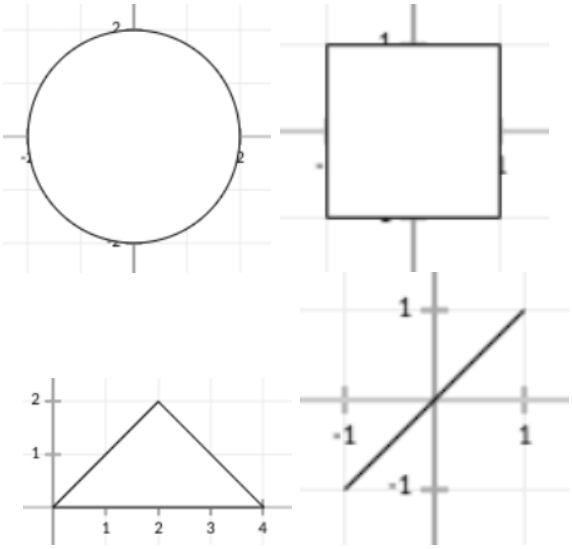

Figura 6: Ejemplos de figuras

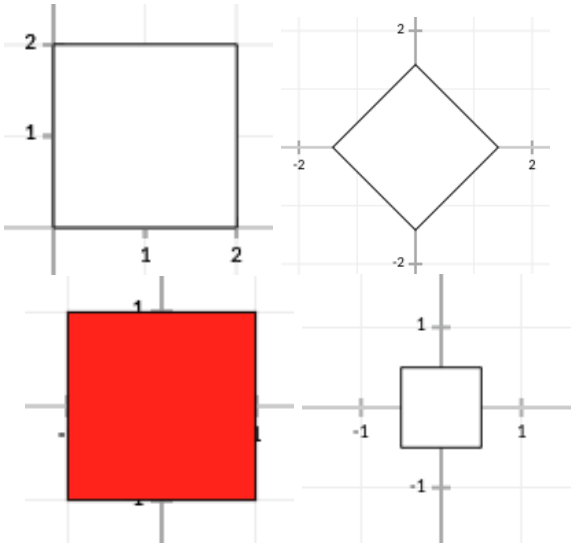

Figura 7: Transformaciones de figuras

números entre 0 y 1 que indican respectivamente el aporte de rojo, verde y azul (*red*, *green*, *blue*), compone el color resultante.

```
fcol = "rgb"
```

La función **aFig** retorna una figura con el texto de un enumerado, **rect** retorna un rectángulo a partir de su base y altura, **circ** retorna un rectángulo a partir de su radio, **linea** retorna un segmento de recta dados dos puntos (**R** X **R**) y **poli** retorna un polígono que une la secuencia de puntos (**R**$*$) que recibe como parámetro. En la Figura 6 se muestran las figuras generadas por las expresiones **circ** (2), **rect** (2,2), **poli** ((0,0) **:** (2,2) **:** (4,0) **:** **[]**) y **linea** ((−1,−1),(1,1)). Las figuras se dibujan en un plano cartesiano; las generadas con **aFig**, **rect** y **circ**, se colocan con centro en el punto (0,0).

```
ffig = "aFig"
     | "rect" | "circ" | "linea" | "poli"
     | "juntar" | "color"
     | "mover" | "rotar" | "escalar"
```

Dos figuras se pueden unir en una nueva figura utilizando la función **juntar**. A una figura se la puede pintar con un color dado utilizando la función **color**. También a una figura se le puede mover, rotar y escalar utilizando las funciones homónimas. En la Figura 7 se muestran las figuras generadas por las expresiones **mover**(**rect** (2,2),(1,1)) , **rotar**(**rect** (2,2),45) , **color**(**rect** (2,2), **Rojo**) y **escalar**(**rect** (2,2),0.5) .

En la Figura 8 se muestra las figura generada por la expresión **juntar**(**color**(**circ** (4), **Rojo**),**rect** (6,1.5)) , que une un círculo rojo con un rectángulo. También se muestra el efecto de mover esa figura al punto (1,1) haciendo **mover**(**juntar**(**color**(**circ** (4), **Rojo**),**rect** (6,1.5)),(1,1)) .

