CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS

DEL

INSTITUTO POLITECNICO NACIONAL

DEPARTAMENTO DE INGENIERIA ELECTRICA

SECCION DE COMPUTACION

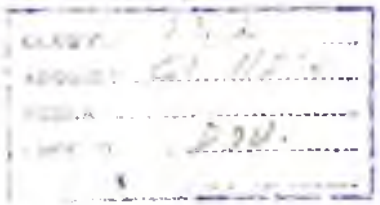
UN SISTEMA DE MANIPULACION SIMBOLICA INTERACTIVO

Tesis que presenta la Ing. Zandra Navarro Villicaña para obtener el grado de MAESTRO EN CIENCIAS en la especialidad de INGENIERIA ELECTRICA con opción en Computación.

Trabajo dirigido por el Dr. Josef Kolar Sabor

Becario de CONACYT

México D.F. a 10 de Julio de 1989



AGRADECIMIENTOS

Agradezco al Dr. Josef Kolar por su dirección y paciencia en el desarrollo de este trabajo, así como a su familia por su agradable compañía en México.

A todos mis profesores por sus consejos y su ejemplo, especialmente al Dr Manuel Guzmán, al Dr Guillermo Morales y al Ing. Javier Longoria.

A la Dra Shirley Bromberg y al M. en C. Oscar Olmedo agradezco enormemente su tiempo y las valiosas observaciones que me hicieron al revisar este trabajo.

A mis compañeros y amigos en México por el aliento que me han brindado siempre. Por su especial colaboración durante la edición a Marco Antonio Rocha, y por su apoyo gracias a: Ruth Delgado, Fernando Vázquez, Jorge Buenabad, Rubén Rusiles y Sandra Navarrete.

Al CINVESTAV, en particular a la Sección de Computación con cuyos recursos se terminó este trabajo.

Al CONACYT por haber facilitado los recursos para financiar mis estudios de maestría.

A mis padres Ofelia y Héctor

A mis hermanos Joel y Néstor

INDICE

INTRODUCCION

El presente documento describe un sistema interactivo de manipulación de simbolos. Se espera que este sistema pueda servir de apoyo a un estudiante que intenta resolver problemas de transformación, derivación e integración de expresiones algebraicas.

Se describe el sistema al usuario y después su implementación. El trabajo contiene 6 capítulos, en los primeros 3 se le detalla al usuario cual es la forma de trabajo con el sistema. En los últimos 3 se describe la implementación, de tal modo que quien se interese pueda realizar modificaciones al sistema ó usar alguna de sus partes según le convenga.

En el capítulo 1 se define cuales son los alcances del sistema, y se explican los conceptos de sesión de trabajo, variables implícitas, ventanas y terminadores, que son conceptos en los que se basa el sistema.

En el capítulo 2 se tiene la descripción de las opciones principales del sistema, para que sirven y como se usan.

En el capítulo 3 se explican las funciones de que dispone el usuario para incluir en una expresión.

Acerca de la implementación del sistema, en el capítulo 4 se presentan cuales fueron las estructuras internas utilizadas para almacenar los datos dentro del sistema.

En el capítulo 5 se encuentra una descripción de la configuración global del sistema, los módulos en que se dividió y su forma de funcionamiento general.

En el capítulo 6 se presenta con mayor detalle cada uno de los módulos descritos en el capítulo 5. Los Apéndices del B al I tienen el código del sistema, y se hace referencia a ellos principalmente en el capítulo 6.

En el Apéndice A se tiene la gramática que define formalmente el lenguaje de entrada del usuario.

En el Apéndice J se presenta un manual de referencia rápido, sin definiciones formales para empezar a usar el sistema.

En el Apéndice K hay una relación de mensajes que ocurren durante la ejecución del sistema.

CAP 1. DESCRIPCION DEL SISTEMA.

En este capitulo se presenta el funcionamiento general del sistema con idea de introducir al usuario a la filosofía de trabajo dentro del mismo. En primer lugar se definen los objetivos del sistema presentando la idea general de éste, después se describen los conceptos de sesión de trabajo, creación y uso de variables implícitas, iniciación de ventanas y su utilidad como parte de una sesión de trabajo, y por último el funcionamiento de un terminador y la repercusión que tiene en la evaluación de las expresiones introducidas, en los resultados que se generan y en la creación de variables implícitas.

1.1 Objetivos del sistema.

El sistema que constituye este trabajo de tesis tiene como objetivo utilizar la computadora como herramienta en el proceso de enseñanza-aprendizaje. Se aprovecha en el trabajo el potencial de la computadora para automatizar procesos mecánicos y repetitivos para el ser humano. El sistema se diseñó e implementó considerando al usuario final como el estudiante de cálculo diferencial e integral que va a resolver problemas planteados como ejercicios en su libro de texto. El sistema entonces le da la posibilidad de utilizar la computadora como la hoja de trabajo de su cuaderno en el proceso de resolución del problema.

También puede ser útil para aquellas personas que dedicadas a otras labores como docencia, investigación o ingeniería requieran en su actividad diaria de la obtención de soluciones a problemas de derivación e integración, que necesiten tratar con ellos para obtener resultados rápidos , compararlos, verificar respuestas, ó simplemente utilizarlos como apoyo en aplicaciones.

En el sistema se provee un medio de trabajo que le da facilidades de uso de ventanas y creación de sesiones, estableciéndose así parte de la interactividad del sistema. Dicha interactividad se extiende dando al usuario la posibilidad de manejar variables implícitas generadas por el sistema, funciones que permiten la

manipulación local de expresiones y terminadores de expresiones que incrementan la libertad del usuario para expresarse. La facilidad de interacción se complementa con una variedad de funciones de las que el usuario dispone para formular sus expresiones.

Para el caso de uso didáctico del sistema, cabe aclarar que no se diseñó para enseñar cálculo, ni sustituye a un profesor en el área. Sólo brinda apoyo principalmente en el trabajo fuera del aula. Tampoco se desea resolver al alumno las tareas, sino facilitarle la realización de las mismas ayudándole con el trabajo mecánico que se supone el estudiante es capaz de realizar por sí mismo, pero que, sin embargo, no constituye parte del objetivo principal de su curso en ese momento. Dentro del aula también puede ser útil en ocasiones en las que algún resultado desea encontrarse rápidamente para proseguir una presentación.

Se trata de que el usuario presente el problema que va a tratar de solucionar, al sistema. El sistema proporciona los medios para la edición del problema, la consulta de tablas, la ejecución de comandos tales como expansión, factorización, transformación de expresiones en general y aplicación de "fórmulas" de derivación e integración.

El sistema objeto de este trabajo no es tampoco un sistema tutorial, ya que, en primer lugar, éste no propone problemas al usuario, sino es el usuario el que presenta los problemas al sistema. En segundo lugar, porque no contiene ningún módulo de evaluación del aprendiz.

Con las herramientas que facilita el sistema, el usuario tiene la posibilidad de resolver el problema usando su propia iniciativa en el desarrollo de la solución.

De este modo a un estudiante le es posible resolver un número más grande de problemas en un tiempo menor y concentrarse en la esencia del material para llegar a conclusiones más rápidas y confiables basadas en los ejemplos.

En el caso de que el uso del sistema no se enfoque al aspecto didáctico se tiene también la posibilidad de resolver problemas de manera inmediata, con lo cual un profesor podría de manera rápida elegir ejemplos expresivos y representativos para exposiciones o tareas. El investigador podría realizar mayor número de experimentos

al utilizar el tiempo ahorrado en la resolución manual, lo que le ayuda también a conjeturar más rápido y con mayor certeza.

Con lo anterior es posible apreciar que las ventajas que presenta el sistema son suficientes como para motivar su construcción.

1.2 Sesión de Trabajo.

Una *sesión de trabajo* es la secuencia de los pasos efectuados durante la interacción del usuario con el sistema, lo cual se encuentra esquematizado en la Fig. 1.1. A cada paso corresponde una entrada, que es la expresión que provee el usuario al sistema, y una salida, que es el resultado que proporciona el sistema al usuario.

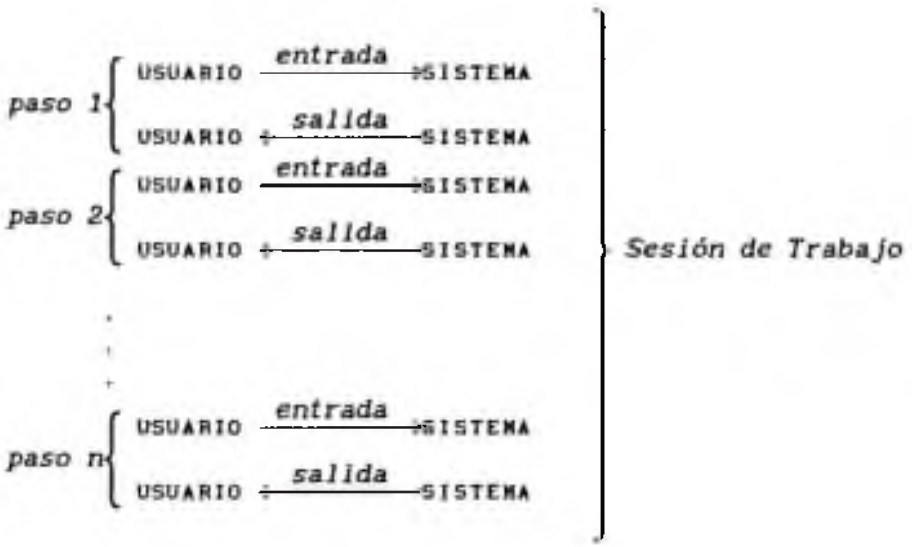


Fig. 1.1 Concepto de Sesión de Trabajo

Podemos establecer una analogía entre el trabajo de un usuario sobre su cuaderno de la manera tradicional y la sesión de trabajo que puede construir, ese mismo usuario, mediante el sistema que se está presentando.

Si consideramos al usuario tradicional, podemos ver que en la hoja de trabajo él va escribiendo una secuencia de formas equivalentes de un problema inicial mediante la aplicación de operaciones válidas.

En el caso del usuario usando el sistema, primeramente introduce la expresión que representa el problema inicial, después, en cada

entrada consecutiva proporciona una expresión indicando la operación que ha de realizarse sobre el problema inicial y el sistema responde con una salida en la que efectúa la operación indicada si es que ésta es válida.

De este modo las transformaciones que sufre una expresión paso a paso se van reflejando en las entradas y salidas consecutivas de una sesión.

Continuando con la analogía vemos que en el caso tradicional el trabajo de la persona ha quedado impreso en su cuaderno y por tanto deseamos que algo semejante sea posible mediante el sistema automático. Por tanto el usuario dispone del medio para guardar una sesión de trabajo en el disco magnético. Asimismo, es posible recuperar una sesión previamente guardada y continuar trabajando sobre ella. Además una sesión es desplegable en pantalla o por impresora en cualquier momento.

Durante la generación de una sesión se pueden realizar sobre ella modificaciones, como borrar algunos de los pasos de la sesión o bien borrarla toda e iniciar una sesión nueva. Podemos observar que una parte de sesión puede ser en sí misma una unidad como sesión, haciendo que la primer entrada constituya un problema inicial.

En la memoria de la computadora pueden permanecer almacenadas varias sesiones en un mismo tiempo pero sólo una de ellas se considera activa y todas las operaciones mencionadas se realizan sobre ella. Esto último está muy ligado al concepto de ventana activa que se revisará en la sección 1.4.

El área sobre la que se desarrolla una sesión es una ventana.

Nos enfocaremos ahora a la forma visual en que una sesión se va desarrollando dentro de una ventana. En el área que le toca aparece el prompt del sistema, indicando que el renglón está disponible para una entrada del usuario. El prompt de entrada del usuario se compone de dos partes. La primera es el número de entrada correspondiente y la segunda es la cadena '?: ', así como aparece en la Fig. 1.2.

En ese momento está posicionado el cursor, parpadeando, donde el usuario puede comenzar a escribir una expresión, cuya forma y estructura se describen en la sección 2.1.1.

Fig. 1.2 Prompt de Entrada

Además del prompt de entrada del usuario existe el prompt de salida o respuesta del sistema, también construido en dos partes, se tiene en primer lugar el signo '@' y en segundo lugar el número consecutivo de entrada, luego ':'. Si el usuario ha introducido algunas expresiones de entrada el desarrollo de la sesión comienza a verse como en la Fig 1.3.

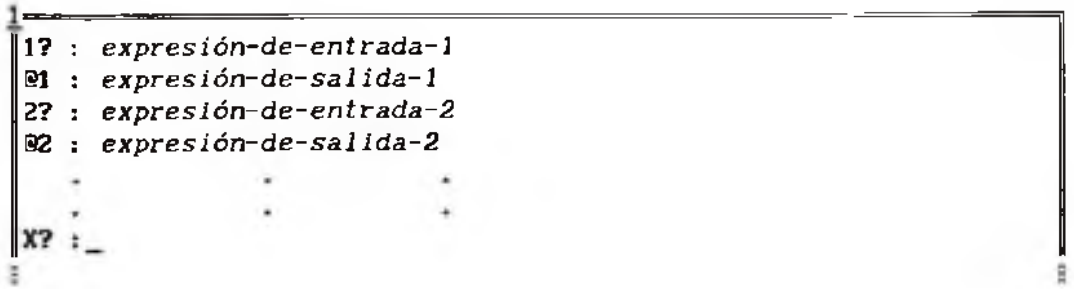


Fig. 1.3 Desarrollo de una Sesión

1.3 Variables Implícitas.

Por medio de las variables implícitas se enriquece la interacción entre el usuario y el sistema.

Las *variables implícitas* son las variables que van adquiriendo como valor la representación de las expresiones de entrada y de salida durante el desarrollo de una sesión de trabajo.

Adquieren su valor a través de una asignación implícita, generada por el sistema, de la expresión correspondiente a un nombre convencional. Esta asignación es como la que se obtendría si el usuario la realizará explícitamente, es decir, si tecleara el nombre de una variable asignándole una expresión de la manera expresada en la Fig. 1.4 a).

Así **var1** tendría como valor "a+b" y podría ser utilizada por medio del nombre **var1** en cualquier expresión subsecuente como se ve en la parte b) de la misma figura.

a) Asignación a **var1**

b) Uso de la variable **var1**

Fig. 1.4 Asignación y uso de una variable

Después de la generación de una variable implícita, ésta se puede referenciar por medio del nombre para formar otra expresión. Los nombres que se van generando están formados de dos partes. La primera es un caracter especial ? ó @, la segunda es un número, que corresponde al número de entrada o salida. Se usa '**?xx**' cuando la expresión que se va a asignar es de entrada y '**@xx**' cuando es de salida.

El efecto que se produce al utilizar en una expresión uno de estos nombres es la sustitución de la expresión correspondiente, sin tomar en cuenta el terminador. Pongamos el ejemplo de la Fig. 1.5.

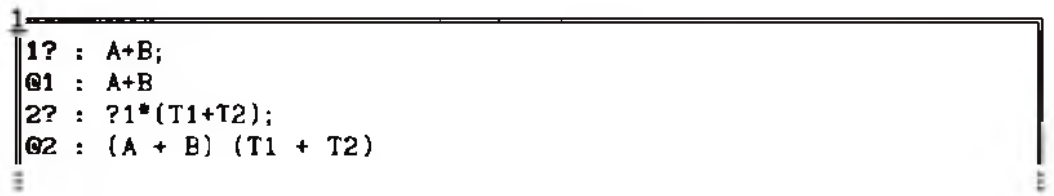


Fig. 1.5 Uso de la variable implícita **?1**.

Puede observarse que en la entrada 2 se utilizó la variable implícita ?1, generada en la entrada 1 y la sustitución ocurrida fue básicamente textual con la omisión del terminador ";" .

Un ejemplo más elaborado podría ser el de la Fig 1.6. En este ejemplo se utilizaron las variables implícitas **?1**, **@2**, **?2**.

```
1?  : 2 A B + 4 A C ;
@1  : 2 A B + 4 A C
2?  : FACTORIZA(?1);
@2  : 2 A *(B + 2 C)
3?  : EXPANDE(@2);
@3  : 4 A C + 2 A B
4?  : ?2;
@4  : 2 A *(B + 2 C)
```

Fig. 1.6 Uso de varias variables implícitas

En el primer caso se utilizó como argumento de la función de factorización la expresión que fue previamente introducida en la entrada 1. La expresión tecleada en la entrada 2 es equivalente a haber tecleado lo siguiente:

```
2? : FACTORIZA (2 A B + 4 A C);
```

De manera similar la entrada 3 es equivalente a haber tecleado:

```
3? : EXPANDE (2 A *(B + 2 C));
```

y la entrada 4 a:

```
4? : FACTORIZA (2 A B + 4 A C);
```

El sistema pues, va realizando automáticamente las asignaciones a las *variables implícitas*, de cada una de las entradas y salidas que forman los pasos de una sesión. El usuario podrá disponer entonces de cualquier expresión que haya introducido previamente, o de cualquiera que haya sido algún resultado anterior, evitándole tener que volver a teclearla completamente, con sólo usar la variable implícita en donde dicha expresión está almacenada.

También existe otro tipo de variables, llamadas *variables especiales* que tienen un funcionamiento semejante al de las variables implícitas. Son propias del sistema y tienen como fin ahorrar al usuario teclear una expresión larga o compleja, a cambio de utilizar

las variables especiales que la representan.

Las variables especiales representan, precisamente, un tipo especial de expresiones que son propiamente equivalencias y son utilizadas sólo por algunas funciones del sistema. Su uso es muy especifico y se explicará como se forman en la sección que se refiere a las tablas de fórmulas de identidades, derivación e integración [2.1.4]. Las variables especiales representan precisamente las fórmulas que ahí se tratan.

1.4 Ventanas.

Una *ventana* es un elemento del sistema cuya utilidad se aprovecha durante la interacción con el usuario.

Cuando un usuario resuelve en su cuaderno problemas de modo tradicional, es frecuente que recurra a realizar algunos cálculos u operaciones en espacios separados. También es costumbre resolver los problemas dividiéndolos en partes y después reuniendo las respuestas parciales para construir un resultado. En el caso de problemas de integración y derivación las tablas de "fórmulas" suelen estar en algún formulario separado del lugar donde se está resolviendo el problema.

Una *ventana* en el sistema es el área de la pantalla donde se desarrolla una sesión y se presentan tablas al usuario. También se utiliza para desplegar las instrucciones de ayuda para uso del sistema.

Las ventanas están numeradas por el sistema de manera secuencial en la esquina superior izquierda. En un momento dado solo es posible mantener una ventana activa. La ventana activa se distingue visualmente por que el número que le corresponde está desplegado en modo inverso.

Las operaciones principales que se pueden realizar con ventanas son: iniciarla, cerrarla y enviar información entre ventanas. Además es posible cambiar la ventana activa a la siguiente o a la previa, respecto de la que está actualmente activa.

Cuando existe más de una ventana en el sistema con una sesión de trabajo se tiene que todas las sesiones son independientes entre sí y

que todas las operaciones que se realicen sobre una sesión se referirán a la que está contenida en la ventana activa. Con las facilidades mencionadas vemos que el usuario puede producir una sesión en el sistema de manera más cercana a la forma tradicional.

1.5 Terminadores.

Los *terminadores* son elementos que tienen como fin primario dar al usuario un medio de indicar que la expresión que introduce ha terminado. Además un *terminador* permite indicar al sistema la forma en que se llevará a cabo el procesamiento de la expresión que se está introduciendo.

En el sistema se planearon 4 formas diferentes de procesar una expresión, en la figura 1.7 se muestra un diagrama en el que se puede ver el proceso que se sigue en cada caso. A continuación se describe cada una de las formas de procesar una expresión.

Forma Automática. El terminador asociado a la forma *automática* de procesar una expresión es el ";". Cuando una expresión es terminada con este caracter se realizan las operaciones siguientes:

a) Se evalúa la expresión de acuerdo a la prioridad de los operadores y funciones que estén involucrados. Se asigna la expresión a una variable implicita.
b) El resultado obtenido se presenta al usuario en la salida correspondiente . También se asigna el resultadoa la variable implícita apropiada.
c) El sistema despliega el prompt de la entrada siguiente y posiciona el cursor.

Considerando que %E9 es la regla de equivalencia siguiente:

$$\text{SEN (A+B)} \rightarrow \text{SEN A COS B + COS A SEN B}$$

se tienen a continuación algunos ejemplos del funcionamiento de este terminador.

```
1? : NUMERADOR((A+B)^2/SEN(A+B));
@1 : (A+B)^2
2? : EXPANDE(NUMERADOR((A+B)^2/SEN(A+B)));
@2 : 2 A B * A^2 + B^2
3? : APLICA(%E9, DENOMINADOR((A+B)^2/SEN(A+B)));
@3 : SEN A COS B * COS A SEN B
```

Lo que se está sucediendo en cada lugar es lo siguiente:

Entrada **1?** : Se pide extraer el numerador de la expresión dada.
Salida **@1** : El sistema escribe que el numerador es (A+B)^2
Entrada **2?** : Se pide expander el numerador de la expresión dada.
Salida **@2** : El sistema escribe la expansión del numerador de la expresión dada.
Entrada **3?** : Se pide al sistema aplicar la regla de equivalencia %E9, sobre la expresión que resulte ser el denominador de la expresión dada.
Salida **@3** : El sistema escribe el resultado de realizar la aplicación de la regla sobre la expresión.

Forma Muda. El terminador asociado a la forma *muda* de procesar una expresión es el "**$**". El procesamiento con el terminador mudo se lleva a cabo internamente igual que el procesamiento con el terminador automático. La diferencia entre ambos está en la parte visible del proceso, pues en el caso del terminador mudo el resultado obtenido NO se presenta al usuario, sin embargo la variable implícita que le toca a la salida se asigna debidamente. Finalmente se despliega el prompt de la siguiente entrada y se posiciona el cursor.

Ejemplificando lo anterior se tendría de modo similar al terminador automático lo siguiente:

```
1? : NUMERADOR((A+B)^2/SEN(A+B))$
2? : EXPANDE(NUMERADOR((A+B)^2/SEN(A+B)))$
3? : APLICA(%E9, DENOMINADOR((A+B)^2/SEN(A+B)))$
4? : @2;
@4 : 2 A B * A^2 + B^2;
```

En este ejemplo el usuario presenta las mismas tres primeras entradas al sistema que en los ejemplos dados para el terminador automático. El sistema hace lo mismo, menos escribir las salidas. En el paso **4** se tiene que:

Entrada **4?** : Se hace uso de la variable implícita @2.
Salida **@4** : El sistema presenta la expresión @2.

Forma Expresiva. El terminador asociado a una forma *expresiva* de procesamiento de expresiones es el "!". Esta forma de procesamiento de expresiones tiene más bien un fin informativo para el usuario. Podría verse como el inverso del terminador mudo. En realidad el procesamiento que se realiza internamente es el mismo que con el terminador automático; pero a diferencia del mudo, que no presenta ninguna respuesta al usuario, éste presenta información relativa al camino seguido por el sistema para obtener el resultado que presenta. Las asignaciones implícitas que se realizan son las mismas que con el terminador mudo.

Un ejemplo del uso de este terminador sería:

```
5? : INTEGRA(X SEN(X), X)!
Se llego a la función para integrar, con los siguientes
argumentos: X SEN(X) , X
Se tiene para derivar la expresión: SEN X
Realizando la(s) asociacion(es):
con U la expresion
   X
Se utiliza la siguiente fórmula de derivación:
d(sen U)/dx → cos U dU/dx

Se tiene para integrar la expresión: X SEN X
Realizando la(s) asociacion(es):
U con la expresión
   X
DV con la expresión
   SEN X
Se utiliza la siguiente fórmula para integrar:
∫ U dv → U V - ∫ V du

Se tiene para derivar la expresión: COS X
Realizando la(s) asociacion(es):
U con la expresión
   X
Se utiliza la siguiente fórmula para integrar
∫ cos U du → sen U
La integral obtenida fue : -X COS X + SEN X
OS: -X COS X + SEN X
```

Forma Dirigida. El terminador que indica una forma *dirigida* de manejar una expresión es "&". Esta forma de procesamiento es muy importante en este sistema pues es el que hace a éste diferente de otros sistemas de manipulación simbólica. No realiza ningún procedimiento de

transformación sobre la expresión. Lo que sucede es que se requiere que un sistema de este tipo brinde al usuario una herramienta para describir de que manera llegaría él al resultado. El usuario debe indicar las transformaciones que desea realizar, por medio de las expresiones que introduce y usando este terminador. En tales expresiones se invocan funciones especiales, que a pesar del terminador, siempre son ejecutadas. De otra manera el sistema se limita en su respuesta presentando como salida la misma expresión de entrada. El uso de este terminador requiere de la creatividad del usuario para resolver el problema de entrada.

Las funciones que siempre son ejecutadas se llaman *funciones ejecutables.*

Se presenta un ejemplo del uso del terminador &, suponiendo que la regla de integración %I10: $\int$ sen U du $\longrightarrow$ cos U

```
?1 : INTEGRA (SEN(4 X),X)&
@1 : ∫ (SEN (4 X)) dx
2? : INTEGRA(4 SUBEXP(?1,INTEGRA(A,X),A),X)&
@2 : ∫ (4 SEN (4 X)) dx
3? : 1/4 @2&
@3 : 1/4 ∫ (4 SEN (4 X)) dx
4? : 1/4 APLICA(%I10, 4 SEN (4 X), 4, U, 4 X)&
@4 : 1/4 (-COS (4 X))
5? : EQEXP (@1, @4);
@5 : Son expresiones equivalentes
```

Donde está sucediendo lo siguiente:

Entrada **?1** : Se presenta al sistema el problema a resolver.
Salida **@1** : El sistema no transforma la expresión con el &.
Entrada **?2** : El usuario escribe el problema, presentado el integrando multiplicado por 4, para tratar de completarlo y poder usar una fórmula.
Salida **@2** : El sistema presenta la expresión reescrita por el usuario.
Entrada **?3** : El usuario escribe el problema, presentándolo de tal manera que sea equivalente al original. Multiplica por 1/4, para equilibrar con el paso que hizo en ?2.
Salida **@3** : El sistema presenta la expresión reescrita por el usuario.
Entrada **@4** : Se pide aplicar la regla de integración %I0, sobre la expresión 4 sen (4 X), indicando que la diferencial de U es 4 y que se ha elegido U como 4 X. El resultado se multiplica por 1/4, para no alterar la expresión.

