Facultad de Ingeniería
Universidad de la República

mac

Matemática Aplicada a Computación

Informe del proyecto

Proyecto Taller V

Cecilia Fernández Costa
Guillermo Fernández Costa

Tutora: Sylvia da Rosa

Instituto de Computación

RESUMEN

La introducción de material informático en cursos en general, y de matemática en particular, está cada vez más extendida.
Tradicionalmente el software es usado en cursos de matemática como material de apoyo con el cuál ilustrar ideas y conceptos y/o con el cuál producir resultados eficientes (cálculos, figuras, gráficos, etc).
De ese modo, ni la metodología, ni el contenido, ni los objetivos de los cursos, son significativamente alterados, ya que el material cumple un papel similar al de la regla y el compás en geometría.
En los últimos años, se ha comenzado a trabajar en otro sentido, que comprende la utilización de lenguajes de programación como herramienta de construcción y manipulación de conceptos matemáticos, intentando producir una transformación significativa en objetivos (para qué enseñamos) y metodología (cómo enseñamos), de los cursos de matemática, que permita analizar los problemas didácticos desde nuevas perspectivas.
Sin embargo, se puede observar que este proceso de integración, programación-matemática, tiene aspectos relativos al contenido (qué enseñamos) que apenas comienzan a ser abordados y que, a nuestro juicio, son de una importancia fundamental.
El principal de estos aspectos tiene que ver con la relación entre el enorme desarrollo que la Ciencia de la Computación ha dado a la Matemática Discreta y la casi ausencia de esta disciplina en el sistema educativo.
El objetivo específico de nuestro proyecto está enmarcado en un objetivo mucho más general, que pretende seguir avanzando en el sentido de lograr reveer tanto la metodología, y objetivos de los cursos de matemática liceales, como el contenido de los mismos. El objetivo es construir un lenguaje de programación que facilite el desarrollo de temas de matemática discreta en cursos de enseñanza de secundaria.
Nuestra tarea específica ha consistido en avanzar en la construcción de un prototipo en desarrollo de dicho lenguaje: MAC (Matemática Aplicada a Computación).

TABLA DE CONTENIDO

OBJETIVOS DEL PROYECTO

Introducción

El uso de material informático en cursos en general, y de matemática en particular, está cada vez más extendida. Tradicionalmente el software es usado en cursos de matemática como material de apoyo con el cuál ilustrar ideas y conceptos y/o con el cuál producir resultados eficientes (cálculos, figuras, gráficos, etc).
De ese modo, ni la metodología, ni el contenido, ni los objetivos de los cursos, son significativamente alterados, ya que el material cumple un papel similar al de la regla y el compás en geometría.
En los últimos años, se ha comenzado a trabajar en otro sentido, que comprende la utilización de lenguajes de programación como herramienta de construcción y manipulación de conceptos matemáticos, intentando producir una transformación significativa en objetivos (para qué enseñamos) y metodología (cómo enseñamos) de los cursos de matemática, que permita analizar los problemas didácticos desde nuevas perspectivas.
Sin embargo, se puede observar que este proceso de integración, programación-matemática, tiene aspectos relativos al contenido de los cursos (qué enseñamos) que apenas comienzan a ser abordados[13][14] y que, creemos, son de una importancia fundamental.
El principal de estos aspectos tiene que ver con la relación entre el enorme desarrollo que la Ciencia de la Computación ha dado a la Matemática Discreta y la casi ausencia de esta disciplina en el sistema educativo. El estudio y la investigación en el campo de la matemática discreta ha quedado restringido al ámbito académico y en el sistema educativo solo se ha incorporado en parte y con gran esfuerzo en la enseñanza superior.
En la facultad de Ingeniería, que imparte 9 títulos de grado, la mitad de los estudiantes que ingresan, lo hacen en carreras de computación, donde ***recién comienzan*** a contar con estudios en matemática relevantes para su carrera.

La formación que adquiere el estudiante en la etapa de la enseñanza secundaria es fundamental, desde los siguientes dos puntos de vista:

- constituye la base de conocimientos previos para abordar estudios universitarios y/o insertarse en el mercado laboral, y
- brinda conocimientos generales que le permitan comprender un mundo en continuo cambio.

Esta deficiencia mencionada en la base formativa de los estudiantes constituye un obstáculo importante no sólo para realizar estudios terciarios en computación, sino para la cultura general que brinda la enseñanza secundaria, aún a aquellos que no continúen con estudios universitarios.

La Ciencia de la Computación y sus aplicaciones, que tanto impacto tiene en la sociedad, debe estar presente en el sistema educativo, interviniendo a través de sus fundamentos matemáticos tanto en la constitución de los conocimientos previos para abordar estudios posteriores o posibilitar la inserción laboral, como brindando esos conocimientos generales que permitan entender el mundo. Realizar los cambios necesarios para ello, es responsabilidad del ámbito científico de la Ciencia de la Computación.
La imprescindible actualización del sistema educativo debe llevarse a cabo como una tarea interdisciplinaria, que cuente con aportes del campo matemático, del campo computacional y del campo educativo, al menos.

Objetivos

El objetivo de este taller, se enmarca dentro del objetivo general de construir un lenguaje de programación que facilite el desarrollo de temas de matemática discreta en cursos de enseñanza secundaria.
Nuestra tarea específica ha consistido en avanzar en la construcción de un prototipo de dicho lenguaje: MAC (Matemática Aplicada a Computación).

Se ha partido de un prototipo inicial desarrollado en el lenguaje funcional Hugs, que comprendía determinadas funcionalidades y una documentación mínima . El prototipo inicial permitía trabajar con conjuntos finitos, relaciones y funciones, al que luego en el año 2000 un grupo de estudiantes en su proyecto de grado, le agregó la representación gráfica de sus objetos, utilizando la biblioteca gráfica Graphics.

Para este proyecto se proponía inicialmente continuar con la extensión del programa, fundamentalmente agregando la posibilidad de definir conjuntos por comprensión y elaborar una documentación completa.

Debido al interés académico del proyecto y al nuestro propio, se planteó como objetivo secundario, la posibilidad de realizar una experiencia práctica de utilización de este lenguaje en un grupo de matemáticas de 5to. año de secundaria, dado que los resultados esperados del taller consistían justamente en contar con un software de fácil integración a cursos de matemática de la enseñanza media, sobre todo en lo que refiere a temas de Matemática Discreta.

Antecedentes y motivación

El problema de los bajos niveles en Matemáticas que alcanzan los estudiantes preuniversitarios es un hecho de honda preocupación, tanto de los actores implicados en la enseñanza secundaria (docentes de matemáticas y autoridades de secundaria), como de los actores universitarios, dado que en los contenidos de las asignaturas universitarias está presupuesto que los estudiantes ya han alcanzado el nivel de conocimiento y maduración que deberían brindar las matemáticas estudiadas en el liceo.
De diversas pruebas que se han venido realizando en los últimos años para evaluar el nivel de los estudiantes en asignaturas tales como matemáticas y física en Uruguay, tanto a nivel de secundaria como de algunas facultades, en particular la de Ingeniería, se ha constatado que se está ante un gran problema en esta área.
La facultad de Ingeniería viene realizando desde hace muchos años una prueba de matemáticas y física a la generación que cada año ingresa a facultad, para luego hacer un seguimiento de la generación durante su carrera universitaria, y poder extraer conclusiones respecto a si el contar con bajos niveles de ingreso determina en alguna medida el fracaso posterior en la carrera.
Esta prueba de ingreso, por sí sola, ya está indicando que los niveles preuniversitarios alcanzados en matemáticas son bajísimos. Se constata a través de los resultados de la prueba que los estudiantes no aprenden conceptos, sino aprenden las fórmulas para resolver determinados problemas y mecánicamente, aplican tales fórmulas. Si se cambia un poco la notación empleada, o el estilo clásico de planteo de un problema, son incapaces de resolverlo. Esto está indicando que no están realmente aprendiendo matemáticas, sino que son entrenados en la resolución de determinados problemas, planteados de determinada forma, utilizando un conjunto de fórmulas y procedimientos aprendidos mecánicamente, por repetición. Es sabido que las matemáticas juegan un papel muy importante en la capacidad de abstracción y de razonamiento de un individuo, es decir, van más allá del conocimiento puntual de qué es la derivada de una función, o el producto cartesiano de dos conjuntos. Por ese papel tan importante que juegan en la capacidad de razonamiento de una persona, en la estructuración mental y la capacidad de abstracción y generalización, es que es un tema que no debiera preocupar únicamente a aquellos que van a seguir alguna disciplina que requiera de conocimientos matemáticos, sino que es importante también para aquellas personas que van a seguir estudios en otras áreas, o incluso a quienes no van a seguir estudios de ningún tipo.
La matemática es una disciplina formadora, que interviene en la estructuración del pensamiento. Esa es su principal contribución al desarrollo intelectual y por eso su importancia. Sin embargo, la mayoría de los estudiantes encuentran que la matemática es algo aburrido y que no les sirve para nada, y que tienen que estudiarla porque está en el programa, pero no les despierta el menor interés, y muchas veces les provoca más miedos y angustias que gusto y desafío intelectual. Las matemáticas vienen siendo, históricamente, el "cuco" temido por un alto

porcentaje del estudiantado. Es común escuchar argumentos del estilo: "seguí humanístico porque odio las matemáticas".

Existe un nuevo enfoque metodológico en la enseñanza de matemáticas usado por investigadores en educación en matemáticas, quienes vienen trabajando en el uso de esta metodología. Este enfoque difiere del tradicional, en que aquí los conceptos matemáticos son introducidos por la vía de ejercicios, preguntas y actividades, de manera tal que el estudiante, sin haber leído acerca de la teoría, comience por pensar él mismo respecto al problema, interactuando y discutiendo con sus compañeros, probablemente obteniendo resultados utilizando alguna herramienta computacional (lenguaje de programación) y teniendo que explicar por qué cree que se producen los resultados que obtiene de utilizar el programa, y de esta forma vaya generando por medio de la intuición y el razonamiento, preconceptos que luego, cuando le sean presentadas las definiciones matemáticas, la teoría, los teoremas, las propiedades, le permitan un más profundo entendimiento de los mismos, que si éstos se le hubiesen presentado, como en la metodología tradicional, es decir, sin haber dedicado tiempo previo a tener una idea intuitiva de las cosas. Esto, creen quienes adhieren a esta nueva metodología, incentiva y despierta el interés de los estudiantes, eliminando aquello de que las matemáticas son aburridas, y generando una actitud positiva que provoque que tengan avidez por más matemáticas.
En el libro de Ed Dubinsky y William Fenton: "Introduction to Discrete Mathematics with ISETL"[1], se describe la aplicación de esta metodología, utilizando un lenguaje de programación matemático, es decir, un lenguaje especialmente diseñado para cursos de matemáticas: ISETL. En los "Comentarios para el Instructor" Ed Dubinsky y William Fenton dicen que han encontrado que los resultados de la aplicación de esta metodología en varios cursos, provee a los estudiantes de la oportunidad de lograr un más profundo entendimiento de los conceptos matemáticos que el que lograrían en cursos que emplearan una enfoque más tradicional.
Plantean también que no hay duda que algunos estudiantes no necesitan este tipo especial de pedagogía e interacción con la computadora que el texto soporta. Dicen que esos son individuos que tienen un talento especial en esta disciplina, y los denominan como " los atletas naturales" de las matemáticas. Sin embargo, dicen, la mayoría de los estudiantes necesitan alguna asistencia, alguna herramienta para ayudarlos a construir los conceptos matemáticos, y éste es el objetivo de ese texto y de esta metodología.
Por otro lado plantean que el hecho de poder determinar si los estudiantes en general siempre necesitarán del tipo de soporte que un enfoque pedagógico constructivista (como llaman a este enfoque) trata de proveerles, es una pregunta abierta. Dicen que puede ser que algunos estudiantes, luego de experimentar con un curso como este, desarrollen la habilidad de construir conceptos matemáticos en sus mentes escuchando una presentación del material, o leyéndolo de un texto, es decir, de una forma más tradicional.

Con respecto al hecho de utilizar un lenguaje de programación en los cursos de matemáticas, creemos que no es un aspecto menor. Para aprender matemática se requiere del conocimiento de un lenguaje especial. Este lenguaje se basa tanto en el lenguaje convencional (o natural) como en el simbólico. Los enunciados y las fórmulas matemáticas no son sólo símbolos, sino instrumentos para el razonamiento lógico, por lo que en matemática, la escritura simbólica pasa a ser más importante y poderosa que la escritura convencional. El poder de los símbolos es muy grande, y la vastedad de su poder ha quedado en evidencia, ya que más que facilitar y abreviar, permiten la generación de nuevos significados. En este sentido los símbolos permiten al pensamiento independizarse de todos los significados de los cuales van cargados. Ese lenguaje simbólico permite una considerable economía en los diferentes procesos de reflexión. La dificultad en la adquisición del lenguaje matemático es uno de los problemas que en el campo educativo aún no se han podido resolver. Aquí es donde se refleja la relevancia de la utilización de un lenguaje de programación.
Al incorporar un lenguaje de programación al ciclo de aprendizaje, el estudiante participa de forma activa en el mismo, lo que redunda en algunas ventajas:

- puede definir un objeto matemático, utilizando un lenguaje muy parecido al propio lenguaje matemático, y puede verificar si su definición es o no correcta

- puede realizar operaciones sobre objetos matemáticos definidos por él, y ver el resultado de tales operaciones, sin la necesidad de esperar que el docente le diga si lo que hizo es o no correcto
- le permite ser creativo e innovar planteándose él mismo nuevos ejemplos para probar y ver si el resultado es el que esperaba
- en cuanto a la notación, utilizando el método de "ensayo y error", el estudiante puede aprender a manejar la simbología matemática de una manera más práctica que en la teoría
- se pueden ver gráficamente los resultados de las operaciones, así como los significados de las diferentes propiedades de los objetos matemáticos y lograr así un más íntegro entendimiento de los conceptos
- se pueden asimilar mejor los conceptos básicos para usarlos como base al momento de tener que abstraer modelos más complejos.

Tomando como base por un lado la convicción de que los cursos de matemáticas a nivel liceal hacen un desmedido énfasis en la matemática continua, en desmedro de la matemática discreta, tan importante tanto en el campo informático como en el de la cultura matemática en general, y por otro lado basados en los defectos que encontraban en los lenguajes de programación matemática existentes (entre ellos Isetl, Matlab, etc.), así como las desventajas de la utilización de lenguajes de programación funcional como Hugs en este tipo de experiencias[8], los autores del prototipo inicial de Mac, Sylvia da Rosa y Gustavo Cirigliano, se embarcaron en la tarea de poner manos a la obra, y empezar la implementación de un lenguaje con la potencia de permitir trabajar con elementos de la teoría de conjuntos, poniendo énfasis en el enfoque de matemática discreta.
Para ilustrar este enfoque, recurrimos a un par de ejemplos, con respecto al lenguaje Isetl[1]:

La relación R formada por los pares ordenados (x,y) tales que x e y pertenecen al conjunto {1,2,3} y cumplen con la propiedad de que x es menor que y, puede definirse en ISETL así:

```
R:= {[1,2],[1,3],[2,3]};
```

La misma relación en MAC, solo puede ser definida indicando el dominio y codominio de la relación:

```
defr R: AxA; R = {<1,2>,<1,3>,<2,3>}
```

donde el conjunto A ha sido definido previamente. De hecho, MAC realiza chequeos que aseguren que el conjunto de tuplas especificado esté contenido en AxA. En caso contrario fallará, dando un mensaje de error que indicará donde se encuentra el problema.
Luego, en MAC es posible saber si la relación R es de equivalencia (simétrica, reflexiva y transitiva), e incluso hallar, por ejemplo, la clausura reflexiva de R, cuyo resultado es la relación de AxA cuyo conjunto de tuplas es: {<1,1>,<2,2>,<3,3>,<1,2>,<1,3>,<2,3>}.