En MateFun, una animación se define como una secuencia de figuras. Entonces por ejemplo la siguiente función produce una animación que rota una figura dada de a 10 grados tantas veces como se le indique.

```
rotFig :: Fig X R -> Fig*
rotFig(fig, cant)
```

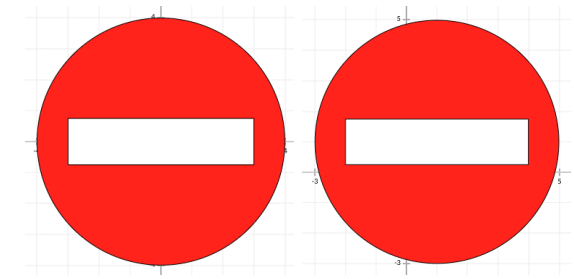

Figura 8: Juntar figuras

```
= [] si cant <= 0
o fig : rotFig(rotar(fig,10),cant-1)
```

III-D. El Intérprete

MateFun es un lenguaje interpretado, es decir que un programa es cargado por un intérprete para que éste realice su ejecución. En los programas MateFun no existe una función principal, que actúe como punto de partida de la ejecución, sino que al cargar un programa todas sus definiciones quedan disponibles en el intérprete para ser utilizadas en su línea de comandos. Tampoco existen funciones de entrada/salida, la interacción con el usuario se produce a través del intérprete.

Entonces, si consideramos que los códigos de las figuras 1, 2, 3, 4 y 5, juntos forman un programa llamado Ejemplos. Al cargarlo en el intérprete tendríamos disponibles, además de las funciones, operadores y conjuntos primitivos, todas las funciones y conjuntos que se definieron en esas figuras. El comando más básico del intérprete consiste en evaluar expresiones, las cuales pueden hacer uso de todas las definiciones disponibles. Un ejemplo de sesión válida sería la de la Figura 9. Esto permite que el estudiante pueda experimentar con la creación y evaluación de expresiones, así como analizar los resultados de aplicar ciertas funciones a distintos argumentos, construyendo por ejemplo tablas de valores.

El entorno de programación integrado web provee de una ventana gráfica en la cual se exhiben las figuras y animaciones

Figura 9: Intérprete

en las que resulten las expresiones de este tipo. Las figuras 6, 7 y 8 son capturas de esta ventana.

Para el caso de funciones numéricas de una sola dimensión (i.e. con dominio y codominio **R** o un subconjunto de **R**) se puede utilizar esa ventana para visualizar sus gráficas. Esto se realiza con el comando `?grafica` del intérprete. Por ejemplo

```
Ejemplos>?grafica cuad
```

genera la gráfica de la Figura 10. Esto permite que el estudiante analice las funciones desde el punto de vista de su representación gráfica.

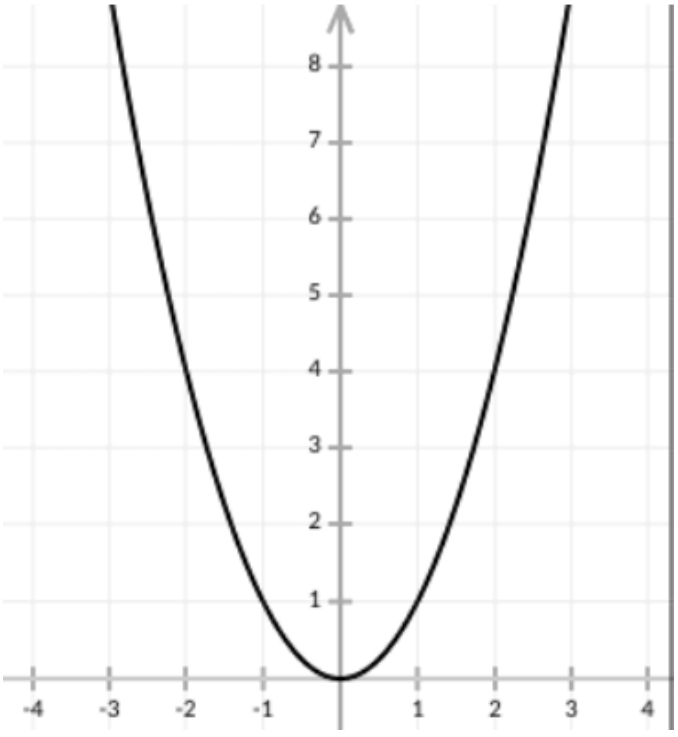

Figura 10: Gráfica de función

III-D1. Chequeo de Errores: Al momento de realizar la carga del programa en el intérprete se realiza un chequeo de errores; i.e. se realiza un chequeo estático.