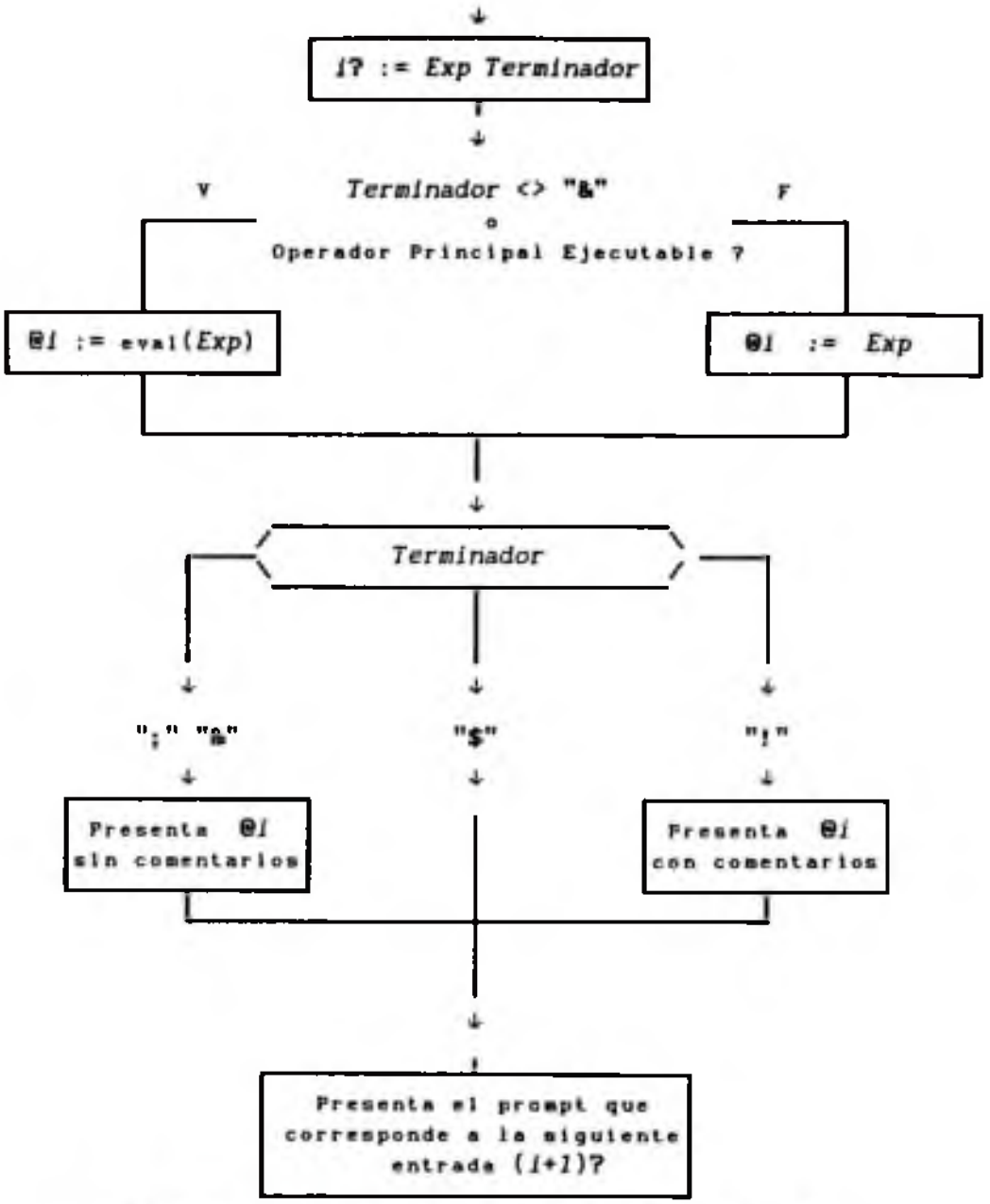


Fig. 1.7 Esquema del efecto de los terminadores sobre expresiones.

Salida ?4 : El sistema presenta el resultado de la aplicación.

Entrada @5 : El usuario intenta confirmar si el resultado obtenido en @4 es equivalente al que obtiene el sistema resolviendo el problema original presentado en @1.

Salida ?5 : El sistema indica que las expresiones dadas son equivalentes. Por lo tanto el usuario puede darse cuenta de que hizo las transformaciones adecuadas sobre el integrando y que aplicó la regla apropiada..

CAP. 2 LA INTERACCION DEL USUARIO CON EL SISTEMA.

En este capítulo se describe detalladamente el funcionamiento del sistema. Se dan a conocer las pantallas que se despliegan, su organización y el orden en que son presentadas las opciones, así como también, la forma de accesarlas . Al término se espera que el usuario pueda interactuar con el sistema y conozca la manera de utilizar todos los recursos que en él se ofrecen.

2.1 Posibilidades de Nivel Superior en el Sistema.

Durante la ejecución del sistema se presentan diversas pantallas. En todas las pantallas desplegadas siempre existe un área de ventanas, enseguida dos renglones de opciones, después un renglón para mensajes y al último un renglón de estado del sistema, como se esquematiza en
la Fig. 2.1.

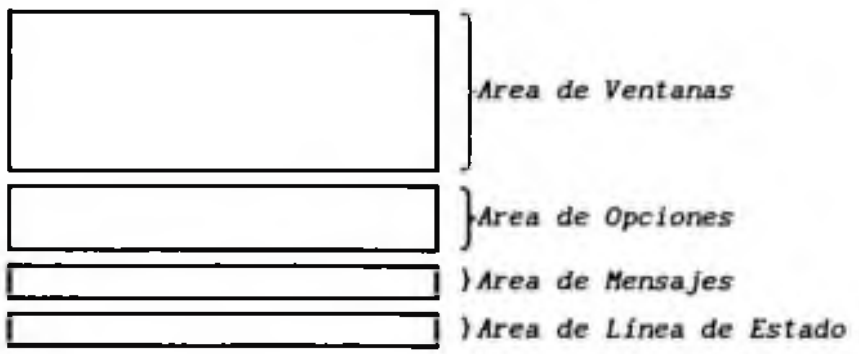


Fig. 2.1 Areas de una Pantalla.

El área para ventanas es el lugar donde usualmente desarrollará el usuario sus sesiones, prácticamente es su espacio para trabajar. En esa área se desplegará el prompt de entradas y salidas.

En el área de opciones se despliega la secuencia de opciones disponibles en el menú. En todas las pantallas hay una forma común para elegir cualquiera de las opciones que son presentadas en el área correspondiente. De todas las opciones, una de ellas está desplegada en modo de video inverso, de tal manera que parece que estuviera

iluminada. Mediante la barra espaciadora se puede lograr cambiar la iluminación a alguna otra de las opciones de la pantalla. Para elegir una opción determinada se debe primero iluminar la opción deseada y entonces oprimir la tecla de retorno de carro.

Haciendo uso de la primer letra de la opción también se puede elegir cualquiera de ellas, solo se oprime la tecla de su inicial y enseguida el sistema accesa la opción indicada.

En el área para mensajes aparecerán mensajes acerca del manejo de opciones y de la entrada de datos en las diferentes opciones del menú, según la relación que se enumera en el Apendice K.

La línea de estado del sistema tiene datos acerca de la sesión activa, como el nombre de la sesión y el número de entrada en que se encuentra dicha sesión.

La primer pantalla que se presenta tiene la forma que se describe en la Fig. 2.2.

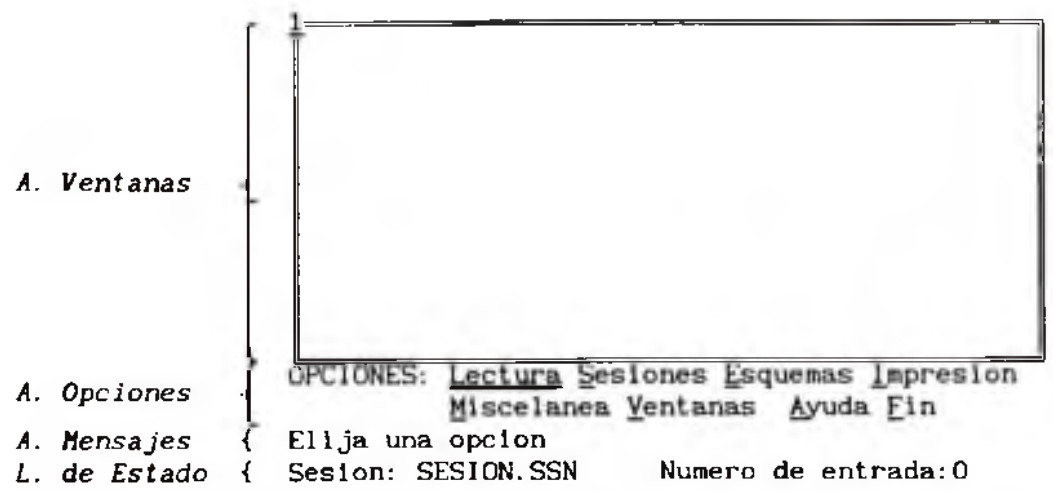


Fig. 2.2 Opciones de nivel superior del sistema

En los renglones correspondientes a las opciones, se puede ver cuales son las que se llaman *Opciones de Nivel Superior*.

Cuando se ha elegido cualquiera de las opciones inicialmente presentadas, ésta es ejecutada.

En lo que resta de esta sección se describirá cual es el objetivo de cada una de las Opciones de Nivel Superior y su uso.

2.1.1 Ciclo de Lectura y Evaluación.

El *Ciclo de Lectura y Evaluación* se refiere al medio ambiente que comienza cuando se elige la opción de LECTURA. El cursor se posiciona adelante del prompt de entrada que aparece en la ventana como se muestra en la Fig 2.3 referente al Inicio de una Sesión.

En este punto el usuario puede teclear una expresión, la cual estará escrita en el lenguaje del usuario. En adelante diremos que una expresión en este lenguaje esta escrita en Forma de Usuario (FU).

Es sencillo generar una expresión correcta en FU. En el Apendice J se dan ejemplos sencillos de ello y de como empezar a usar el sistema por primera vez. La manera formal en que se define el lenguaje de entrada es por medio de una gramática libre de contexto, la cual se presenta en el Apendice A.

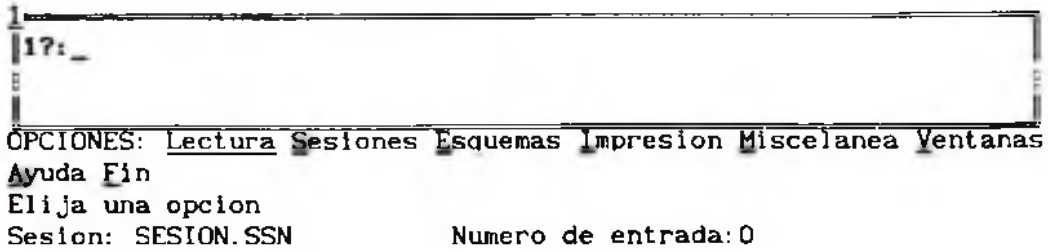


Fig. 2.3 Inicio de una sesión.

El ciclo de *Lectura y Evaluación* consiste de los siguientes dos pasos:

a) se despliega el prompt de entrada del sistema, delante del cual el usuario escribe una expresión .

b) según el terminador de la expresión, ésta es evaluada. Con la evaluación se obtienen los efectos laterales correspondientes tanto de asignación a variables implícitas, como de impresión.

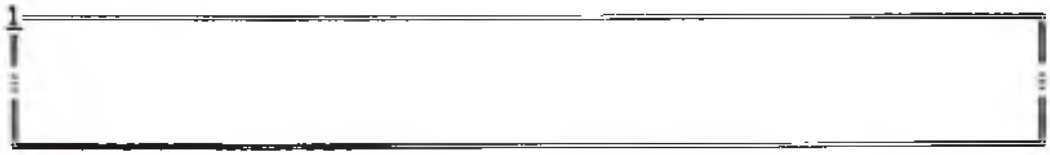
Esto se repite una y otra vez hasta que se presiona <ESC> en una entrada para volver a la pantalla inicial.

2.1.2 Sesiones.

En esta sección se presentan las opciones para manejar una sesión. En la figura 2.4 aparece un esquema de la pantalla que es

desplegada al elegir la opción de SESIONES en el menú inicial de OPCIONES.

Pueden existir en memoria tantas sesiones como ventanas haya abiertas, y cualquiera de las facilidades de la pantalla de SESIONES a la cual no le sea especificado un nombre de sesión, presupone que se hace referencia a la sesión que reside en la ventana activa. Además, cuando para llevar a cabo la tarea elegida es necesario desplegar resultados, éstos se reflejan en la ventana activa.



SESIONES: Despliega-Sesion Guarda-Sesion Carga-Sesion
Modifica-Sesion
Elija una opcion
Sesion: SESION.SSN Numero de entrada:0

Fig. 2.4 Menú de Sesiones.

Despliega-Sesión Esta facilidad permite desplegar la sesión que se encuentra en la ventana activa. Se puede utilizar cuando el tamaño de una sesión ya no permite que ésta se pueda visualizar dentro del espacio disponible de la ventana. Al ya no caber en este espacio, la sesión comenzó a desplazarse por la parte superior de la ventana y las primeras entradas sólo son accesibles nuevamente a través de la opción que nos ocupa.

Guarda-Sesión Esta opción permite almacenar en memoria secundaria la sesión residente en la ventana activa. Es necesario dar un nombre a la sesión y éste es requerido por el sistema al invocar la opción *Guarda-Sesion*. El nombre debe ser un nombre válido para archivos en el sistema MS-DOS. La extensión por omisión que añade el sistema es "SSN".

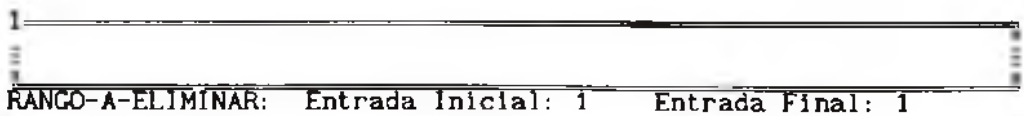
Carga-Sesión Por medio de *Carga-Sesión* puede ser llevada una sesión de memoria secundaria a memoria principal. Específicamente se coloca como la sesión residente en la ventana activa. El usuario debe determinar entonces si desea deshacerse de la sesión que se encuentra

en ese momento en la ventana activa. El sistema solicita el nombre en disco de la sesión que desea cargarse. La extensión por omisión es "SSN".

Modifica-Sesión Esta opción permite al usuario eliminar las partes de la sesión activa que ya no requiere. En todas las sesiones siempre corresponde una salida a cada una de las entradas y todas están identificadas de manera única como ha sido explicado en la sección 1.1. La eliminación de un rango de entradas elimina automáticamente las salidas correspondientes, por lo tanto, para dar el rango a eliminar solo es necesario dar el número de la entrada inicial y el número de la entrada final sin la identificación "?" que significa una entrada. El sistema presenta los prompts necesarios para introducir el rango deseado, Fig 2.5. Los efectos laterales de asignación que hayan sido realizados por alguna de las entradas eliminadas permanecen. De manera similar, no importa si en alguna de las expresiones del rango eliminado se usaron variables implícitas, pues desde el momento en que fueron usadas el sistema las substituye por la expresión que representan, al igual que con las asignaciones. Por ejemplo. si se ha tecleado la siguiente sesión:

```
1? : A+B;
@1 : A+B
2? : ?1*@1;     (ambas VI son substituidas inmediatamente por A+B
@2 : (A+B)^2
```

y después se borra de la sesión la entrada 1, se puede hacer referencia a ?2, que usaba variables implícitas y tendrá como expresión asociada (A+B)^2. Pero la numeración será recorrida y se convierte en la entrada 1.




```
RANGO-A-ELIMINAR:  Entrada Inicial: 1     Entrada Final: 1
Numero de entrada INICIAL. Use <TAB> para dar entrada FINAL
Sesion: C:SESION.SSN                  Numero de entradas: 4
```

Fig. 2.5 Datos para la opción *Modifica-Sesion*

Borra-Sesión Se encarga de borrar de memoria la sesión activa permitiendo que una sesión nueva pueda comenzarse en esa área. Inicializa el medio ambiente para una sesión.

2.1.3 Impresión.

La opción de IMPRESION en el menú principal de opciones da la facilidad de mandar la sesión residente en la ventana activa a la impresora. En la impresión que se obtiene no figuran los caracteres especiales que se ven en la pantalla, como los signos usados para el símbolo de la integral y de la diferencial. En caso de que exista algún problema físico con la impresora, éste es reportado por el sistema.

2.1.4 Esquemas.

En esta sección se describen las opciones que permiten desplegar en la ventana activa las tablas de equivalencias, de derivación e integración disponibles en el sistema.

Al invocar la opción de ESQUEMAS se presenta un submenú como el que figura en el área para opciones de la Fig. 2.6. Por medio de este submenú puede el usuario elegir la tabla que necesite y usando las teclas de PgUp, PgDw, Flecha izquierda y Flecha derecha puede buscar algún esquema específico en la tabla desplegada. Sea cual fuere la tabla escogida, ésta es desplegada en el área correspondiente a la ventana activa. En caso de que en la ventana activa haya una sesión, ésta es borrada únicamente de la pantalla, no de la memoria, para presentar la tabla. En la Fig 2.6 se abrieron, como ejemplo, dos ventanas. En la ventana 1 están desplegadas las tablas de equivalencias, que se refieren principalmente a identidades trigonométricas. La ventana 2 de esa misma figura contiene las tablas de Derivación del sistema.

Cada una de las fórmulas que son desplegadas en estas tablas representa una equivalencia. El lado izquierdo puede convertirse en el lado derecho en cada caso. Todas las identidades están identificadas por medio de una letra y un número. La "E" se refiere a las

identidades trigonométricas y logaritmicas. La "D" se refiere a las tablas de derivación y la "I" a las tablas de integración.

La identificación se usa para formar un tipo especial de variables que son usadas por la función APLICA cuyo uso y significado se explica en el siguiente capítulo. Uno de los parámetros de APLICA es una variable especial que empieza con el signo "%" y a continuación la identificación de un esquema de alguna de las tablas.

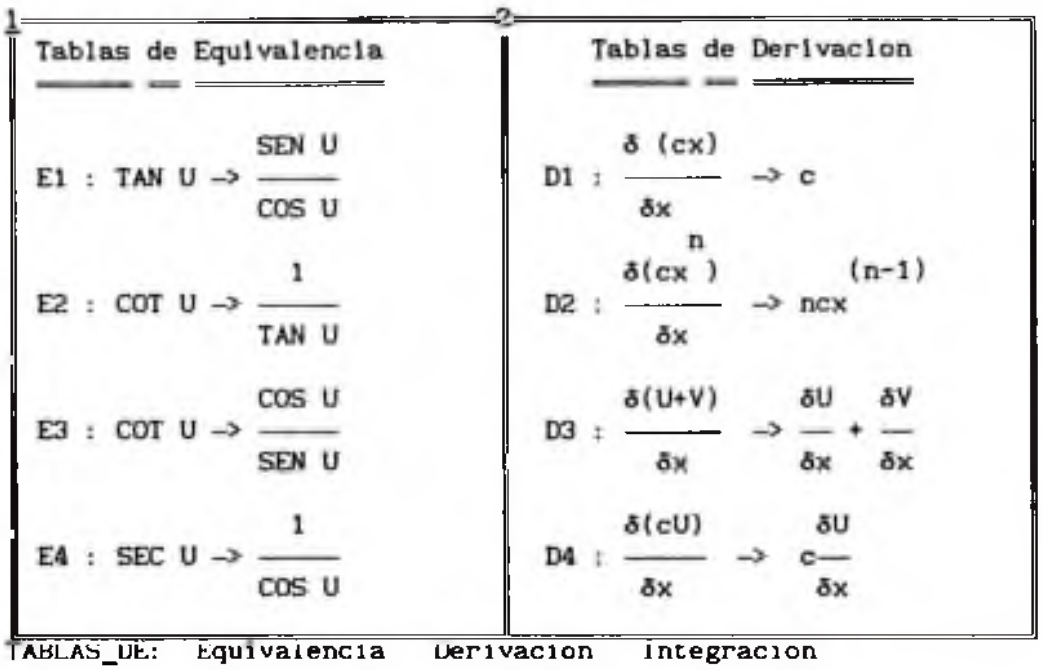


Fig. 2.6 Uso de la opción de ESQUEMAS.

Dichas variables son llamadas *variables especiales* y tienen la forma %Exx, %Dxx o %Ixx. Donde xx es el número que le corresponde a la fórmula en su tabla respectiva.

2.1.5 Miscelánea.

La opción de MISCELANEA permite realizar las siguientes operaciones:

a) configurar los colores a usar en las distintas áreas de la pantalla

b) configurar la pantalla de acuerdo al tipo de monitor

c) encender o apagar la campana de error

d) ejecutar comandos de MS-DOS.

Al elegir la opción de MISCELANEA el sistema presenta el submenú de la Fig. 2.7.

Al accesar la opción *Colores*, se desplegarán los prompts necesarios para introducir el código del color que el usuario desea usar para el texto dentro de las ventanas, las opciones del menú, la línea de prompt, la línea de estado, el fondo y el borde de la pantalla. Los códigos de color válidos son los números enteros del 0 al 15 y proporcionan los siguientes colores:

0 Negro	**4** Rojo	**8** Gris Oscuro	**12** Rojo Claro
1 Azul	**5** Magenta	**9** Azul Claro	**13** Magenta Claro
2 Verde	**6** Cafe	**10** Verde Claro	**14** Amarillo
3 Azul Cielo	**7** Gris Claro	**11** Azul Cielo Claro	**15** Blanco

Si la opción de *Monitor* se selecciona, el sistema presenta los prompts necesarios para dar la información acerca de las siguientes variantes de configuración:

a) Modo de pantalla (Texto o Grafico)

b) Resolución de pantalla (Media o Alta)

c) Cuando el adaptador es tipo EGA se debe especificar si el monitor es a Color, Enhanced o Monocromático.

MISCELANEA: Colores Monitor AvisaError Sistema_Dos

Elija una opcion
Sesion: SESION.SSN Numero de entrada:0

Fig. 2.7 Submenú de MISCELANEA

La opción de *AvisaError* permite activar o desactivar la campana que, cuando está activa, suena si el usuario teclea en algún menú una opción que no existe.

La opción *Sistema_Dos* permite mantener el sistema en memoria e ir a ejecutar comandos de MS-DOS, cuando el archivo COMMAND.COM está accesible. Se puede ejecutar un solo comando o varios.

Para ejecutar solamente un comando, éste debe invocarse cuando se presenta el prompt de MS-DOS.

Para ejecutar varios comandos se teclea <RETURN> cuando es presentado el prompt de MS-DOS. A a partir de ahí el sistema es controlado totalmente por MS-DOS y para regresar al sistema de manipulación simbólica se teclea el comando EXIT.

2.1.6 Ventanas.

La opción del menú principal VENTANAS permite el manejo conceptual de ventanas. El submenú que se despliega al escoger esta opción es presentado en la Fig 2.8.

Ya han sido mencionadas las operaciones que se realizan sobre una ventana, a continuación se describen y se presenta la manera en que se llevan a cabo en el sistema utilizando el submenú de la Fig 2.8.

a) **Iniciar una ventana.** Lo que significa que el usuario tiene la posibilidad de crear ventanas con un medio ambiente limpio para comenzar una sesión. Al principio existe una ventana inicializada que tiene el número de identificación 1. Cada vez que se crea una ventana, el sistema asigna a ésta el siguiente número consecutivo disponible para una ventana.

VENTANAS: DividirVentana CerrarVentana SiguienteVentana
PreviaVentana EnviaVentana
Elija una opcion
Sesion: SESION.SSN Numero d entrada:0

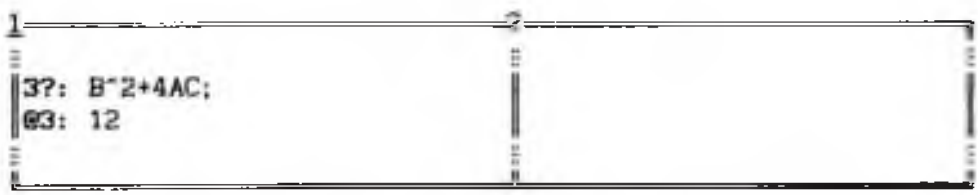
Fig. 2.8 Submenú correspondiente a la opción de VENTANAS.

Para iniciar una ventana se usa la opción *DividirVentana*, pues para crear una nueva ventana es necesario hacer más pequeña alguna de las que ya existen y por tanto se va a dividir. La

ventana candidata para dividir es siempre la ventana activa. El sistema deja que el usuario decida de que manera se realizará la división, si de modo VERTICAL u HORIZONTAL y solicita un número de columna o renglón respectivamente para determinar el lugar donde se realizará la división de la ventana activa.

b) **Cerrar una Ventana.** Es la operación inversa a Iniciar una Ventana. Se usa cuando el usuario desea eliminar alguna de las ventanas en que se está desarrollando algúna sesión o que fue utilizada para desplegar tablas o pedir ayuda. La opción indicada para realizar esta operación es *CerrarVentana*. El sistema desaparece la ventana que el usuario le indique, verificando con el usuario, si existe sesión residente, que ésta haya sido guardada en memoria secundaria o que pueda ser eliminada.

c) **Enviar información entre ventanas.** Esta operación da la facilidad al usuario para que tome resultados parciales que se desarrollan en alguna sesión y los siga desarrollando en un espacio separado, es decir en alguna otra ventana. Para ello necesita transferir información de una ventana a otra. La información que puede transferirse es una entrada o una salida que haya sido obtenida en el desarrollo de alguna sesión residente en cualquiera de las ventanas abiertas.



TRANSFIERE-INFORMACION : Enviar:Entrada Salida Numero: 3
Ventana Fuente: 1 Ventana Destino: 2
<Espacio> para elegir Entrada o Salida. <TAB> para ir a num
Sesion: SESSION.SSN Numero de entradas: 3

Fig. 2.9 Datos para el envio de información.

La opción usada para está operación es *EnviaVentana*. Cuando se elige, el sistema presenta los prompts desplegados en la Fig. 2.9, para que el usuario introduzca la descripción de la

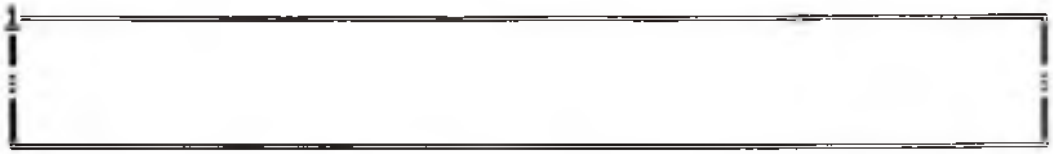
información a enviar, así como la fuente y el destino de ésta.
En la descripción de la información que se desea enviar se debe especificar si se elige enviar una entrada o una salida y el número correspondiente de ésta.
Para especificar la fuente se da el número de identificación de la ventana que contiene la información que se desea enviar. El sistema verifica que tal exista.
De manera similar se requiere el número de la ventana que recibirá la información. La entrada o salida seleccionada por el usuario pasará a ser la siguiente entrada disponible en la ventana elegida como destino.
En la Fig. 2.9 hay dos ventanas. La primera contiene varias entradas. Se podría elegir mandar la entrada 3 de la ventana 1 a la ventana 2.

d) **Cambiar de Ventana.** Esta operación simplemente permite al usuario cambiar el número de la ventana activa, hacia la que sigue en número de identificación respecto a la ventana actual o a la que la antecede. Esto lo realiza por medio de las opciones de *SiguienteVentana* y *PreviaVentana* respectivamente. La tecla de función <F1> realiza también la operación.