Análogamente, en ISETL se puede definir una función como un conjunto de pares, pero tampoco es posible incluir en la definición de la función dominio y codominio de la misma. Por ejemplo, la función f(x) = 2x se puede definir de la siguiente manera:

```
f:= func(x);
return 2* x;
end;
```

Como se puede observar, no se incluye en la definición dominio y codominio de la función, con lo cual se está definiendo para todos aquellos argumentos x para los que esté definido 2*x. Esto impide chequear propiedades tales como la inyectividad, sobreyectividad, biyectividad de una función, y con ello saber si es invertible y este hecho, no solo no aporta a la claridad del concepto matemático de función, sino que por el contrario, oscurece, lo hace ambiguo. Creemos que esta ambigüedad proviene de considerar los conceptos dentro del campo de la

[1] Por una descripción más profunda de Isetl y su comparación con MAC, dirigirse al apéndice B

matemática continua, donde tradicionalmente no se indica dominio y codominio de las funciones ya que los mismos son en la mayoría de los casos, el conjunto de los números reales.

Es así que Mac surge no solamente como un lenguaje de programación para usar en cursos de matemáticas, sino que pretende además influir en la revalorización de la Matemática Discreta en la formación básica de los estudiantes[7][9]. Al mismo tiempo introduce a los estudiantes en el mundo de la computación, y en particular de la programación. Puede permitir ver bajo un mismo punto de vista, conceptos que pertenecen tanto a matemáticas como a programación (funciones, algoritmos, programas, pruebas, etc.), introduciendo, entonces, algunas ideas de la ciencia de la computación.

Mac permite fundamentalmente:

1. El manejo de definiciones y operaciones sobre la Teoría de Conjuntos.
2. La definición de funciones y relaciones debiendo indicar dominio y codominio de las mismas.
3. La posibilidad de definir funciones como métodos (notación similar al cálculo lambda)
4. Visualización de los objetos del lenguaje, utilizando representaciones basadas en los diagramas de Venn y representaciones matriciales de relaciones y funciones.
5. La utilización de funciones booleanas para chequear si los objetos del lenguaje cumplen determinadas propiedades (ej: saber si una función es o no inyectiva, si una relación es o no simétrica, si un elemento pertenece o no a un conjunto, etc.)

Tan importante como contar con un lenguaje apropiado, es el papel que éste tiene en el proceso educativo. Tradicionalmente el enfoque de los matemáticos y educadores que han venido de hace años trabajando en el tema es el considerar al lenguaje solamente como una herramienta que debe adaptarse al objeto de estudio. Nosotros coincidimos con los autores de MAC en la convicción de que la informática no solo aporta a las matemáticas la herramienta (el lenguaje de programación que les permita interactuar con una computadora y ver los resultados de las definiciones y operaciones que se realicen), sino que también aporta a la comprensión del objeto de estudio de las matemáticas: un lenguaje de programación es necesariamente formal, más formal que las propias matemáticas, y que puede influir incluso en las propias definiciones y conceptos matemáticos. Puede incluso (y lo ha hecho) dar nuevas perspectivas a los conceptos, verlos desde otro punto de vista y lograr una mayor formalización.

El punto de partida de nuestro proyecto

El prototipo inicial de MAC, que constaba de funcionalidades mínimas, fue retomado en el año 2000 por un grupo de estudiantes de ingeniería, quienes, como objetivo principal de su proyecto de taller V, le incorporaron al programa la posibilidad de graficar los objetos del lenguaje para los que tenía sentido realizar alguna representación de este estilo (conjuntos, relaciones, funciones).
Como resultado de su proyecto, además de agregarle al prototipo esta potencia gráfica de la que carecía hasta ese entonces, realizaron alguna documentación, entre la que se encontraba una Guía de Uso[10] donde aparecían documentados algunos "bugs" identificados y una lista de posibles mejoras a realizar.
Esta Guía de Uso introducía informalmente y en forma ambigua tanto las funcionalidades que soportaba el prototipo como la sintaxis del lenguaje.

El código del programa no se encontraba comentado, así como tampoco se incluía una documentación técnica completa.
La Guía Técnica[11] con la que contamos para empezar, contenía básicamente el código con algunos comentarios de los módulos que implementaban las características gráficas del lenguaje, y luego una lista de los otros módulos (los que constituían el prototipo inicial), con las interdependencias entre ellos y una breve descripción de lo que hacía cada uno.

Debido a que el objetivo del proyecto de taller anterior no involucraba tocar el prototipo inicial, sino extender sus funcionalidades incorporando la parte gráfica, es que el mismo no estaba prácticamente documentado.

Al momento de comenzar nuestro proyecto, recibimos la documentación descripta antes, junto con el informe del proyecto[9] y el programa que era la evolución del prototipo inicial, al que el grupo de taller anterior le agregó las características gráficas.

A continuación se listan los "bugs" detectados en el proyecto anterior y extraídos de la Guía de Uso[10]:

- Si se aplican los operadores "**esiny**" (es inyectiva?), "**esbiy**" (es biyectiva?) o "**essob**" (es sobreyectiva?) a una relación (objeto matemático que para MAC es de un tipo de datos base: el tipo relación) que no sea de A -> A el resultado es: "la relación debe ser de A -> A". El problema es que "**esiny**", "**esbiy**" y "**essob**" son operadores que se aplican sobre funciones y no sobre relaciones, por lo que el mensaje de error debiera ser otro.

- Si se aplican los operadores "**essim**" (es simétrica?), "**esref**" (es reflexiva?) o "**estrans**" (es transitiva?) a una función (objeto matemático que para MAC es de un tipo de datos base: el tipo función) siempre devuelve True.[2]

- Si se definen mediante línea de comandos dos conjuntos con igual nombre, (MAC tiene dos formas de ingresar definiciones al entorno o ambiente de trabajo: leyéndolas de archivos mediante la utilización de los comandos "**cargar**" y "**agregar**" o ingresándolas directamente en la línea de comandos utilizando los comandos y sintaxis correspondientes[3]) las definiciones quedan superpuestas, quedando incorrecta la definición de ambos conjuntos.

- Si se define una función escribiendo:

  ```
  deff f: {a} -> {a};
  f= [v1] a
  fo
  ```

  Entra en loop.
  (Lo anterior es una forma sintáctica permitida por Mac para definir que el elemento f es una función de dominio el conjunto compuesto por el string "a", codominio el mismo conjunto, y tal que la función se calcula, para cualquier valor del dominio, v1, como el resultado de evaluar la expresión "a". Es decir, es la función constante que toma siempre el valor "a". La forma de definición anterior es la de "función por abstracción" y es una notación similar a la del cálculo lambda. El "bug" entonces especifica que si se quiere definir la función constante anterior de la forma mencionada, entonces el programa entra en loop.)

En la misma Guía de Uso se señalaban las siguientes mejoras a introducir:

- Al agregar definiciones de objetos en el **entorno** o **ambiente**[4] a partir de un archivo, solo permite agregar definiciones de objetos con nombres nuevos. Si al leer el archivo para agregar definiciones se encuentra la definición de un objeto cuyo nombre coincide con el de otro objeto definido en el entorno, se deja de cargar desde el archivo,

[2] En particular, al comenzar a investigar los "bugs" anteriores, encontramos que este bug no estaba descripto correctamente: el resultado no era siempre True, sino era siempre el resultado de aplicar "essob" a funciones. Esto fue descubierto luego, al estudiar el código.

[3] Por una descripción de los elementos del lenguaje dirigirse al apéndice A

[4] Por una descripción del entorno o ambiente y de los elementos del lenguaje, ir al Apéndice A

agregando solo las definiciones anteriores a la del objeto de nombre duplicado y no se despliega ningún mensaje de error.

- No se pueden definir dos conjuntos con el mismo nombre.
  Si se definen dos conjuntos con el mismo nombre quedan los dos en el entorno, pero solo uno es alcanzable. Por lo tanto al ejecutar el comando "**mostrar**", el cual lista los objetos definidos en el entorno, se muestran ambas definiciones, aunque solo una tenga validez.

- Cuando se intenta definir una función utilizando la notación similar a la lambda ("funciones por abstracción"), si se incluye en la expresión correspondiente el operador de multiplicación, es necesario hacerlo a través del operador "x" y no el "*" que es el que se suele utilizar más.
  Ejemplo:

```
deff f:{1,2}->{2,4};
f = [v1]v1 x 2,
fo
```

  En lugar de escribir:

```
deff f:{1,2}->{2,4};
f = [v1]v1 * 2,
fo
```

  (Aquí se quiere definir la función f como en la siguiente definición matemática:
  f(x) = 2x)

- No puede definirse una función vacía por lo que al menos un elemento del dominio debe tener una imagen en el codominio[5]

- Cuando se aplica una función, debe colocarse entre paréntesis el argumento o la expresión completa. Así, para hacer: "f 1" se debe escribir: (f 1) o (f (1)).

- Al graficar la **composición** (operador matemático de composición de relaciones o funciones) en modo grilla, utilizando el comando "**dibujarM**" (realiza una representación matricial de una función o relación[6]), se muestra sólo una representación de la relación o función resultante. Podría utilizarse una representación similar a la del comando "**dibujar**" donde además del resultado, se muestran las funciones o relaciones intermedias.

- Posibilidad de definir conjuntos y relaciones por comprensión

- Permitir un método de representación de las relaciones de A->A en forma de grafo. Esta representación es útil para verificar si una relación es o no reflexiva.

Detectamos que algunas de las mejoras señaladas arriba, eran en realidad errores que se producían como consecuencia de un mal funcionamiento del programa y fueron corregidos, así como también los reportados como "bugs".
No se trabajó la parte gráfica, ya que estaba fuera de nuestros objetivos, y como explicamos en la próxima sección, si bien la posibilidad de definir conjuntos por comprensión estaba planteada dentro del alcance inicial del proyecto, finalmente no pudo ser incluida, por lo cuál la hemos incluido como mejora a futuro, para próximas versiones del programa.

[5] Las funciones podían ser parciales en la implementación de Mac recibida. Esto fue cambiado en nuestro proyecto, y las funciones deben ser totales, en correspondencia con lo que se da en secundaria.

[6] Por una descripción más detallada de este comando, así como de la operación de composición, dirigirse a los capítulos 2 y 3 del Manual de Usuario de Mac[12] entregado junto con este informe.

INCLUIRIMAGENINCLUIRIMAGENINCLUIRIMAGEN**NUESTRA PROPUESTA**

La planificación inicial del proyecto implicaba el testeo exhaustivo del programa de modo de detectar errores y corregirlos, y de forma tal de contar con un producto confiable y al mismo tiempo elaborar la documentación. Las condiciones de la misma eran, dejar al sistema bien documentado, en cuanto al código, y contar con un buen y completo manual de usuario, que permita a cualquiera aprender a utilizar el lenguaje simplemente leyendo el manual e interactuando con el programa.

Debido a nuestro interés en trabajar en áreas de aplicación de la informática en la educación, específicamente en el área matemática, dejamos planteada desde un comienzo la intención de realizar una experiencia práctica de aplicación de este nuevo lenguaje matemático a un grupo de 5to. año liceal que estuviera estudiando conceptos de Teoría de Conjuntos. Sabíamos que podíamos contar con el apoyo de algunos docentes de secundaria que se prestaran para una experiencia de este tipo, y que la realización de este tipo de experiencia nos acercaría mucho al objetivo general en el que está enmarcado el proyecto.

Requerimientos previos

Para realizar el proyecto fue necesario leer o investigar ciertos temas. Algunos de ellos en el comienzo del proyecto y otros en el transcurso del mismo.
En primer lugar, debimos repasar el lenguaje de programación funcional con el que estaba implementado el programa: Hugs.
Como la tarea fundamental del proyecto era corregir los "bugs" identificados del programa, y algunos de ellos tenían que ver con el "parser" del lenguaje, así como también se planteaba dentro del alcance la incorporación de nuevas funcionalidades, tuvimos que estudiar la construcción de "parsers" en lenguajes de programación funcional. Para ello recurrimos primeramente a las Lecture Notes: "Advanced Functional Programming" , en su capítulo 1 de Jeoren Fokker[2] que profundizaba en la definición y estudio de "combinadores de parsers" en lenguajes funcionales, para hacer lenguajes de programación.
Además bajamos de la Internet varios artículos también referentes a la construcción de "parsers" con programación funcional[3][4]. Este estudio nos posibilitó la compresión del "parser" de MAC para poder corregir los "bugs" identificados por el grupo de proyecto anterior en su Guía de Uso[10], en primera instancia, y luego, esa comprensión del "parser" nos condujo, en segunda instancia, a tomar la decisión de rehacer buena parte del mismo.

En segundo lugar, debimos consultar libros de matemática discreta[5], en busca de formalismos matemáticos, definiciones y ejemplos. De estos textos también sacamos ideas de nuevas funcionalidades, que no estaban implementadas en MAC, la mayoría de las cuales fueron incorporadas al programa posteriormente. También consultando estos libros encontramos que algunas de las funcionalidades ya implementadas debían ser modificadas, para que el programa se comportara de la forma que entendíamos correcta a partir de lo que se definía en estos textos.
El documento de usuario fue realizado teniendo como idea mantenerse apegados a esos formalismos matemáticos, por lo que fue de consulta asidua el siguiente libro: "Matemática discreta y lógica, con aplicaciones a la Ciencia de la Computación" de Grassmann y Tremblay[5].

Por último, para prepararnos para la experiencia en Secundaria, comenzamos a elaborar ejemplos acerca de cómo se escribiría un capítulo de un libro de texto para los alumnos de un curso de matemáticas discretas que utilizaran MAC[10].

HIPERVÍNCULO

[10] Estos ejemplos figuran en el apéndice C

Esta idea surgió al consultar el libro "Introduction to Discrete Mathematics with ISETL" de Fenton y Dubinsky[1]. Este es un libro de texto escrito para ser utilizado en cursos de matemáticas discretas, siguiendo el enfoque pedagógico que comentamos en la introducción de este informe, y que supone que el estudiante va a interactuar con el lenguaje de programación matemática ISETL, similar al lenguaje MAC, y del que ya hablamos anteriormente.
De este libro extrajimos la idea de incorporar a MAC funcionalidades que no habíamos contemplado anteriormente, y que en la lectura del mismo se nos presentaron como importantes, debido a que responden a conceptos matemáticos que es importante que los estudiantes comprendan y utilicen.

Otros materiales consultados fueron, un artículo no publicado de Sylvia da Rosa: "An experience with ISETL"[7], y material elaborado por Sylvia da Rosa para su trabajo en el grupo SecUTUniv[11] "Consideraciones sobre el curso de 5to. año del Bachillerato", tendiente a mostrar la importancia de las matemáticas discretas en la formación de los estudiantes.

Problemas y soluciones

Con respecto al planteo inicial, aproximadamente en la mitad del tiempo que debíamos dedicar al proyecto, surgió la siguiente disyuntiva, con respecto al camino a seguir de ahí en más:

- Continuar mejorando el programa, tanto en corrección de errores, testeo, incorporación de nuevas funcionalidades y documentación,
- Diseñar e implementar la funcionalidad que permitiera definir conjuntos por comprensión, o
- Realizar la experiencia con un grupo de 5to. año de secundaria

Nuestra elección fue la primera opción, a pesar de que la segunda estaba definida en el alcance inicial del proyecto y la tercera coincidía con nuestro interés personal.