Además del análisis sintáctico, que se realiza al parsear el programa, se realiza un análisis semántico para chequear unicidad y existencia de los nombres declarados y utilizados, así como la equivalencia y compatibilidad de los conjuntos utilizados. En caso de existir errores el programa no se carga y se despliegan en el intérprete los mensajes de error correspondientes.

En una primera pasada se verifica que no existan definiciones duplicadas de conjuntos y funciones. El siguiente programa, que llamamos `Errores1`, contiene una serie de errores de este tipo.

```
1  conj C1 = { Elem1 }
2  conj C1 = { Elem2 }
3
4  foo :: R -> R
5  foo (x) = x
6
7  foo :: R -> R
8  foo(x) = x + 1
```

Si intentamos cargar este programa en el intérprete se despliegan los siguientes errores, notificando la definición duplicada de C1 y foo.

```
Cargando Errores1
Error:  El conjunto C1 se definio mas de una vez.
Definido en:
- {archivo: Errores1 linea: 1 columna: 1}
- {archivo: Errores1 linea: 2 columna: 1}
Error:  La funcion foo se definio mas de una vez.
Definida en:
- {archivo: Errores1 linea: 4 columna: 1}
- {archivo: Errores1 linea: 7 columna: 1}
```

Si no existen funciones ni conjuntos duplicados se pasa a chequear que no existan definiciones duplicadas de elementos de un conjunto y argumentos de una función,que coincida el número de argumentos con el dominio de la función y que los conjuntos, variables y funciones utilizadas estén definidas. El siguiente programa contiene esta clase de errores.

```
1  conj C1 = { Elem1, Elem1 }
2
3  foo :: R X R -> R
4  foo(x,x) = x
5
6  bar :: R -> R
7  bar(x,y) = x + 1
8
9  baz :: J -> R
10 baz(x) = qux(x) + y
```

Por lo que al intentar cargarlo se desplegarían los siguientes mensajes de error.

```
Cargando Errores2
Error: {archivo: Errores2 linea: 1 columna: 1}
 En la definicion del conjunto C1 elementos
 duplicados: ["Elem1"].
Error: {archivo: Errores2 linea: 3 columna: 1}
 En la definicion de foo argumentos duplicados: ["x"].
Error: {archivo: Errores2 linea: 6 columna: 1}
 En la definicion de bar el numero de argumentos no
 coincide con el dominio de la funcion: 1 contra 2.
Error: {archivo: Errores2 linea: 9 columna: 8}
```

```
 El conjunto J no esta definido.
Error: {archivo: Errores2 linea: 10 columna: 10}
 La funcion  qux no esta definida.
Error: {archivo: Errores2 linea: 10 columna: 19}
 La variable y no esta definida.
```

En conjunto con los chequeos anteriores, también se verifica que las expresiones (y sub expresiones) retornen valores pertenecientes a conjuntos compatibles con los especificados. Esto es lo que en lenguajes de programación se llama chequeo de tipos. En nuestro caso no hay inferencia de tipos, dado que en MateFun siempre se debe especificar la signatura de las funciones. Algunos ejemplos de errores son los siguientes.

```
conj C = { Elem1, Elem2 }

foo :: R -> R
foo(x) = x + Elem1

bar :: C -> R
bar(x) = x + 2

baz :: R -> C
baz(x) = x
```

```
Cargando Errores3
Error: {archivo: Errores3 linea: 4 columna: 14}
 Se esperan elementos de R pero se encontro C.
Error: {archivo: Errores3 linea: 7 columna: 10}
 Se esperan elementos de R pero se encontro C.
Error: {archivo: Errores3 linea: 10 columna: 10}
 Se esperan elementos de C pero se encontro R.
```

En el primer y segundo caso, la suma espera que sus operandos sean reales, pero uno de ellos pertenece al conjunto C. En el tercer caso el resultado de la función debe pertenecer al conjunto C, pero la expresión que la define retorna un real.