2.1.7 Ayuda

La opción de AYUDA es útil en diversas ocasiones, cuando el usuario, que está familiarizado con los conceptos de terminadores, ventanas y sesiones, requiere consultar la sintáxis, el uso o el significado de alguna función, o bien requiere encontrar en donde está alguna facilidad en el árbol de opciones del menú y saber para que sirve.

La información que provee está totalmente contenida en el capítulo 2 y 3 de este trabajo escrito. Al elegir el usuario esta opción, se le presenta la posibilidad de solicitar tres tipos de información: generalidades, información acerca de las funciones disponibles para las entradas válidas e información acerca de las opciones del menú, según se presenta en la Fig 2.10.



AYUDA: Generalidades Funciones Opciones-del-Menu

Elija una opcion
Sesion: SESION.SSN Numero de entrada:0 AYUDA

Fig. 2.10 Submenú disponible para solicitar AYUDA.

El acceso a cualquiera de los submenues que se presentan en está opción tiene como efecto lateral la presentación en la ventana activa de las explicaciones correspondientes a la opción elegida. En la línea de estado aparece un indicador que permite distinguir que se está en la opción de ayuda y no en las opciones de nivel superior del sistema.

2.1.8 Fin

Esta opción del menú permite terminar la ejecución del sistema. No sin que éste verifique antes que el usuario guarde en memoria secundaria las sesiones y medios ambientes que podría requerir en otra ocasión que use el sistema.

CAP 3. FACILIDADES PROVISTAS EN EL SISTEMA.

Antes de presentar las facilidades que brinda el sistema por medio de funciones, se establecen algunas convenciones en cuanto a nombres de constantes predefinidas, terminología y orden de variables en una expresión.

Después se presentan las funciones que pueden llegar a formar parte de una expresión. Todas ellas se pueden ver como operadores prefijos que actuan sobre varios argumentos, por tanto, junto con las funciones se agregó una descripción de los operadores aritméticos que se pueden incluir en una expresión. Los argumentos de una función se escriben después del nombre de ésta, encerrados entre paréntesis y separados por comas. Esto último se puede ver con más precisión en donde se describe la gramática para el lenguaje de entrada del sistema, el Apendice A.

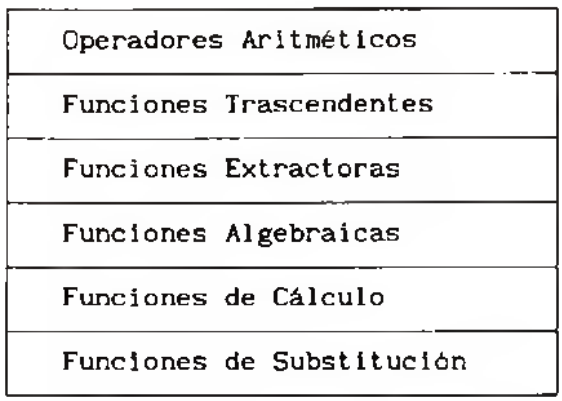
Las funciones provistas por el sistema son clasificadas de acuerdo a su área de aplicación, primero se tratan de manera independiente los *operadores aritméticos*, enseguida las *funciones trascendentes*, después se describen aquellas funciones que se utilizan para obtener una parte de una expresión y son llamadas *funciones selectoras* pues sirven para seleccionar alguna parte de una expresión que luego puede ser tratada de manera independiente igual que una expresión. Otro tipo de funciones son las que realizan manejo algebraico con una expresión, y son llamadas *funciones algebraicas*.

Además se tienen las funciones que sirven para derivar e integrar expresiones, con otras dos funciones que complementan el módulo de cálculo y son llamadas *funciones de cálculo*. Por último existe otra función que sirve para realizar substituciones de expresiones de acuerdo a una "fórmula" o "esquema", el cual se provee según las tablas de esquemas que se detallaron el la sección 2.1.4 y su clasificación es de *funcion de substitución*.

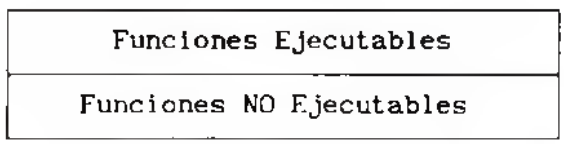
Todas las facilidades del sistema, que están provistas como funciones, se pueden colocar dentro de alguna de estas seis categorías, que fueron determinadas de acuerdo al área de aplicación de la función.

Además de este criterio se consideró uno basado en un principio distinto que el del área de aplicación de la función. Se trata de clasificar las funciones según el efecto que realiza el terminador dirigido sobre ellas. Bajo dicho criterio existen entonces *funciones ejecutables* y son aquellas que se evaluarán cuando se encuentren incluidas en una expresión que ha sido terminada con "&", el terminador dirigido.

Los criterios de clasificación de funciones que han sido establecidos pueden esquematizarse como se muestra en la figura 3.1.



a) De acuerdo al área de aplicación.



b) De acuerdo al efecto de "&".

Fig. 3.1 Clasificaciones de Funciones

A continuación se agrupan primeramente las funciones del sistema de acuerdo a la clasificación de área de aplicación. En todos los casos en los que se hace referencia a una expresión <expl> se supone que se está hablando de una expresión correctamente escrita de acuerdo a la gramática del apéndice A.

En la descripción de cada función se incluyen básicamente tres aspectos, en primer lugar su USO, poniendo la forma en que puede ser usada la función que se está describiendo.

Después se explica el SIGNIFICADO del uso de la función, es

decir, se describe lo que hace la función sobre sus argumentos, así como la forma y tipo que se requiere para éstos.

Finalmente se dan algunos EJEMPLOS específicos para usarla.

En la última sección de este capítulo se ve con detalle el segundo criterio de clasificación mencionado y se determina cuales son las funciones ejecutables.

3.1 Convenciones.

Antes de proceder a la presentación de las funciones ya clasificadas es conveniente establecer algunas convenciones de utilidad en cuanto a nombres de constantes predefinidas, terminología y orden de los términos y factores dentro de una expresión.

En primer lugar tenemos dos nombres especiales que representan dos constantes matemáticas muy usuales.

Estas son, el número π que se conocerá dentro del sistema como **%PI** y en el número *e*, base de los logaritmos naturales, que se conocerá dentro del sistema como **%E**.

Dentro de la descripción de funciones se mencionaran en algunos casos varios términos de los que a continuación se describen:

Expansión Completa. El producto de dos sumas y la potencia entera de una suma se pueden expander usando la siguiente transformación:

$$(X + Y) (Z + W) \rightarrow X Z + Y Z + X W + Y W$$

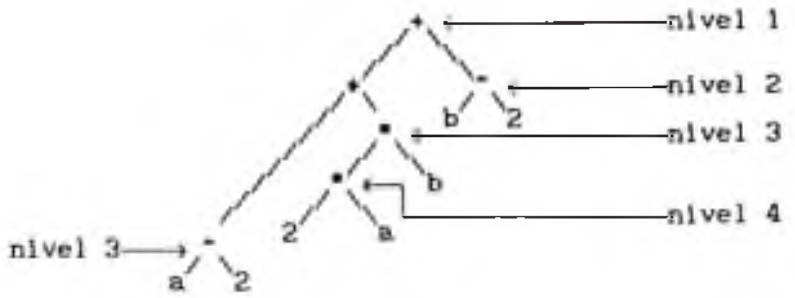
y una expresión se considera completamente expandida si todos los productos y potencias como las que se mencionan han sido expandidas.

Expresión Semifactorizada:

Una expresión está *semi-factorizada* cuando los factores que se extrajeron son el máximo común divisor obtenido de los términos de la expresión. No son el resultado de haber encontrado un factor que dividiera exactamente a la expresión.

Estructura de una Expresión:

Se llama estructura de una expresión al árbol de asociación de los operadores que en ella se involucran. Para hacer ésto más claro vamos a poner algún ejemplo:



A la expresión a^2 + 2*a*b + b^2 corresponde el anterior árbol para asociar sus operadores, de acuerdo a la jerarquía convencional para estos operadores y como resultado de la asociatividad que les correponde. Los operadores + y * se asocian por la izquierda y el operador ^ por la derecha.

Se dice que la *estructura de la expresión* tiene cuatro niveles. Se puede describir la *estructura de la expresión* desde cualquier nivel dando nombres a las ramas que deseen abreviarse. De tal manera que la estructura de la expresión de nuestro ejemplo puede describirse de varias maneras:

$X + Y$ haciendo que X sea la rama $a^2 + 2\ a\ b$
y Y sea la rama b^2.

o bien

$X^2 + 2\ Y + Z$ haciendo que X sea a
que Y sea a b
y Z sea b^2.

Dentro de una expresión que es una suma los términos se ordenan de la siguiente manera: números, variables, productos, potencias y funciones. Se ignoran los coeficientes numéricos de un término.

Cuando una expresión es un producto sus factores son ordenados de la siguiente manera: números, variables, sumas, potencias y funciones.

3.2 Operadores Aritméticos.

El sistema tiene implementado manejo aritmético racional exacto, que permite realizar sumas, restas, productos, cocientes y potenciación de números racionales. Los símbolos para denotar estas operaciones son respectivamente +, -, *, / y ^.

El tamaño de los números racionales se limita sólo por el espacio de memoria disponible en la computadora.

El símbolo correspondiente a la multiplicación usualmente se puede omitir, salvo cuando se utiliza antes de una expresión entre paréntesis, con el fin de distinguir el producto del uso de una función.

Con el símbolo de "/" se expresan fracciones, éstas siempre son simplificadas a su mínima expresión.

La jerarquía definida para los operadores es la que se utiliza convencionalmente. Las primeras operaciones en realizarse son las potenciaciones (asociando por la derecha), a continuación el menos unario, después las multiplicaciones y las divisiones (asociando por la izquierda) y finalmente las sumas y las diferencias (asociando por la izquierda).

3.3 Funciones Trascendentes.

Las funciones trascendentes que maneja el sistema son de cuatro tipos:

a) Funciones trigonométricas directas: SEN, COS, TAN, COT, SEC, CSC.

b) Funciones trigonométricas inversas: ASEN, ACOS, ATAN, ACOT, ASEC, ACSC.

c) Funciones trigonométricas hiperbólicas: SENH, COSH, TANH, COTH, SECH, CSCH.

d) Funciones logaritmicas: LOG, LN, exponenciación ej. $e^{f(x)}$. Para la exponenciación se usa el operador aritmético ^, como se presentó en la sección anterior.

Las funciones del tipo a), b) y c) funcionan similarmente. Si **NomFun** es el nombre de cualquiera de ellas, se tiene lo siguiente en todos los casos:

USO: **NomFun** *(exp1)*

SIGNIFICADO: Obtiene *NomFun* de la expresión *exp1*. *Exp1* es una expresión que el sistema espera recibir en radianes.

En los resultados nunca se intenta aproximar funciones trigonométricas que sean irracionales.

Los senos y cosenos de ángulos que son múltiplos numéricos de π son reducidos a senos y cosenos en el rango $[0, \pi/4]$.

EJEMPLO:

```
SEN ( %PI ) = 0                    ACOS(2-X^2)=%PI/2-ASEN(2-X^2)
COS ( %PI/6 ) = 3^(1/2)/2
```

Para el caso de LOG y LN se tiene lo siguiente:

LOG

USO: **LOG** (*exp, base*)

SIGNIFICADO: Obtiene la expresión que representa el logaritmo de *exp* en base *base*.

LN

USO: **LN** (*exp*)

SIGNIFICADO: Significa lo mismo que LOG(*exp*, %E).

3.4 Funciones Selectoras.

El objetivo principal de las funciones selectoras es obtener subexpresiones. Estas pueden ser utilizadas como expresiones independientes, sobre las cuales se pueden realizar todas las operaciones propias para una expresión.

NUMERADOR

USO: **NUMERADOR** (*exp*)

SIGNIFICADO: Si la *exp* tiene numerador, entonces la función da como resultado el numerador de *exp*. De cualquier otro modo el resultado será la misma *exp*.

EJEMPLO:

NUMERADOR(X^2/(X+Y))=X^2 NUMERADOR(X^2+X^3+X^4)=X^2+X^3+X^4

DENOMINADOR

USO: **DENOMINADOR** (*exp*)

SIGNIFICADO: Si la *exp* tiene denominador, entonces la función da como resultado el denominador de *exp*. De cualquier otro modo el resultado será 1.

EJEMPLO:

DENOMINADOR(X^2/(X+Y)) = X+Y DENOMINADOR(X^2+X^3+X^4)= 1

Es importante comentar que NUMERADOR y DENOMINADOR son funciones complementarias y que la relación que se escribe a continuación siempre se mantiene:

$$exp = \frac{\text{NUMERADOR} \quad (\ exp\ )}{\text{DENOMINADOR}\ (\ exp\ )}$$

COEFICIENTE

USO: **COEFICIENTE** (*exp*)

SIGNIFICADO: Si *exp* es numérica, entonces la función siempre regresa *exp* . Si *exp* no es un producto, entonces la función siempre regresa 1. Cuando la función sí representa un producto, entonces el resultado es una expresión que es el coeficiente de *exp*, es decir, los factores numéricos de *exp*.

EJEMPLO:

COEFICIENTE(3 SEN(A+B)) = 3 COEFICIENTE(A+B) = 1

BASE

USO: **BASE** (*exp*)

SIGNIFICADO: Cuando *exp* tiene la forma base^exponente, la función regresa una expresión que es base. Cuando la *exp* no tiene esa forma la función regresa *exp*.

EJEMPLO:

BASE(COS(X)^SEN(X))=COS(X)

EXPONENTE

USO: **EXPONENTE** (*exp*)

SIGNIFICADO: Cuando *exp* tiene la forma base^exponente, la función regresa una expresión que es exponente. Cuando la *exp* no tiene esa forma la función regresa 1.

EJEMPLO:

EXPONENTE(COS(X)^SEN(X))=SEN(X)

Igual que NUMERADOR Y DENOMINADOR, las funciones BASE Y EXPONENTE son complementarias y respecto a ellas, la relación siguiente siempre se mantiene.

exp = BASE (*exp*) ^ EXPONENTE (*exp*)

SUBEXP

Esta función es muy poderosa y usándola sola puede extraer cualquiera de las partes que se obtienen con las funciones anteriores, NUMERADOR, DENOMINADOR, COEFICIENTE, BASE y EXPONENTE, sin embargo, áquellas se proveen para abreviar trabajo al usuario y se recomienda utilizarlas en casos muy particulares.

USO: **SUBEXP** (*exp-1*, *exp-2*, *exp-3*)

SIGNIFICADO: La función obtiene como resultado una expresión con la estructura de *exp-3* y en la cual se substituye el empatamiento obtenido al realizar una comparación entre *exp-1* y *exp-2*, si es que *exp-1* y *exp-2* tuvieron la misma estructura. El caso en que *exp-1* y *exp-2* no tengan la misma estructura sin importar el nivel de detalle, entonces la llamada a SUBEXP no es válida y se recibe el mensaje "Arg1 y Arg2 no tienen la misma estructura". La operación no se efectua y la entrada y salida correspondiente son anuladas regresando la sesión al estado anterior.

EJEMPLO:

```
SUBEXP(X^2 + TAN(X+Y)/2 , A+B/2, A) = X^2
SUBEXP(X^2 + TAN(X+Y)/2 , A+B/2, A^2 + B^2)=(X^2)^2 + TAN(X+Y)^2
SUBEXP(X^2 + TAN(X+Y)/2 , A+TAN(C)/2, C ) = X+Y
```

3.5 Funciones Algebraicas.

Este tipo de funciones realizan transformaciones algebraicas sobre su argumento.

CUADRADO

USO: **CUADRADO(** *exp1*)

SIGNIFICADO: Esta función eleva su argumento a la segunda potencia. Lo mismo sucedería si se elevara la expresión *exp* al cuadrado.

CUADRADO(*exp*) = *exp* ^ 2

EJEMPLO:

```
CUADRADO(CSC(X))= CSC(X)^2
```

EXPANDE

USO: **EXPANDE** (*exp*)

SIGNIFICADO: La función regresa como resultado una expresión que es equivalente a *exp* con un denominador completamente expandido y distribuido sobre los términos de un numerador también completamente expandido.

EJEMPLO:

```
EXPANDE((A+B)^2);
2 A B + A^2  + B^2
```

EXPD

USO: **EXPD** (*exp*)

SIGNIFICADO: El llamado a la función regresa como resultado una expresión que es equivalente a *exp*, donde el numerador está completamente expandido sobre un común denominador completamente expandido.

EJEMPLO:

```
EXPD(2 + X/(1+X)) = (2 + 3 X) / (1 + X)
```

FACTORIZA

USO: **FACTORIZA** (*exp*)

SIGNIFICADO: El llamado a la función regresa como resultado una expresión que es equivalente a *exp* con un numerador semi-factorizado sobre un denominador semi-factorizado. A esta extracción del máximo común divisor de los términos del denominador se le llama factorización contenida.

EJEMPLO:

```
FACTORIZA(X^2 - 2 X Y + Y^2)= X (X - 2 Y) + Y^2
FACTORIZA(X^3+X Y^2 + 3 X^2 Y + 2 X Y^2 + Y^3)=
                                        X*(3 Y*(X+Y) + X^2) + Y^3
```

RACIONALIZA

USO: **RACIONALIZA** (*exp*)

SIGNIFICADO: El llamado a esta función da como resultado una expresión equivalente a *exp*, en la cual se han eliminado exponentes fraccionarios del denominador, mediante la operación de multiplicar numerador y denominador por una expresión que elimie a éstos del denominador.

EJEMPLO:

```
RACIONALIZA(X / 2^(1/2) ) = 2^(1/2) X/2
```

EQEXP

USO: **EQEXP** (*exp1, exp2*)

SIGNIFICADO: Esta función sirve para que el sistema determine si *exp1* y *exp2* son dos expresiones equivalentes. Regresa como valor la una frase que indica si éstas fueron o no equivalentes.

EJEMPLO:

```
EQEXP (SEN(X)/COS(X), TAN(X))
Son expresiones equivalentes
EQEXP (1/CSC(X), TAN(X))
NO son expresiones equivalentes
```

3.6. Funciones de Cálculo.

Son funciones que sirven para encontrar derivadas parciales de una expresión y para calcular la integral de una expresión. También son necesarias las funciones para definir la dependencia entre variables.

DEPENDE

USO: **DEPENDE** (*VAR-DEP VAR-IND1 VAR-IND2 ... VAR-INDn)*

SIGNIFICADO: **DEPENDE**, sirve para establecer que *VAR-DEP* es una variable que depende de las variables *VAR-IND1, VAR-IND2 ... VAR-INDn.*

EJEMPLO:

DEPENDE(U, X, Y ,Z) establece que u = f(x,y,z)

NODEPENDE

USO: **NODEPENDE** *(VAR-DEP VAR-IND)*

SIGNIFICADO: Sirve para eliminar las dependencias que hayan sido establecidas por DEPENDE.

EJEMPLO:

Suponiendo que se había establecido que DEPENDE(U,X,Y,Z)

NODEPENDE(U, Y) indica que ahora u = f(x,z) solamente.

DERIVA

USO: **DERIVA** (*exp-1, VarAndOrder, VarAndOrder, ...*)

Donde *VarAndOrder* es una variable o una variable y un entero positivo separados por una coma.

SIGNIFICADO:

DERIVA (*exp, var*) Regresa la derivada parcial de primer orden de *exp* con respecto a *var.*

DERIVA (*exp, var, n*) Regresa la enésima derivada parcial de *exp* respecto a *var.*

DERIVA (*exp, var1, var2*) Regresa la segunda derivada parcial mezclada de *exp* con respecto a *var1* y *var2.*

DERIVA (*exp, var1, n1, var2, n2, ...)* deriva la *exp* , *n1* veces con respecto a *var1, n2* veces con respecto a *var2,* etc

EJEMPLO:

```
DERIVA(1/X,X) = -1/X^2                DERIVA(SEN(3 X),X)= 3 COS (3 X)
DERIVA(3 X^2 Y, X, 2) = 6 Y           DERIVA(LN(A), X) = 0
DERIVA(3 X^2 Y^3 , X , Y) = 18 X Y^2
DERIVA(4 X^2 Y^3 + 5 X^3 Y^5, X,2, Y,3)=48 + 1800 X Y^2
```

INTEGRA

USO: **INTEGRA(** *expl, varint* **)**

SIGNIFICADO: Esta función encuentra la integral de *expl* con respecto a *varint*, si es que el sistema es capaz de encontrarla. En caso de que no sea calculable por el sistema, éste regresa como resultado la misma expresión de entrada.

EJEMPLO:

```
INTEGRA(1/X, X) = LOG (ABS(X), %E)
INTEGRA(X^2, X) = X^3 / 3
```

3.7 Función de Substitución.

El propósito principal de la función de *substitución* APLICA es que el usuario elija de las tablas provistas por el sistema la "fórmula" adecuada para una expresión y la substitución mecánica sea realizada por el sistema.

APLICA

USO 1: **APLICA(** *Var-Especial , Expl* **)**

SIGNIFICADO 1: *Var-Especial* es una variable del tipo **%Exx, %Dxx** o **%Ixx** donde **xx** es el número de esquema de las tablas de equivalencia, derivación o integración respectivamente.

Cada uno de los esquemas tiene dos partes, un antecedente y un concecuente.

exp-antecedente → exp-concecuente

La expresión *Expl*, debe tener la misma estructura que la expresión *exp-antecedente*.

El sistema realiza la asociación adecuada de variables, como en SUBEXP, entre las expresiones *Expl* y *exp-antecedente*. Con el resultado de la asociación, el sistema ejecuta el esquema substituyendo la

asociación encontrada en la expresión *exp-concecuente*.

EJEMPLO:

Se tienen los siguientes esquemas de las tablas:

%E12: cos(A-B) → cos A cos B + sen A sen B

%D5 : d(UV)/dx → U dV/dx + V dU/dx

%I3 : $\int (U + V)\, dx \rightarrow \int U\, dx + \int V\, dx$

```
APLICA(%E12, cos(x^2 - 4 y^3)) = cos x^2 cos 4 y^3 +
                                 sen x^2 sen 4 y^3

APLICA(%D5, DERIVA(X^2 sen(X + Y),X)) = X^2 d(sen(X+Y))/dx +
                                        sen(X+Y) d(X^2)/dx

APLICA(%I3, INTEGRA(X^2 + 2 X^3,X)) =
                                    ∫ X^2 dx + ∫ 2 X^3 dx
```

USO 2:

APLICA(*Var-Especial , Exp1, DifU,* U, *ValU, ..., VarN, ValVarN*)

SIGNIFICADO 2: Este uso es válido para realizar aplicaciones de fórmulas de integración, cuando en los esquemas se especifica una diferencial de U, donde U se supone una función de x. En este caso es necesario que el usuario especifique cual es la substitución que desea realizar según la expresión que tiene y la fórmula que desea aplicar.

Var-Especial es según se explicó en el uso 1. *Exp1* es exclusivamente la expresión que es el integrando. Todas las fórmulas que se pueden aplicar tienen especificada una parte del integrando como la diferencial de U. *DifU* debe ser la parte de *Exp1* que se elige como la diferencial de U. U es la función de x, de la cual se tiene la diferencial contenida en el integrando. *ValU* es la parte de la expresión *Exp1* que desea elegirse como la función U. *Varx* son las variables contenidas en la fórmula y que también se les tiene que asignar una parte de la expresión *Exp1* como su valor, el cual deberá ser indicado en *Varx*.

EJEMPLO:

Dado el esquema

%I10 : $\int$ sen U du → - cos U

```
APLICA(%I10, 2 X SEN(X^2+3), 2 X, U, X^2+3)= -COS(X^2+3)
```

3.8 Funciones Ejecutables.

Las funciones ejecutables son aquellas que se efectúan siempre sin considerar el terminador de la expresión en que intervienen. Las funciones ejecutables funcionan como se describió en la sección 3.7. Son *SUBEXP* y *APLICA*.

Suponiendo que se tiene la siguiente regla de derivación en las tablas, y la sesión que se describe en seguida, puede apreciarse un ejemplo:

```
%D9: d(sen(x))/dx = cos (x)

?1: SEN(X) + X^2;
@1: SEN(X) + X^2
?3: SUBEXP(?1,A+B,A)&
@3: SEN(X)
?4: APLICA(%D9, DERIVA(SUBEXP(?1,A+B,A),X))&
@4: COS(X)
```

Las entidades principales con que se trabaja en este sistema son las expresiones matemáticas, que se representan internamente por medio de listas en LISP. La representación de las expresiones varía de forma a lo largo del sistema de acuerdo al módulo en que se encuentran y, más que nada, a qué tan cerca está el módulc del usuario o bien de la manipulación de la expresión.

Además, como elemento esencial de la interactividad se tienen las pantallas que se usan en el sistema.

En el presente capitulo se describe la estructura de datos usada para estos elementos.

4.1 Formas de una expresión en el sistema.

Las tres formas que adquiere una expresión en el sistema son la Forma de Usuario (FU, para abreviar), la Forma Interna Prefija (FIP) y la Forma Interna Prefija Normalizada (FIPN).

Esta sección tiene como objetivo definir cada una de las formas mencionadas, que juegan un papel muy importante en cómo están diseñados los algoritmos en los módulos de manipulación de símbolos del sistema.

4.1.1 Forma de Usuario (FU).

Se le ha denominado así porque es la forma de la expresión que es más comprensible para el usuario. Su forma se asemeja a la manera convencional en que una persona escribiría una expresión en notación matemática, con la salvedad de que en la pantalla la forma gráfica que tiene un cociente y una exponenciación solo se refleja en forma lineal. El cociente $\frac{\text{numerador}}{\text{denominador}}$ debe expresarse como numerador/denominador y la exponenciación A^B como A^B. Por otra parte, para el caso particular del operador de integración tenemos que se usa un símbolo especial, el cual no se encuentra disponible en el momento en que el usuario escribe una expresión. En su lugar se utilizó la

abreviación INT, de tal manera que la operación

$$\int f(x)\,dx$$

se describa como

INT (f(x) , x).