Las razones de peso por las cuáles nos inclinamos a tomar el camino que finalmente seguimos tienen que ver con el hecho de que el proceso intenso de desarrollo en el que nos encontrábamos, tanto del prototipo como de la documentación, nos colocó en una situación de conocedores en profundidad del lenguaje. Esto nos permitía avanzar más y rápidamente para lograr un producto mucho mejor, que facilitara tanto continuar con su desarrollo como su aplicación al área de la enseñanza.
Por otro lado, la idea de la experiencia con un grupo liceal fue descartada, mediante una decisión tanto coyuntural como estratégica: nos encontramos con que las clases en secundaria habían finalizado, y aún no teníamos una versión lo suficientemente estable del programa, como para ser considerada como una versión beta.
Además, realizar una experiencia de este tipo de modo de obtener resultados relevantes, no es una tarea menor: hay que plantear un conjunto de ejercicios lo suficientemente completo y testado, que permitan utilizar el lenguaje para aprender en profundidad los conceptos subyacentes, hay que formar a los docentes en el uso de la herramienta, para que éstos preparen a los estudiantes, hay que recolectar resultados y evaluarlos

Principales problemas en el desarrollo

De lo descrito anteriormente, se desprende que nuestra tarea se concentró en la corrección y desarrollo de funcionalidades y en la documentación.

Nuestra primer tarea para entender cómo funcionaba el programa y cómo debía comportarse, fue probarlo como usuarios: es decir, con la Guía de Uso[10] al lado, fuimos aprendiendo sobre la herramienta. Esto nos permitió diseñar una sintaxis BNF para las expresiones y comandos válidos del lenguaje, como primera parte de la documentación. A partir de ahí pudimos realizar

[11] http://www.fing.edu.uy/secutuuniv

un testeo más profundo del programa, detectando errores no reportados antes, de los que se presentan algunos ejemplos:

- **eval # A U {1,2}**

  Aquí estamos queriendo evaluar el cardinal de la unión del conjunto A, previamente definido, con el conjunto conformado por los enteros 1 y 2. El resultado que devolvía MAC era: "error de tipo en los argumentos". El problema en este caso era que el operador cardinal (#) esperaba como operando un conjunto (elemento canónico) y no una expresión de conjunto como la formada por la unión de dos conjuntos, por lo que MAC evaluaba la expresión #A devolviendo un entero y luego trataba de realizar la unión de ese entero con el conjunto {1,2} resultando el error antes mencionado. Como puede verse en los diagramas de sintaxis que figuran en el Manual de Usuario[12] que entregamos con este proyecto, el operador cardinal es un operador unario que participa en la formación de expresiones de tipo entero. Su operando no tiene por qué ser un elemento canónico del tipo conjunto, sino que puede ser una expresión de conjuntos. Esto acompaña la idea de que cualquier operador puede aplicarse a expresiones del tipo de datos de sus operandos.

- **defr R = S o G**.

  Aquí estamos queriendo definir una relación R cuyo valor es el resultado de evaluar la expresión de tipo relación que resulta de componer otras dos relaciones previamente definidas. Si Mac permitía definir una variable de tipo conjunto, dándole como valor el resultado de evaluar una expresión de tipo conjunto, entonces también debía permitir definir cualquier variable utilizando expresiones a la derecha de la asignación.
  Sin embargo, el programa no permitía definir una relación como una expresión de relación. Ejemplo: no permitía definir una relación como la inversa de otra relación, o como la clausura reflexiva, simétrica o transitiva de otra. Esto mismo ocurría con las funciones.

- El "parser" aceptaba expresiones que no terminaran con un fin de línea, o un punto o una coma en el caso de cargar definiciones desde un archivo. Se "parseaba" hasta que la expresión fuera válida, sin importar lo que viniera después. Esto provoca resultados inciertos, ya que el resultado devuelto solo indica el resultado de evaluar hasta donde pudo "parsear" el programa.

- Si se definía en Mac la siguiente relación R:

  defr R: {1,2} -> {1,2}; R= {<1,2>, <2,2>}

  la cual claramente no es reflexiva ya que el entero 1 no está relacionado consigo mismo, o, en otras palabras, el par <1,1> no está en la relación, y luego se ingresaba en la línea de comandos:

  eval True And esref R,

  donde se quiere evaluar la expresión booleana (True And (esref R)), es decir, evaluar el resultado de aplicar el operador binario booleano And, al booleano True y al booleano resultante de evaluar la subexpresión esref R, el resultado daba True. Sin embargo, claramente la evaluación debería dar True si y solo si la relación R es reflexiva, y en este caso no lo es.

  Aquí ocurrían dos problemas:
  1. las operaciones de booleanos (And, Or, etc) solo esperaban como operandos los booleanos True y False y no expresiones booleanas, como la formada por esref R.

2. el parser identifica como válida la expresión: eval True, y por eso es que el resultado devuelto fue True (esto tiene que ver con el punto anterior).

- **defc A = { 1 , 2 , 3 , ({1,2}->{3,4}; {<1,3>,<2,4>}) }**

  Aquí se está queriendo definir un conjunto A cuyos elementos son los enteros 1, 2, 3 y una relación sin nombre. La única forma de utilizar objetos de tipo relación era definiendo una variable de ese tipo. Es decir, no podía hacerse algo como lo anterior. De hecho, si observamos el elemento de tipo relación que estamos queriendo incluir en el conjunto A, en realidad no sabemos si se trata de una relación o de una función. Esto muestra que no podían, pues, especificarse como elementos de un conjunto ni relaciones ni funciones sin nombre, es decir, sin estar definidas previamente en el entorno como variables de tipo relación o función. En cambio sí se permitía especificar como elementos de un conjunto otros conjuntos sin nombre (ej: defc A = {1,2,3,{"mas","menos"}}). Esto no era razonable, dado que tanto conjuntos como relaciones o funciones son objetos base del programa y debían por tanto tener exactamente el mismo comportamiento. De hecho, una de las características de MAC es que permite trabajar con funciones y relaciones como elementos de los tipos de datos correspondientes, como objetos análogos a los conjuntos, o a los enteros, así que esto debía ser corregido. Adicionalmente se cambió la notación para especificar los dominios de una relación (en lugar de separarlos por "->" como se hace con las funciones, a partir de nuestra versión se separan por "x", respondiendo al hecho de que ese es justamente el operador producto cartesiano, y es la notación matemática que se utiliza para las relaciones), de lo que no surge ninguna ambigüedad cuando se quiere definir una relación sin nombre como elemento de un conjunto.

- **defc U = { 1 , 2 , 3 }**

  El programa permitía definir un conjunto U, siendo U la letra que se utiliza para denotar la operación de unión de conjuntos, así como también permitía definir un conjunto de nombre **eval**, que es el comando utilizado para evaluar expresiones. En definitiva el programa no tenía palabras reservadas. Decidimos, pues, incluir un conjunto de palabras reservadas del lenguaje, y cada vez que se quiere definir una nueva variable en el entorno, se chequea que el identificador no esté en ese conjunto.

- Al definir una relación o función se deben hacer varios chequeos, tanto sintácticos como semánticos. Los errores sintácticos son los más comunes. Es fácil olvidar una llave o un punto y coma. Como ejemplos de errores semánticos podemos mencionar entre otros: un elemento del dominio de una función tiene más de una imagen en el codominio, o se está queriendo definir una función sin asignarle una imagen a todos los elementos del dominio (función total), o los elementos de las tuplas definidas en una relación no pertenecen a los dominios definidos. Hay una amplia gama de errores posibles, pero fuera cual fuera el error, el resultado en la versión de Mac que nos entregaron era: "error en ..." donde lo que seguía era la expresión completa que se había ingresado en la línea de comandos.
  Aquí no estamos hablando de que el programa se estuviera comportando mal, pero claramente es mejor poder ser un poco más específicos en los mensajes de error, y en la medida de lo posible dar al usuario información más precisa de lo que ocurrió.

- Era muy común, mucho más de lo medianamente tolerable, que el programa abortara su ejecución. Esto a veces ocurría al ingresar una expresión no esperada y no controlada, así como también si al escribir en la línea de comandos el usuario debía recurrir al backspace para corregir errores en el string tipeado. El programa se caía también cuando se intentaba cargar un archivo con definiciones donde el nombre del archivo que el usuario tipeaba no era un nombre de archivo existente. Esto era una de las cosas que debíamos arreglar, si queríamos que el programa fuera utilizado por alguien en el futuro. El programa no puede caerse ante entradas inesperadas, y menos

aún cuando el usuario se equivoca en lo que está tipeando y lo modifica con el backspace. Eran errores intolerables.

- **eval 5 – 2**
  Al ingresar el comando anterior en la línea de comandos, Mac devolvía "Resultado 2", es decir, el operador "-" estaba funcionando como división.

- **eval 2 * 2**
  Al ingresar el comando anterior en la línea de comandos, Mac devolvía "Resultado 1", es decir, el operador "*" estaba funcionando como And.
  Estos ejemplos muestran el gran problema que existía dentro del programa con los operadores.

La mayoría de los errores detectados señalaban deficiencias en el "parser", siendo la principal de ellas la existencia de una función genérica que realizaba el "parseo" de todas las expresiones válidas. Esto provocaba que se pudieran formar expresiones matemáticamente inválidas, combinando tipos de datos incompatibles.
Nuestra propuesta fue la de implementar distintas funciones para las distintas expresiones, de manera de tener un claro dominio de las expresiones que son válidas en el lenguaje.
Las alternativas eran: o mantener las estructuras internas de las expresiones, de los objetos base y del combinador de "parsers"[12], o dejar de lado todo esto y empezar de cero. Tomamos la primera opción ya que la otra era inviable. Prácticamente significaba tirar el programa anterior y empezar de la nada (incluyendo la parte gráfica, resultado del proyecto anterior, ya que la misma estaba basada en las estructuras internas de los objetos base y de las expresiones del programa) y estos cambios hubieran escapado a los plazos y al alcance original del proyecto.
De esta forma, perfeccionamos y extendimos la sintaxis BNF de las expresiones válidas y basándonos en el "parser" anterior, construimos el "parser" actual.

Resultados

Podemos dividir los resultados en dos grandes áreas:

1. el programa propiamente dicho, es decir el código, y
2. la documentación

1. Mejoras al programa

Se corrigieron los tipos de errores que ilustran los ejemplos mencionados anteriormente y los detectados en el proyecto anterior.
Se introdujeron nuevas expresiones válidas.
Se mejoró un poco la interfaz con el usuario, dejando espaciados entre prompts en la línea de comandos y mejorando la legibilidad de las salidas.
Se incluyeron nuevas funcionalidades, la mayoría de las cuáles se detallan a continuación:

- Posibilidad de definir una relación como el resultado de evaluar una expresión de relación, por ejemplo dadas R, G y H relaciones puedo definir:
  - defr R = G (defino la relación R igual a la relación G)
  - **defr R= G o H** (defino la relación R como la relación resultado de realizar la composición de las relaciones G y H)
  - **defr R= inv G** (defino R como la relación inversa de la relación G)

[12] Conjunto de funciones que sirven como base para la construcción de "parsers" de expresiones. Ejemplos de estas funciones: leer una cadena de caracteres (literal "eval"), o las nociones de secuencia de expresiones (exp1 seguido de exp2) o la noción de ó (exp1 ó exp2).

- Posibilidad de definir una función como el resultado de evaluar una expresión de función, por ejemplo dadas f , g y h funciones puedo definir:
  - **deff f = g** (defino la función f igual a la relación g)
  - **deff f = g o h** (defino la función f como la función resultado de realizar la composición de las funciones g y h)
  - **deff f = inv g** (defino f como la función inversa de la función g)

- Tipo Boolean.
  Se agregó la constante Boolean para representar este tipo de datos. Este es un conjunto con los valores True y False. Está definido en el entorno o ambiente cuando se inicializa el programa, y no puede ser redefinido, ni eliminado.
  Esto por ejemplo permite definir una función cuyo codominio sea Boolean, sin tener que especificar {True, False}.

- Definición de objetos booleanos (comando: **defb**).
  Como antes existía la posibilidad de definir conjuntos, relaciones y funciones en el entorno utilizando los comandos **defc**, **defr** y **deff** respectivamente, decidimos agregar la posibilidad de hacerlo con los booleanos, es decir, definir variables de tipo Boolean.
  Ejemplo: **defb B=True**

- No booleano (operador: **Not**).
  Se ha agregado el operador Not para representar el no booleano, antes solo se representaba con el "_" (infraguión).
  Por ejemplo, se puede definir una variable booleana A como No B, siendo B el de la definición anterior.
  Ejemplo: **defb A= Not B**

- Inclusión estricta de un conjunto en otro (operador: **inclEstricto**).
  Función que dados dos conjuntos retorna True si y solo si los conjuntos no son iguales y el primero es un subconjunto del segundo.
  Ejemplo: **eval {1,2} inclEstricto {1,2,3,4}**,
  cuyo resultado es True.
  Este operador no existía.

- Incluido igual (operador: **incluido**).
  Igual a la función anterior pero sin la restricción de igualdad. Este operador tampoco existía.

- Igualdad (operador: **==**) y distinto (operador: **!=**).
  Agregamos estos operadores de comparación, que no existían en la versión anterior de MAC. Ahora están definidos para todos los tipos base, excepto para las funciones. No creímos que fuera muy importante comparar funciones, razón por la cuál no incluimos este caso. Puede considerarse una mejora a futuro, cuya única dificultad radica en el hecho de que las funciones tienen actualmente dos formas de representarse. Por más detalles sobre la implementación de la igualdad de funciones mirar la sección de mejoras a futuro.

- Pertenencia (operador: **en**).
  Esta función retorna un booleano resultado de evaluar la pertenencia de un objeto base en un conjunto.
  Ejemplo: **eval "azul" en {"rojo", "amarillo", "azul"}**, cuyo resultado es True.
  Este operador no existía.

- Definición de Universo.
  Se ha incluido la posibilidad de definir un universo. Este es un concepto no menor en la teoría de conjuntos, ya que permite definir la operación de complemento de conjuntos.

Para definir el universo o conjunto universal se debe definir una variable de tipo conjunto y de nombre **universo**, ya que por defecto no hay definido un universo en el entorno. La palabra universo está reservada solo a estos fines por lo que cualquier otro uso que se le quiera dar dará lugar a un mensaje de error. Una vez definido el universo el programa no chequeará que todos los objetos base definidos en el ambiente previamente estén contenidos en este universo, sino que por el contrario, solo chequeará cada objeto que se quiera definir o usar a partir de la definición del universo. El **universo** se puede eliminar más adelante con el comando **eliminar** y más tarde volver a definirlo.
Ejemplo:

defc A = {1, 2, "hola"}, se define el conjunto A.
defc universo = {1,2,3,4,5,6,7,8,9}, no se chequean objetos definidos antes (A no está incluido en el universo).
defc C = A U {3,4}, causa error ya que el elemento base "hola" perteneciente al conjunto A no pertenece al universo.

Esta funcionalidad no estaba implementada, como tampoco lo estaba, lógicamente, la que sigue.

- Complemento conjuntos (operador: **c**).
  Una vez definido un universo se puede realizar la operación de complemento de un conjunto que devuelve el conjunto de los elementos del universo no pertenecientes al conjunto. Si el universo no fue definido se despliega un mensaje de error.
  Ejemplo: **defc C= A c** , define un conjunto C igual al complemento de A (según el universo definido)

- Definición de tuplas (comando: **deft**).
  Al igual que con los booleanos decidimos incorporar la posibilidad de definir variables de tipo tupla.
  Ejemplo: **deft t = <1, "luna", {2,3}>**.

- Evaluación de tuplas.
  Hemos incorporado la posibilidad de determinar qué elemento base está en determinada posición de la tupla.
  Ejemplo: **eval t (2)**, lo que devuelve el base que está en la segunda posición de la tupla t definida en el ejemplo anterior, esto es "luna".

- Nuevas operaciones sobre relaciones: Unión (operador: **U**), Intersección (operador: **I**), Diferencia (operador: **-**), Diferencia Simétrica (operador: **/**).
  Como las relaciones contienen conjuntos, en particular conjuntos de tuplas, se pueden aplicar estas operaciones sobre ellas. Implementamos todas las operaciones, tomando en cuenta según la operación los dominios de la relación resultante.

- Inversa de funciones (operador: **inv**).
  Se implementó la operación inversa para funciones que antes solo estaba disponible para relaciones. También se modificó la parte de gráficos para poder graficar la inversa de una función. Si la función no es invertible, al intentar ejecutar esta operación dará lugar a un mensaje de error.