Para el caso de los conjuntos definidos por comprensión, la situación es un poco más compleja, dado que la pertenencia o no de un elemento a un conjunto depende de la evaluación de una expresión que tenga en cuenta a este y otros elementos. Se puede decir que estamos ante un caso (simplificado) de tipos dependientes [27]. Algunos chequeos se pueden realizar de forma estática si las expresiones son constantes. Para identificar la mayor cantidad de casos posibles realizamos una propagación de constantes sobre el programa. Entonces por ejemplo en el siguiente programa.

```
conj RNo0 = { x en R | x /= 0 }

foo :: R -> RNo0
foo(x) = 0 + 0

bar :: R -> RNo0
bar(x) = foo(x)
```

La expresión 0+0 se puede reducir a 0 en tiempo de carga y detectar el error de no pertenencia a RNo0.

```
Cargando Errores4
Error: {archivo: Errores4 linea: 4 columna: 10}
 El valor 0 no pertenece al conjunto RNo0
 porque no se cumple: [0 /= 0].
```

Notar que la función bar también cae en el mismo error, pero el mismo no puede ser detectado al momento de la carga. Para resolver esto MateFun emplea una estrategia híbrida de chequeo de tipos estático y dinámico. El chequeo de errores como el que inserta la función bar se pospone a tiempo de ejecución.

Como se puede ver en el siguiente programa, las funciones bar y baz pueden resultar en errores en tiempo de ejecución. Esta estrategia también se utiliza para permitir la sobrecarga de operadores y funciones, por ejemplo, el operador + que está definido para los reales, puede ser utilizado en su subconjunto RNo0.

```
conj RNo0 = { x en R | x /= 0 }

foo :: RNo0 -> R
foo(x) = x

bar :: R -> RNo0
bar(x) = x

baz :: RNo0 X RNo0 -> RNo0
baz(x,y) = x + y
```

Si habilitamos la funcionalidad de que, además de los mensajes de error, se desplieguen mensajes de advertencia, al cargar el programa anterior se desplegarán los siguientes mensajes.

```
Cargando Advs
Advertencia: {archivo: Advs linea: 7 columna: 10}
 El conjunto RNo0 requerido es subconjunto
 del resultante R.
 Por lo que existe la posibilidad de que su valor
 quede fuera del conjunto.
Advertencia: {archivo: Advs linea: 10 columna: 12}
 El conjunto RNo0 requerido es subconjunto
 del resultante R.
 Por lo que existe la posibilidad de que su valor
 quede fuera del conjunto.
```

A continuación se muestran algunas expresiones para las cuales se detectan errores al momento de ser ejecutadas.

```
Advs>bar(0)
Error: {archivo: Advs linea: 7 columna: 10}
 El valor 0 no pertenece al conjunto RNo0
 porque no se cumple: [0 /= 0].
Advss>baz(1,-1)
Error: {archivo: Advs linea: 10 columna: 12}
 El valor 0 no pertenece al conjunto RNo0
 porque no se cumple: [0 /= 0].
Advs>baz(0,3)
Error: {Interprete columna: 5}
 El valor 0 no pertenece al conjunto RNo0
 porque no se cumple: [0 /= 0].
```

En los casos de bar(0) y baz(1,−1) el error es que su resultado no pertenece al conjunto declarado como codominio. El caso de baz(0,3) es distinto, y por eso el error se identifica como perteneciente al intérprete, dado que el error consiste en que intentamos aplicar la función a un elemento que no pertenece a su dominio.

IV. Trabajo en aula

Si bien no es el enfoque principal del artículo, a continuación se describen algunos puntos que formaron parte de la introducción piloto de la herramienta MateFun al trabajo en aula en educación secundaria. Se realizó una intervención con 40 adolescentes de 13.3 años (sd=.45) de edad promedio, pertenecientes a dos grupos de segundo año, para evaluar la interacción de los estudiantes con la plataforma y el lenguaje, y los efectos sobre el aprendizaje de funciones matemáticas. Se trabajó con un diseño quasi experimental de evaluación pre-post con una intervención de 8 instancias de 80 minutos cada una en promedio, a lo largo de un mes. En dicha intervención el grupo experimental (N=20) realizó las actividades sobre programación funcional, y el grupo control tuvo el mismo tiempo de intervención con enseñanza de funciones matemáticas de la forma tradicional.

Se diseñó una prueba junto a los profesores de Matemática del curso para evaluar el desarrollo y aprendizaje del pensamiento algebraico centrado en las nociones de función. La misma consideró aspectos no formales como por ejemplo la función como relación, pero también la capacidad de comprensión y manejo de los símbolos, habilidades para la generalización, organización y representación de aspectos vinculados a funciones matemáticas.