Lo anterior es válido durante la fase en la que el usuario introduce información al sistema. Cuando ocurre el proceso inverso, de que el sistema presenta información al usuario, se trató de escribir la expresión de una manera más natural para las personas; pero sin dejar de ser lineal. Esto último se logró regresando la abreviación de INT a su forma usual, ya que se dispone del caracter especial en la pantalla.

En el capítulo 6 se profundizará en los detalles de la entrada al sistema de una expresión escrita por el usuario, así como también la manera en que del interior del sistema se reescriben las expresiones en la forma de usuario para entregar resultados.

Para automatizar el primer punto que mencionamos es necesario definir de alguna manera el universo de expresiones que se permitirá escribir al usuario, es decir, el lenguaje válido de entrada al sistema, así como las reglas para obtener expresiones sintácticamente correctas en el lenguaje. Para ello se utilizó como herramienta la gramática descrita en el apéndice A. Dicha gramática describe la sintaxis de la forma de las expresiones para el usuario, FU.

4.1.2 Forma Interna Prefija (FIP).

La FIP es la primera forma interna que obtiene el sistema a partir de la FU.

El procedimiento mediante el cual es obtenida esta forma se describirá en el módulo encargado de la transformación FU-FIP en la sección 6.2.2.

La FIP consiste en representar todas las expresiones mediante una lista de LISP cuyo primer elemento es un operador y el resto de la lista contiene los argumentos para dicho operador. Ver Fig. 4.1.

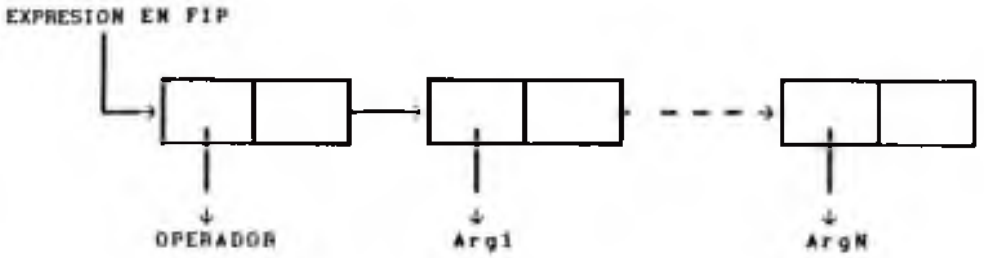


Fig. 4.1 Lista para guardar una expresión en FIP.

Como puede observarse en la gramática de la sección 2.1.1, los operadores básicos que pueden ser usados en cualquier expresión son los que se presentan en la Fig. 4.2.

Operador	Operacion	Ejemplo
+	mas binario	A + B
-	menos unario, binario	-A , A - B
/	cociente binario	A / B
*	producto binario	A * B
^	exponenciacion binaria	A ^ B

Fig. 4.2 Operadores Básicos.

También el nombre de cualquiera de las funciones permitidas en el sistema se considera como un operador para propósitos de esta forma de representación de expresiones, y son todas aquellas comprendidas dentro de cualquiera de las que fueron mencionadas en el capítulo anterior y cuya forma natural es en sí prefija.

Para los argumentos de un operador, que tienen su lugar en el resto de la lista, se puede tener la opción de tener a su vez una expresión. Una expresión ahí debe tener la sintaxis definida en la categoría <EXP> de la gramática que define la FU.

Debido a la estructura de la gramática que define la FU, el módulo transformador FU-FIP obtiene la FIP que refleja el orden apropiado en que deben ejecutarse todas las operaciones, según la

jerarquía asignada a los operadores y la alteración presentada en ésta por paréntesis introducidos por el usuario en una expresión.

Además de los operadores ya considerados, están otros elementos que se manejan también como operadores para efectos de la representación en FIP. Sin embargo difieren en cierta forma de los que ya han sido mencionados. Son los terminadores, que funcionan como operadores postfijos y el operador de asignación, "=". Estos también se colocan en FIP como los anteriores; pero se encuentran en categorías sintácticas que no son permitidas como argumentos.

4.1.3 Forma Interna Prefija Normalizada (FIPN).

La FIPN tiene como objetivo representar en forma única una expresión.

La razón de que sea necesario este requerimiento son los módulos de manipulación simbólica, que simplifican sus tareas con expresiones normalizadas, que utilizan menos operadores y tienen una forma más uniforme que permite evitar ambigüedades.

Por ejemplo, la siguiente expresión:

$$\frac{a\,(\,b - c\,)}{d\ f}$$

puede representarse en FIP así:

```
( /  ( * a  ( - b c) )
     ( * d  f) )
```

o bien:

```
( *  ( * a ( - b c))
     ( ^ ( * d f ) -1) )
```

La misma expresión puede tener en FIP distintas representaciones. Todas ellas reflejan adecuadamente el orden en que deben ejecutarse los operadores según su jerarquía y según el orden establecido por los paréntesis introducidos por el usuario.

Los módulos de manipulación simbólica se simplifican usando la representación en FIPN, pues se utiliza menos variedad de operadores en la representación y la representación que se obtiene es más

uniforme .

La normalización se efectúa a partir de la FIP eliminando los operadores de division "/" y de diferencia "-" y permitiendo el uso del uno negativo, -1.

La normalización toma en cuenta que:

- Cualquier cociente puede ser convertido a un producto.
- Cualquier diferencia puede convertirse en una suma.

Las reglas a aplicar son las que se describen en la tabla de la Fig. 4.3.

FU	FIP	FIPN
A / B	(/ A B)	(* A (^ B -1))
A - B	(- A B)	(+ A (* B -1))

Fig. 4.3 Reglas para normalización de expresiones.

Aplicando ambas reglas al ejemplo anterior, considerándola como un cociente o como un producto, se obtiene la siguiente FIPN para la expresion del ejemplo:

```
( * ( * a ( + b ( * c -1)) )
    ( ^ ( * d f) -1 ) )
```

4.2 Forma Interna de Ventanas.

Las pantallas que se despliegan en el sistema y la forma de manejo de menúes están codificadas en el módulo del manejador de ventanas.

Las utilerías de Mu-LISP contienen un manejador de ventanas , que mejor dicho, es un manejador de pantallas y menúes.

Dicho manejador se tomó como base para elaborar uno como el sistema lo requería.

En esta sección se explican algunos conceptos propios del manejador de pantallas y se describen las estructuras para representar las entidades que se usaron.

4.2.1 Conceptos.

El manejador divide físicamente la pantalla en cuatro áreas. La primera es el área de ventanas, la segunda el área para opciones, la tercera es para mensajes y la última es una línea de estado.

Para el manejador de ventanas un programa se denomina aplicación.

Cada aplicación tiene un menú asociado, en donde se definen las opciones disponibles para la aplicación y eventualmente sus submenúes y respectivas opciones.

El editor de Mu-LISP 87, el rastreador de programas y el sistema objeto de este trabajo son ejemplos de aplicaciones.

Todas las aplicaciones son independientes unas de otras.

El área dedicada a las ventanas puede dividirse en varias ventanas más pequeñas, cada una de ellas corresponde a una aplicación.

No es necesario que todas las ventanas que haya en la pantalla tengan el mismo tipo de aplicación, ni tampoco se restringe a que todas tengan que ser de diferente tipo.

La potencialidad del manjeador de ventanas permite que una ventana esté formada por uno o varios cristales, aunque solo es posible ver uno a un tiempo, porque un cristal ocupa todo el espacio disponible para su ventana.

Un cristal se puede imaginar como un vidrio contenido en un marco del ventanal de una casa. Usualmente a todo el conjunto se le acostumbra llamar ventana. Todas las ventanas de una casa tienen un sólo cristal; pero si hubiera posibilidad de tener cristales intercambiables podría tenerse el ventanal con un color distinto cada día de acuerdo al gusto o a la necesidad.

El ejemplo imaginario de los cristales intercambiables es una posibilidad que el manejador de ventanas del sistema de Mu-LISP contempla.

La aplicación que corresponde al sistema de manipulación simbólica que nos ocupa no requiere nunca de más de un cristal por

ventana.

El menú de las aplicaciones es desplegado en el área de la pantalla destinada a las opciones. Si en un momento dado existe más de una ventana en el área de ventanas sólo se desplegará ahí el menú correspondiente a una de ellas. Este menú será el que esté asociado a la aplicación de la llamada ventana activa.

La ventana activa determina la aplicación que se está ejecutando y si hay en otras ventanas el mismo tipo de aplicación establece entonces la instancia que se ejecuta.

El número que corresponde a la ventana activa está guardado en la variable *CURRENT-WINDOW*. Su expresión externa consiste en que el número que se despliega para ella en la pantalla está escrito en modo de video inverso y se encuentra en la esquina superior izquierda de la ventana.

En el sistema de manipulación de símbolos el único tipo de aplicación que se maneja se llama "Modelo". Debido a ésto el menú que se despliega es siempre el mismo. Cada vez que se mencione la aplicación ha de entenderse que se refiere a la de este tipo.

4.2.2 Información General e Información Especifica de ventanas.

Acerca de una ventana se puede decir que existen dos tipos de información. La información general de una ventana, en donde se especifican básicamente los cristales que tiene la ventana y la posición y tamaño que guarda en la pantalla.

La información especifica es la que da el detalle de los cristales de una ventana.

Debido a que, como anteriormente se ha dicho, una ventana del sistema de manipulación de símbolos, solo tendrá un cristal, se podrá hablar indistintamente de información específica de una ventana e información del cristal de la ventana.

La información general de una ventana tiene seis elementos:

Elementos referentes a sus cristales:

1.- Posición que guarda la información del cristal visible, dentro de la lista que es el 2o. elemento de la información general de

una ventana.

2.- La lista que guarda la información específica de una ventana, o bien también descrita como la información acerca de los cristales de una ventana.

Elementos referentes a su posición:

3.- El renglón base del área de ventanas de la pantalla donde se posiciona la ventana.

4.- La columna base del área de ventanas de la pantalla donde se posiciona la ventana.

Elementos referentes a su tamaño:

5.- El número de renglones de altura para la ventana.

6.- El número de columnas de ancho para la ventana.

Como puede el lector apreciar, la información específica de una ventana es el segundo elemento de la información general de la misma.

Ahora bien, la lista donde se guarda la información específica de una ventana tiene tantos elementos como variables globales necesiten manejarse para codificar la aplicación. La asociación de las variables ahí contenidas con sus respectivos valores cuando una aplicación se activa, es controlado por el manejador de ventanas.

Como consecuencia de que cada ventana de la aplicación sólo tiene un cristal, el segundo elemento de la información general de una ventana, que es una lista que contiene información para cada cristal, siempre tiene longitud igual a uno.
Por el mismo motivo la posición que guarda dicha información en la lista es la 0.

En la aplicación tipo "Modelo" son 6 elementos los que forman la información específica que nos interesa.

1. Nombre del tipo de la aplicación "Modelo" (siempre).
2. Lista de entradas correspondientes a la instancia de la aplicación.
3. Lista de salidas correspondientes a la instancia de la aplicación.
4. Lista de variables usadas por el usuario en la instancia de la

aplicación.

5. Bandera de detección de cambios en el estado de la sesión.
6. Nombre por omisión de una sesión.

4.2.3 Estructura Interna.

Para guardar la información necesaria acerca de las ventanas es necesario tener listas de información general para tantas ventanas como las que haya abiertas en el sistema.

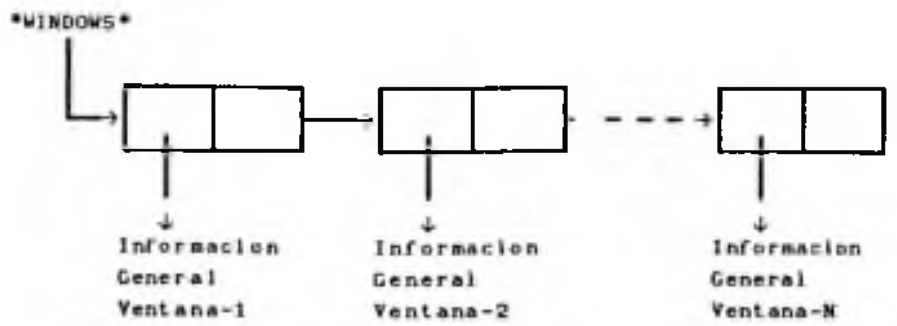


Fig. 4.4 Estructura de *WINDOWS*

El valor de la variable *WINDOWS* es una lista cuya longitud es igual al número de ventanas abiertas en el sistema, y cada uno de sus elementos es la información general de una de ellas. *WINDOWS* es indexada generalmente por medio de *CURRENT-WINDOW*, por lo cual puede concluirse entonces que *CURRENT-WINDOW* es, más que la ventana actualmente activa, la posición de ésta dentro de *WINDOWS*.

La estructura que se utiliza para *WINDOWS* se puede esquematizar en la Fig. 4.4. La información que contiene según las necesidades expuestas para este sistema es la siguiente:

```
*WINDOWS* =
(
  (0    (("Modelo"    *ENTRADAS*    *SALIDAS*    *VARIABLES*    *HUBOCAMBIOS*))
     RBase CBase Rengs Cols)
  (0    (("Modelo"    *ENTRADAS*    *SALIDAS*    *VARIABLES*    *HUBOCAMBIOS*))
     RBase CBase Rengs Cols)
  :                                                                    :
  (0    (("Modelo"    *ENTRADAS*    *SALIDAS*    *VARIABLES*    *HUBOCAMBIOS*))
     RBase CBase Rengs Cols) )
```

CAP. 5. DESCRIPCION GLOBAL INTERNA DEL SISTEMA.

En este capítulo se da un panorama de la configuración del sistema como un todo. Se enumeran los módulos que lo integran, describiendo su función y se sitúa cada uno dentro del marco del sistema completo.

Es sumamente importante conocer la forma en que se representan los datos en cada módulo, pues la estructura de los algoritmos está ligada estrechamente con la estructura de los datos, por tanto dicha información también se proporciona aquí.

Cada módulo del sistema no es una función única de LISP, que tome una entrada y regrese una salida. Más bien se constituyen por una secuencia de definiciones y funciones de LISP cuya labor conjunta se puede describir como la función operativa del módulo al que peretenecen.

5.1. Los módulos del sistema.

Manejador de Ventanas

En este módulo está programado el manejo de pantallas y de menúes del sistema. Es particularmente importante porque, después de todo, es el que da la presentación última al usuario. Su implementación está estrechamente ligada a las características del intérprete de LISP. Las funciones principales que se encuentran en este módulo son *WINDOWS* y *"Modelo"*. La primera sustituye al manejador del sistema LISP, que es la función implementadora del ciclo de lectura-evaluación-impresión propia del intérprete. La segunda es una función parámetro de la primera y es donde está codificado el núcleo del sistema de manipulación de símbolos de este trabajo. Se encarga de establecer las condiciones iniciales de trabajo, la actualización periódica de parámetros existentes en las diferentes ventanas del sistema y las condiciones que permiten iniciar y terminar la instancia de una aplicación.

Transformador FU-FIP

El transformador FU-FIP se compone de una serie de funciones que

realizan la traducción de las expresiones introducidas por el usuario a la primer forma interna que toman, FIP. Durante el proceso en el que se analiza la entrada del usuario para su traducción, se reconoce si una expresión dada es sintácticamente correcta. El reconocimiento se realizó por medio del método de descenso recursivo. Este método consiste en tener un proceso por cada categoría sintáctica en la gramática que define el lenguaje válido de entrada. El proceso correspondiente a la categoría sintáctica más alta es el que comienza a realizar el reconocimiento. Recursivamente se van invocando todos los procedimientos según la grámatica lo va marcando. El proceso recursivo termina cuando se encuentra un símbolo terminal, cuya validez puede ser determinada léxicamente.

Transformador FIP-FIPN

Las funciones del transformador FIP-FIPN son invocadas cuando en una expresión intervienen operadores "-" o "/" con el fin de convertir una expresión que se encuentra en FIP a FIPN. La FIPN es la forma adecuada para realizar operaciones simbólicas.

Módulo Aritmético

En este módulo se realizan las operaciones aritméticas requeridas en las expresiones introducidas por el usuario. Además éste es el primer módulo que contiene algoritmos de manipulación simbólica. Las funciones en él codificadas tienen implementadas las operaciones básicas en forma simbólica. Esta son la suma, la multiplicación y la exponenciación. También están incluidas en este lugar algunas funciones primitivas de apoyo al manejo simbólico de otros módulos más complejos.

Módulo Algebraico

Las funciones que componen este módulo realizan operaciones algebráicas que ya están directamente disponibles para el usuario. Las principales funciones codificadas en este módulo permiten realizar operaciones de expansión, factorización y selección de subexpresiones.

Módulo de diferenciación e integración.

Las funciones principales que componen este módulo son básicamente dos, la que se encarga de encontrar la derivada de una expresión y la que encuentra la integral de una expresión. En este módulo se encuentran también otras funciones que son de apoyo para encontrar derivadas parciales y de ordenes mayores, utilizando siempre como base la función primitiva de derivación. Otras funciones de ayuda en este módulo son las que se encargan de determinar la dependecia entre variables según la definición del usuario. Además se encuentran otras funciones que complementan al proceso de integración.

Módulo de Codificación de Tablas.

Las funciones de este módulo contienen la definición matemática de funciones trigonométricas, de equivalencias, de derivadas, primitivas trigonométricas y logaritmicas. Estas funciones pequeñas y compactas son básicamente definiciones y son utilizadas principalmente por el módulo anterior.

Manejador de Terminadores.

En este módulo se encuentra la función que inicializa el proceso de reconocimiento del lenguaje de entrada, el cual termina cuando se obtiene la FIPN para la entrada recién introducida y aceptada. Debido a que la notación en LISP es prefija, el evaluador de LISP evalúa muchas de las expresiones que se encuentran en FIPN. Sin embargo, cuando se encuentra una función cuyo operador es uno de los terminadores ";", "$", "!" ó "&" se requiere de un tratamiento especial cuyas funciones de apoyo se encuentran en este módulo. En él se tienen también algunas funciones complementarias que realizan el empatamiento de esquemas de derivación e integración con expresiones dadas por el usuario.

Transformador FIPN-FU

Las funciones codificadas en este módulo son invocadas cuando el resultado obtenido después de la manipulación simbólica va a ser presentado al usuario. Ellas se encargan de tomar la expresión resultante que aún está descrita por medio de una lista en LISP con

las convenciones de la FIPN y despliegan sobre el dispositivo de salida los caracteres más adecuados para expresarla al usuario. Esta forma final de presentación es nuevamente una expresión en FU.

5.2. Interrelación de las partes del sistema.

La parte principal de esta sección es un diagrama que tiene como objetivo mostrar la relación que guardan entre sí los módulos numerados en la sección anterior. Dicha relación se encuentra establecida en base a la secuencia de llamados entre ellos.

El marco que brinda la presentación al usuario es el que provee el manejador de ventanas. Su funcionamiento está ligado de manera muy estrecha al funcionamiento del intérprete de LISP, por lo que va a ser descrito de manera general.

Después se presenta el diagrama de relación entre los módulos y se da una explicación acerca de él.

5.2.1. El intérprete de Lisp

Mu-LISP es un intérprete del lenguaje LISP. El intérprete Mu-LISP comienza su ejecución preparándose un espacio para datos y después entra en un ciclo, el cual se identifica como Ciclo de Nivel Superior (CNS).

```
(LOOP
     (CATCH NIL (FUNCALL DRIVER)
     ( ((throw was not active?))
       ((EQ *THROW-TAG* 'SYSTEM)
             (cerrar archivos actualmente abiertos)
             (reestablecer la forma normal del cursor)
             (terminar la ejecucion de Mu-Lisp) )
       ((EQ *THROW-TAG* "Memory-Full")
             (BREAK NIL "Memory Full") )
       ((EQ *THROW-TAG* "Disk Full")
             (BREAK NIL "Disk   Full") ) ) )
```

El CNS invoca a la función *DRIVER*, la cual implementa a su vez el ciclo llamado de Lectura-Evaluación-Impresión (LEI).

Si durante el ciclo de LEI se detecta una expresión que llame al sistema operativo o hubo algún error de almacenamiento saturado, el ciclo se interrumpe. En el primer caso regresando el control al sistema operativo, y en el segundo, enviando un mensaje de error para que el usuario determine si finaliza el ciclo de LEI o bien, de que manera va a ser éste reestablecido.

```
(DEFUN DRIVER ()
    (LOOP
      (WRITE-PROMPT "$ ")
      (CATCH NIL
             (SETQ - (READ T))      ; LECTURA
             (SETQ * (EVAL -))      ; EVALUACION
             (WRITE * T)             ; IMPRESION
             (TERPRI NIL T) )
      ( ((throw was not active?))
        ((EQ *THROW-TAG* 'DRIVER)
             (desactivate throw))
        ((EQ *THROW-TAG* 'RETURN)
             (desactivate throw)
             (RETURN *))
        (THROW *THROW-TAG* *) )))
```

El ciclo de LEI consiste en el desplegado del prompt de LISP, que es un signo "$". Después de que se introduzca una expresión completa en LISP y se teclee retorno de carro se lee la expresión, se evalúa e imprime el resultado de la evaluación en el comienzo de una nueva línea. El ciclo de LEI se repite indefinidamente hasta que se dé el comando SYSTEM en alguna de las expresiones tecleadas, regresando al CNS para terminar la ejecución del intérprete.

Para efectos de la implementación del Manejador de Ventanas se substituyó la función *DRIVER*, donde está codificado el ciclo LEI, por la función *WINDOWS* que implementa un ciclo adecuado a las necesidades del manejo de pantallas y menúes.

El esquema general de esta nueva función manejadora es de la siguiente manera:

```
(DEFUN WINDOWS variables-locales

    (INICIALIZACION)
    (LOOP
      (SETQ EXPN
            (CATCH NIL
                   (APPLY "Modelo" 'Run-Window)))
      (
        ((EQ *THROW-TAG* 'CLOSE-WINDOW)
              (eliminar ventana a cerrar)
                   (desactivate throw))
        ((EQ *THROW-TAG* 'SWITCH-WINDOW)
              (cambiar hacia la nueva ventana elegida)
                   (desactivate throw))
        ((EQ *THROW-TAG* 'DIVISION-WINDOW)
              (crear ventana nueva)
                   (desactivate throw))
        ((EQ *THROW-TAG* 'QUIT-PROGRAM)
                   (salvar sesiones)
                   (terminar)
                   (desactivate throw))
        ((EQ *THROW-TAG* 'SWITCH-SCREEN)
                   (dibujar los marcos del sistema)
                   (desactivate throw))
        ((EQ *THROW-TAG* 'DRIVER)
                   (desactivate throw))
        (THROW *THROW-TAG* EXPN)  )))
```

5.2.2. Diagrama de llamado entre los módulos.

El diagrama que se muestra en la fig. 5.1 describe gráficamente la relación que existe entre los diferentes módulos de acuerdo a la serie de llamados que suceden a partir de que comience a ejecutarse el sistema.

En primer lugar se ejecuta automáticamente la función *WINDOWS* del manejador de ventanas que es invocada por el CNS del intérprete de LISP.

La ejecución de la función *"Modelo"* ocurre al ser invocada por el ciclo implementado en *WINDOWS*.

"Modelo" utiliza una lista llamada *MODELO-OPTIONS*, que es donde se encuentra codificado el árbol de opciones del sistema, acompañado de las funciones respectivas que deben ejecutarse según sea la opción elegida por el usuario.

La opción principal del árbol de opciones es la primera y aparece

en el menú de opciones como **"Lectura"** . Las otras funciones son de las utilerías referentes al manejo de sesiones, desplegado de tablas, impresión, opciones de miscelánea y opciones de manejo conceptual de ventanas. Con la elección de estas opciones es interrumpida la evaluación normal de la función *"Modelo"* por medio de la instrucción THROW, propia de LISP, que envía el control a diferentes puntos de la función *WINDOWS*.

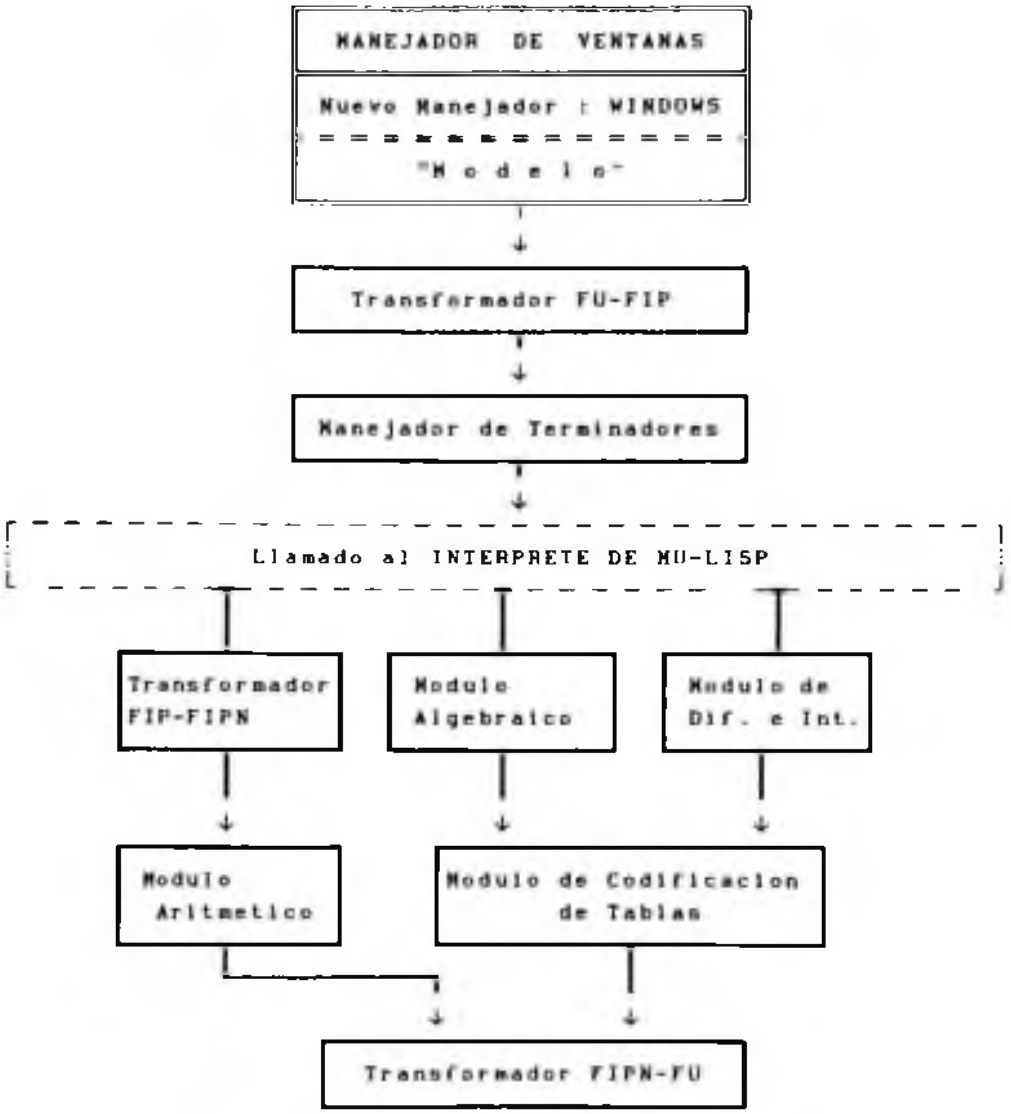


Fig. 5.1 Esquema general del sistema

Cuando la evaluación continúa normalmente dentro de la opción de Lectura es llamado el módulo que transforma la FU a FIP.