- Definición de enteros (comando: **defe**).
  Al igual que para booleanos y tuplas ahora podemos definir enteros.
  Ejemplo: **defe Num = 4 + 3**, defino la variable entera Num con el valor 7 resultado de evaluar 4 + 3.

- Operación de módulo para enteros (operador: **mod**).
  La operación de módulo no estaba definida.

- Menos unario (operador: **-**).

Con la introducción del menos unario en los enteros podemos tener enteros negativos. En la versión anterior de MAC esto no era posible.
En la versión que entregamos, el tipo de datos base entero corresponde al concepto matemático de entero. Además, existe el operador unario de enteros, que le cambia el signo al operando. El tener definido el tipo de datos base Entero como natural positivo, negativo o cero, posibilitó que los enteros negativos pudieran ser elementos de un conjunto, ya que los elementos de un conjunto solo pueden ser elementos canónicos, y no pueden ser expresiones compuestas.

- Uso de palabras reservadas.
  Se ha definido un conjunto de palabras reservadas, las que al intentar ser usadas darán un mensaje de error que explica este hecho.

- Ahora los objetos base de tipo string deben estar entre comillas (ej: "a", "b"), para diferenciarlos de los identificadores que no llevan comillas. De esta manera por ejemplo si dentro de los objetos base dentro de un conjunto aparecen A y "a" , la A hace referencia a un objeto base cuyo identificador es A y "a" es un objeto base de tipo string.

- Los identificadores deben comenzar con una letra y van sin comillas, se chequea que no sean palabras reservadas.

- Mejoras en mensajes de error.
  Antes prácticamente no había un manejo de errores, cuando ocurría uno se desplegaba el mismo mensaje: "error en... " sin expresar claramente cuál era el error ocurrido, ya sea semántico o sintáctico, ó directamente se caía el programa al no manejarse una entrada incorrecta. Para esta versión de MAC resolvimos el manejo de errores semánticos, los cuales son los más difíciles de identificar sin un mensaje apropiado.
  Ejemplos de los nuevos mensajes de error:
  - Al intentar usar una palabra reservada como nombre de un objeto.
    defc U = {2}, trato de definir un conjunto de nombre: U, pero se usa como el operador de la unión.
    Resultado: "Error: U es palabra reservada"
  - Al definir una relación o una función correcta sintácticamente pero no semánticamente, en este caso se despliega un mensaje indicando cuál fue el error.
    deff f : {1,2} -> {True, False}; f = {<1,True>} defino la función f que no es total
    Resultado: "Error: la función no es total"
    defr S: {1} x {2,3} x {5}; S = {<1,2,5>, <1,5>}, defino la relación S.
    Resultado: "Error: Tupla de relacion no tiene la aridad correcta"
  - Al usar un elemento base que no pertenece al universo en un conjunto.
    defc universo = {1,2,3}
    defc C = {1,5}
    Resultado: "Error: Entero no incluido en el universo"

- Eliminación de definiciones.
  Se agregó el comando "**eliminar**", para permitir borrar del entorno objetos previamente definidos. Sintaxis: eliminar B, siendo B una variable en el entorno. Antes no era posible eliminar una definición en particular. Solo existía el comando "**limpiar**" para borrar todas las definiciones del entorno.

- Mejoras en el uso de archivos para cargar definiciones.
  Se le ha brindado mayor flexibilidad al programa para cargar definiciones desde archivo. Antes las definiciones debían de estar en un orden establecido, ahora no se necesita un orden específico.

Antes, al agregar definiciones nuevas desde un archivo con el comando: "**agregar**", se cortaba la carga si un objeto base ya estaba en el ambiente, agregándose los anteriores objetos base, sin dar ningún tipo de mensaje de error. Ahora si un objeto que se está intentando agregar está en el ambiente, es sobreescrito. Esto es lo mismo que ocurre al definir en la línea de comandos un objeto base definido anteriormente.
Si al intentar cargar un archivo ocurre un error sintáctico, se listan las definiciones a partir de la línea en la que ocurrió el error. De esta manera es más fácil identificar cuál fue la definición que causó el error.

- Uso de "**x**" en vez de "**->**" para separar los dominios de una relación.
Las funciones las podemos definir con el formato de una relación, esto es una lista de dominios, en este caso 2 dominios, y una lista de tuplas. ¿Pero cómo identificar una relación de una función si pueden tener la misma representación? Bueno, en la línea de comandos se utilizan comandos distintos, por lo que queda claro en cada caso de qué estamos hablando; pero al cargar un archivo con definiciones no se utilizan comandos. El mismo problema surge cuando se quiere incluir como parte de un objeto que se está definiendo, una relación o función sin nombre. Antes para separar los dominios de una relación como de una función se utilizaba la flecha (->). Por esta razón es que el programa antes siempre asumía que el objeto que se mencionaba era una relación. Esto nos llevó a introducir una modificación y utilizar la letra "x" para separar los dominios en una relación, y ahora sí podemos diferenciar los dos tipos base. Además, generalmente en matemáticas se utiliza el símbolo "x" para indicar producto cartesiano, que es justamente la idea de las relaciones.

Se determinaron algunas mejoras a futuro, que se detallan a continuación:

Incluimos como mejoras a futuro dos limitaciones que tiene MAC en su versión actual, ya que con un poco de esfuerzo podrían ser levantadas en próximas versiones. Una es la definición de ciertas funciones por abstracción y la otra ocurre en el producto cartesiano.

- **Producto cartesiano.**
Este punto constituye una limitación temporal que consideramos debería eliminarse en una versión posterior. El operador de producto cartesiano es binario, por lo que si queremos definir el producto cartesiano de más de dos conjuntos, las tuplas del conjunto resultado no serán de la forma usual: tuplas con tantos elementos como conjuntos participan en el producto.
Matemáticamente podemos definir el producto cartesiano de más de dos conjuntos, por ejemplo AxBxC siendo A, B y C conjuntos, en donde las tuplas resultantes tendrán tantos elementos como conjuntos participen, de manera que las tuplas tendrían la siguiente forma:
< elemento_de_A, elemento_de_B , elemento_de_C >,
pero actualmente si queremos hacer esto obtenemos:
< <elemento_de_A , elemento_de_B> , elemento_de_C >, esto es, tuplas tal que el primer elemento es otra tupla, a la cual, si participaran más conjuntos, le ocurriría lo mismo.
A modo de ejemplo, si hago el producto cartesiano de {1}x{"a"}x{True} tendré como resultado un conjunto con la tupla: < <1 , "a"> , True>, en lugar de: <1,"a",True>.
¿Qué es lo que hace el programa?. El programa no diferencia el operador "x" de cualquier otro operador binario, por lo tanto en el ejemplo anterior primero hace el producto cartesiano de los primeros dos conjuntos para luego realizar el producto cartesiano de este conjunto de tuplas con el tercer conjunto, por lo que obtiene tuplas de aridad 2, tal que el primer elemento es una tupla del conjunto resultado del producto anterior y el segundo elemento es del tercer conjunto.
En realidad este es un problema de las matemáticas ya que el símbolo "x" que se utiliza para el producto cartesiano es un operador sintácticamente binario, al recibir solo 2 conjuntos, pero dependiendo del contexto puede estar utilizándose como terciario o de mayor aridad, en cuyo caso el resultado es distinto. Creemos que para mantener una

coherencia más que un operador de producto cartesiano debiera haber una función que recibiera una serie de conjuntos.
De todas maneras quizás con un poco de trabajo se podría programar algo como para tener en cuenta este caso particular en el comportamiento de un operador para solucionar el problema. Este punto muestra claramente algo que mencionábamos al comienzo de este informe, y es la influencia de un lenguaje de programación en la formalización de los conceptos matemáticos. Aquí es donde la computación está aportando a las matemáticas.

- Otra limitación temporal es la imposibilidad de definir ciertas "**funciones por abstracción"**. Una "función por abstracción" está formada por el dominio y el codominio, una lista de variables locales y una expresión que posiblemente contenga a las variables locales. El problema ocurre en aquellas funciones en las que exista una variable local dentro de una subexpresión.
Para ver cuál es el problema tenemos que sumergirnos un poco en el código del programa. La expresión que tiene la función como abstracción, es una expresión como cualquier otra.
El problema es que el programa al "parsear" expresiones va identificando subexpresiones las cuales evalúa de manera de quedarse con los objetos base resultado de dicha evaluación, en vez de guardar toda la expresión. Por ejemplo: si tengo una expresión formada por el operador de unión sobre dos subexpresiones de conjunto, el programa evalúa cada subexpresión de manera que la expresión resultado será la formada por dos conjuntos y un operador.
¿Pero qué pasa si quiero evaluar una subexpresión en donde participa una variable?, es claro que no se puede esperar un resultado ya que por algo es una expresión. Sin dudas el problema es que el programa al ir "parseando" una expresión va evaluando cada subexpresión que la forma (la semántica de MAC es estricta), en vez de guardar toda la expresión sin ninguna evaluación.
Por lo tanto el problema no ocurre en todos los casos. Si las variables no participan en subexpresiones no se registra ningún problema, ya que las variables son objetos base y por lo tanto no tiene que evaluarlas.
Por lo tanto, sí podemos definir funciones de la forma:

```
deff F:{ {1,2} , {} , {"a","b","c"} , {True} } -> {0,1,2,3};
F = [v1]  # v1
```

en donde el dominio es un conjunto formado por varios conjuntos, y la función F es tal que devuelve el cardinal de su argumento. En este caso la variable (v1) no participa en una subexpresión.

En cambio la siguiente función no será correcta:

```
deff G: { {1,2} , {}, {"a","b","c"}, {True} } -> {1,2,3,4};
G = [v1] (# v1) + 1
```

Esta función es igual a la anterior pero al resultado le suma uno. No será correcta la función porque la variable participa en la expresión entera # v1 , la cuál es una subexpresión de la expresión final.

- Actualmente si quiero definir un objeto base como una expresión, se guarda el objeto base resultante de evaluar la expresión. Por ejemplo, si quiero definir un conjunto como la unión de otros dos conjuntos, sería bueno que se almacene internamente esta expresión (A U C) y no el conjunto resultado de evaluar la expresión, ya que de esta manera perdemos la forma en que fue construida. Además nos limita bastante el uso de conjuntos de muchos elementos, sobre todo para la representación gráfica.
Otro ejemplo: definimos un conjunto que entre sus objetos cuenta con una relación R antes definida. Entonces si queremos graficar el conjunto veremos entre los elementos la representación interna de R, esto es, dominios y lista de tuplas, en lugar de

simplemente la letra R como sería deseable. Esta modificación además serviría para solucionar la actual limitación de las "funciones por abstracción".
Para realizar esto habría que modificar la estructura interna de las expresiones ya que actualmente es una simple lista de elementos base, y habría que agregarle por ejemplo parentización.

- Como funcionalidad relevante a ser agregada es la definición de conjuntos por comprensión. Para realizar esto, sin duda habría que agregar al tipo base conjunto (set), otro tipo de expresión, además de la actual lista de objetos base, de manera de poder representar conjuntos por extensión y por comprensión.

- También debería estar en planes futuros la posibilidad de definir funciones por comprensión. Para poder realizar esto al igual que para definir conjuntos por comprensión se debería introducir una nueva forma interna de representar las funciones, además de las actualmente utilizadas, la lista de tuplas y la representación como abstracción.

- Implementar la igualdad de funciones, para lo cual si ambas funciones están representadas de distinta forma hay que comparar los dominios y si coinciden comparar la evaluación de cada objeto del dominio en cada función, esto ocurre por ejemplo si una función está representada como abstracción y la otra como relación.

- Actualmente podemos evaluar una función en un objeto base dando como resultado otro base, pero no podemos utilizar este resultado dentro de una expresión, cosa que sería interesante. Si modificáramos esto, podríamos por ejemplo definir: defe e = f (2) en donde se define un entero e como el resultado de evaluar la función f en el objeto base de tipo entero, 2, o podemos realizar la siguiente evaluación eval f (2)+4.

- Para esta versión de MAC se ha incluido la posibilidad de obtener el objeto base ubicado en determinada posición de una tupla como ya hemos explicado, pero al igual que para funciones no podemos utilizar este resultado en una expresión.

- Hemos mejorado mucho el manejo de los errores, como ya hemos visto anteriormente, aunque creemos que aun se podría hacer más. Para esta versión hemos hecho hincapié en el manejo de errores semánticos, por lo que en un futuro se le podría agregar más inteligencia al programa, para identificar errores sintácticos, y poder dar mensajes apropiados.

- Representación gráfica de funciones por abstracción.
Esta limitación es heredada de la versión anterior de MAC, y queda pendiente para próximas versiones.

- Representación gráfica de relaciones en forma de grafo. Este método de representación es útil para verificar si una relación es reflexiva o no.

- Al graficar la composición en modo grilla, se muestra solo el resultado. Podrían mostrarse las dos relaciones participantes en la composición, y a continuación el resultado.

- Adaptar Mac al entorno Windows, utilizando las versiones de winhugs y Graphics liberadas recientemente (comentarios respecto a nuestras pruebas de adaptación de MAC con estas versiones liberadas pueden ser encontradas en el apéndice B de este informe, donde se trata MAC vs ISETL)

2. Documentación

En esta área podemos mencionar tres puntos:

- Manual de usuario
  Como complemento al programa hemos desarrollado un completo manual de usuario, de más de 120 páginas, en el que invertimos bastante tiempo y esfuerzo.
  En este documento hemos puesto énfasis en describir de manera clara y formal los distintos elementos que forman el lenguaje, con numerosos ejemplos y muestras de las salidas gráficas que tiene el programa.
  Se van presentando los distintos tipos base (conjuntos, relaciones, etc.) que tiene MAC, con las operaciones que podemos aplicar sobre ellos, y acompañado de las definiciones matemáticas de los conceptos que se van mostrando.
  Se incluyen diagramas de sintaxis BNF de las expresiones que acepta el programa.

- Código mayormente comentado
  A medida que fuimos estudiando el código fuimos comentando las funciones, de manera de facilitar la comprensión a futuros desarrolladores de MAC.

- Descripción de cada módulo del sistema y de sus interdependencias
  También hemos dejado un archivo (Readme.txt), en la misma carpeta que están los archivos con todas las funciones que forman el programa, en el que se hace una descripción de todos los archivos que conforman el código, y las interdependencias entre los distintos archivos. Creemos que esto puede ser muy útil ya que son muchos los archivos que intervienen.

- Adicionalmente, incluimos como apéndice C dentro de este informe, un ejemplo de cómo podría ser escrito un capítulo de un libro de texto para ser utilizado por estudiantes de secundaria en un curso de matemáticas enfocado a la parte de Teoría de Conjuntos. Esto compensa un poco el no haber podido realizar la experiencia liceal cómo hubiésemos deseado.

Conclusiones

El objetivo central de nuestro trabajo fue el de continuar con el desarrollo del prototipo MAC. Esto incluía la corrección de algunos "bugs" detectados, y la extensión de las funcionalidades, principalmente la implementación de conjuntos definidos por comprensión, así como la documentación completa del sistema.

Creemos que nuestro proyecto ha sido exitoso, ya que al haber finalizado el mismo, el prototipo se encuentra en una etapa bastante superior de su desarrollo, en el que se incluye una documentación completa.

Entre los puntos fundamentales que hacen al éxito del proyecto citamos:

- El programa es más estable y correcto y por tanto más confiable,
- A partir de un profundo testeo se han detectado y corregido muchos errores
- Se le han agregado nuevas funcionalidades que creemos indispensables en un producto de estas características, que trabaja con elementos de la Teoría de Conjuntos (pertenencia, inclusión, universo, complemento, etc., eran operaciones que no podían estar ausentes), así como se han modificado algunas de las funcionalidades preexistentes para adecuarlas a las definiciones y formalismos matemáticos en los que nos basamos.
- Se ha mejorado un poco la parte de comunicación con el usuario, dando mensajes de error más aclarativos, dejando líneas en blanco entre prompt y prompt de manera que sea más legible la línea de comandos, etc.
- Se han identificado y documentado varias mejoras a futuro
- Se le ha dotado al programa de una buena documentación, tanto técnica como de usuario, indispensable no solo para quienes utilicen el programa como usuarios, sino para quienes se encarguen en el futuro de mantenerlo y evolucionarlo.