La intervención del grupo experimental consistió en una serie de 8 tareas a las cuales se accedía por una plataforma tipo Moodle. Las primeras actividades fueron diseñadas con el objetivo de familiarizarse con la plataforma, la interacción con el intérprete y las sintaxis de MateFun. Para eso podían desde realizar operaciones aritméticas en el intérprete hasta utilizar los comandos de ayuda o desplegar las funciones primitivas del lenguaje. Luego de esta etapa los estudiantes debían programar funciones e interactuar con diferentes conjuntos por ejemplo para componer funciones que hicieran figuras, modelaran o resolvieran problemas matemáticos. En la figura 11 se muestra uno de los problemas utilizados en las actividades.

Tanto los profesores como los estudiantes que participaron manifestaron que el período inicial de acercamiento y reconocimiento de la plataforma y del lenguaje fue un poco arduo. Los profesores tenían experiencia en lenguajes de programación imperativos, por lo que tuvieron que enfrentarse al cambio de paradigma. Por el lado de los estudiantes, el problema fundamental estuvo en lo simbólico, en entender cómo debían escribir y que tenían que seguir ciertas reglas que no conocían. También hay que tener en cuenta que tanto el lenguaje como su interfaz web estaban en etapa experimental al momento de desarrollarse la intervención, por lo que muchas de las dificultades atravesadas sirvieron de retroalimentación para implementar mejoras. Superados los obstáculos iniciales sobre el uso de MateFun y el paradigma de programación, el trabajo en matemáticas se desarrolló con naturalidad.

Se destaca sobre todo el uso de funciones vinculadas a obtener resultados gráficos que fue de mucha ayuda para la apropiación de la herramienta por parte de los estudiantes, pero también este componente visual, hizo que el resultado de la aplicación de las funciones fuera tangible y reconocible lo que facilitó el aprendizaje del concepto de función. Otros

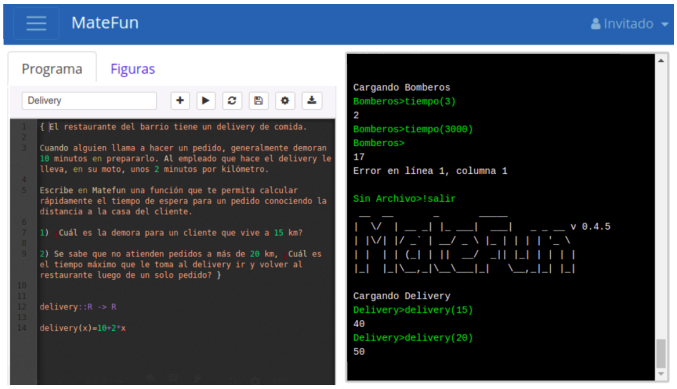

Figura 11: Entorno de la Plataforma y ejemplo de Actividades

elementos interesantes para tener en cuenta desde el punto de vista del aprendizaje de la programación fue la relativa facilidad para aplicar la recursión presentada como posibilidad de resolver problemas matemáticos.

Los resultados de la pruebas matemáticas diseñadas por los docentes para evaluar el aprendizaje de funciones matemáticas no mostraron diferencias significativas entre el grupo experimental y el grupo control. Una posible explicación es que la prueba no logró discriminar con precisión los avances ya que ambos grupos partieron de niveles altos de desempeño. Sin embargo los estudiantes que participaron de la experiencia piloto lograron avances similares al grupo que recibió la instrucción tradicional, a la vez que incorporaron nociones elementales de la programación funcional. Como trabajo futuro es necesario afinar el tipo de actividades de intervención así también como la incorporación de una evaluación del aprendizaje de programación funcional. Por otro lado la percepción tanto del docente como de los estudiantes sobre la incorporación de MateFun al trabajo del aula fue muy positiva.

V. Conclusiones

Tal como se describe en el apartado II Marco Teórico, el empleo de actividades de programación para apoyar el aprendizaje de matemáticas es un tópico de interés pero aún en desarrollo en el mundo. Los antecedentes en Uruguay son escasos y en este sentido el proyecto MateFun es una experiencia valiosa que propone una alternativa hacia la mejora de las prácticas de enseñanza y en particular de matemáticas y programación en educación media.