Ahora bien, como puede observarse en el diagrama, en este punto

ocurre un llamado al intérprete Mu-LISP.

En Mu-LISP existen implementadas las funciones aritméticas básicas. Estas son suma, resta, multiplicación y división. Dichas funciones solamente efectuan operaciones aritméticas y los argumentos que reciben deber ser argumentos aritméticos, de acuerdo a lo que está especificado en el manual de referencia del lenguaje Mu-LISP. Cuando alguna de estas funciones recibe el número y tipo correcto de argumentos, realiza las operaciones aritméticas requeridas.

Cuando reciben más de dos argumentos, asocian sucesivamente de izquierda a derecha los argumentos recibidos y realizan la operación adecuada.

Para cada función aritmética de Mu-LISP hay una función de manejo de error, que es llamada cuando el tipo de los argumentos no es numérico.

OPERADOR	Funcion de manejo de error
+	+TRAP
-	-TRAP
*	*TRAP
/	/TRAP

Fig. 5.2 Funciones de LISP para manejo de error

Cada función de manejo de error se encarga de llamar a la rutina de ruptura de LISP, la función BREAK, para que sea desplegado el mensaje de error adecuado. Sin embargo en este punto se intercepta la secuencia descrita anteriormente.

Las rutinas de manejo de error son llamadas cuando los argumentos recibidos por sus respectivas operaciones no son números. Entonces significa que tal vez se requiera un tratamiento simbólico para efectuar la operación.

En el caso de los operadores "+" y "*", sus funciones de manejo de error fueron modificadas y en su lugar se definieron las funciones escritas en el Módulo Aritmético para realizar suma y multiplicación simbólica.

En el caso de los operadores "-" y "/" las funciones respectivas de manejo de error invocan al módulo Transformador de FIP a FIPN. La FIPN solo contiene operadores "+", "*" y "^" cuya implementación simbólica ya se encuentra en el Módulo Aritmético.

En el intérprete no se encuentra implementado el operador de exponenciación, "^", así es que el Módulo Aritmético también contiene definición para él. Su definición comtempla exponenciación tanto aritmética como simbólica.

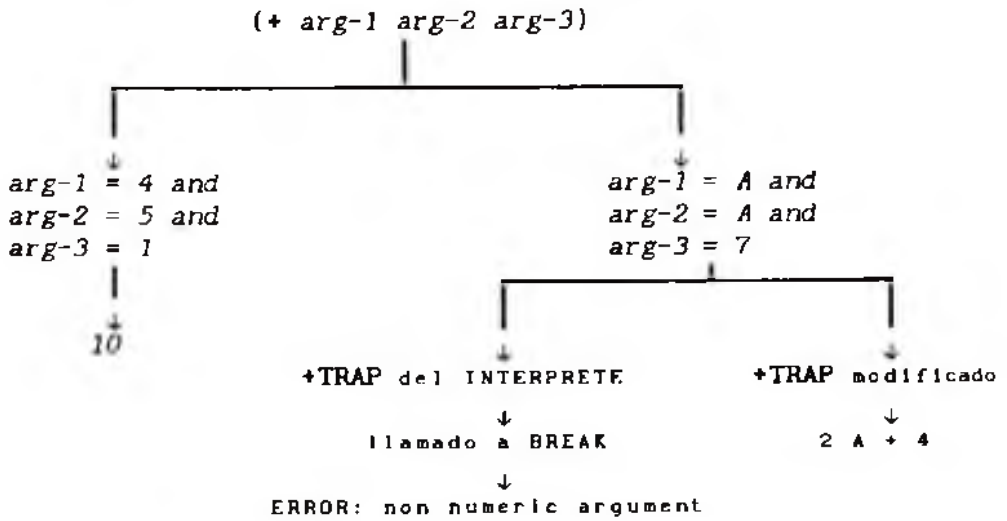


Fig. 5.3 Intercepción de la función de manejo de error *+TRAP*

En lo referente a las funciones que no son operadores, tenemos que, cuando se trata de funciones de tipo algebraico son ejecutadas del Módulo Algebraico. Cuando se trata de funciones de cálculo se ejecutan del Módulo de Diferenciación e Integración. Y de manera especial son tratadas las evaluaciones para los terminadores en el Módulo de Terminadores.

Especialmente las rutinas implementadas en el Módulo de Diferenciación e Integración hacen uso de las definiciones codificadas en el Módulo de Codificación de Tablas.

Cuando se terminan de ejecutar las funciones de manipulación simbólica, la forma de las expresiones es aún FIPN. El transformador FIPN-FU es llamado para presentar los resultados al usuario.

CAP. 6. DESCRIPCION DETALLADA ACERCA DEL FUNCIONAMIENTO DE LOS MODULOS DEL SISTEMA.

Este último capítulo tiene como objetivo describir la manera en que están codificadas las funciones principales de cada módulo. Además de la codificación de las definiciones matemáticas que son utilizadas por dichas funciones.

El máximo detalle de cada función es el código de la misma, y puede ser consultado en los apéndices. Para cada módulo existe un apéndice asociado que contiene el código correspondiente. La descripción que aquí se dé, es complemento a los comentarios realizados sobre el código fuente para entender como está hecho el programa.

Al término del estudio de este capítulo se espera que el lector pueda ser capaz de comprender la lógica seguida en la implementación del sistema y de poder llevar a cabo modificaciones, ampliaciones o hacer uso de cualquiera de los módulos sin tener necesidad de investigar otras fuentes.

6.1. Funcionamiento del Manejador de Ventanas.

Uno de los objetivos del sistema es que el usuario se retroalimente mientras esta resolviendo algún problema. Se intenta que el sistema sea lo más amigable posible. Un sistema interactivo que permita usar ventanas y elegir facilmente opciones de un menú resulta conveniente.

Este módulo contiene los algoritmos propios para ello y su código de puede consultarse en el apéndice B.

Ya se ha mencionado que las dos **funciones principales implementadas en el manejador de ventanas son** *WINDOWS* **y** *"Modelo"*.

La primera substituye al ciclo LEI del sistema Mu-LISP. Su estructura consiste de una inicialización y después se implementa un ciclo.

En la inicialización se pintan las ventanas la primera vez, se llena *WINDOWS* con información de una ventana y se les ponen los valores iniciales a algunas variables, las que conservarán ese valor

todo el tiempo.

En el ciclo lo primero que se hace es invocar a la función *"Modelo"* con el comando RUN-WINDOW para provocar que se ejecute una instancia de la aplicación.

Dependiendo de la opción que ejecute el usuario, la evaluación de *"Modelo"* regresa el control a *WINDOWS* en algún punto específico.

Los casos que se contemplan para continuar con la evaluación en *WINDOWS* son:

a) CLOSE-WINDOW En este punto se elimina de la pantalla y del sistema la ventana que haya sido seleccionada por el usuario para cerrarse. Después se mandan dibujar los marcos de las ventanas segun hayan quedado.

b) SWITCH-WINDOW En este punto se cambia el número indicador de la ventana activa a la ventana previa o a la siguiente, respecto a la ventana activa actual, según haya sido el caso elegido por el usuario. La posición de la nueva ventana que quedó activa está indicada por medio de la variable EXPN, y este valor se le dá a *CURRENT-WINDOW*.

c) DIVISION-WINDOW Aquí se crea una ventana nueva en el sistema según la elección que el usuario haya indicado, si la ventana que quiere abrir es vertical u horizontal, así como su tamaño según el espacio disponible. Se refleja sobre la pantalla la nueva ventana.

d) QUIT-PROGRAM Se verifica ventana por ventana si la sesión residente en su medio ambiente fue salvada y se finaliza la ejecución del sistema.

e) SWITCH-SCREEN En este punto se redibujan los marcos de las ventanas del sistema.

Las **funciones de ayuda a WINDOWS** son varias, primeramente se encuentra la de *ACTUALIZA-VENTANAS*.

ACTUALIZA-VENTANAS se encarga de limpiar el área de la pantalla de cada una de las ventanas que hayan sido abiertas en el sistema. Su estructura es básicamente un ciclo que se ejecuta tantas veces como ventanas abiertas haya. Esta última información se encuentra en *WINDOWS*, pues su longitud es precisamente el número de veces que se ejecuta un llamado de la función *UPDATE-WINDOW*.

UPDATE-WINDOW llama a la función *"Modelo"* por medio del comando

UPDATE-WINDOW para que realice la limpieza del área de la pantalla que corresponde a la ventana indicada en *CURRENT-WINDOW*.

Las funciones *UPPER-LEFT*, *UPPER-RIGHT*, *LOWER-LEFT* y *LOWER-RIGHT* sirven principalmente para cerrar una ventana y determinar el caracter gráfico que es adecuado escribir en cada coordenada de la pantalla cuando se dibujan los marcos de las ventanas.

Otras de las funciones importantes codificadas en el módulo del manejador de ventanas son las que sirven como utilerías para manejar en la estructura *WINDOWS* la información general de las ventanas. Permiten consultar los parámetros ahí guardados y también modificarlos.

Existe una función para cada uno de los parámetros de la información general que tiene una ventana. Cualquiera de ellas puede llamarse enviándole 0, 1 ó 2 valores.

a) Cuando son llamadas con un solo valor, éste se toma como el número de la ventana de la que se quiere obtener el parámetro correspondiente. De la siguiente manera:

(*WINDOW-PANE* N)	Obtiene el número de cristal activo de la ventana N.
(*WINDOW-ROW* N)	Obtiene el renglón base de la ventana N.
(*WINDOW-COL* N)	Obtiene la columna base de la ventana N.
(*WINDOW-ROWS* N)	Obtiene el número de reglones de la ventana N.
(*WINDOW-COLS* N)	Obtiene el número de columnas de la ventana N.
(*WINDOW-STATES* N)	Obtiene la lista de la información especifica de la ventana N.

b) Si cada una de estas funciones es llamada sin valor, la información respectiva que obtiene se refiere a la de la ventana activa, *CURRENT-WINDOW*.

c) El llamado a alguna de estas funciones especificando dos valores, modifica el parámetro correspondiente en la ventana indicada por el primer valor. El segundo valor se usa como la información nueva que se va a colocar.

Por ejemplo, si *WINDOWS* tiene la siguiente información:

```
*WINDOWS* =
( (0 (("Modelo" NIL NIL NIL NIL "SESION.SSN"))  1  1 9 37)
  (0 (("Modelo" NIL NIL NIL NIL "SESION.SSN"))  1 39 9 40)
  (0 (("Modelo" NIL NIL NIL NIL "SESION.SSN")) 11  1 9 78)
)
```

y *CURRENT-WINDOW*=2 (recordar que *CURRENT-WINDOW* es la posición en *WINDOWS* de la ventana actual), entonces (WINDOW-COLS) regresa 78, (WINDOW-COL 1) regresa 39, (WINDOW-STATES 2 '(("*Modelo*" NIL NIL NIL T "TRABAJO1.SSN"))) substituye la información general de la ventana tres, por (("*Modelo*" NIL NIL NIL T "TRABAJO1.SSN")) .

Las funciones *EXECUTE-OPTION*, *MUESTRA-OPTIONS*, *MAX-LENGTH*, *MUESTRA-OPTION* y *SELECT-OPTION* sirven para el **manejo de presentación de mensajes y elección de opciones**.

Otras funciones incluidas en este módulo son **utilerías que permiten presentar prompts para la lectura de datos** en el área correspondiente al desplegado de las opciones. La función encargada de ello es *MODE-QUERY* que recibe dos parámetros. El primero de ellos es TITLE, y es el letrero a imprimir antes de los prompts para leer datos. El segundo, OPTIONS, es una lista que tiene tantas listas como datos se vayan a solicitar al usuario.

Cada lista tiene seis elementos. Los primeros tres son valores de salida:

1. El valor leído por *MODE-QUERY* para el dato solicitado al usuario.
2. Renglón donde se desplegará el prompt para solicitar el dato.
3. Columna donde se desplegará el prompt para solicitar el dato.

y los tres últimos son valores de entrada para *MODE-QUERY*.

4. Mensaje a desplegar en el área de mensajes al solicitar el dato.
5. Prompt para solicitar el dato.
6. Espacio en caracteres que ocupará el valor a ser tecleado por el usuario. O bien una lista de valores alternativos para que el usuario elija.

BORDER-WINDOWS Es la función principal usada para el dibujo de los marcos de las ventanas. Utiliza caracteres gráficos.

Las rutinas que siguen son para **activar alguna de las áreas de la pantalla**.

CURRENT-WINDOW El área de ventanas, específicamente la que se refiere a la ventana activa.

OPTION-WINDOW El área donde se despliegan usualmente las opciones.

PROMPT-WINDOW El área de mensajes.

STATUS-WINDOW El área para el desplegado del estado de las sesiones.

También están incluidas en este módulo definiciones de algunas variables como *INIT-SCREEN*, *RESET-SCREEN*, *BORDER-CHARS*, *WINDOW-COLORS* y de algunas funciones tales como *NORMAL-VIDEO, INVERSE-VIDEO, BLINK-WRITE-SCREEN, CURSOR-ON, CURSOR-OFF, CONSOLE-BYTE* que sirven para manejar la lectura de las teclas y la forma del desplegado sobre la pantalla, según el tipo de monitor.

La función *"Modelo"* tiene en su estructura una entrada para cada uno de los casos bajo los cuales es llamada por la función *WINDOWS*, cuando la instancia de la aplicación requiere realizar alguna operación. Para indicar la operación deseada utiliza el primer argumento, COMMAND. Los demás argumentos se refieren a la información específica de la instancia de la aplicación.

La función *"Modelo"* contempla cuatro posibles comandos para atender a *WINDOWS* [2]. A continuación se describe lo que se realiza con cada uno de ellos.

a) Cuando se crea una instancia para la aplicación *"Modelo"*, el manejador utiliza el comando CREATE-WINDOW para obtener una lista con los valores apropiados que debe contener la infomación específica de una ventana. Esta información debe ser

   (*"Modelo"* NIL NIL NIL NIL "SESION.SSN")

   de acuerdo a lo expuesto en la sección 4.2.

b) Similarmente *WINDOWS* utiliza el comando CLOSE-WINDOW. Para esta entrada la aplicación verifica si se acepta cerrar una ventana. Si

los cambios realizados ya han sido salvados en dicha ventana entonces se puede cerrar. De lo contrario regresa NIL. A menos de que el usuario no desee conservar la sesión que haya sido generada en esa ventana.

c) El comando UPDATE-WINDOW es usado por *WINDOWS* cuando es necesario limpiar y actualizar la información de la ventana indicada.

d) Finalmente, el comando RUN-WINDOW solicita que *"Modelo"* ejecute una instancia de la aplicación.

La ejecución de la instancia de una aplicación consiste básicamente en llamar a la función *EXECUTE-OPTION* que se encarga del manejo de la elección de opciones y del llamado de los procedimientos apropiados para la opción seleccionada. Después se llama a la función *UPDATE-STATE* para actualizar la información de *WINDOWS* .

En seguida de la definición de la función *"Modelo"* se encuentran varias definiciones.

En primer lugar la definición del árbol de opciones de la aplicación *"Modelo"* que es una lista del tipo que recibe la función *EXECUTE-OPTION*. Contiene un árbol de opciones como el que se esquematiza en la Fig 6.1.

A continuación se encuentran definidas las funciones *PERMITE-CERRAR-PATRON* y *LINEA-ESTADO*.

PERMITE-CERRAR-PATRON es una función que verifica que se hayan salvado los cambios realizados en el medio ambiente de una ventana para poder permitir que dicha ventana sea cerrada.

LINEA-ESTADO despliega en el área correspondiente la información de estado general de la sesión residente en la ventana activa.

Las funciones anteriores son la implementación que corresponde al núcleo de la función *"Modelo"* y que realizan el manejo adecuado para las ventanas.

A continuación se describen los grupos de funciones que son llamados para implementar las opciones que se brindan en el menú y en los submenues que presenta el sistema.

La función *LECTURA* lleva el cursor al área que le toca a la ventana activa e invoca al módulo transformador de FU-FIP para que tome la secuencia de entradas tecleadas por el usuario y genere la continuación de la sesión de la ventana.

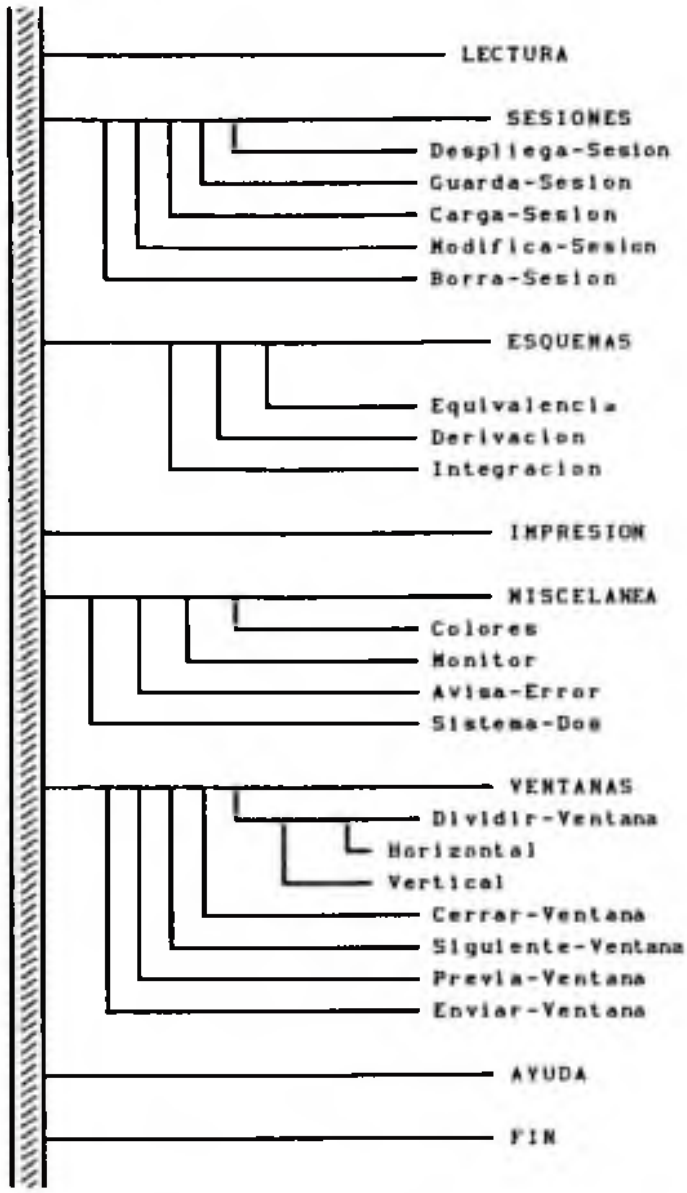


Fig. 6.1 Arbol de opciones del sistema.

Si aún no se inicia una sesión en esa ventana se crea entonces una sesión nueva.

Si se realizaron cambios en el estado de la sesión, estos son actualizados.

Después se encuentra el grupo de **funciones propias para el manejo de sesiones**.

DESPLIEGA-SESION permite al usuario desplegar el contenido completo de la sesión de la ventana activa. Con el fin de que el usuario pueda ver la secuencia que ésta ha seguido desde que se inició. Para ello recorre las listas *ENTRADAS* y *SALIDAS* desplegando alternativamente un elemento de cada una de ellas.

Por la forma en que éstas se fueron construyendo la primera entrada es el último elemento de las listas. Por tanto para desplegarlas en el orden adecuado en que se sucedieron es necesario invertirlas primero.

GUARDA-SESION pregunta el nombre para el archivo en disco con el que va a guardarse la sesión residente en la ventana actual.

Si el nombre del archivo ya existe, pregunta al usuario si reescribe sobre él, o en su defecto vuelve a pedir otro nombre.

Llama a la función *ESCRIBE-ARCH-SESION* para que grabe físicamente la sesión. Se guardan el el archivo las listas *ENTRADAS*, *SALIDAS* y *VARIABLES* que son las entradas dadas por el usuario en una sesión, las salidas que obtiene el sistema con esas entradas y la lista de los nombres de las variables que tienen un valor, respectivamente. Con lo cual queda almacenada la información de una sesión.

CARGA-SESION pregunta el nombre del archivo en disco de la sesión que el usuario desea cargar en la ventana actual.

Si hay algo en la ventana actual lo borra, preguntando previamente si es necesario guardar el contenido.

Lee mediante la función *LEE-ARCH-SESION* el contenido del archivo, inicializando con él las listas de *ENTRADAS*,*SALIDAS* y *VARIABLES*, recuperando así la información que define la sesión y de donde puede obtenerse su medio ambiente.

Finalmente con *ENTRADAS* y *SALIDAS* reconstruye el medio ambiente de la sesión realizando los efectos laterales que haya habido en alguna de las entradas, que consiste en dar a las variables de la lista *VARIABLES* los valores que tenían.

MODIFICA-SESION pide al usuario los datos necesarios para

eliminar entradas en la sesión de la ventana actual.

Elimina las entradas requeridas por el usuario de las listas*ENTRADAS* y *SALIDAS*.

En seguida está un grupo de **funciones propias para desplegar las tablas de equivalencias matemáticas, fórmulas de derivación y fórmulas de integración**.

Son básicamente tres funciones muy semejantes, *T-EQUIVALENCIA, T-TABS-DER, T-TABS-INT.*

Realizan un llamado a la función que despliega la pantalla de tablas, *DESP-PANTALLA*, pasando como parámetro la lista con la información de los caracteres para las tablas de equivalencia, de derivación y de integración, que son *PANT-EQU*, *PANT-DER* y *PANT-INT* respectivamente.

La función que despliega la información anterior es *DESP-PANTALLA.*

DESP-PANTALLA limpia el área de la ventana actual y despliega las tablas llamando a las funciones que permiten al usuario subir o bajar un renglón, *LINEA-ARRIBA* y *LINEA-ABAJO* respectivamente y *PAGINA-ARRIBA* y *PAGINA-ABAJO* para subir y bajar una página, de manera correspondiente.

Sigue la **función para la impresión de sesiones**, *IMPRESION.* Esta función es semejante a la de *DESPLIEGA-SESION* con la diferencia de que el archivo de salida no es la pantalla sino la impresora.

El siguiente grupo de **funciones son propias para las utilerías de miscelánea**: *SET-COLOR*, *SET-DISPLAY*, *SET-MUTE* y *GO-DOS*.

Las primeras tres funciones realizan un llamado a *MODE-QUERY* para solicitar los datos que el usuario desea cambiar y colocan los valores nuevos en las listas adecuadas para guardar la información de colores en *WINDOW-COLORS*, la bandera del sonido seleccionada por el usuario queda en el primer elemento de la lista SET-MUTE, y en el caso de la información leida para el estado y características del monitor éstas son ajustadas y guardadas en la lista SET-DISPLAY.

La última función de miscelánea es *GO-DOS* que permite salir del sistema para ejecutar momentáneamente comandos de MS-DOS usando la

función *EXECUTE* del intérprete de Mu-LISP.

A continuación se encuentra el grupo de funciones referentes al manejo conceptual de ventanas, que permiten operaciones como abrir y cerrar ventanas, moverse de una ventana a otra y enviar información entre ellas.

DIVISION-HORIZONTAL y *DIVISION VERTICAL* piden al usuario la información necesaria para abrir una nueva ventana y la añaden en *WINDOWS* inicializando sus coordenadas. Después regresan el control de la evaluación a la función *WINDOWS* en el punto etiquetado DIVISON-WINDOW.

CLOSE-WINDOW pide el número de la ventana que se desea cerrar y regresa el control de la evaluación a la función *WINDOWS* en el punto etiquetado 'CLOSE-WINDOW.

LAST-WINDOW y *NEXT-WINDOW* regresan el control de la evaluación a la función *WINDOWS* en el punto etiquetado SWITCH-WINDOW.

ENVIA-WINDOW pide los datos necesarios para enviar información de la sesion de una ventana a otra, los valida, realiza en envio y regresa el control de la evaluación a *WINDOWS* en el punto etiquetado SWITCH-SCREEN.

6.2. Funcionamiento del transformador FU-FIP y función que lo invoca, LEE-PROCESA.

El texto fuente que corresponde a este módulo puede ser consultado en el apéndice C.

En el mismo texto se introdujo una función muy importante del sistema, que es básicamente la que realiza los llamados a todos los demás módulos del sistema. El primer módulo que invoca es precismente el de transformación FU-FIP, por lo cual va a ser descrito en esta sección y antes que el transformador.

6.2.1 La función *LEEPROCESA*.

La estructura de la función *LEEPROCESA* es un ciclo que termina con ESC para regresar a la función de *LECTURA* que es la que la invoca.

En cada iteración se realizan las siguientes acciones:

1) Se inicializa el medio ambiente para llamar al reconocedor sintáctico (parser), mediante:
   a) La colocación de NIL en las banderas de detección de errores.
   b) La colocación de toda la entrada leída en la variable #ENTRADAACTUAL, para tomar de ahí elemento léxico por elemento léxico de la expresión (Tokens).
   c) Se toma el primer elemento léxico de la expresión leída y se pone en la variable #TK usando la función *SCAN*.
   d) Invoca al reconocedor sintáctico (parser) a través de la función que reconoce la categoría más alta de la gramática: *COMANDO*.
2) Cuando la función *COMANDO* regresa NIL significa que durante el proceso fue detectado un error, el cual pudo haber sido de tipo léxico o sintáctico y entonces debe haber quedado encendida alguna de las banderas #ERROR-LEX# ó #ERROR-SIN# respectivamente, con el código de error adecuado para desplegar el mensaje.
3) En caso de que la expresión introducida por el usuario se haya detectado válida, su representación en FIP quedó en la variable #ENTRADA.
4) La expresión en FIP se evalúa de acuerdo a su operador principal.
   a) Si el operador principal es un terminador, el módulo de terminadores es invocado.
   b) Si contiene operadores aritméticos se invoca al módulo aritmético.
5) El resultado de la evaluación de la expresión se asigna a la variable de salida correspondiente, donde la expresion se coloca en FIPN y se llama al transformador FIPN-FU para presentar el resultado al usuario.