En el debe de este proyecto se encuentra el hecho de no haber podido implementar la definición de conjuntos por comprensión. La decisión que nos llevó a desestimar esta funcionalidad, y a dejarla por tanto como mejora a futuro, ya fue discutida en este informe. En resumen, creemos firmemente que fue una buena decisión, pues entendimos más importante asegurar la correctitud del núcleo central del prototipo, que claramente no incluía esta funcionalidad, antes que agregarle funcionalidades avanzadas, sin asegurarnos que éstas últimas estuvieran construidas sobre una base firme.
Gracias a esta decisión, creemos que ahora sí están dadas las condiciones como para diseñar e introducir ya en el próximo proyecto, tanto esta funcionalidad, como otras que aparecen como mejoras a futuro en este informe.

También creemos que la experiencia práctica de utilización de este prototipo en un curso de matemáticas de secundaria, que se planteó en cierto punto del proyecto, y de la que desistimos a pesar de nuestro real interés, está en condiciones de llevarse adelante en una próxima instancia. Los avances logrados fundamentalmente en estabilidad y correctitud, como en documentación, son imprescindibles para pensar en la realización de esta experiencia, la cual creemos que aportará mucho tanto al prototipo en sí, obteniendo un "feedback" que permita identificar cambios, errores y mejoras a introducir, como al l objetivo global del proyecto, ya que será un puntapié inicial en el camino de intervenir en metodología, contenido y objetivos de los cursos de matemáticas.

De aquí en más, visualizamos cuatro caminos posibles para continuar en la consecución del objetivo global en el que este proyecto estuvo inmerso, y que no tienen por qué ser contrapuestos:

- planificar cuidadosamente la realización de una experiencia con un 5to. año liceal, preparando material tanto para los estudiantes como para los docentes, y formando a

los mismos en la utilización del programa. Preparar formas de seguimiento y evaluación de la experiencia, etc.
- Adaptar Mac para que pueda correr sobre la versión de winhugs y la biblioteca gráfica Graphics recientemente liberadas, de manera tal de ganar en amigabilidad con el usuario, ya que correría sobre Windows en lugar de sobre DOS
- Incorporarle funcionalidades avanzadas, como la posibilidad de definir conjuntos, relaciones y funciones por comprensión
- Modificar las estructuras internas que representan las expresiones de MAC, y la forma estricta en que éstas se evalúan, para lograr implementar varias mejoras ya mencionadas en este documento

Nosotros nos inclinamos en la dirección de la adaptación de Mac al ambiente Windows, para que ya sobre este ambiente, se le realicen las mejoras que correspondan.

En conclusión, si bien debimos tomar decisiones que hicieron que nos apartáramos un poco del alcance inicial, y de nuestros intereses personales, estamos igualmente muy satisfechos por el resultado alcanzado, y el aporte que realizamos al proyecto, que creemos que redundará en la consecución más temprana que tarde de los objetivos globales en los que se enmarca este trabajo.

APÉNDICES

A. Elementos del lenguaje MAC

Ambiente o entorno

Cada vez que se inicia una ejecución de Mac, se inicializa el entorno o ambiente, que contendrá todas las definiciones que se vayan realizando durante la ejecución del programa.
Este entorno contendrá todas las variables que el usuario defina utilizando los comandos correspondientes para tales fines (ya sea que cargue las definiciones desde archivos mediante los comandos **cargar** o **agregar**, o las ingrese en forma interactiva mediante la línea de comandos).

Variables Globales y Locales

Una variable es global, cuando una vez que se define, queda ingresada en el entorno, y su valor es recuperado posteriormente, cada vez que se menciona su nombre. Tiene existencia desde que se la define hasta el final de la ejecución del programa, o hasta que se utiliza alguno de los comandos para eliminar variables (ya sea el comando **eliminar**, o el comando **limpiar**).
Existen también variables locales, y corresponden a identificadores que se utilizan para definir funciones por abstracción. Todo intento de recuperar el valor de una variable local, dará lugar a un mensaje de error.

Objetos base

En MAC tenemos definidos los siguientes tipos base o básicos:

- Enteros
  Los enteros son los números enteros matemáticos, es decir, tanto los positivos, como los negativos y el cero.

- Conjuntos (set)
  Se definen como una lista de elementos base, entre los que pueden estar, en particular, otro conjunto, una relación o una función.
  Sintaxis: { elemento_base, ... , elemento_base }

- Tuplas
  Representan una serie de objetos base con determinado orden.
  Sintaxis: < elemento_base , ... , elemento_base >

- Cadenas de caracteres (strings)
  Los cuales se definen como una serie de caracteres entre comillas.

- Booleanos
  Tienen el valor True o False.

- Relaciones

Internamente están integradas por una lista de conjuntos, los cuales representan a los dominios, y por un conjunto, en el que están las tuplas de la relación.
Sintaxis: Conjunto x .... x Conjunto ; { tupla , ... , tupla }

- Funciones
  Tienen dos representaciones internas:
  - como relación, está formada por una lista de conjuntos que forman los dominios, y un conjunto con las tuplas.
    Sintaxis: Conjunto -> Conjunto ; { tupla , ... , tupla }
  - funciones de una o varias variables definidas por abstracción. Están integradas por: una lista de conjuntos con los dominios, una lista con variables locales a la función (v1, v2, ...) y una expresión en donde probablemente participarán las variables antes definidas. Este tipo de funciones se basan en la sintaxis del cálculo-λ (cálculo lambda).
    Sintaxis: Conjunto -> Conjunto ; [v1, .. , vN] expresión

  Además a la hora de definir una función podemos utilizar esta otra sintaxis:

```
deff F : Conjunto - >Conjunto ;
        F x = case x of
                elemento_base -> elemento_base, ... ,
                elemento_base -> elemento_base
        fo
```

- Operadores
  Los operadores (aritméticos, lógicos, relacionales, unarios, de conjuntos) también están definidos como objetos base.

Expresiones

Una expresión es una secuencia de operadores y operandos que especifican una operación determinada, y de la que, por tanto, al ser evaluada se obtiene un valor. El tipo de datos de una expresión es el tipo de datos del valor que resulta de evaluarla. Una expresión en MAC está formada internamente por un objeto base ó por una expresión y un base. Esto se puede lograr gracias a la definición de los operadores como objetos base.

Comandos

- cargar, agregar
  Ambos comandos reciben un archivo con definiciones y lo cargan en el ambiente, el comando cargar antes de leer las nuevas definiciones borra el ambiente, a diferencia del agregar que no borra las anteriores.

- defe, defb, deft, deff, defr, defc
  Estos comandos se utilizan para definir objetos base, agregándolos al ambiente. defe – enteros, defb – booleanos, deft – tuplas, deff – funciones, defr – relaciones, defc – conjuntos.

- eval
  Este comando evalúa la expresión que le sigue, devolviendo su valor, que será de uno de los tipos de datos estudiados.

- dibujar, dibujarM
  Estos comandos realizan una representación gráfica del resultado de evaluar la expresión que le sigue. dibujarM realiza una representación gráfica de tipo matricial.

- mostrar
  Este comando despliega una lista por cada uno de los tipos de datos existentes: boolean, entero, conjunto, tupla, relación y función, con las variables definidas para cada uno de esos tipos de datos.

- limpiar, eliminar
  Estos comandos se utilizan para eliminar variables del entorno. En el caso de "limpiar" elimina todas las variables ingresadas por el usuario, en cambio "eliminar" elimina la variable cuyo identificador aparece a su derecha.

- fin
  Este comando cierra la ejecución del programa.

Por una definición más detallada de los tipos base, operadores, expresiones válidas o comandos dirigirse a los capítulos 2 y 3 del Manual de Usuario de MAC[12] que estamos entregando junto con este informe.

B. COMPARACIÓN DE MAC E ISETL

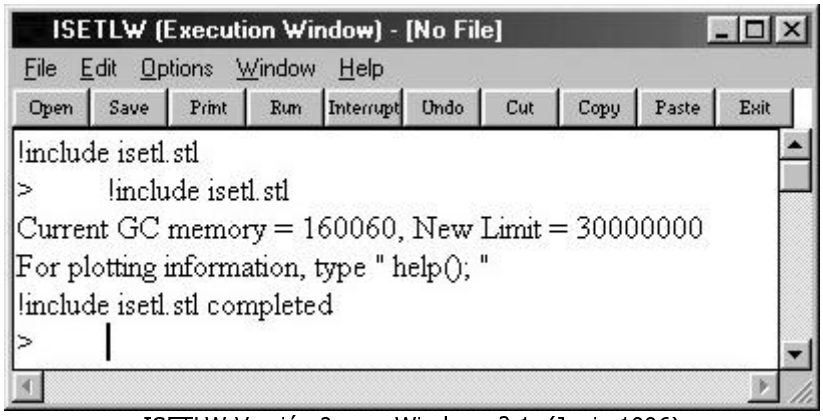
ISETL viene de Interactive SET Language (Lenguaje Interactivo de Conjuntos). ISETL es también un lenguaje de programación matemática, que recuerda el lenguaje de conjuntos y funciones utilizado por los matemáticos. Pueden definirse conjuntos, operaciones binarias y funciones y testar propiedades sobre ellos, utilizando los cuantificadores existencial y universal. SETL fue desarrollado por Jack Schwartz. Años más tarde Gary Levin desarrolló ISETL (Interactive SETL) para Unix, DOS, y Macintosh en la Universidad Clarkson. **ISETLW** (ISETL para Windows) fue desarrollado por John Kirchmeyer en el Mount Union College.
Este lenguaje mucho le debe a Ed Dubinsky, cuya idea fue usar SETL para enseñar y aprender Matemáticas. Él influenció a Gary a crear ISETL y animó a John a crear ISETLW.
ISETLW está disponible en la Home Page de ISETL en el Mount Union College[14]

Existen, por tanto, dos versiones de ISETL: la primera de ellas es una versión para DOS, Unix y Macintosh, y la segunda es una versión para Windows (ISETLW). Las versiones para Windows y para DOS difieren en que mientras para la primera se pueden realizar las operaciones mediante el mouse, disponer del portapapeles y realizar representaciones gráficas, para la segunda esto no es posible. La versión DOS tiene una apariencia muy similar a la de la implementación de MAC que estamos entregando con este proyecto. Esto se debe a que la versión DOS de ambos lenguajes interactúa con los usuarios a través de una ventana DOS.

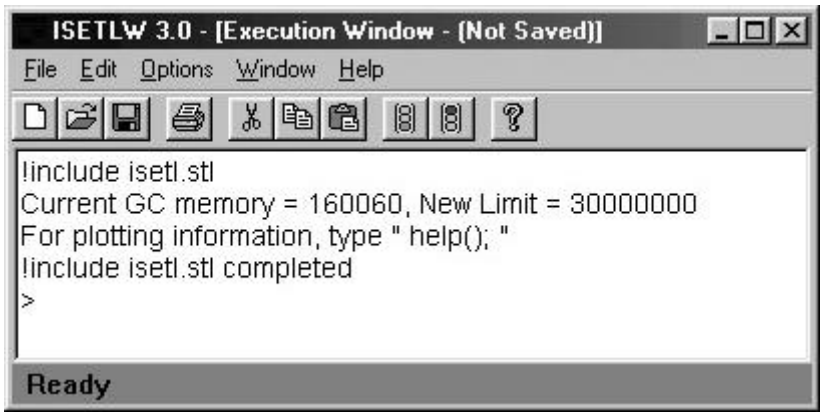
Como comentamos más adelante en este apéndice, creemos que con las versiones recientemente liberadas de Hugs98 y de la librería gráfica Graphics, será posible evolucionar MAC de manera de realizar una implementación para Windows, que soporte perfectamente bien toda la representación gráfica que MAC provee. Esto constituirá un avance significativo en la amigabilidad del producto, y lo pondrá más a la par de los lenguajes existentes, como ISETLW.

ISETLW fue escrito en 1995 y revisado en 1996 como un programa de 16-bits para Windows 3.1. Hasta el primer semestre de 2001, John Kirchmeyer, creador del lenguaje en su versión para Windows no pudo seguir trabajando en él, y recién en ese semestre, tomándose su año sabático, retomó el trabajo, para liberar la siguiente versión de ISETLW. El 6 de Junio de 2001 se liberó la versión beta de ISETLW 3.0, que es lo más reciente que existe hasta el momento del producto. Entre los principales cambios y mejoras que introduce esta nueva versión (aún en estado de beta a esta fecha), es que se ha convertido en un programa de 32-bits, diseñado para Windows 95/98/Me/NT4/2000/XP. Con ello se ha transformado en un programa muy amigable, que supera ampliamente a su versión anterior. Entre otras mejoras, ha incluido "toolbars" y opciones de "menues" que lucen y se comportan como los usuales de Windows. En la imagen que sigue se muestra esta diferencia entre las versiones 2 y 3.0 (beta) en cuanto al "look and feel" de ISETLW y a continuación se muestra la pantalla inicial de Mac, en su versión actual:

[14] http://isetlw.muc.edu/isetlw/default.asp

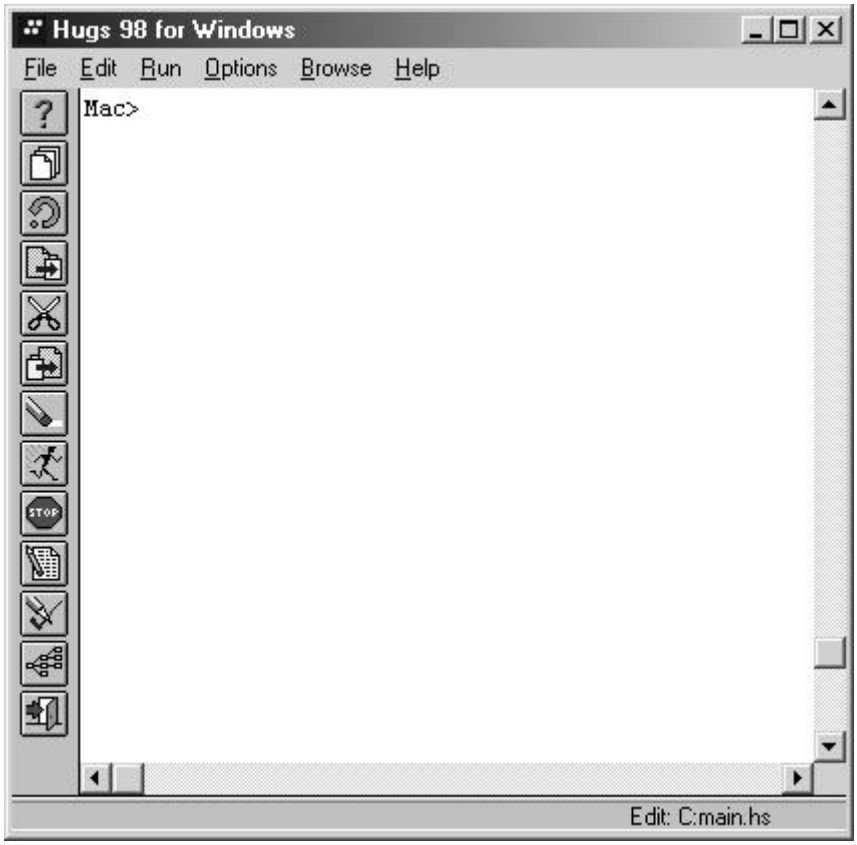


ISETLW Versión 2 para Windows 3.1 (Junio,1996)