Entendemos que es particularmente valioso promover discusiones con una firme base teórica en las que se favorezca la interacción entre educadores e investigadores. Actualmente las actividades de programación suelen visualizarse como una excusa para el aprendizaje de la interacción con los aparatos. De aquí resulta un aprendizaje superficial con menos posibilidades de sentar bases para el desarrollo de pensamiento computacional, así como la de realizar “transferencias” a otras áreas del conocimiento. En este sentido es importante abrir espacios de intercambio que permitan avanzar hacia un nivel más profundo de aprovechamiento de la enseñanza de la programación.

Es importante enfatizar en que el paradigma de programación funcional y la sintaxis del lenguaje MateFun, muy cercana a la notación matemática, acortan la distancia entre la programación y la matemática que maneja el estudiante de secundaria. Esto permite que la herramienta funcione como puente de conceptos de ambas disciplinas. Otro punto a destacar del diseño del lenguaje es su sintaxis minimal, que sumada a lo anterior facilita su uso y compresión; esto fue observado en aula al momento que los jóvenes se iniciaban en el mundo de la programación de manera relativamente natural, luego de superar los inconvenientes normales de enfrentarse por primera vez a un lenguaje de programación textual. Además es importante puntualizar que la simpleza del lenguaje no va en desmedro de su expresividad a la hora de resolver problemas motivantes para el estudiante, que pueden incluir componentes gráficos y animaciones.

Los planes a futuro en el proyecto MateFun se concentran tanto en el uso como en el desarrollo del lenguaje. En cuanto al uso se está planificando una intervención y evaluación de mayor escala con estudiantes y docentes de secundaria, incluyendo ciclos más avanzados de educación secundaria, como bachillerato, para poder explotar otras funcionalidades. También planeamos realizar talleres con profesores de secundaria, tanto de matemática como de informática, de manera de incorporarlos al proyecto y mejorar el lenguaje y su interfaz con sus retroalimentaciones.

Las principales líneas de desarrollo de MateFun se están orientando a: (a) el desarrollo de la internacionalización del lenguaje, (b) integración a sistemas de gestión de aprendizaje (Learning Management System, LMS) y (c) mejoras en la visualización de gráficas 2D y 3D. Al incorporar la internacionalización se podrá traducir tanto las palabras clave como los mensajes del intérprete y de esta forma permitir cambiar el idioma de MateFun. Actualmente es posible trabajar con español e inglés. Respecto de la integración a LMS, si bien la aplicación web actual cuenta con algunos elementos en este sentidos (i.e definición de grupos, entrega de tareas y corrección) se espera aprovechar al máximo las capacidades del gestor de aprendizaje en relación a las necesidades propias de la interacción entre docentes y estudiantes al utilizar MateFun. Sobre la visualización de las gráficas, se puede decir que si bien en su versión actual se pueden graficar funciones continuas y discretas esto último se hace de una forma muy artificial, se plantea agregar el conjunto Z y permitir graficar sobre Z, R y cualquier enumerado, así como crear figuras 3D.

AGRADECIMIENTOS

Los autores quieren agradecer a Marie Noël Delger, directora del Colegio Nacional José Pedro Varela, por la coordinación que permitió el desarrollo de una intervención piloto en dicho colegio, a los docentes de cada grupo, y a Sylvia da Rosa, profesora del Instituto de Computación de la Facultad de Ingeniería, por el apoyo y las propuestas de actividades educativas. También, queremos agradecer a Gonzalo Cameto y Martín Méndez por el desarrollo de la aplicación web MateFun.

REFERENCIAS

[1] ANEP-PISA, "Uruguay en PISA 2015. Primer informe de resultados," 2016.

[2] D. Carraher, A. D. Schliemann, and B. M. Brizuela, "Early algebra, early arithmetic: Treating operations as functions," in *Presentado en the Twenty–second annual meeting of the North American Chapter of the International Group for the Psychology of Mathematics Education, Tucson, Arizona*, 2000.

[3] A. D. Schliemann, *Bringing out the algebraic character of arithmetic: From children's ideas to classroom practice*. Routledge, 2013.

[4] L. Linchevski and D. Livneh, "Structure sense: The relationship between algebraic and numerical contexts," *Educational studies in mathematics*, vol. 40, no. 2, pp. 173–196, 1999.