6.2.2 Funcionamiento del transformador FU-FIP.

El módulo que se encarga de la transformación FU-FIP, utiliza como herramienta la gramática del apéndice A, mediante la cual es sencillo determinar automáticamente si la expresión introducida en FU es válida o no para el sistema.

Se utiliza un parser TOP-DOWN de descenso recursivo [4], durante la transformación FU-FIP. Se va ejecutando un conjunto de procedimientos recursivos para procesar la entrada. Se asocia cada uno de estos procedimientos a un símbolo no terminal de la gramática.

La gramática del apéndice A no puede usarse directamente pues algunas producciones hacen que un parser de descenso recursivo entre en un ciclo.

Las producciones de este tipo en la gramática la hacen recursiva por la izquierda. Son las que corresponden a la descripción de <EXP1> y <TERM>. Más precisamente las dos primeras de cada una de estas categorías sintácticas.

```
<EXP1> ::= <EXP1> + <TERM>
<EXP1> ::= <EXP1> - <TERM>
<EXP1> ::= <TERM>

<TERM> ::= <TERM> * <FACTR>
<TERM> ::= <TERM> / <FACTR>
<TERM> ::= <FACTR>
```

El ciclo sucede al tratar de reconocer, por ejemplo <EXP1>, para lo cual se incurre, de acuerdo al consecuente, en tratar nuevamente de reconcer <EXP1> y así sucesivamente sin terminar nunca de consumir toda la parte derecha de la producción.

Existe un método de eliminación de recursividad izquierda en una gramática [4]. El método consiste básicamente en cambiar cada conjunto de producciones

$$A \rightarrow A\ \alpha$$
$$A \rightarrow \beta$$

por el siguiente grupo de producciones

$$A \rightarrow \beta\ A'$$
$$A' \rightarrow \alpha\ A'$$

$$A' \rightarrow \varepsilon$$

Entonces a la gramática del apéndice A se le eliminó así la recursividad izquierda.

El grupo de producciones mencionadas arriba para <EXP1> se substituyó por:

```
<EXP1>  ::= <TERM> <REXP1>
<REXP1> ::= + <TERM> <REXP1>
<REXP1> ::= - <TERM> <REXP1>
<REXP1> ::= ε
```

y similarmente las producciones para <TERM> se substituyeron por:

```
<TERM> ::= <FACTR> <RTERM>
<RTERM> ::= * <FACTR> <RTERM>
<RTERM> ::= / <FACTR> <RTERM>
<RTERM> ::= ε
```

Con la gramática resultante, sin recursividad izquierda, se codifica una rutina para cada una de las categorías sintácticas definidas en dicha gramática.

Tales rutinas son las funciones *COMANDO, EXP, EXP1, REXP1, TERM, RTERM, FACTR, PRIMARY, LISTAARGS.*

El reconocedor sintáctico (parser) está invocado por la función *LEEPROCESA.*

Esta función realiza el llamado a la función *COMANDO*, para reconocer una expresión o un comando válido para el sistema.

La lógica que sigue *COMMANDO* es la misma lógica que siguen todas las demás funciones del reconocedor sintáctico, siguiendo cada una su producción asociada de la siguiente manera.

```
(DEFUN  COMANDO  (#E  #R)
  (  (NULL  (SETQ  #E  (EXP)))  #E  )
  (  (TERMINADOR-P #TK)
           (SETQ   #R   #TK)
           (SCAN)
           (LIST  #R  #E)  )
  (ASIGNACODIGO-SIN      3)
)
```

Se supone que al invocar cada una de las funciones del reconocedor sintáctico se encuentra en la variable #TK el siguiente elemento léxico que se va a analizar de la entrada.

Cuando una función de estas no reconoce la categoría sintáctica que le corresponde entonces debe regresar NIL, con el objeto de que la función que la llama detecte adecuadamente la existencia de un error durante el análisis.

La función *COMANDO* debe reconocer una forma tipo <EXP> y luego uno de los terminadores '$', '!', '&', ';'. Las producciones que la definen son:

```
<COMANDO> ::=  <EXP> $
<COMANDO> ::=  <EXP> ;
<COMANDO> ::=  <EXP> !
<COMANDO> ::=  <EXP> &
```

Se llama a la función *EXP* para tratar de reconocer una forma descrita por la categoría sintáctica <EXP>. Si al llamarla ésta regresa NIL significa que no fué reconocida dicha forma. Como no fue reconocida, una de las variables de error debe haber sido encendida durante las llamadas consecutivas cuando se detectó el error, y entonces debe regresar NIL la función *COMANDO* y no continuar con el reconocimiento de dicha categoría.

Si fue reconocida una <EXP> se guarda, y se trata de verificar si el siguiente símbolo a analizar es un terminador. En caso afirmativo se lee de la entrada con *SCAN* para ver si hay más elementos que analizar, y se regresa la lista de la expresión leída en FIP. Si no se reconoce un terminador como el siguiente elemento, se enciende la bandera de error sintáctico, #ERROR-SIN# con el código 3 que correponde al mensaje de "Terminador Esperado".

Como puede verse cuando se requiere reconocer un símbolo terminal de la gramática se invocan las funciones de reconocimiento léxico.

Las **funciones principales de reconocimiento léxico** se enumeran a continuación.

SCAN Obtiene el siguiente símbolo (token) a analizar de la lista que contiene los caracteres de la entrada actual, #ENTRADAACTUAL, usando para ello la función *RASTREATK*.

Las funciones auxiliares de reconocimiento de caracteres son *SIGNOESPECIAL-P*, *PORCENTAJE-P*, *INTARRO-P*, *VAR-IMPLICITA*, *NUM-SIMBOLICO* y las de edición de línea *LEEENTRADA*, *EDITA-LINEA*, *MAPEAASCII*,

TECLA-F, IDENT-FUNCION y BORRA-IMPRESION.

6.3. Funcionamiento del Transformador FIP-FIPN.

Este módulo solo contiene dos funciones y están en el texto del apéndice D, donde también está el código del módulo aritmético.

Las dos funciones son llamadas cuando en una expresión en FIP se encuentran operadoreas de división, "/", ó de diferencia "-".

-TRAP es llamada por el intérprete de Mu-LISP cuando éste trata de evaluar una forma (- *arg1 arg2 . . . argn*) donde los argumentos no son numéricos sino simbólicos. El intérprete la llama, enviando sólo dos argumentos y asociándolos de izquierda a derecha .

La función regresa la suma del primero, más el segundo multiplicando por -1. Es decir

a - b = a + (-1 * b)

que es la forma de la expresión normalizada.

/ es llamada por el intérprete cuando éste trata de evaluar una forma.

(*/ arg1 arg2*)

La función regresa la multiplicación del primero por el segundo elevado a la -1, es decir ,

a / b = a * (b ^ -1)

en donde la expresión equivalente es ya una forma normalizada.

6.4 Funcionamiento del Manejador de Terminadores

Entre los objetivos iniciales de este sistema se consideró deseable permitir que el usuario *dirigiera* la forma en que debe resolverse un problema. En tal caso el sistema no debe dar automáticamente una solución al mismo. También se deseaba que en algunas ocasiones el sistema no diera el resultado automáticamente sin justificar absolutamente nada acerca del proceso. Es así como surgen formas diferentes de que un usuario pida al sistema como evaluar una

expresión. Para ello se da la facilidad de que el usuario indique la manera en que desea evaluar una expresión, y se crearon los terminadores.

El código del manejador de terminadores puede consultarse en el apéndice E. Las funciones principales codificadas en este módulo son ';', '$', '!' y '&'.

El módulo de terminadores se invoca en la función *LEEPROCESA*. Cuando se ha reconocido que una función es sintácticamente correcta se llama una de estas funciones, según el terminador de la expresión. El llamado se efectua de la siguiente manera:

(APPLY (CAR *expresion*) (CDR *expresion*))

en donde *expresion* es una lista que contiene la FIP de la expresión admitida.

La función que corresponde al terminador automático es ";", esta función debe entonces llevar a cabo de manera automática toda la operación que se indica en la expresión tecleada por el usuario. Como ésta se encuentra en FIP se llama a la función que evalua la expresión indicando cual es el terminador. La función propia para ello es *EVALUA-EXP*.

La función para el terminador mudo es "$", que realiza exactamente lo mismo que la función para el terminador automático. Después de que termina la evaluación, en el lugar donde es llamada se impide la impresión del resultado.

Para el terminador explicito se tiene la función "!". En esta función se prende la bandera #TRACE, para indicar que al realizar la evaluación, también usando la función *EVALUA-EXP*, sea registrado el proceso realizado en un stack llamado #STACK-TRACE. Se usa la función *IMP-PROCESO* para que imprima, de acuerdo a #STACK-TRACE, una sencilla descripción del proceso realizado durante la evaluación de la expresión. Se inicialiIza el stack y se apaga la bandera de #TRACE, para que se efectue la evaluación de la próxima entrada de manera correcta.

El proceso de rastreo utiliza tres funciónes *TRACE-FUN, TRACE-RET, y TRACE-PTO*. Estas tres funciones registran entradas en el stack #STACK-TRACE que van a indicar partes del proceso durante la evaluación de la expresión.

La primera, *TRACE-FUN*, registra una entrada en el stack de la forma (NOMFUN_IN (ARG1 ARG2 ... ARGN)). Dicha entrada en el stack indica que se llegó a la función NOMFUN, con los argumentos ARG1, ARG2 ... ARGN. El postfijo _IN indica llegada a la función. Esta función se utiliza al principio de las funciones que se desean registrar para el proceso, en este caso *DERIVA* e *INTEGRA*. Su llamado es para cada una de ellas *(TRACE-FUN 'DERIVA LST-ARGUMENTOS)* y *(TRACE-FUN 'INTEGRA EXP1 VAR-INT)*, de manera respectiva.

La función *TRACE-RET* registra en el stack una entrada de la forma (NOMFUN_OUT REGRESO) que indica que se terminó la función NOMFUN y que su valor de regreso fue REGRESO. Es importante que esta función siempre regrese REGRESO, pues es usada en cada una de las funciones que se desean registrar en el proceso, en todos los posibles puntos en los que la función pueda terminar y es la última expresión a evaluar en la función.

La función *TRACE-PTO* registra varios tipos de entrada en el stack. Su forma general es (NOM-PTO EXP1 EXP2 ... EXPN), la diferencia está en NOM-PTO, que es un nombre único para cada llamada, e identifica un punto estratégico del proceso, que se va a describir. Las expresiones que se registran dependen de lo que se requiera guardar en cada uno de los puntos.

Es importante hacer notar que la información que se registra en cada punto estratégico debe ser elegida de tal manera que describa algún paso que signifique algo importante para el usuario. La descripción que puede dársele se refiere a la manera en que el algoritmo realiza las operaciones, pues es la única información que se tiene; y no como se realizaría humanamente dicho proceso. Entonces es necesario elegir los puntos de la siguiente manera:

a) escogiendo partes del algoritmo que tiene algo en común con la forma humana de realizar una operación.
b) identificar la información que tienen las variables y el medio ambiente en ese lugar.
c) usar la función *TRACE-PTO* para registrar en el stack las expresiones necesarias.

Para registrar el uso de fórmulas se debe identificar lo siguiente:

a) El tipo de fórmula que se está usando, si es de integración o de derivación y cual de ellas es. Registrado en X y xx respectivamente de la forma siguiente.

(EXP1 %Xxx (VAR1 ASOC1) (VAR2 ASOC2)... (VARN ASOCN))

b) La expresión que se utiliza para aplicar dicha fórmula, EXP1.

c) Las asociaciones de variables que se realizan.

Para el caso especial del registro de puntos no se utilizan las mismas listas que describen las fórmulas de las tablas de derivación e integración que se despliegan al usuario en la opción de ESQUEMAS, sino una lista, de forma semejante a éstas que se llama *ESQUEMAS-EXP*.

Otro tipo de punto es el que registra que una expresión se transforma en una equivalente, para lo cual es necesario identificar lo siguiente:

a) Cual es la expresión antes de que se realice una transformación, EXP-ANTES.

b) El tipo de transformación que se realiza, n.

c) Cual es la expresión después de que se realizó la transformación, EXP-DESPUES.

Lo cual queda registrado en un punto de la siguiente forma:

(TRANSFn (EXP-ANTES EXP-DESPUES))

Las tres funciones anteriores solamente realizan registros en el stack cuando está prendida la bandera #TRACE.

La función *IMP-PROCESO* toma como entrada el stack que llenaron las funciones *TRACE-FUN, TRACE-RET y TRACE-PTO* y despliega strings con la explicación propia para cada punto, que se completa con las expresiones que están en las entradas del stack.

Finalmente la función para el terminador dirigido es "&". Esta función llama a la función *EVALUA-EXP*. Esta última, se encarga de evitar la evaluación que realiza el evaluador de LISP en los casos que requiere el terminador dirigido. En una expresión que termina con el evaluador dirigido las únicas funciones que deben calcularse son las funciones evaluables, que son *APLICA* y *SUBEXP*, para permitir que el usuario indique explícitamente el camino del proceso.

La función *EVALUA-EXP* tiene dos argumentos, el primero es la expresión a evaluar y el segundo es el nombre de la función que la mando llamar, que puede ser cualquiera de las que correponden a los terminadores. Recorre en toda su profundidad la expresión, buscando los operadores de cada subexpresión que contenga. Cuando el terminador que llamó fue ";", "$", o "!" se permite que realice la evaluación normal el evaluador de LISP, *EVAL*. Si el terminador llamador es "&" se distinguen las funciones *APLICA* y *SUBEXP* para realizar su evaluación, llamándolas con sus argumentos evaluados con *EVALUA-EXP* y no con *EVAL* de LISP. En cualquier otro caso se regresa la expresión de entrada sin evaluar.

6.5. Manipulador Simbólico.

6.5.1. Funcionamiento del Módulo Aritmético.

El código correspondiente a este módulo se encuentra en el texto del apéndice D.

Las rutinas principales codificadas en él, son las que se encargan de definir las operaciones simbólicas de suma, multiplicación, exponenciación y logaritmos.

Para la **definición de la suma simbólica** están las funciones *+TRAP, MERGETERM, MERGESUM, ADDTERM.*

+TRAP Es llamado por el intérprete sólo con dos elementos, de los que al menos uno es no numérico. Sobre dichos argumentos debe realizarse una suma simbólica.

+TRAP tiene un parámetro que es la lista LEX1 con 2 elementos. Los elementos de LEX1 son expresiones en FIPN que representan las expresiones simbólicas a sumar.

LEX1 = (<FIPN-EXP1> <FIPN-EXP2>)

Esta función puede recibir potencialmente n argumentos para sumarlos, sin embargo la asociación de izquierda a derecha se encarga de hacerla el intérprete al llamarla y sólo le envia dos elementos a

la vez de la manera esquematizada en la Fig 6.2.

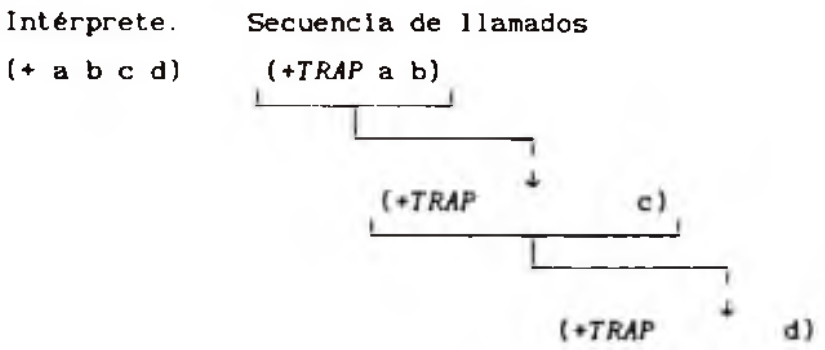


Fig. 6.2 Secuencia de llamados del intérprete a *+TRAP*

+TRAP tiene definida localmente una lista, LEX2, que sirve para almacenar los términos que resulten al sumar las expresiones de LEX1.

LEX2 = (<FIPN-TERM> . . . <FIPN-TERM>)

La lógica que sigue la función en cuestión es la siguiente:

Efectúa un ciclo que se ejecuta tantas veces como expresiones haya que sumar en LEX1. En cada iteración se realiza lo siguiente:

a) Se toma la expresión de LEX1 que toca sumar y junto con los términos que ya hayan sido generados se envia a la función *MERGESUM*. La lista de los términos resultantes que se obtiene se asigna a LEX2 para actualizarla.

b) Cuando se han terminado de sumar todas las expresiones, los términos resultantes de la suma han quedado en LEX2. Se construye, usando la función *MKSUM*, la lista que representa la suma de los términos que hay en LEX2.

MERGESUM recibe dos argumentos EX1 y LEX1 cuya forma es la siguiente:

EX1 = <FIPN-EX1>
LEX1 = (<FIPN-TERM> . . . <FIPN-TERM>)

Además está definida una variable local, LEX2, que sirve como auxiliar para ir descartando los términos de LEX1 que ya han sido sumados. Tiene la misma forma que LEX1.

EX1 es la expresión que representa el elemento a sumar, es

propiamente la representación en FIPN de una <EXP1>. LEX1 tiene los términos que ya existen en la suma que se está realizando.

La lógica de la función *MERGESUM* es la siguiente:

1) Si no hay términos previamente en LEX1 se revisa la forma de la expresión que se desea sumar.
   a) Cuando es una suma, se elimina el operador +, y se regresa una lista con todos los términos que formaban la expresión.
   b) Cuando la expresión no es suma, significa que se trata de una expresión que está formada de un solo término y se regresa una lista que contiene el término.
2) Cuando ya hay términos en LEX1, se llama a la función *MERGETERM*, para que se encargue de sumar los términos que se encuentren en la expresión y se consideran los dos casos del punto anterior
   a) Cuando la expresión es una suma, se realiza un ciclo de acuerdo al número de términos que contiene la expresión. En cada iteración se toma un término de la expresión y se usa LEX1 para ir acumulando los términos resultantes. La suma de términos se realiza por medio de un llamado a *MERGETERM* en cada vuelta del ciclo.
   b) Si la expresión no es suma, solamente se realiza un llamado a *MERGETERM*, enviando como término la expresión y acumulando el resultado en LEX1.
3) *MERGESUM* regresa una lista de la forma indicada para LEX1. En tal lista ya se encuentran incorporados los términos que hayan sido obtenidos de la expresión. La función *MERGETERM* se encargó de realizar la simplificación de los términos semejantes.

MERGETERM recibe dos argumentos EX1 y LEX1 cuya forma es la siguiente:

EX1 = <FIPN-TERM>

LEX1 = (<FIPN-TERM> . . . <FIPN-TERM>)

Las variables definidas para el medio ambiente creado por la función *MERGETERM* son EX2, EX3, LEX2 y LEX3. Las dos primeras son de la forma de EX1 y las dos últimas de la forma de LEX1 y funcionan como meras variables auxiliares.

EX1 es una expresión de un solo término, que es el que se va a

sumar. Cuando ya haya término semejante a él en LEX1 se sumará ahí. Sino se creará una entrada nueva en LEX1.

LEX1 es la lista que contiene las expresiones que representan los términos que ya existen en la suma.

La lógica de la función *MERGETERM* es como sigue:

Ejecuta un ciclo que se efectúa tantas veces como términos haya en LEX1. A menos de que el término a sumar sea independiente. En cada iteración se verifica lo siguiente.

1) Si ya se terminaron de revisar todos los términos que originalmente estaban en LEX1, buscando que alguno representara un término independiente. Si no se encontró ninguno se llega a este punto en el que se busca entonces, por medio de un ciclo, si existe en LEX1 un término semejante al que está representado en EX1.

   Para verificar si un término es semejante a otro, se obtienen los factores no numéricos de los términos a comparar, usando la función *CODIV* [6.5.2], y se comparan por medio de la función *EQUAL* de LISP.

   a) Si se encuentra algun término semejante a EX1, se obtienen los coeficientes, factores numéricos, de ambos y se suman usando la función *ADDTERMS*. Se construye una lista que sea la representación de la suma de los términos semejantes y se ordena en la lista que regresará *MERGETERM*.

   b) Si no se encuentra, se regresa una lista construida con los términos que venian originalmente en LEX1 con una entrada más para el nuevo tipo de término que venía en EX1.

2) Si se ha encontrado un término independiente en LEX1 que pudiera ser sumado con EX1 por medio de la función *ADDTERMS*. En cuyo caso la suma encontrada se incorpora con todos los demás términos, los que ya se hubieran revisado previamente y los que quedaron sin verificar, por medio de la función *MERGESUM*.

Para la **definición de la multiplicación simbólica** se usan las siguientes rutinas **TRAP*, *MERGEPROD*, *MERGEFACT*, *MULTFACTS*.

Si se observa el esquema de la Fig. 6.3, donde se describe la secuencia de llamados entre las funciones que realizan la suma

simbólica y las correspondientes a la multiplicación simbólica, se puede observar que su funcionamiento es muy semejante.

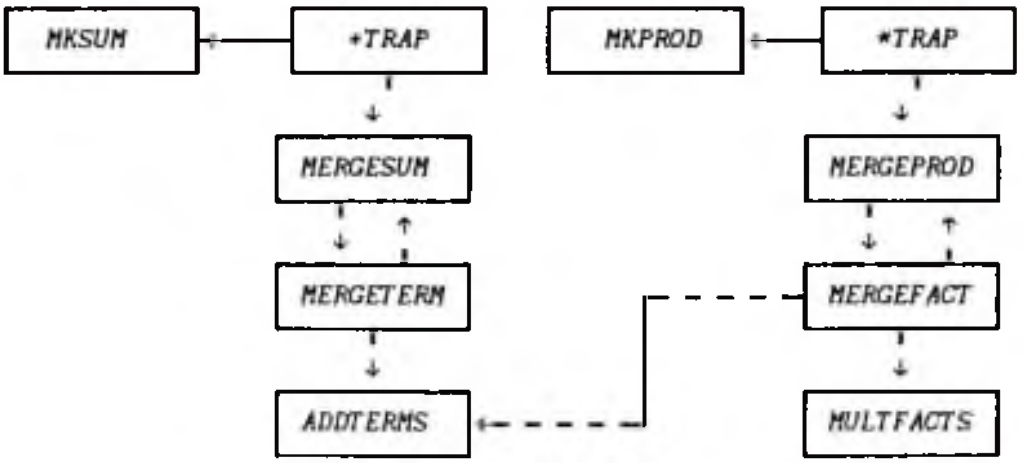


Fig. 6.3 Rutinas para suma y para multiplicación simbólica.

**TRAP* es la función paralela a *+TRAP*.

MERGEPROD es la función correspondiente a *MERGESUM*, asi como *MERGEFACT* a *MERGETERM* y *MULTFACTS* a *ADDTERMS*.

Su estructura es igual y cada una de estas funciones para la multiplicación sigue la misma lógica que la de la suma. La diferencia es que una suma contiene términos, en tanto que en la multiplicación se habla de factores.

La otra diferencia es que el elemento neutro para la suma es el cero, en tanto que para la multiplicación es el uno. Esta diferencia se refleja hasta que se realiza el llamado a las funciones *ADDTERMS* y *MULTFACTS*.

Las **funciones para la definición de la exponenciación** son *EXPT* y "^".

La principal de las dos es "^". Realiza varias referencias a ciertas propiedades de los átomos BASE y EXPONENTE. Dichas propiedades están codificadas en el módulo de codificación de tablas.

Las propiedades BASE y EXPONENTE se definen para un átomo que es un operador o bien el nombre de una función. La propiedad <*atomo*> de BASE es una función. Esta función define el resultado apropiado que se obtiene cuando en una exponenciación el operador principal de la base

es *<atomo>*. De manera similar, en la propiedad *<atomo>* de EXPONENTE se define el resultado que se obtiene cuando en una exponenciación el operador principal del exponente es *<atomo>*.

"^" Recibe dos parámetros, EX1 y EX2. EX1 es la base de la exponenciación y EX2 es el exponente.

La primera parte de la función corresponde a la potenciación numérica. Esta sólo se realiza cuando, tanto la base como el exponente, son enteros.

Realizando primeramente las simplificaciones siguientes:

a) $1^X = 1$ Sin importar que entero sea X.

b) 0^0 no está definido, se marca error.

c) 0^{-X} no está definido, pues provoca división entre 0, pues $0^{-X} = \frac{1}{0^X}$ sin importar que entero sea X.

Luego se manda realizar la exponenciación a la función *EXPD*.

Después se verifican los casos en los que el exponente es numérico y la base no es un entero. La base es simbólica, es decir, puede ser cualquier simbolo, ej. A, %PI, %I, una lista que represente una suma, o bien puede ser un número racional representado por una lista, por ej, p/q , con p y q enteros estaría representado en la lista (* p (^ q -1)).

a) $X^1 = X$ siempre.

b) $X^0 = 1$, si esta encendida la variable de control #ZEROEXPT que permite realizar dicha simplificación,

A continuación se verifican los casos en los que la base es atómica y representa al número imaginario *i*.

Si la base es simbólica, tal vez no sea atómica y deben considerarse los casos en los que la base es una expresión. En este punto se invoca a la función definida para el operador principal de la base como propiedad de BASE.

Si la base no fue el átomo simbólico del número *i*, ni fue una expresión, entonces el resultado es la lista que representa la exponenciación simbólica de EX1 elevado a la potencia EX2, que es

entera.

Cuando el exponente no es entero, se verifican ahora los casos para los que la base es simbólica y además atómica.

a) 1^{X} entonces el resultado es 1. X no es entero.

b) 0^{-X} provoca división entre 0; pues 0^{-X} es $\frac{1}{0^{X}}$.

Después se consideran los casos en que tanto la base como el exponente son átomos simbólicos.

A continuación está contemplado el caso en que la base es un símbolo atómico y el exponente es una expresión. Con ello se llama a la propiedad EXPONENTE que corresponde al operador principal del exponente.

Finalmente está el caso en el que tanto la base como el exponente son expresiones simbólicas y se utilizan las propiedades de los átomos BASE y EXPONENTE según el operador del exponente y de la base respectivamente.

LOG es la **función encargada de obtener logaritmos**. Recibe dos argumentos EX1 y EX2. EX1 es la expresión de la que se va a buscar el logaritmo y EX2 es la expresión que representa la base del logaritmo.

La variable #LOGBAS se considera la base por omisión del logaritmo.

De manera similar que funcionan para la exponenciación las propiedades de los átomos BASE y EXPONENTE, en la obtención de logaritmos se tienen propiedades para el átomo LOG.