ISETLW Versión 3.0 (Beta) para Windows 95/98/Me/2000/XP

Mac en su versión actual (para DOS, corriendo con hugs98 versión Feb2000)



Mac en su versión actual (para Windows corriendo con winHugs versión Feb/2000)

Como el alcance de nuestro proyecto no incluía los aspectos gráficos de MAC, que constituyeron la parte medular del proyecto del grupo que nos precedió en el desarrollo de este lenguaje, no tuvimos casi oportunidad de meternos en los módulos que implementan la representación gráfica de los objetos del lenguaje. Como mencionaba este grupo en su Informe Final[9], la imposibilidad de utilizar winHugs para Mac radicaba en el hecho de que la versión de winHugs de febrero del 2000 (la más nueva al momento de realizar su proyecto) tenía un "bug" que provocaba que al intentar visualizar cualquier gráfica de Mac, el programa dejara de reponder. Por este motivo, todo el desarrollo que el grupo realizó se basó en la versión para DOS (hugs). Actualmente (Diciembre, 2001) se han liberado nuevas versiones, tanto de winhugs como de la librería gráfica Graphics[16], que nos hacen sospechar que ya es posible adaptar los módulos que implementan la parte gráfica de Mac, utilizando estas nuevas versiones, de forma tal que pueda correr sobre Windows, ganando, de esta forma, en todos los aspectos, máxime si se pretende que sea utilizado por alumnos de secundaria, ya habituados a interactuar con Windows y poco tolerantes a la hora de interactuar con programas que no provean de facilidades gráficas.
Al momento de escribir estas líneas, bajamos de la página de Hugs los siguientes archivos: GraphicsLib.msi y hugs98-Dec2001.msi, los cuales, al ser ejecutados, instalan tanto hugs y winhugs, como la librería gráfica Graphics, en sus últimas versiones. Intentamos ejecutar MAC en esta nueva plataforma, sin el menor éxito en la tarea. Al investigar un poco los motivos, encontramos que los cambios introducidos tanto en la librería Graphics, como en los archivos que implementan hugs requieren de algunos cambios en los módulos de Mac que implementan la representación gráfica de los objetos. Algunos módulos de la librería gráfica que son importados por los módulos gráficos de MAC dejaron de existir, y varias de las funciones de Graphics cambiaron de nombre o tienen tipos de datos distintos.
Así es que la adaptación de Mac para que funcione en la versión Windows con estas nuevas versiones de winhugs y de Graphics no es una tarea menor; requiere de cierto tiempo y esfuerzo, y de un conocimiento más cabal de la implementación gráfica de MAC, que por cierto no tenemos, debido a que esto no estaba incluido dentro del alcance de nuestro proyecto.

[16] http://glass.cse.ogi.edu/Hugs/pages/downloading.htm Versiones Dic. 2001

Este es un posible próximo camino en la evolución del prototipo Mac: intentar utilizar como plataforma de desarrollo la de winhugs, adaptando todos los módulos del lenguaje.

PRINCIPALES DIFERENCIAS ENTRE MAC E ISETLW

A continuación destacamos algunas de las diferencias más importantes entre MAC e ISETL, desde el punto de vista de los objetos matemáticos que permiten definir:

Definición de Conjuntos

Las definiciones de conjuntos en ambos lenguajes son muy similares. La idea fundamental que hay atrás de cualquiera de estos lenguajes de programación, es que sean en esencia muy parecidos al lenguaje matemático.
La diferencia más importante que existe actualmente entre MAC e ISETL, es la posibilidad que tiene el segundo de definir conjuntos por comprensión. Esta es una de las mejoras a futuro de MAC, que había sido planteada dentro del alcance inicial del proyecto, y que como mencionamos oportunamente en este Informe, no pudimos implementar, por lo que constituye una de las carencias más importantes que tiene MAC en este momento, frente a ISETLW.

Definición de Relaciones y de Funciones

Aquí es donde aparece uno de los motivos principales para la creación de MAC: en ISETL si bien la definición de relaciones y funciones sigue una notación parecida a la matemática, sin embargo no permite una definición "completa" de estos objetos, ya que no permite definir el dominio y codominio de una relación o función. Estrictamente hablando, cuando uno define matemáticamente una función o una relación, especifica tanto el domino como el codominio de la misma. Si esto no se hiciera, entonces sería imposible chequear propiedades tan importantes como la inyectividad, sobreyectividad y biyectividad de una función (y con esto poder precisar si una función es o no invertible, y en caso de serlo hallar su inversa) y en las relaciones, saber si una relación es de equivalencia, y más particularmente, si es reflexiva, simétrica o transitiva, y poder hallar su clausura reflexiva, simétrica o transitiva.
Esto no es un hecho menor, ya que hace al entendimiento de estos objetos matemáticos por parte de los alumnos, con total precisión y sin ambigüedades.

Por ejemplo, para describir la relación R formada por los pares ordenados (x,y) tales que x e y pertenecen al conjunto {1,2,3} y cumplen con la propiedad de que x es menor que y, escribimos en ISETL:

```
R:= {[x,y]: x,y in {1..3} | x<y};
```

O también:

```
R:= {[1,2],[1,3],[2,3]};
```

Lo mismo en MAC, solo puede ser escrito enumerando las tuplas que pertenecen a la relación:

```
defr R: AxA; R = {<1,2>,<1,3>,<2,3>}
```

donde el conjunto A suponemos que fue definido previamente como {1,2,3}. Pero observar que en MAC la definición del conjunto universal de pares (AxA) es parte de la definición de la relación, y no puede ser omitido. De hecho, MAC realiza chequeos que aseguren que el conjunto de tuplas especificado esté contenido en AxA. En caso contrario fallará, dando un mensaje de error que indicará donde se encuentra el problema.
Luego es posible saber si la relación R es de equivalencia (simétrica, reflexiva y transitiva) (haciendo en MAC "eval essim R And esref R And estrans R") e incluso hallar, por ejemplo, la clausura reflexiva de R, " ref R " cuyo resultado es la relación de AxA cuyo conjunto de tuplas es: {<1,1>,<2,2>,<3,3>,<1,2>,<1,3>,<2,3>}.

De igual manera, en ISETL se puede definir una función como un conjunto de pares, pero tampoco es posible incluir en la definición de la función dominio y codominio de la misma.

En ISETL se pueden definir funciones de muchas maneras. De hecho, ISETL incluye tanto una forma declarativa de especificación como una imperativa[7].
Se puede definir una función como una regla que aplicada a distintos argumentos, calcula su valor. Por ejemplo, la función f(x) = 2x se puede definir de la siguiente manera:

```
f:= func(x);
return 2* x;
end;
```

Como se puede observar, no se incluye en la definición dominio y codominio de la función, con lo cual se está definiendo para todos aquellos argumentos x para los que esté definido 2*x.
En cuanto a las funciones de alto orden podemos decir que ellas pueden ser definidas en ISETL. Por ejemplo, se puede definir la función compuesta de f y g como:

```
compose:= func(f,g);
return | x -> (f(g(x)))|;
end;
```

Como puede observarse, no se incluye en la definición (ni se infieren) los tipos de las funciones, los que son determinantes para que la composición esté bien definida.

Para definir en MAC una función, podemos hacerlo de dos maneras:
1) Escribiendo explícitamente los pares ordenados que la componen, o
2) usando una notación similar al cálculo lambda (funciones por abstracción).

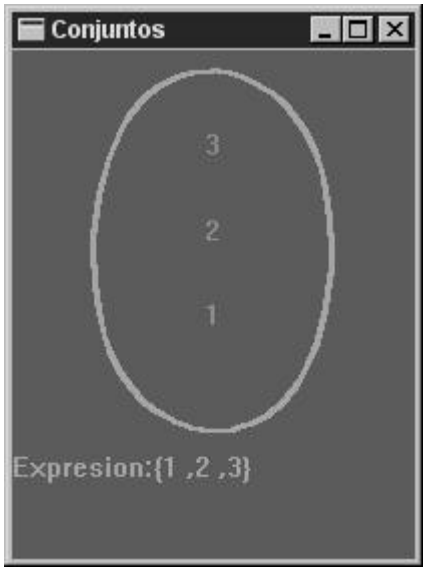
Cualquiera sea la forma de definición empleada, será necesario incluir en la definición en MAC los elementos del dominio y codominio de la función a definir.
Los conjuntos domino y codominio nos permiten afirmar si una función cumple con las propiedades de inyectividad, biyectividad, sobreyectividad, si la composición de dos funciones está bien definida, etc. Esto quiere decir que si no conocemos estos conjuntos no podemos concluir si una función cumple o no con dichas propiedades. Lo único que nos brinda ISETL en estos casos es la posibilidad de conocer los conjuntos imagen y preimagen.

La representación gráfica de los distintos objetos manejados por MAC constituye una diferencia importante con ISETL.
En MAC pueden representarse gráficamente tanto conjuntos, como relaciones y funciones, así como operaciones entre ellos.
En ISETL, en cambio, los aspectos gráficos existentes no parecen ser del todo buenos en el esclarecimiento los conceptos matemáticos que hay detrás de las representaciones. La idea que se persigue con estos lenguajes es que faciliten el aprendizaje de matemáticas por parte de los alumnos. En ese sentido, las representaciones gráficas deben contribuir al más cabal entendimiento de los conceptos, y por lo tanto deben ser bien claras.

A continuación presentamos las diferencias gráficas entre MAC e ISETL, extraídas del Informe del Proyecto[9] del grupo anterior:

Representación gráfica de conjuntos

- Representación gráfica del conjunto A = {1,2,3} en MAC



Como puede verse, se utiliza la representación de diagramas de Venn.

- Representación gráfica del conjunto A = {1,2,3} en ISETLW
  No es posible graficar conjuntos en ISETLW

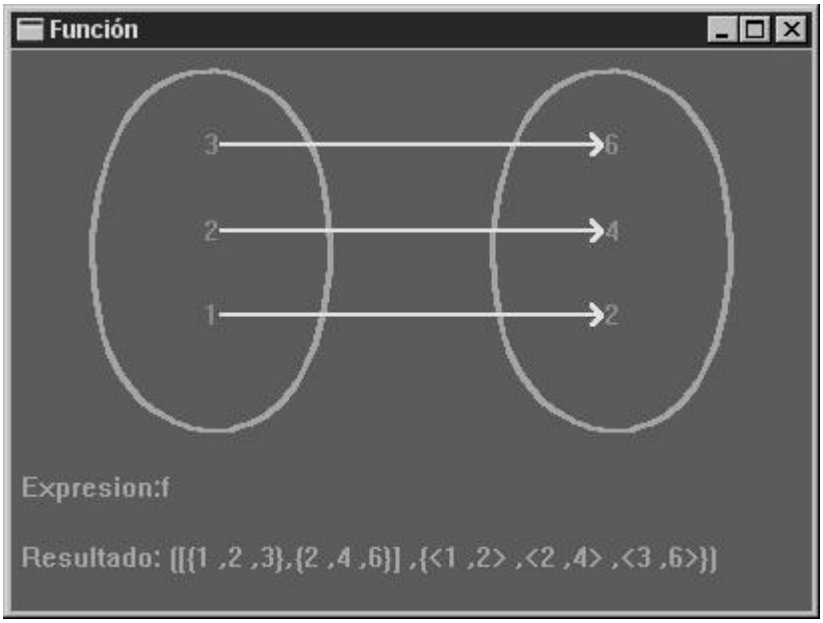
Representación gráfica de funciones

- Representación gráfica de la función f(x) = 2x en MAC

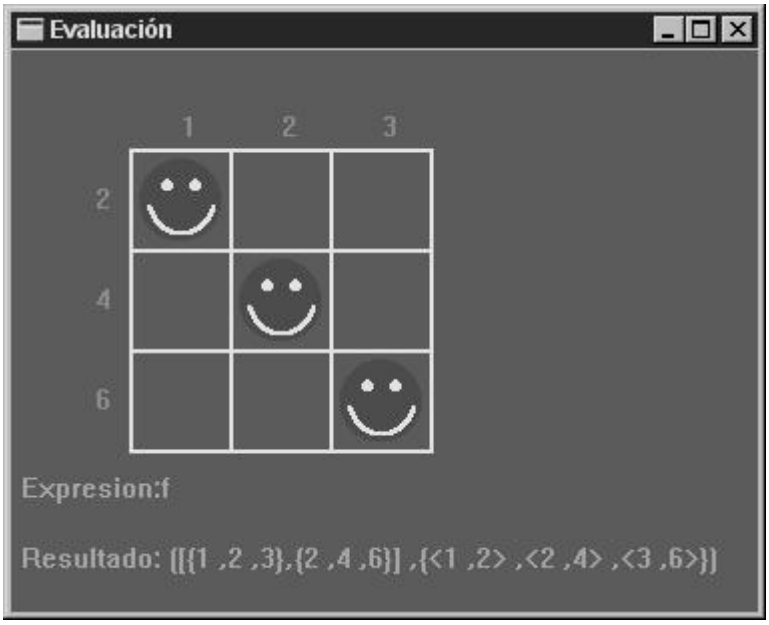
  En MAC debemos incluir dominio y codominio de una función en su definición. Si definimos en MAC la función anterior como:

```
deff f:{1,2,3}->{2,4,6};
f x = case x of
        1 -> 2,
        2 -> 4,
        3 -> 6
fo
```

  y utilizamos el comando "dibujar f" obtenemos la siguiente gráfica:



Por otro lado, también contamos en MAC con una representación matricial de las funciones. Si en lugar de utilizar el comando "**dibujar**" utilizamos el "**dibujarM**", haciendo: "dibujarM f" obtenemos la siguiente representación gráfica:



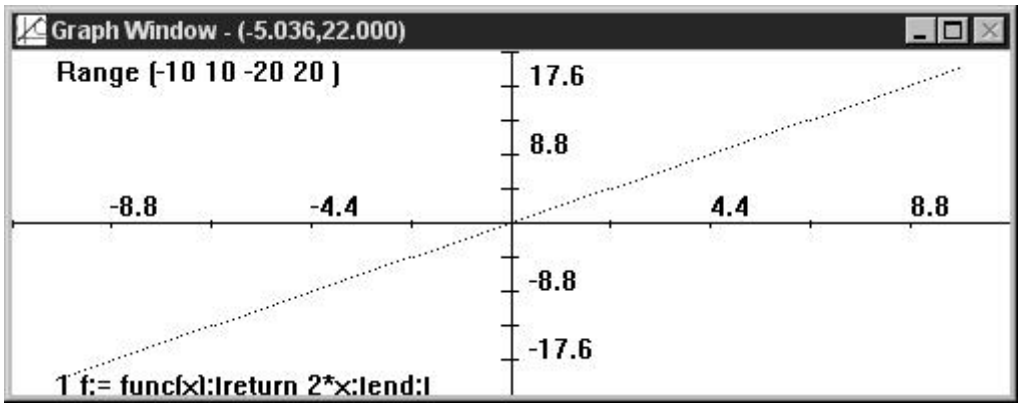
- Representación gráfica de la función f(x) = 2x en ISETLW

  Mediante ISETLW solamente se pueden representar funciones numéricas (a diferencia de MAC que permite representar funciones de cualquier tipo). La representación gráfica de funciones se lleva a cabo a través del plano cartesiano.