[5] C. Butto Zarzar and T. Rojano Ceballos, "Pensamiento algebraico temprano: El papel del entorno logo," *Educación matemática*, vol. 22, no. 3, pp. 55–86, 2010.

[6] L. Booth, D. Johnson, S. S. R. C. G. Britain), Strategies, and E. in Secondary Mathematics Project, *Algebra: Children' Strategies and Errors: A Report of the Strategies and Errors in Secondary Mathematics Project*. Nfer Nelson, 1984. [Online]. Available: https://books.google.com.uy/books?id=W5ROAAAAYAAJ

[7] N. M. McNeil, "U-shaped development in math: 7-year-olds outperform 9-year-olds on equivalence problems." *Developmental Psychology*, vol. 43, no. 3, p. 687, 2007.

[8] W. Feurzeig, S. Papert, M. Bloom, R. Grant, and C. Solomon, "Programming-languages as a conceptual framework for teaching mathematics," *SIGCUE Outlook*, vol. 4, pp. 13–17, 1970. [Online]. Available: http://dx.doi.org/10.1145/965754.965757

[9] S. Marlow, "Haskell 2010 Language Report," 2010.

[10] J. A. Statz, "The development of computer programming concepts and problem-solving abilities among ten-year-olds learning logo," 1973.

[11] J. A. Howe, P. Ross, K. Johnson, F. Plane, and R. Inglis, "Teaching mathematics through programming in the classroom," in *Computer Assisted Learning*. Elsevier, 1981, pp. 85–91.

[12] W. Feurzeig, S. A. Papert, and B. Lawler, "Programming-languages as a conceptual framework for teaching mathematics," *Interactive Learning Environments*, vol. 19, no. 5, pp. 487–501, 2011.

[13] Y. B. Kafai, *Minds in play: Computer game design as a context for children's learning*. Routledge, 2012.

[14] D. W. Shaffer, "Studio mathematics: The epistemology and practice of design pedagogy as a model for mathematics learning. wcer working paper no. 2005-3." *Wisconsin Center for Education Research (NJ1)*, 2005.

[15] J. R. Anderson, C. F. Boyle, A. T. Corbett, and M. W. Lewis, "Cognitive modeling and intelligent tutoring," *Artificial intelligence*, vol. 42, no. 1, pp. 7–49, 1990.

[16] K. R. Koedinger, J. R. Anderson, W. H. Hadley, and M. A. Mark, "Intelligent tutoring goes to school in the big city," 1997.

[17] J.-F. Nicaud, D. Bouhineau, and H. Chaachoua, "Mixing microworld and cas features in building computer systems that help students learn algebra," *International Journal of Computers for Mathematical Learning*, vol. 9, no. 2, pp. 169–211, 2004.

[18] T. Rojano, "Modelación concreta en álgebra: balanza virtual, ecuaciones and sistemas matemáticos de signos," *Números. Revista de Didáctica de las Matemáticas*, vol. 75, pp. 5–20, 2010.

[19] J. Liu, "Dragonbox: Algebra beats angry birds," *Wired, June*, 2012.

[20] T. C. Bell, I. H. Witten, and M. Fellows, *Computer Science Unplugged: Off-line activities and games for all ages*. Citeseer, 1998.

[21] G. Michaelson, *An introduction to functional programming through lambda calculus*. Courier Corporation, 2011.

[22] R. Lee, "Teaching algebra through functional programming: An analysis of the bootstrap curriculum," 2013.

[23] D. Wampler, *Functional Programming for Java Developers: Tools for Better Concurrency, Abstraction, and Agility*. .º'Reilly Media, Inc.", 2011.

[24] K. Hinsen, "The promises of functional programming," *Computing in Science & Engineering*, vol. 11, no. 4, 2009.

[25] E. T. Schanzer, "Algebraic functions, computer programming, and the challenge of transfer," Ph.D. dissertation, 2015.

[26] F. Miller, A. Vandome, and M. John, *Extended Backus-Naur Form*. VDM Publishing, 2010. [Online]. Available: http://www.ics.uci.edu/~pattis/misc/ebnf2.pdf

[27] P. Martin-Löf, *Intuitionistic type theory*, ser. Studies in Proof Theory. Lecture Notes. Bibliopolis, Naples, 1984, vol. 1, notes by Giovanni Sambin.