Cuando al tratar de obtener Log_{EX2} EX1, si la EX1 es una expresión no numérica, la propiedad del operador principal de EX1 del átomo LOG define el resultado apropiado para ese caso.

Cuando la variable de control #PRRCH es diferente de NIL se considera realizar las siguientes transformaciones

$$\log_a 1 = 0$$

$$\log_a a = 1$$

$$\log_a a^{x} = x$$

Cuando la base del logaritmo y la expresión son enteros, es calculado el logaritmo con ayuda de las funciones MODULO y QUOTIENT.

Si la función fué invocada para convertir bases se verifica que

calculado el logaritmo con ayuda de las funciones MODULO y QUOTIENT.

Si la función fué invocada para convertir bases se verifica que la variable de control #LOGEXPD sea un múltiplo positivo de 2 y que #LOGBAS no sea igual a la base EX2 para realizar la siguiente conversión:

$$\log_a x = \frac{\log_b x}{\log_b a}$$

Al final se hace uso de la propiedad adecuada del operador principal de EX1 del átomo LOG para obtener $\log_{EX2}$ EX1 cuando EX1 es una expresión.

Las propiedades de LOG definidas permiten reflejar las leyes de los logaritmos, tales como

$$\log_x A\,B = \log_x A + \log_x B$$
$$\log_x A^B = B \log_x A$$

y están codificadas en el módulo de codificación de tablas.

Después de las funciones principales están codificadas funciones de apoyo para definir propiedades, para redefinición de la función *BREAK* del sistema y *QUOTIENT*. Además de funciones de prueba para identificar si un número es -1, o si es non o entero, funciones de ordenamiento, de verificación de múltiplos negativos y positivos, de reconocimiento de tipo de expresión, es decir, si una expresión es suma, producto, exponenciación o logaritmo, o bien alguna forma especial de función.

6.5.2 Funcionamiento de Módulo Algebraico.

El texto donde se encuentra el código fuente de este módulo puede ser consultado en el apéndice F.

En primer lugar se describen las **funciones propias para extraer subexpresiones**. Se usa el concepto de definir funciones apropiadas para algunos átomos, bajo un indicador que es operador. Estas funciones regresen el valor apropiado para cuando la expresión que se

Primeramente están las funciones *NUMERADOR* y *DENOMINADOR*, definidas de manera similar.

NUMERADOR El numerador de una expresión cuya forma interna es atómica, es la expresión misma. Para este caso la función *DENOMINADOR* debe dar como resultado 1.

Si la forma interna de la expresión de la que se desea obtener su *NUMERADOR* o su *DENOMINADOR* no es atómica, entonces su representación interna, que está canonizada, es una lista de cualquiera de las siguientes tres formas:

a) $(+\ arg_1\ arg_2)$

b) $(*\ arg_1\ arg_2)$

c) $(\hat{}\ arg_1\ arg_2)$

En el primer caso, la expresión es una suma, donde el numerador, es pues, la misma expresión y el denominador es 1.

En el segundo caso, la expresión representa un producto y el numerador de un producto es el producto de los numeradores de cada uno de sus argumentos. De manera semejante, el denominaor de un producto es el producto de los denominadores de sus argumentos. Se usa la propiedad * del átomo NUMERADOR y la del átomo DENOMINADOR, donde se encuentra definido el resultado apropiado para cada uno.

En el último caso tenemos que la lista representa una exponenciación y puede darse en sus argumentos uno de los siguientes casos respecto a su signo:

$$arg_1^{\,arg_2} \rightarrow arg_1^{\,arg_2}$$

$$arg_1^{\,-arg_2} \rightarrow \frac{1}{arg_1^{\,arg_2}}$$

$$-arg_1^{\,arg_2} \rightarrow -arg_1^{\,arg_2}$$

$$-arg_1^{\,-arg_2} \rightarrow \frac{1}{-arg_1^{\,arg_2}}$$

Como se puede observar, si el segundo argumento es el reciproco aditivo de arg_2, se tiene que el numerador de la expresión es 1. Hablando del denominador se observa que es el primer argumento elevado al segundo; pero con el signo invertido, es decir, positivo.

Hablando del denominador se observa que es el primer argumento elevado al segundo; pero con el signo invertido, es decir, positivo.

Si no es un recíproco aditivo el segundo argumento, el numerador es la misma expresión y el denominador es 1.

Estos últimos resultados son los que regresan las funciones definidas para los átomos NUMERADOR y DENOMINADOR bajo el operador *.

En seguida están las funciones *EXPONENTE* y *BASE*. Cuando una expresión se eleva a la primera potencia su representación interna no va a ser una potencia, puesto que el exponente 1 en cualquier expresión es implícito y es claro que el exponente de una expresión así es 1, en tanto que la base es la misma expresión.

Si la representación interna de la expresión comienza con el operador ^ puede suceder una de las siguientes dos cosas:

a) Que la exponenciación sea la representación del recíproco de cualquier número, es decir, algo como $\frac{1}{x}$, es decir x^{-1} , en cuyo caso no se trata de una potenciación propiamente dicha y el exponente de tal expresión es 1, en tanto que la base es la misma expresión.

b) Que sí sea una potenciación propiamente dicha, que tiene como primer argumento la base y como segundo argumento el exponente, entonces el exponente es el tercer elemento de la representación interna de la expresión (el primer elemento de la representación es el operador ^) , y el segundo elemento es la base.

COEFICIENTE Si se desea obtener el coeficiente de una expresión entera, entonces el coeficiente es la misma expresión.

Después de lo anterior tenemos la posibilidad de que la expresión tenga solamente una de las siguientes formas:

a) (* arg_1 arg_2)

b) (^ *int -1*), es decir, una potenciación que represente el recíproco de un número entero.

c) (+ arg_1 arg_2) ó (^ arg_1 arg_2) , donde la exponenciación no representa el recíproco de un número.

respectivamente un número entero y la representación de un número entero, si ésto es así entonces el coeficiente es el producto de ellos, o sea $arg_1 * arg_2$.

En el último caso tenemos que el coeficiente de una suma o de una potenciación es 1.

En el caso de que la expresión de la que se desea obtener coeficiente sea una suma, tenemos que el coeficiente de dicha expresión es simplemente 1.

SUBEXP En el primer argumento se encuentra en FIP la expresión de la cual se desea obtener cierta parte, #EXP1. En el segundo argumento, #EST, se encuentra la descripción del usuario de la estructura de la expresión en FIP. El último argumento, #EXPGEN, indica el usuario la expresión que desea obtener a partir de EXP1.

La lógica de función *SUBEXP* consiste en llamar a la función *EMPATA* para encontrar la lista de asociaciones que el usuario indica que existe entre #EXP1 y #EST.

Si la lista en donde se guardan las asociaciones #ALIST es consistente, entonces la utiliza para generar la expresión indicada por el usuario en #EXPGEN, de lo contrario envia un mensaje de error indicando que las expresiones #EXP1 y #EST no guardan la misma estructura.

La función *EMPATA* es un sencillo algoritmo de correspondencia en el que se verifica que la cabeza de las expresiones a empatar sea la misma. Si esto es así, se puede proceder a encontrar el empatamiento entre los argumentos, usando nuevamente la función *EMPATA*.

6.5.3 Módulo de derivación e integración.

Este módulo puede ser consultado en el apéndice G. En él están codificados tres grupos de funciones.

Las funciones *DEPENDE* y *NODEPENDE* que sirven para marcar la **dependencia o independencia de una variable respecto a otra**.

La marca se realiza utilizando la lista de propiedades del átomo que representa a la variable.

Cuando un átomo Y tiene asociada la propiedad DEPENDS bajo el

que representa a la variable.

Cuando un átomo Y tiene asociada la propiedad DEPENDS bajo el indicador X, significa que la variable Y depende de la variable X.

DEPENDE recibe en el parámetro LST la lista de los argumentos que el usuario envia a *DEPENDE*, en tal lista el primer elemento representa a la variable que se va a hacer dependiente de las variables representadas por el resto de los elementos.

La función se ayuda de la función *DEPEND1*, que implementa el ciclo en el que se le da el valor adecuado a la lista de propiedades de la variable dependiente.

NODEPENDE tiene un argumento, LST, de la misma manera que *DEPEND*. La función *NODEPENDE* es inversa que *DEPENDE*, se ayuda de la función *UNDEPEND1* para eliminar las asociaciones que indican dependencia.

El siguiente grupo de funciones son las que sirven para llevar a cabo la derivación de una expresión. Son las funciones *DERIVA*, *DIFN*, *DIF1*. Estas funciones guardan la secuencia de llamadas que describe la Fig. 6.4.

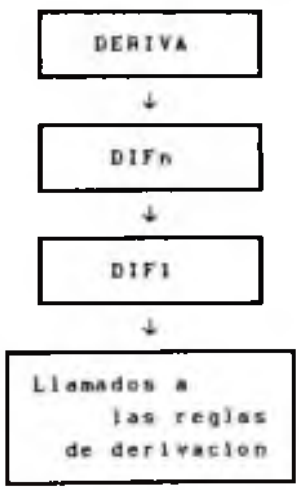


Fig. 6.4 Secuencia de llamados durante la Derivación.

La lógica que se sigue para la derivación se describe a continuación.

La función principal de derivación es *DERIVA*, pues es la que recibe los argumentos del usuario e invoca a las funciones que le

una lista, cuyo primer elemento es la expresión a derivar. Los demás elementos de LEX1 son variables y grados de derivación. LEX1 tiene pues la siguiente forma:

LEX1=(*exp var varororder varororder ....varororder)*

DERIVA separa la expresión a derivar de las variables y grados de derivación e invoca a *DIFN* con dicha información ya clasificada.

La función *DIFN* tiene dos argumentos *EX1* y *LEX1*. El primero es la expresión a derivar y el segundo es la secuencia de variables de derivación y los grados de ésta. Las variables auxiliares usadas son *EX2*, que es la variable con respecto a la que se deriva actualmente. *EX3* que es el número de veces que ha de derivarse con respecto a *EX2*.

DIFN se encarga de realizar la secuencia de derivaciones que se indican en *LEX1*, sobre la expresión. Toma uno a uno los elementos de *LEX1* y efectua las derivaciones individuales.

La lógica de la función es un ciclo en el que en cada paso realiza lo siguiente:

a) Toma el la primer variable de *LEX1* y la coloca en la variable auxiliar *EX2*, que debe haber sido validado que fuera una variable, no un número.
b) Ejecuta un ciclo en el que, usando la función *DIF1*, realiza la derivación de la expresion *EX1* con respecto a *EX2* tantas veces como se indica en el siguiente elemento de *LEX1*, si es que éste es un número y lo elimina de *LEX1*.
c) Cuando el primer elemento de *LEX1* no es un número, realiza la derivación con respecto a *EX2* una sola vez.
d) Termina el ciclo cuando ya no hay elementos en *LEX1*, que indiquen seguir derivando.

La función *DIF1* realiza una derivación sencilla. Recibe dos argumentos *EX1* e *INDET*. El primero es la expresión a derivar. *INDET* contendrá siempre la variable actual de derivación.

DIF1 realiza las siguientes verificaciones :

a) Si *INDET* no depende de *EX1* entonces DERIVADA(*EX1, INDET)* = 0.
b) Si *INDET=EX1* entonces DERIVADA(*EX1, INDET)* = 1.

a) Si *INDET* no depende de *EX1* entonces DERIVADA(*EX1*, *INDET*) = 0.

b) Si *INDET=EX1* entonces DERIVADA(*EX1*, *INDET*) = 1.

c) Cuando el operador principal de la expresión es *DERIVA*, se llama a la función auxiliar *DIFMERGE* para que se incorporen las variables de derivación, de tal manera que se realice primero la derivación interna.

d) Se realiza un llamado a la regla de derivación adecuada según el operador principal de la expresión a derivar.

INTEGRA es la función para realizar la integración de una expresión, la cual recibe en su primer argumento al integrando, EX1, y su segundo argumento corresponde a la variable de integración INDET. El algorimo de integración no utiliza un solo método para realizar cualquier integración, sino que usa varios métodos, los cuales son propios solo para cierto tipo de integrandos.
Se ayuda de las funciones *INT1*, *INT2*, *INT3*, *INTPROD* y *DRVDIV*. Se encuentra fuertemente apoyada en las definiciones para integración del módulo de definición de tablas.

Lo primero que se hace al empezar esta función, es transformar el integrando de tal manera que la expresión inicial sea expandida y reducida lo más que sea posible, colocándo valores a algunas variables que controlan la forma de la expresión. Se llama a la función *INT1* para realizar intentos de integrar la expresión.

Cuando falla algún intento se retransforma la expresión, convirtiendo lo más que sea posible a senos y cosenos para intentar integrar con esta nueva forma.

Cuando un intento falla se regresa NIL para seguir con otros intentos, ya sea transformando la expresión o tratando de aplicar alguna fórmula según el operador principal en el integrando.

La función *INTPROD* intenta realizar una integración por partes, tiene dos argumentos EX2 y EX3, que son los dos factores de una expresión inicial EX1 con la forma EX2*EX3. EX2 fue elegida como U y EX3 fue elegida como DV. *INTPROD* se invocó en la definición del módulo de integración para el átomo INTEGRA cuando el operador principal de EX1 fue *. Si *INTPROD* falla significa que la elección de U y DV fue equivocada o que no es posible integrar por partes. *INTPROD* se apoya

medio del sistema el usuario recibe como respuesta la misma expresión de entrada que él dió.

6.5.4 Módulo de codificación de tablas.

En este módulo se encuentran codificadas las tablas matemáticas. El código fuente puede ser consultado en el apéndice H.

Es fácilmente entendible que, entre mayor sea el número de definiciones registradas para el sistema, más poderoso puede ser éste. El apéndice H es bastante grande, si lo comparamos con el código usado en módulos anteriores. Debido a ello, la gama de operaciones que el sistema es capaz de realizar es más o menos grande, pues en él se apoyan todos esos módulos.

En la primera parte del módulo de codificación de tablas se encuentran las definiciones propias de las funciones trigonométricas, primero la definición del SEN y COS de una expresión. A partir de éstas se definen entondes la TAN, COT, SEC y CSC.

De manera similar sigue la definición de las funciones trigonométricas inversas.

A continuación viene la definición de las funciones hiperbólicas, donde el SENH y el COSH se define en términos del número e, base de los logaritmos naturales. Las funciones para TANH, COTH, SECH y CSCH se definen en función de SENH y COSH.

En la segunda parte del apéndice G se encuentra la definición de propiedades de operadores y fórmulas, para la mayoria de las cuales se hizo uso de las listas de propiedades que se pueden asociar a los átomos en el lenguaje LISP. El **cdr** de un átomo apunta a la lista de propiedades del átomo, que contiene tanto propiedades como banderas.

Una lista de propiedades de un átomo, es una lista de puntos pareja. El **car** de cada punto pareja es la llave o nombre de la propiedad, en tanto que el **cdr** es el valor asociado a dicha propiedad. El valor de la propiedad de un átomo puede ser una función, cualidad cual es ampliamente explotada para la codificación de las definiciones matemáticas utilizadas en el sistema.

En LISP estan disponibles las funciones para el manejo de

matemáticas utilizadas en el sistema.

En LISP estan disponibles las funciones para el manejo de propiedades, que facilitan la construcción de bases de datos cuya información puede ser consultada de manera sencilla.

La función *PROPERTY* se realizó con la ayuda de estas funciones. Tiene tres argumentos, el primero es el nombre del átomo, el segundo es el nombre de la propiedad y el tercero el valor de ésta. *PROPERTY* se encarga de registrar esa información.

En primer lugar se encuentran las definiciones de las fórmulas o equivalencias trigonométricas para suma de ángulos y para ángulos múltiplos.

La propiedad + del átomo **NomFun** tiene codificada la fórmula adecuada para encontrar **NomFun**(A+B).

La propiedad * del átomo **NomFun** tiene codificada la fórmula para encontrar **NomFun**(n*A).

Ejemplo:

Codificación de: sen(A+B) = sen A cos B + cos A sen B

```
(PROPERTY 'SEN '+ (LAMBDA (EX1 EX2)
      ( (NEGMULT #TRGEXPD S)
                (+ (* (SEN EX1) (COS EX2)
                   (* (COS EX1) (SEN EX2) )) )))
```

El valor de la variable #TRGEXPD indica si tal conversión debe efectuarse o no.

En seguida se encuentran las definiciones de identidades de combinación de funciones trigonométricas y de funciones hiperbólicas.

La propiedad **Nom-Fun** del átomo * indica la fórmula para encontrar la multiplicación de **Nom-Fun**(A)*Otra-Fun(A).

Ejemplo:

Codificación de: TAN(A) * COS(A) = SEN(A)
TAN(A) * COT(A) = 1

```
(PROPERTY '* 'TAN (LAMBDA (EX1 EX2)
    ((EQUAL (CADR EX1) EX2)
            ((EQ (CAR EX1) 'COS) (SEN EX2))
            ((EQ (CAR EX1) 'COT) 1 ) ) ) )
```

Nom-Fun1(Nom-Fun(a))

A continuación vienen las propiedades **Op** del átomo **Base** y las del átomo **Exponente**. Estas fueron utilizadas en la definición de la exponenciación en el módulo aritmético.

Las propiedades de los logaritmos se codificaron en las propiedades * y ^ del átomo **LOG** y en la propiedad **LOG** de los átomos + y *.

Ahí se encuentran definidas algunas otras propiedades banderas que indican números que siempre son positivos y para identificación del cuadrante en que caen algunas funciones trigonométricas.

La segunda parte del apéndice H contiene la codificación de las tablas y propiedades de derivación e integración.

El átomo que guarda las tablas de derivación es el átomo DIF1. El valor de la propiedad SEN del átomo DIF1 guarda la fórmula de la derivada del seno de una expresión.

Ejemplo:

Codificación de: DERIVADA(sen(U),X) = cos U DERIVADA(U,X)

```
(PROPERTY 'DIF1 'SEN '(LAMBDA (EX1)
                        (* (COS EX1)
                           (DIF1 EX1 INDET))))
```

En la sección anterior se puede ver la definición de la función DIF1.

Se codificaron tablas de derivación que incluyen suma, multiplicación, exponenciación, logaritmos, valores absolutos, derivadas de funciones trigonométricas, trigonométricas inversas e hiperbólicas.

Por último se incluyen en este módulo las definiciones de fórmulas de integración.

Las fórmulas básicas incluidas son propiedades del átomo **INTEGRA**, **INTPWR**, **INTEXPTMS** y **INTMONTMS**.

Para el átomo **INTEGRA** se tiene definición para la suma , multiplicación, exponenciación y algunas funciones trigonométricas de base, SEN, COS, ASEN, ATAN y LOG.

Para el átomo **INTEGRA** se tiene definición para la suma , multiplicación, exponenciación y algunas funciones trigonométricas de base, SEN, COS, ASEN, ATAN y LOG.

El átomo **INTPWR** tiene definiciones para el operador +, LOG, SEN y COS que indican integrar expresiones con las formas (EXP1 + EXP2) ^ n , LOG(EXP1,EXP2) ^ N, SEN(EXP1) ^ N y COS(EXP1) ^ N, respectivamente considerando ciertas restricciones. Por ejemplo para LOG, SEN y COS que N sea negativa.

Las definiciones para los otros dos átomos ya no corresponden a fórmulas muy generales.

6.6. Funcionamiento del Transformador FIPN-FU.

Este último módulo está disponible en el apéndice I. la función principal en él codificada es la función *PRINTEXP* que recibe una expresión en FIP y la escribe en la pantalla en FU.

PRINTEXP recibe un argumento, EXP1, que es la expresión a imprimir en FU y no regresa una lista con la expresión en FU, sino que más bien se distingue por los efectos laterales que provoca, y que consisten en imprimir los caracteres necesarios para formar la expresión correspondiente a EXP1, en FU.

Esta transformación se apoya bastante en la jerarquía definida para los operadores. las funciones definidas en este módulo son auxiliares de *PRINTEXP* y son *PRINTOPER, PRINTPOINT, PRINTFUNC, INTSPC, PRINTLIST, PRINTATOM, PRINTPAR* y *PRINTDELTA.* Cada una de ellas tiene como objetivo imprimir una expresión de un tipo especifico.

Hay definiciones de funciones para el átomo **PRINTEXP** bajo los operadores de +, *, ^, -, LOG, DERIVA e INTEGRA donde se define la impresión de una expresión suma, producto, exponenciación y menos unario, así como los casos especiales para imprimir el llamado a las funciones de Log, Deriva e Integra.

CONCLUSIONES

Para llevar a cabo la construcción del sistema se tomó apoyo de varios tipos de herramientas ya existentes.

El desarrollo del sistema requirió de usar algunos conceptos de varias áreas, entre las principalmente explotadas estan:

Gramáticas y Lenguajes Formales. Específicamente el manejo de una gramática libre de contexto para reconocer el lenguaje de entrada del sistema. Se utilizó como base una gramática para reconocimiento de expresiones aritméticas muy sencilla, que incluye los operadores infijos de suma, resta, multiplicación, división y potenciación algunas veces. Una gramática para ello se puede encontrar casi en cualquier libro acerca del tema. Siguiendo el esquema observado en estas gramáticas sencillas se elaboró la gramática propia para el sistema, con más requerimientos, como operadores prefijos para denotar el llamado a funciones. Operadores infijos como en el caso del operador de asignación requerido por el sistema. Operadores postfijos como los terminadores introducidos. Categorías sintácticas para reconocer listas de argumentos de funciones.

Compiladores. Más precisamente se utilizó el método indicado en [4], para elaborar un reconocedor de descenso recursivo para que el sistema fuera capaz de admitir el lenguaje descrito por la gramática mencionada anteriormente. Para ello también se utilizó el método de eliminación de recursividad izquierda descrito en [6] en una gramática libre de contexto.

Algebra elemental, Cálculo Diferencial e Integral. Básicamente se utilizaron conceptos elementales de la extensa área de las matemáticas. Los operadores y operaciones que se manejan guardan las leyes y propiedades convencionales de conmutatividad y asociatividad. Su jerarquía tampoco fue alterada con respecto a lo usual.

Se usan resultados obtenidos de compendios de tablas y fórmulas matemáticas, de los cuales también existe una gran variedad. Ninguno de estos resultados se presta a alguna duda acerca de su veracidad y

por tanto no fueron objeto de ningún tipo de demostración.

IA, Manipulación simbólica. Se utilizó lo que se refiere a métodos para manipulación de expresiones algebraicas. Para este punto no se encontró la suficiente bibliografía acerca del diseño o de la contrucción de métodos para tales fines. Para ello se revisaron entonces algunos artículos que hablaban acerca de sistemas de manipulación simbólica ya existentes, como MACSYMA, MAPLE, SAINT, REDUCE, Mu-MATH. Se trabajó intensamente con la implementación de la versión 83 de éste último, que junto con la de REDUCE fueron las únicas disponibles para una computadora PC. Eligiéndose la de Mu-MATH por su mayor manejabilidad puesto que es un sistema más propio para PC que REDUCE, pues REDUCE fue hecho originalmente para una máquina más grande.

Mu-MATH está implementado en un lenguaje interpretado llamado Mu-SIMP, que tiene algún parentezco con LISP. Es un intérprete y los códigos en Mu-SIMP de los algoritmos usados para Mu-MATH pueden ser accesados. Se estudiaron y eligieron los métodos que podían servir al sistema en construcción, para codificarlos en LISP. Los algoritmos de Mu-SIMP para realizar operaciones simbólicas, los algoritmos para efectuar sistemáticamente la derivación e integración de una expresión fueron extraídos de Mu-MATH y son los que se decriben en el capítulo 6.

Lenguajes de Programación. Exactamente del lenguaje LISP y su filosofía. La elección del lenguaje LISP para implementar el sistema responde a varias ventajas que presenta respecto a una posible implementación en PROLOG.

- LISP es un lenguaje más propicio para la implementación de aplicaciones interactivas que PROLOG, independientemente de la versión que se utilice. Una aplicación interactiva es más difícil establecerla en términos descriptivos. Ciertamente, la versión de Turbo-PROLOG, que es la versión que más facilidad brinda para implementar interacción da opción de usar varios predicados propios para el manejo de ventanas y elección de opciones; pero no para la comunicación usuario - consola. La versión de Mu-LISP 87 también trae un módulo con funciones

semejantes que se prestó para ser explotado y modificado para la contrucción del sistema.

- Es indispensable codificar algoritmos de empatamiento en LISP, pues no da facilidades para ello. PROLOG, en cambio, permite realizar asociación entre patrones con las estructuras más complejas. Sin embargo el tipo de empatamiento que se requiere en el sistema se simplifica enormemente por la característica infija de la representación de las expresiónes elegida y las expresiones que LISP maneja.

El módulo de "manejo de ventanas" que trae la versión 87 de Mu-LISP es propiamente un manejador de pantallas y de menúes. Este manejador fue adaptado para ser usado por el sistema en desarrollo.

En cuanto a la evaluación general del sistema pueden sugerirse algunas mejoras.

- En primer lugar, en la parte de los transformadores de FU a forma interna y viceversa podría ser muy deseable que la forma de usuario se presentara en dos dimensiones. Con ello podría tenerse un sistema aún más amigable. Se presentarían los cocientes en tres renglones, los exponentes como superíndices, las extracción de raices con el símbolo convencional, en lugar de exponentes fraccionarios.

- En cualquier momento podrían incrementarse las definiciones del módulo de codificación de tablas para aumentar el poder de cálculo del sistema.

- Sería también conveniente aumentar el número de funciones ejecutables para permitir mayor versatilidad al terminador dirigido. Por ejemplo un comando que permitiera resolver una expresión en una variable. Si se tuviera manera de obtener las raices de un polinomio o de algúnas expresiones dentro del sistema, los estudiantes podrían obtener las derivadas y hacer uso de esa facilidad para determinar máximos y mínimos.

- Un módulo de graficación de funciones traería consigo una herramienta más para que se pudiera dar una aplicación más práctica al sistema.

- Para lograr una mejora también con el terminador explicativo pueden buscarse más puntos estratégicos para su registro.

APENDICE A.