  Si definimos en ISETLW la función f como:

```
f:= func(x);
return 2*x;
end;
```

y luego utilizamos el comando "plot(f)" obtenemos la siguiente representación gráfica:

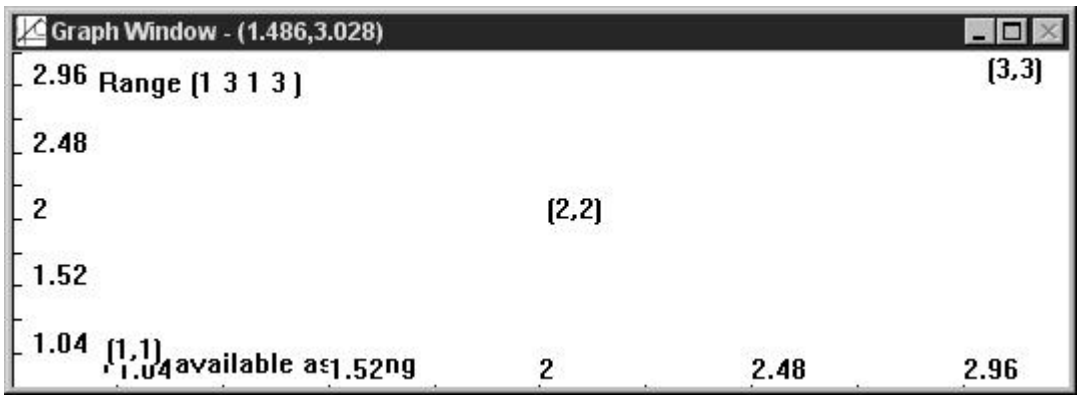


Como puede verse, aquí no se indican dominio y codominio e ISETLW realiza una representación "continua" de la función, es decir, supone dominio y codominio reales.

Representación gráfica de relaciones

- Representación de una relación R en ISETL

ISETL no permite la representación gráfica de relaciones por defecto. ISETL solamente permite representar funciones, por lo que si una relación es también función, entonces ella podrá ser representada.
Por ejemplo, la relación R definida como R:= {[x,y]: x, y in {1..3} | x<y} no puede ser representada mediante el comando "plot" en ISETL, debido a que no es una función.
Si, en cambio, definimos la siguiente relación R2:
R2:= {[x,y]:x,y in {1..3} | x=y}; esta relación es función, por lo cuál puede ser representada gráficamente. Al hacer "plot(R2);" obtenemos la siguiente gráfica:

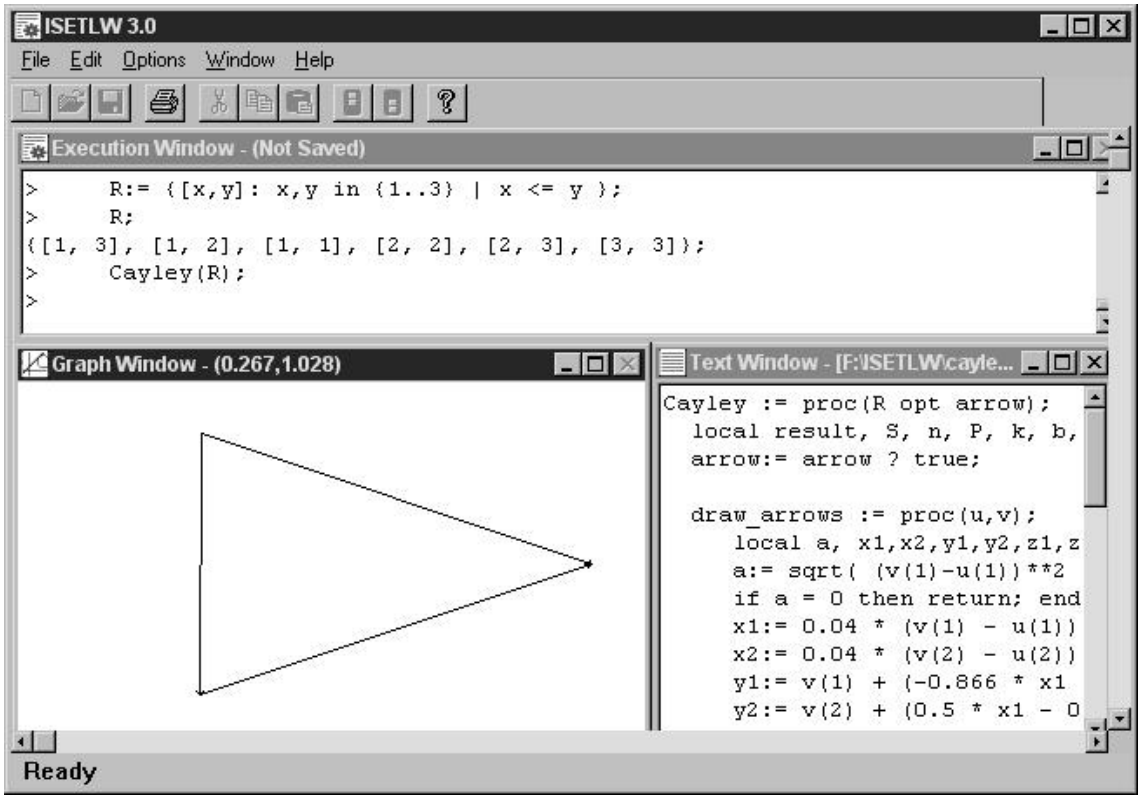


Otra opción que existe en ISETL es definir nuestras propias funciones.
Utilizando este mecanismo, definimos una nueva función: "Cayley" que se encuentra disponible en libros de referencia[1] y que transcribimos a continuación:

```
Cayley := proc(R opt arrow);
  local result, S, n, P, k, b, Set;
  arrow:= arrow ? true;

  draw_arrows := proc(u,v);
     local a, x1,x2,y1,y2,z1,z2;
     a:= sqrt( (v(1)-u(1))**2 + (v(2)-u(2))**2);
     if a = 0 then return; end;
     x1:= 0.04 * (v(1) - u(1))/a;
     x2:= 0.04 * (v(2) - u(2))/a;
```

```
      y1:= v(1) + (-0.866 * x1 - 0.5 * x2);
      y2:= v(2) + (0.5 * x1 - 0.866 * x2);
      z1:= v(1) + (-0.866 * x1 + 0.5 * x2);
      z2:= v(2) + (-0.5 * x1 - 0.866 * x2);
      move(v(1),v(2)); draw(y1,y2);
      move(v(1),v(2)); draw(z1,z2);
   end;

   S:= domain(R) union image(R);
   n:= #S; P:= {}; k:= 2*Pi/n;
   for i in [1..n] do
     take b from S;
     P:= P union { [b, [cos(i*k), sin(i*k)]]};
   end;
   Set:= {[P(x), P(y)]: [x,y] in R};
   graphics(true);

   while result /= 'q' do
      scale(-1.2,1.2,-1.2,1.2);
      for [u,v] in Set do
         move(u(1),u(2)); draw(v(1),v(2));
         if arrow then draw_arrows(u,v); end;
      end;
      result := zoomer();
   end;
   graphics(false);
end;
```

A continuación se muestra la pantalla de ISETLW 3.0 ejecutando esta función para la relación R:= {[x,y]: x,y in {1..3} | x<= y}:



La función Cayley no viene implementada con ISETL, sino que debemos escribir un archivo que la implementa e incluirlo en ISETL antes de ejecutarlo. El resultado de la aplicación de dicha función es la representación en forma de grafo de una relación.

Observar en la pantalla gráfica de ISETLW que esta representación no nos dice qué elemento representa cada uno de los nodos del grafo, y tampoco permite distinguir cuando un elemento se relaciona consigo mismo, ya que no dibuja lazos en el grafo, lo que impide, por ejemplo, saber si la relación es o no reflexiva.
Es decir, si representamos mediante la función Cayley la relación R3 definida como:

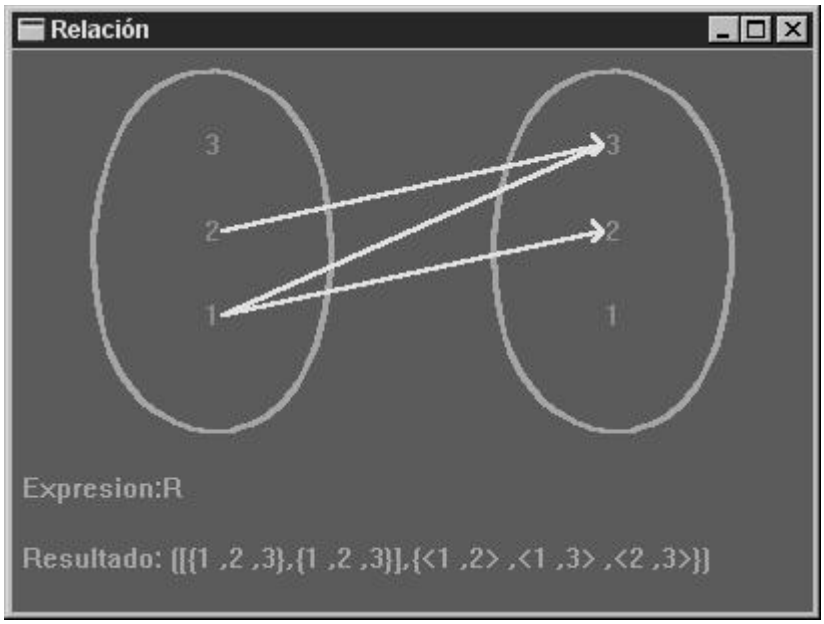
```
R3:={[x,y]: x,y in {1..3} | x<y};
```

la gráfica obtenida será la misma que antes.

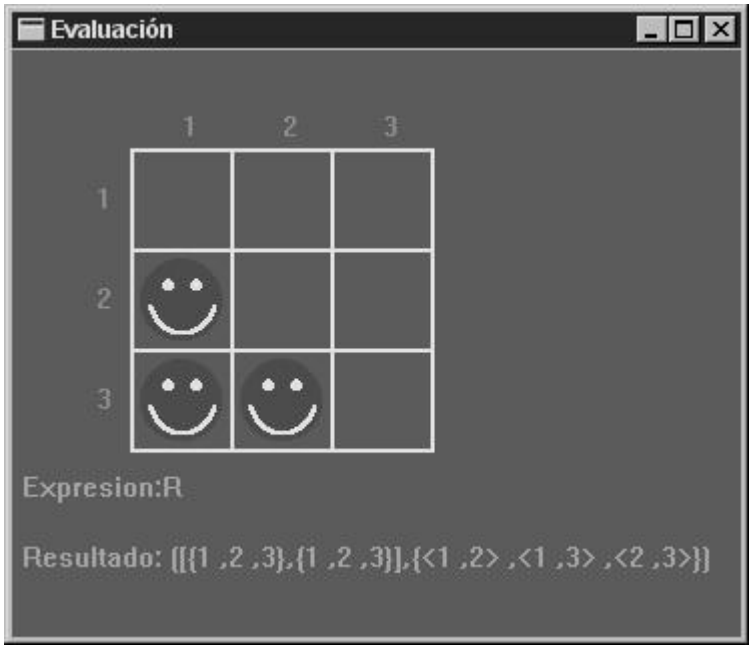
- Representación de una relación R en MAC

Las representaciones soportadas por MAC se basan en diagramas de Venn y la representación matricial. Las gráficas de la relación R = {<1,2>,<1,3>,<2,3>} en cada una de estas representaciones son respectivamente:

Representación basada en diagramas de Venn, obtenida de ejecutar en MAC el comando "**dibujar R**":

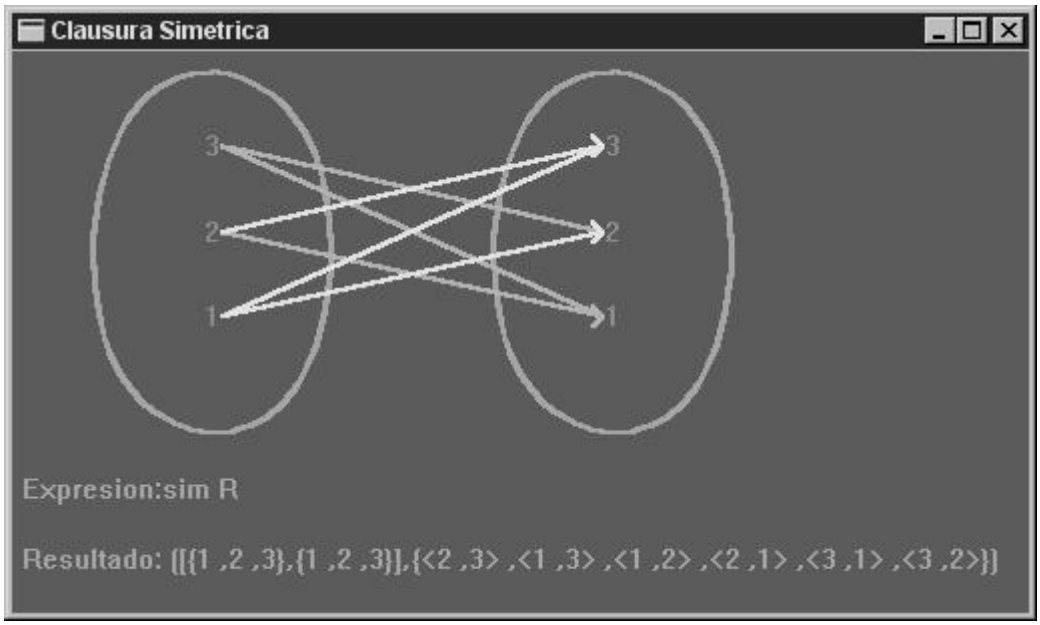


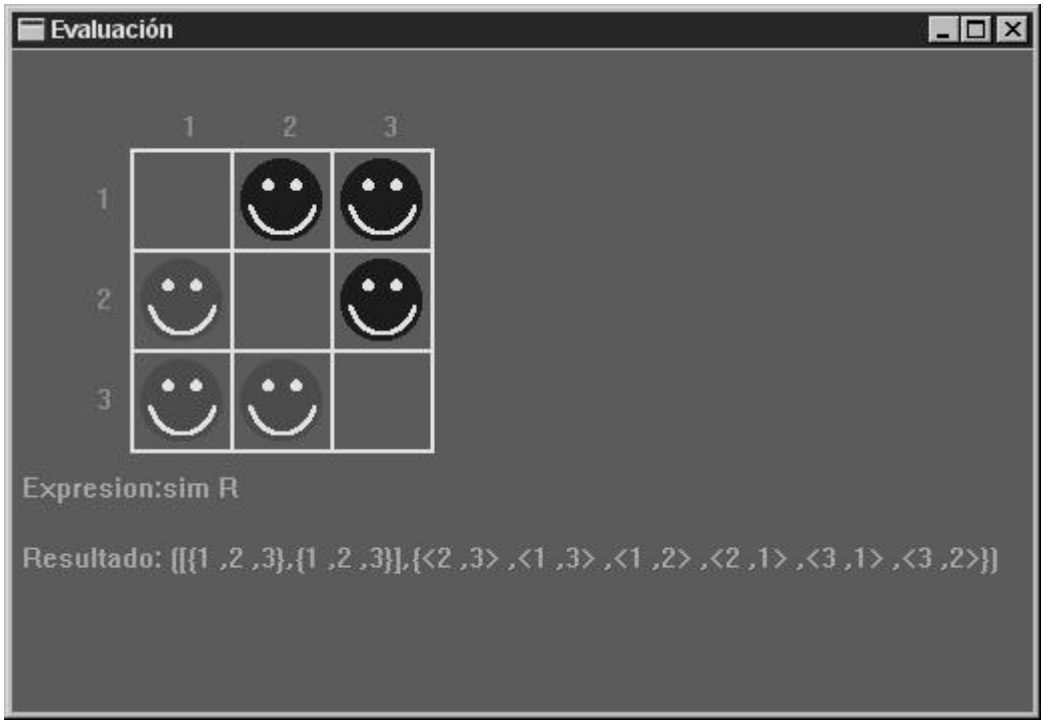
Representación matricial obtenida de ejecutar en MAC el comando "**dibujarM R**":



- Representación gráfica de operaciones

  A diferencia de ISETL, MAC también permite la rápida visualización de conceptos como simetría de relaciones, entre otras operaciones sobre objetos representables gráficamente. Por ejemplo, si deseamos visualizar la clausura simétrica de la relación R vista antes, podemos utilizar los comandos "**dibujar**" y "**dibujarM**" con argumentos "**sim R**" en ambos casos, y obtenemos respectivamente:





Para profundizar más en las representaciones gráficas soportadas por MAC, dirigirse al Manual de Usuario de Mac[12], que estamos entregando con esta documentación.

INCLUIRIMAGEN

ℂ. MATEMÁTICA DISCRETA USANDO MAC

A continuación se presentan algunos ejemplos acerca de cómo puede utilizarse MAC para un curso liceal de matemáticas discretas.
Creemos que en próximos proyectos deberá avanzarse en este sentido, enfocándose fundamentalmente a la parte de enseñanza utilizando la herramienta, ya que a través de la interacción entre el desarrollo y la aplicación de la herramienta se obtendrán resultados concretos que permitan evaluarla y adaptarla a la realidad.
Para presentar los ejemplos, utilizamos como guía el libro "Introduction to Discrete Mathematics with ISETL" de William E. Fenton y Ed Dubinsky.
El por qué de la elección de este libro como guía, es sencilla: si bien el enfoque en el que se basa la concepción de MAC (***M**atemática **A**plicada a la **C**omputación*) es diferente del de ISETL, el trabajo de los autores del libro realiza aportes valiosos para el mejoramiento de la enseñanza de matemática, en el mismo sentido que nosotros planteamos. Las diferencias de enfoque que señalamos anteriormente tienen su origen en que ISETL aparece como una herramienta computacional que apoya fuertemente la actividad de aprender matemáticas en los cursos existentes , mientras que Mac está estrechamente vinculado a la enseñanza de matemática discreta y programación en forma integrada, para lo cual no existen aún cursos en la realidad.
Los autores del libro mencionado presentan una metodología según la cual ISETL facilita la tarea de interiorizar los conceptos matemáticos, en base a la hipótesis de que la mejor forma de aprender matemáticas para la media de los estudiantes, es primero trabajar de forma intuitiva con los elementos matemáticos, sin tener previamente una presentación formal de los conceptos; jugar un poco, si se quiere, con ejercicios y problemas que se plantean específicamente para este fin, aún sin tener las herramientas formales para resolverlos, y una vez que se dedicó el tiempo necesario para tener un primer acercamiento intuitivo a estos elementos, recién ahí presentar los elementos teóricos que los sustentan. A este enfoque le llaman "enfoque constructivista de la enseñanza", donde los estudiantes construyen por ellos mismos los conceptos matemáticos apropiados para entender y resolver problemas. Para ello, el libro en cuestión desarrolla el conocimiento utilizando un estilo circular, que imitamos en los ejemplos que escribimos utilizando MAC. Primeramente se plantean algunas actividades prácticas, para sentar una base experimental previa, que sirva de cimiento para los conceptos que se plantearán luego, y permita a los estudiantes llegar, por tanto, a un más temprano y eficaz entendimiento de los mismos. El dedicar algún tiempo y esfuerzo a estas actividades, ya sea que se llegue a respuestas correctas o no, entienden los autores, trae beneficios claros, luego, a la hora de entender los conceptos teóricos que hay detrás. Es por ello que las definiciones, teoremas, pruebas son presentados recién después que el estudiante tuvo la oportunidad de trabajar con las ideas relacionadas a los conceptos a ser formalizados.
La forma de lograr que los estudiantes se involucren con este estilo de aprendizaje, y que realmente sientan necesidad de más matemática, es justamente no dándoles todos los conceptos servidos, sino haciendo que los vayan descubriendo por sí mismos, con la ayuda de una herramienta de software, como lo es, en el caso del libro mencionado, el lenguaje ISETL, y en nuestro caso, MAC.
En esta introducción simplemente queremos presentar algunos ejemplos que ilustren en alguna medida la presentación de conceptos comunes a matemática y programación.
Los autores del libro definen el ciclo de aprendizaje utilizado como ciclo A.C.E. (Activities, Class Discusión Material, Exercises), donde dividen el texto en secciones, cada una a ser cubierta por la media de la clase en un tiempo determinado. Una sección consiste de Actividades, material de discusión en clase, que es donde se presentan los conceptos teóricos, y un conjunto de ejercicios, para poner en práctica los conceptos, luego de haber trabajado con ellos, primero de forma intuitiva en las actividades y luego en forma teórica en la discusión. Usamos dicha metodología en la presentación de los ejemplos. (Por razones de tiempo, solo se trabajó parcialmente según el ciclo A.C.E., con el tema "Conjuntos").
Un análisis más completo de las funcionalidades de MAC puede encontrarse en el Manual de Usuario, perteneciente a la documentación de nuestro proyecto.

CONJUNTOS

"La teoría de Conjuntos fue desarrollada por Boole y Cantor a fines del siglo XIX, y ha tenido una profunda influencia en el desarrollo de la Matemática en el siglo XX..."

DEFINIENDO CONJUNTOS

ACTIVIDADES

1. ¿Cuáles de los siguientes conjuntos son iguales? Trata de contestar antes de utilizar MAC. Luego chequea A = B; A = C; etc. (haciendo en Mac: "eval A=B", etc.) Explica tus resultados, y explica cualquier diferencia entre tus resultados y los de la computadora.

   A = {1,2,3},
   B = {3,2,1},
   C = {1,2,3,1,2,3},
   D = {{1}, {2}, {3}},
   E = {{1,2,3}},
   F = {},
   G = {{}},
   H = {"L","o","s","r","e","d","o","n","d","i","t","o","s","d","e","r","i","c","o","t","a"},
   J = {"L","o","s","r","e","d","n","i","t","c","a"},
   K = {"Los Buitres"},
   L = {"Los", "Buitres"}

2. ¿Cuántos elementos tiene cada uno de los conjuntos anteriores? Escribe tu respuesta, luego ve a MAC y prueba #S, siendo S el nombre de cada conjunto, para calcular esos valores. Explica tus respuestas, y explica cualquier diferencia entre tus respuestas y las de la computadora. Hazlo también para los conjuntos que siguen:

   S1 = {"a", "b", True}
   S2 = {S1, "True"}
   S3 = { 1, { 1, {1}}}

DISCUSIÓN: CONJUNTOS Y SUS ELEMENTOS

En Matemáticas, la palabra ***conjunto*** se emplea para representar una colección no ordenada de objetos considerada como una sola entidad. Las colecciones designadas con nombres tales como "rebaño", "tribu", "estudiantado", "equipo" y "electorado" son todas ejemplos de conjunto. Los objetos que constituyen la colección se llaman ***elementos*** o ***miembros*** del conjunto, y de ellos se dice que ***pertenecen*** o que ***están contendidos*** en el conjunto. A su vez, se dice que el conjunto ***contiene*** o ***está compuesto*** de sus elementos. Los objetos pueden ser cualquier tipo de cosas: gente, lugares, palabras, números, valores booleanos, conjuntos (esto es interesante, observa en el ejercicio 2, los conjuntos S2 y S3) , animales, vegetales, minerales, relaciones, funciones, operadores y así.
Para utilizar un conjunto es importante saber precisamente qué elementos contiene el conjunto. Esto puede ser fácil de explicar o difícil. Por ejemplo, {1,0,3,0,-1, 1} es un conjunto **expresado por extensión**, es decir, **se listan todos los elementos que conforman el conjunto**. ¿Cuántos elementos contiene este conjunto? Si la respuesta es distinta de 4, piénsalo un rato más. ¿Qué elementos contiene este conjunto? Eso te llevará más fácilmente a la solución. ¿El conjunto anterior y el {1,0,3,-1} son o no iguales?
Revisa las respuestas que dio Mac a las igualdades del ejercicio 1. ¿Los conjuntos A y C no son iguales? ¿Los conjuntos H y J no son iguales?

Por otro lado, decimos que un conjunto es una colección **no ordenada** de objetos. ¿Qué significa esto? ¿Los conjuntos A y B del ejercicio 1 son o no iguales?
Un conjunto queda determinado cuando se puede saber para cualquier objeto que uno tome, si este objeto pertenece o no al conjunto. Hasta ahora hemos visto una sola forma de definir un conjunto: por extensión, es decir, dando una lista de los elementos que pertenecen al conjunto. Pero existen otras formas de determinar un conjunto, como lo son las definiciones por comprensión, que veremos más adelante.

Temas a desarrollar

Los ejemplos que se presentan a continuación han sido pensados para introducirse de la misma manera que los anteriores. A pesar de estar inconclusos, se incluyen a modo de guía futura.

Tuplas y Relaciones

Nuestra propuesta de curso incluye el tema Lógica al principio del mismo.

Las tuplas definidas en MAC por:

```
deft T1 = <True,True>
deft T2 = <True,False>
deft T3 = <False,True>
deft T4 = <False,False>
```

representan los posibles valores de p y q en una tabla de verdad.
Se puede representar con relaciones cualquier tabla de verdad y testar las equivalencias lógicas. Por ejemplo, para testar la equivalencia lógica de $p \rightarrow q$ y $(\text{not } p) \vee q$, podemos construir en Mac una relación entre los valores de (p,q) y (not p, q) :

```
defr notpq : (Boolean x Boolean) x (Boolean x Boolean);
     notpq = {<T1,T3>,<T2,T4>,<T3,T1>,<T4,T2>}
```

y la relación or :

```
defr or : (Boolean x Boolean) x Boolean;
     or = {<T1,True>,<T2,True>,<T3,True>,<T4,False>}
```

y la relacion "implica".

```
defr impl : (Boolean x Boolean) x Boolean;
     impl = {<T1,True>,<T2,False>,<T3,True>,<T4,True>}
```

La composición de la relación notpq con la relación or, es una relación R : (Boolean x Boolean) x Boolean, tal que $(T,t) \in R$ sii existe T' tal que $(T,T') \in$ notpq y $(T',t) \in$ or.
Es fácil comprobar que son los valores de la tabla de verdad de $(\text{not } p) \vee q$.
Podemos comprobar usando MAC que $(\text{not } p) \vee q$ es equivalente a $p \rightarrow q$:

```
eval notpq o or == impl
Resultado:  True
```

Funciones

Se definen funciones cuyo dominio y/o codominio no es exclusivamente un conjunto de números

```
defc A = {1,2,3,4}
deff f : P(A) -> Boolean;
      f = [x] 2 en x

deff g : Boolean -> {"verdadero","falso"};
      g x = case x of
               True -> "verdadero",
               False -> "falso"
            fo

eval g o f
eval esiny g o f
eval essob g o f

defc A = {0,1,2,3,4}
deff f : A -> P (A);
      f = [x] {x}

deff g : P(A) x P(A) -> P(A);
       g = [x,y] x U y
```

Divisibilidad

Dado el conjunto A = {1,2,3,4,5,6}, definir la relación Divisores, tal que (x,y) $\in$ Divisores sii x es divisor de y.
Se puede usar la definición x es divisor de y sii y mod x == 0.
Divisores : A x A;
{<1,1>,<1,2>,<1,3>,<1,4>,<1,5>,<1,6>,<2,2>,<2,4>,<2,6>,<3,3>,<3,6>,<,4,4>,<5,5>,<6,6>}

Definir la relación Divisores en Mac y resolver:

¿Es la relación, reflexiva, simétrica y transitiva ? Si alguna de las propiedades no se cumple, explica por qué y halla la clausura correspondiente. ¿Esta nueva relación es reflexiva, simétrica y transitiva ? En caso de que no, ¿por qué no ? Halla la clausura correspondiente.

Definir el conjunto A = {0,1,2,3,4,5,6,7,8,9}. ¿Cuál deberá ser el codominio de una función f que tome un elemento x de A y devuelva x mod 3 ?
Definirla por abstracción y comparar la respuesta con la de MAC.
Definir un conjunto de subconjuntos de A, tal que para todos los elementos de A, se cumpla que dos elementos cuales quiera de A, x e y, se cumple x e y $\in$ al mismo subconjunto sii f (x) == f (y).
¿Cuál es la unión y la intersección de todos los subconjuntos? ¿Alguno es vacío ?
Define una relación R tal que (x,y) $\in$ R sii f(x) == f (y).
Comprueba con MAC que R es de equivalencia.
Este ejercicio sirve para introducir los conceptos de aritmética módulo n, partición, etc.

¿Cuál es el resultado de (-8) mod 3? Compara tu respuesta con la de MAC. ¿Para qué sirve la condición de *resto menor que dividendo* en la definición de división?

El lenguaje matemático y el lenguaje de Mac.

Dadas las definiciones:

```
defc True = {1}
defc Y = {1,2,Y}
defc Y = {1,2,"Y"}
defc A = {1,2,Y}
defc A = {1,2,x}
defc Y = {1,2,Y}
```

explicar respuestas de MAC. Observar el orden: la última definición es igual a la segunda, pero el resultado es diferente.
Dada la definición :

```
deff f : {1,2,3} -> {1,2,3,4};
     f = [x] x + 1
```

Explicar respuesta de MAC. ¿Por qué usa MAC este tipo de definición para las funciones? Nuestro curso incluye el estudio de funciones como algoritmos (procesos) y su formalización a través de una introducción al Cálculo Lambda.

Dada la definición:

```
deff g : {0,2} -> {0,2,4};
     g x = case x of
               0 -> 2 * 0
               2 -> 2 * 2
           fo
```

Explicar respuesta de Mac.

En las definiciones de f y g usamos el mismo nombre x. ¿Es correcto? ¿Podemos usar otro nombre?
Explica la diferencia entre los símbolos de MAC = y == (y i=) y di cuáles son los símbolos matemáticos correspondientes.

Ejemplos de mensajes de error

Trabajar con algunos mensajes de error de Mac, puede contribuir a la reafirmación de conceptos. Por ejemplo,
Dada la siguiente definición, explicar la respuesta de MAC y corregir la definición.

```
deff and : Boolean x Boolean  -> Boolean;
     and t = case t of
               T1 -> True,
               T2 -> False,
               T3 -> False,
               T4 -> false
             fo
```

Error : la función no es total

REFERENCIAS

[1] FENTON, William & DUBINSKY, Ed. *Introduction to Discrete Mathematics with ISETL*. New York, Springer, 1996.

[2] FOKKER, Jeroen. *Lecture Notes in Computer Science: "Advanced Functional Programming"*, Cap.1

[3] HUTTON, Graham & ERIK, Meijer. *FUNCTIONAL PEARLS: Monadic Parsing in Haskell*. Universidades de Nottingham y de Utrecht. Paper extraído de internet

[4] HUTTON, Graham & ERIC, Meijer. *Monadic Parser Combinator*. Universidades de Nottingham y de Utrecht. Reporte Técnico NOTTCS-TR-96-4, 1996

[5] GRASSMANN, Winfried & TREMBLAY, Jean Paul. *Matemática Discreta y Lógica: Una perspectiva desde la Ciencia de la Computación*. Prentice Hall, Madrid 1996.

[6]THOMPSON, Simon. *The Craft of Functional Programming*. Addison-Wesley

[7] DA ROSA, Sylvia. An experience with ISETL. Tech. rep.

[8] DA ROSA, Sylvia & CIRIGLIANO, Gustavo. *Ensayo sobre Matemática Aplicada a Computación*, Tech. rep., 1999.

[9] ACUÑA, Juan & YEMINI, Gigliola. *Informe proyecto de Taller V.* Facultad de Ingeniería,2000

[10] ACUÑA, Juan & YEMINI, Gigliola. *Guía de Uso. Proyecto Taller V.* Facultad de Ingeniería, 2000.

[11] ACUÑA, Juan & YEMINI, Gigliola. *Guía Técnica. Proyecto Taller V.* Facultad de Ingeniería, 2000.

[12] FERNÁNDEZ, Cecilia & FERNÁNDEZ, Guillermo, *Manual de Usuario de MAC*. Facultad de Ingeniería, 2002

[13] BACK, Ralph-Johan & VON WRIGHT, Joakim, *Structured Derivations: a Method for Doing High-School Mathematics Carefully.* Abo Akademi University, Department of Computer Science, articulo no publicado.

[14] WRAITH , G.C. *An Educational Role for Gofer ?.* Departamento de Matemáticas. Sussex University, articulo no publicado.