Gramática libre de contexto que define el lenguaje de entrada del usuario.

```
<COMANDO>        ::=  <EXPRESION> $
<COMANDO>        ::=  <EXPRESION> ;
<COMANDO>        ::=  <EXPRESION> !
<COMANDO>        ::=  <EXPRESION> &

<EXPRESION>      ::=  <VAR> = <EXPRESION>
<EXPRESION>      ::=  <EXP1>

<EXP1>           ::= <EXP1> + <TERM>
<EXP1>           ::= <EXP1> - <TERM>
<EXP1>           ::= <TERM>

<TERM>           ::= <TERM> * <FACTR>
<TERM>           ::= <TERM> / <FACTR>
<TERM>           ::= <FACTR>

<FACTR>          ::=  <PRIMARY> ^ <FACTR>
<FACTR>          ::=  <PRIMARY>

<PRIMARY>        ::=  <CONSTANTE>
<PRIMARY>        ::=  % P I
<PRIMARY>        ::=  % E
<PRIMARY>        ::=  <VAR>
<PRIMARY>        ::=  <VAR-ESPECIAL>
<PRIMARY>        ::=  <VAR-IMP-ENT>
<PRIMARY>        ::=  <VAR-IMP-SAL>
<PRIMARY>        ::=  - <PRIMARY>
<PRIMARY>        ::=  ( <EXP1> )
<PRIMARY>        ::=  <FUNC> ( <LISTARGS> )

<LISTARGS>       ::=  <EXP1><LARGS>

<LARGS>          ::= ε
<LARGS>          ::= , <LISTARGS>

<FUNC>           ::= <TRASCENDENTAL>
<FUNC>           ::= <SELECTORA>
<FUNC>           ::= <ALGEBRAICA>
<FUNC>           ::= <CALCULO>
<FUNC>           ::= <SUBSTITUCION>

<TRASCENDENTAL>  ::= SEN | COS | TAN | SEC | CSC
                     | COT | ASEN | ACOS | ATAN | ASEC |
                     ACSC | ACOT | SENH | COSH | TANH
                     | SECH | CSCH | COTH
```

```
    ("Ventanas" VENTANAS (
                          ("Dividirventana" "DIVISION DE VENTANA" (
                                   ("Horizontal" . DIVISION-HORIZONTAL)
                                   ("Vertical" . DIVISION-VERTICAL) ))
                          ("Cerrarventana" . CLOSE-WINDOW)
                          ("Siguienteventana" . NEXT-WINDOW)
                          ("Previaventana" . LAST-WINDOW)
                          ("Enviaventana" . SWAP-WINDOW)
                          )
    )
    ("Ayuda" . AYUDA)
    ("Fin" . QUIT-PROGRAM)
  )
)

(DEFUN PERMITE-CERRAR-PATRON ()
; Regresa T si el usuario aprueba abandonar el programa o actualizacion.
  (LIKE? *FLUJO-CAMBIOS*))
  (PROMPT-YN "¿Abandono las correcciones realizadas (S/N)?")
)

(DEFUN LINEA-ESTADO (&KEY(MENSAJE)
; Actualiza el desplegado de la linea de estado de "Modelo"
  (COND ((NULL BUFFERDEC) (SETQ BUFFERDEC ""))
  (STATUS WINDOW)
  (CLEAR SCREEN)
  (WRITE-STRING "Sesion: ")
  (WRITE-STRING (SIXTH (FIRST (WINDOW-STATES))))
  (SET-CURSOR 0 37)
  (WRITE-STRING "Numero de entradas: ")
  (WRITE FILESTO (SECOND (FIRST (WINDOW-STATES))))
  (SET-CURSOR 0 (- 80 (LENGTH BUFFERDEC)))
  (WRITE-STRING BUFFERDEC)
  0
)

;-------------------------------------------------------------------------
;     SALIDA
;-------------------------------------------------------------------------

(DEFUN SALIDA ()
  (CLEAR-WINDOW)
  (CLEAR-SCREEN)
  (SLEEP 0.25)
  (SETQ *FLUJO-CAMBIOS* (LIST-PROCESS))
  (ACTUALIZA-VISIBLES)
)

;-------------------------------------------------------------------------
;     DISPLIEGA-SESION
;-------------------------------------------------------------------------
```



```
(DEFUN DESPLIEGA-SESION (
  LisEnt LisSal Ent Res TEMPORAL)
  (CLEAR-WINDOW)
  (CLEAR-SCREEN)
  (WRITE-STRING "Sesion actual:")
  (WRITE-STRING *NOMBRE-SESION*)
  (TERPRI)
  (setq LisEnt (reverse *Entradas*)
        LisSal (reverse *Salidas*)
        Ent 0
        Res 0)
  (loop
      ((and (null LisEnt)
            (null LisSal)))
      (COND ((NOT (NULL LisEnt))
                  (WRITE-STRING "*")(princ Ent)(WRITE-STRING ": ")
                  (SETQ TEMPORAL (CAR LisEnt))
                  (LOOP
                      ((NOT (MEMBER (CAR TEMPORAL '(0 1 2 3))))
                      (SETQ TEMPORAL (CDR TEMPORAL)))
                  )
                  (PRINT TEMPORAL T)
                  (TERPRI)
                  (SETQ LisEnt (CDR LisEnt)
                        Ent (+ Ent 1))
      )
      (COND ((NOT (NULL LisSal))
                        (WRITE-STRING "R")(princ Res)(WRITE-STRING ": ")
                        (SETQ TEMPORAL (CAR LisSAL))
                        (LOOP
                            ((NOT (MEMBER (CAR TEMPORAL '(0 1 2 3))))
                            (SETQ TEMPORAL (CDR TEMPORAL)))
                        )
                        (PRINT TEMPORAL T)
                        (TERPRI)
                        (SETQ LisSal (cdr LisSal)
                              Res (+ Res 1))
      )
      (EXPERA)
  )
)

;-------------------------------------------------------------------------
;     GUARDA-SESION
;-------------------------------------------------------------------------

(DEFUN GUARDA-SESION (
  BROWSER-SESIONES
  (SETQ BROWSER-SESION *WINDOW-SESION*)
  (LOOP
    (SETQ BROWSER-SESION
      (PROMPT-TEXT-FILE "Nombre para la Sesion en Disco: " BROWSER-SESION))
```

```
        (241 . -117)        (Shift-Keypad 1      Ctrl-End
        (141 . -120)        (Enter               Alt-1
        (205 . -120) ))     (Shift-Enter         Alt-!
))
```



```
;*****************************************************************
;                  PROGRAMA DE MENUS Y PANTALLAS
;*****************************************************************

;Ordena (ASC) la lista de aplicaciones
(SETQ *WINDOW-TYPES* (SORT (ADJOIN "Modelo" *WINDOW-TYPES*) 'STRING<))
(MENU '* "$m)
(SETQ $A(+VER '*WINDOWS)

(DEFUN "Modelo" (COMMAND
                 "ENTRADAS" "SALIDAS" "VARIABLES" "NUMS-CAMBIOS"
                 "NOMBRE-SESION")
  (DEV COMMAND 'CREATE-WINDOW)
    (LIST "Modelo" NIL NIL NIL NIL "SESION.ESM"))
  (DEV COMMAND 'CLOSE-WINDOW)
    (PERMITE-CERRAR-PATRON) )
  (DEV COMMAND 'UPDATE-WINDOW) ; Este encargarse de limpiado de pantalla
    (CURSOR-WINDOW)
    (CLEAR-REGION)
  )
  (LINEA-ESTADO)
  (UNWIND-PROTECT
    (LOOP
      (CURRENT-WINDOW)
      (EXECUTE-OPTION 'OPCIONES *MODELO-OPT)(NOT)
    (UPDATE-STATE
          (*"ENTRADAS" *SALIDAS* *VARIABLES* *NUMS-CAMBIOS* *NOMBRE-SESION*)))
  )
)

(SETQ *MODELO-OPTIONS* '(
   ; Este es el arbol de opciones correspondiente al programa MODELO.4
   ("Lectura" . LECTURA)
   ("Sesiones"  SESIONES (
                          ("Despliega-Sesion" . DESPLIEGA-SESION)
                          ("Guarda-Sesion" . GUARDA-SESION)
                          ("Carga-Sesion" . CARGA-SESION)
                          ("Modifica-Sesion" . MODIFICA-SESION)
                          ("Borra-Sesion" . LIMPIA-SESION)))
   ("Ecuaciones"  FORMULAS_DE (
                          ("Equivalencia" . T-Equivalencia)
                          ("Derivacion" . T-DERIVACION)
                          ("Integracion" . T-INTEGRACION)
                          )
   )
   ("Impresion" . IMPRESION)
   ("Miscelanea" OPTIONS (
          ("Colores" . SET-COLOR)
          ("Monitor" . SET-DISPLAY)
          ("AvisoError" . SET-MUTE) ; Que al elegir mas una opcion,
                                    ; suene la campana
          ("Sistema_DOS" . DOS)))
```

```
        ;Validarlos
        ( SAME
            (EXISTE MEN MENTRADA-INICIAL
                    (ADDT *CURRENT-WINDOW*)) ;Si existe la entrada inicial
            (EXISTE MEN MENTRADA-FINAL
                    (ADDT *CURRENT-WINDOW*)) ;Si existe la entrada final
            (>= MENTRADA-INICIAL MENTRADA-FINAL))
                  (REALIZA-MODIF MENTRADA-INICIAL
                                 MENTRADA-FINAL)
                  (THROW 'SWITCH-SCREENS) )
        (ERROR "BEEP 5) ;Bangio invalido
      )
    ); Ctrl set mode video-mode
)

(DEFUN SET-MSESSION ^(
 (** 3 0
   "Numero de entrada INITIAL. Use <Tab> para dar entrada FINAL"
   "Entrada Inicial" 5)
 (** 2 0
   "Numero de entrada FINAL."
   "Entrada Final" 3)
                )
)

(DEFUN REALIZA-MODIF (MEINICIAL MEFINAL &LST-TEMP MENTRADAS MSALIDAS)
    ; Obtener entrada a enviar
    (SETQ &LST-TEMP (WINDOW-STATE))
    (CURRENT-WINDOW)
    (SETQ MENTRADAS (REVERSE (OBTENER &LST-TEMP))
          MSALIDAS  (REVERSE (TRIM2 &LST-TEMP)))
    (SETQ MEFINAL (ADDT (- MEFINAL MEINICIAL)))
    (LOOP
       ((ZEROP MEFINAL))
       (SETQ MENTRADAS (DELETE-nth MENTRADAS (1+& MEINICIAL)))
       (SETQ MSALIDAS  (DELETE-nth MSALIDAS  (1+& MEINICIAL)))
       (SETQ MEFINAL   (1-& MEFINAL))
    )
    (REPLACE-nth (REVERSE MENTRADAS) &LST-TEMP 1)
    (REPLACE-nth (REVERSE MSALIDAS ) &LST-TEMP 2)
    (WINDOW-STATE *CURRENT-WINDOW* (LIST &LST-TEMP))
)

;-------------------------------------------------------------------------
;     LIMPIA-SESION
;-------------------------------------------------------------------------

(DEFUN LIMPIA-SESION ()
  ( (FORMITE-CERRAR-MATRIZ)
    (SETQ *NUMERO-SESION* *SESION-SSS*)
    (SETQ *NUMERO-CARACTER* NIL)
    (CURRENT-WINDOW)
```



```
    (CLEAR-SCREEN)
    (SETQ *ENTRADAS* ()
          *SALIDAS* ())
    (SETQ *VARIABLES* (RESET-VARS))
    (LINEA-ESTADO)
    0 )
)

(DEFUN RESET-VARS ()
  (LOOP
    ((NULL *VARIABLES*) NIL)
    (SET (CAR *VARIABLES*) (POP *VARIABLES*))
  )
)

;-------------------------------------------------------------------------
;     Y-EQUIVALENCIA
;-------------------------------------------------------------------------

(DEFUN Y-EQUIVALENCIA ()
  (ECHO (NULL *PANT-EQU*) (LIST LISTA '*PANT-EQU*))
  (DISP-PANTALLA (WINDOW-DIMS) *PANT-EQU*)
)

;-------------------------------------------------------------------------
;     Y-1485-DOS
;-------------------------------------------------------------------------

(DEFUN Y-1485-DOS ()
  (ECHO (NULL *PANT-DOS*) (LIST-LISTA '*PANT-DOS*))
  (DISP-PANTALLA (WINDOW-DIMS) *PANT-DOS*)
)

;-------------------------------------------------------------------------
;     Y-1485-INT
;-------------------------------------------------------------------------

(DEFUN Y-1485-INT ()
  (ECHO (NULL *PANT-INT*) (LIST-LISTA '*PANT-INT*))
  (DISP-PANTALLA (WINDOW-DIMS) *PANT-INT*)
)

(DEFUN DISP-PANTALLA (WINDOW-DIMS *PANTALLA*
                      &OPTIONAL
                      &AUX (current-window)
                      (LARGO-CAD
                      (FACTA-PANTALLA 0 (- WINDOW-WIDTH 1) 2)
                      (SETQ X-INI 0)
                      (MAESTRO PROMPT
                               "Usar  Pgdn, Pgup, Flecha hacia arriba o Flecha hacia abajo")
                      (SET-CURSOR X Y)
```

```
    ((EQL (NOMBRE-SESION **) NIL)
    ((AND (SB ;(ES (NOMBRE-SESION *NOMBRE-SESION*)
              (NOT (PROBE-FILE (NOMBRE-SESION))
              (PROMPT-RE
                "Ya existe sesion con ese nombre. Escribo sobre el(s S/N?"))
          (OPEN-OUTPUT-FILE (NOMBRE-SESION))
      (SETQ *NOMBRE-SESION* (NORMALIZA-NOMBRE-ARCHIVO (OUTPUT-FILE) **)
            *OUTPUT-FILE*)
      (LINEA-ESTADO)
      (UNUSED-PROJECT
        (ESCRIBE-ARCH-SESION (CONS *ENTRADAS*
                                   (CONS *SALIDAS*
                                         *VARIABLES*) )))
      (SETQ *HUBO-CAMBIOS*)
      0 )
    (ERROR-BEEP) )
)

(SETQ *DEFAULT-TYPE* *SES*)

(DEFUN ESCRIBE-ARCH-SESION (ARCHIVO)
  *OUTPUT-ECHO*)
  (MUESTRA-PROMPT "Salvando sesion...")
  (SETQ *OUTPUT-FILE* T)
  (CATCH *DISK Full*
     (WRITE (MAPLIS '(LAMBDA (X) . PTOCONA)) ARCHIVO))
     (WRITE-BYTE 26) )
    (CLOSE-OUTPUT-FILE) )
  (FILE-WRITE-POSITION 0)
  (CLOSE-OUTPUT-FILE)
  (MUESTRA-PROMPT "Disco lleno; borre algunos archivos e intente de nuevo")
  (ERROR-BEEP)
  (TONE NIL 2000) )

;-----------------------------------------------------------------------
;     CARGA-SESION
;-----------------------------------------------------------------------

(DEFUN CARGA-SESION (
    &NOMBRE-SESION &OBJ)
  (SETQ &NOMBRE-SESION *NOMBRE-SESION*)
  (LOOP
    (SETQ &NOMBRE-SESION (PROMPT-TEXT-FILE
                           "Nombre de la sesion: " &NOMBRE-SESION))
    ((EQ &NOMBRE-SESION **) NIL)
    ((AND (PROBE-FILE &NOMBRE-SESION)
          (LIMPIA-SESION)
          (OPEN-INPUT-FILE &NOMBRE-SESION))
       (SETQ *NOMBRE-SESION* (NORMALIZA-NOMBRE-ARCHIVO (INPUT-FILE) **))
       (LINEA-ESTADO)
       ;Asignar *entradas* y *salidas* a lo que lee
       (SETQ &OBJ (LEE-ARCHIVO-SESION))
```



```
          (SETQ *ENTRADAS* (CAR &OBJ)
                *SALIDAS*  (CADR &OBJ)
                *VARIABLES* (CDDR &OBJ))
          (ASIGNA-VALORES (REVERSE *ENTRADAS*))
       0 )
    (ERROR-BEEP)
  )
)

(DEFUN REVISA-ASIGNACION (&REXP &VALOR)
  ((NOT (EQ (CAR &REXP) 'SETQ)) (LOOP
                                  ((NULL &VARS-AFECTADAS))
                                  (SET (POP &VARS-AFECTADAS) &EXP)
                                ))
  (SETQ &VARS-AFECTADAS (APPEND () LIST (CADR &REXP)) &VARS-AFECTADAS))
  (REVISA-ASIGNACION (CADDR &REXP))
)

(DEFUN ASIGNA-VALORES (&LIST &VARS-AFECTADAS)
  (LOOP
     ((NULL &LIST))
     (SETQ &VARS-AFECTADAS NIL)
     (REVISA-ASIGNACION (CADR (POP &LIST)))
  )
)

(DEFUN LEE-ARCHIVO-SESION (
    LST *TAB-EXPAND* *INPUT-ECHO*)
  (MUESTRA-PROMPT "Leyendo sesion...")
  (SETQ *TAB-EXPAND* 8)
  (LOOP
    ((NOT (LISTEN)))
    (PUSH (MAPLIS '(LAMBDA (X) X)
                  (READ-FROM-STRING (READ-LINE))) LST) )
  (CLOSE-INPUT-FILE)
  (CAR LST)
)

;-----------------------------------------------------------------------
;     MODIFICA-SESION
;-----------------------------------------------------------------------

(DEFUN MODIFICA-SESION (
   &ENTRADA-INICIAL &ENTRADA-FINAL &MSG)
   ((SETQ MODE (VIDEO-MODE))
     ((EQ MODE 255) NIL)
     (REPLACE (FIRST SET-MODESESION) 1)
     (REPLACE (SECOND SET-MODESESION) 1)
     (LOOP
       ((NOT (MODE-SELECT "Ranges-a-Eliminar SET-MODESESION) NIL)
       (SETQ &ENTRADA-INICIAL (PARSE-INTEGER (CAR  (FIRST  SET-MODESESION))))
       (SETQ &ENTRADA-FINAL   (PARSE-INTEGER (CAR  (SECOND SET-MODESESION))))
                              *
```

```
(CURRENT-WINDOW)
(LOOP
   (SETQ DF (LEE-TECLA-DESEADA))
   (ERR DF 0))
   (COND ((ERR DF 1) (LINEA-ARRIBA))
         ((ERR DF 2) (PAGINA-ARRIBA))
         ((ERR DF 3) (LINEA-ABAJO))
         ((ERR DF 4) (PAGINA-ABAJO))
   )
)
(CURSOR-OFF)
)

(DEFUN ESCRIBE-LINEA (LST
                      CONT MAXIMO)
  (SETQ CONT 2)
  (LOOP
    (NULL LST))
    (PRINC (ASCII) (POP LST)))
    (INCR CONT)
  )
  (SETQ MAXIMO (APPLY 'MAX (MAPCAR 'LENGTH *PANTALLA*)))
  (LOOP
    ((ER CONT MAXIMO))
    (PRINC #\ESPACIO)
    (INCR CONT)
  )
)

(DEFUN PINTA-PANTALLA (S-I S-F ABASE
                       CONT)
  (SETQ CONT ABASE)
  (LOOP
     ((ER (- CONT ABASE) (+ (- S-F S-I) 1)))
     (SET-CURSOR CONT 0)
     (ESCRIBE-LINEA (NTH (+ (- CONT ABASE) S-I) *PANTALLA*))
     (INCR CONT)
  )
)

(DEFUN LINEA-ARRIBA()
  ((NOT (NULL (SET-CURSOR (- (ROW) 1 ) 0)))
  ((= 0 (NI @@
       (SET-CURSOR 0 0)
       (INSERT-LINES 1)
       (SET-CURSOR 0 0)
       (SETQ N-INI ( N-INI 1))
       (ESCRIBE-LINEA (NTH N-INI *PANTALLA*))
       (SET-CURSOR (ROW) 0) )
   (WRITE-BYTE 7)
)
```



```
(DEFUN PAGINA-ARRIBA()
 ((NOT (NULL (SET-CURSOR (- (ROW) 1) 0)))
 ((= ) 0-INI (- WINDOW-ROWS 2)) 0)
       (SET-CURSOR 0 0)
       ;Numero de renglones que alguen igual son 2
       (INSERT-LINES (- WINDOW-ROWS 2))
       (SET-CURSOR 0 0)
       (SETQ N-INI (- N-INI (- WINDOW-ROWS 2)))
       (PINTA-PANTALLA N-INI
        (- (+ N-INI (- WINDOW-ROWS 2)) 1) 0) )
 (SETQ N-INI 0)
 (PINTA-PANTALLA 0 (- WINDOW-ROWS 1) 0)
 (SET-CURSOR (- WINDOW-ROWS 2) 0)
)

(DEFUN LINEA-ABAJO()
  ((NOT (NULL (SET-CURSOR (+ (ROW) 1) 0))))
  ( (= (+ N-INI WINDOW-ROWS)
       (LENGTH *PANTALLA*))
       (SET-CURSOR 0 0)
       (DELETE-LINES 1)
       (SET-CURSOR (- WINDOW-ROWS 1) 0)
       (SETQ N-INI (+ N-INI 1))
       (ESCRIBE-LINEA (NTH (- (+ N-INI WINDOW-ROWS) 1) *PANTALLA*))
       (SET-CURSOR (ROW) 0) )
  (WRITE-BYTE 7)
)

(DEFUN PAGINA-ABAJO()
  ((NOT (NULL (SET-CURSOR (+ (ROW) (- WINDOW-ROWS 2)) 0))))
  (SET-CURSOR 0 0)
  (DELETE-LINES (- WINDOW-ROWS 2))
  (SET-CURSOR (- WINDOW-ROWS (- WINDOW-ROWS 2)) 0)
  (SETQ N-INI (+ N-INI (- WINDOW-ROWS 2)))
  (PINTA-PANTALLA (+ N-INI 2) (- (+ N-INI WINDOW-ROWS) 1) 2)
)

(DEFUN LEE-TECLA-DESEADA (TECLA)
 (LOOP
    (SETQ TECLA (TECLA-F (READ-BYTE)))
    ((AND (NOT **UNCION*)
          (EQ TECLA 27))       0)     ) Esc
    ((AND *FUNCION* (EQ TECLA 72)) 1) ; Flecha para arriba
    ((AND *FUNCION* (EQ TECLA 73)) 2) ; PgUp
    ((AND *FUNCION* (EQ TECLA 80)) 3) ; Flecha para abajo
    ((AND *FUNCION* (EQ TECLA 81)) 4) ; PgDown

    (WRITE-BYTE 7)
 )
)
```



```
(DEFUN TECLA-F (BYTE TEMP)
  ((EQ BYTE 255)
       (SETQ TEMP (READ-BYTE))
       (SETQ **UNCION* T)
       TEMP)
  (SETQ *FUNCION* NIL)
  BYTE
)

(DEFUN LEE-LISTA (ENCHLST ELST ELINEA #C)
  (COND ( (EQ #NCHLST '*PANT-EDO*)
              (COND ( (OPEN-INPUT-FILE "C:\\VRLISP\\PROGRAMA\\EQUALDAD.LSP) )
                    ( (OPEN-INPUT-FILE "A:\\EQUALDAD.LSP) ))
        )
        ( (EQ #NCHLST '*PANT-DZD*)
              (COND ( (OPEN-INPUT-FILE "C:\\VRLISP\\PROGRAMA\\DERIVADA.LSP) )
                    ( (OPEN-INPUT-FILE "A:\\DERIVADA.LSP) ))
        )
        ( (EQ #NCHLST '*PANT-INT*)
              (COND ( (OPEN-INPUT-FILE "C:\\VRLISP\\PROGRAMA\\INTEGRAL.LSP) )
                    ( (OPEN-INPUT-FILE "A:\\INTEGRAL.LSP) ))
        )
  )
  (LOOP
     ((NOT (LISTEN)))
     (SETQ #C (READ-BYTE))
     (COND ((EQ #C 13)
                (COND ((NULL #LINEA)
                            (SETQ #LINEA '(32)))
                (PUSH (REVERSE #LINEA) #LST)
                (READ-BYTE) ; Eliminar el LineFeed
                (SETQ #LINEA ()))
           ((PUSH #C #LINEA)))
  )
  (SET #NCHLST (REVERSE #LST))
  (CLOSE-INPUT-FILE)
)

(LOAD "A:\\PROGRAMA\\EDICION.LSP")

;-----------------------------------------------------------------
;     IMPRESION
;-----------------------------------------------------------------

(DEFUN IMPRESION()
  (CURRENT-WINDOW)
  (CLEAR SCREEN)
)

;-----------------------------------------------------------------
;     SET-COLOR
;-----------------------------------------------------------------
```



```
(DEFUN SET-COLOR (
   #CNT #UM2 #UM3 #UM4 #UM5 #UM6 #UM7 #UM8)
 ((SETQ MODE (VIDEO-MODE))
   ((EQ MODE 255) NIL)
   (MAPC 'REPLACA '(SET-COLOR *WINDOW-COLORS*)
   (REPLACA (REVERSE SET-COLOR) (BORDER-COLOR))
   (REPLACA (CDDR SET-COLOR) (PALETTE-COLOR))
   (LOOP
      ((NOT (MODE-QUERY 'COLORES SET-COLOR)) NIL)
      (SETQ #UM1 (PAPLE-INTEGER (CAR (FIRST SET-COLOR)))
            #UM2 (PAPLE-INTEGER (CAR (SECOND SET-COLOR)))
            #UM3 (PAPLE-INTEGER (CAR (THIRD SET-COLOR)))
            #UM4 (PAPLE-INTEGER (CAR (FOURTH SET-COLOR)))
            #UM5 (PAPLE-INTEGER (CAR (FIFTH SET-COLOR)))
            #UM6 (PAPLE-INTEGER (CAR (SIXTH SET-COLOR)))
            #UM7 (PAPLE-INTEGER (CAR (SEVENTH SET-COLOR)))
            #UM8 (PAPLE-INTEGER (CAR (EIGHTH SET-COLOR))) )
      ((AND (COLORP #UM1) (COLORP #UM2) (COLORP #UM3) (COLORP #UM4)
            (COLORP #UM5) (COLORP #UM6) (COLORP #UM7) (COLORP #UM8))
       (SETQ *WINDOW-COLORS* (LIST #UM1 #UM2 #UM3 #UM4 #UM5 #UM6))
       (BORDER-COLOR #UM7)
       (PALETTE-COLOR #UM8)
       (THROW 'SWITCH-SCREEN) )
      (ERROR-BEEP 4) ) ) )

(SETQ SET-COLOR '(
    (*** 0 0 "Introducir codigo de color (0-15) (USAR TABS)" "Encuadre" 3)
    (*** 0 0 "Introducir codigo de color (0-15) (USAR TABS)" "Texto" 3)
    (*** 0 0 "Introducir codigo de color (0-15) (USAR TABS)" "Opciones" 3)
    (*** 0 0 "Introducir codigo de color (0-15) (USAR TABS)" "Mensajes" 3)
    (*** 0 0 "Introducir codigo de color (0-15) (USAR TABS)" "Estado" 3)
    (*** 0 0 "Introducir codigo de color (0-15) (USAR TABS)" "Fondo" 3)
    (*** 0 0 "Introducir codigo de color (0-15) (USAR TABS)" "Borde/Color" 3)
    (*** 0 0 "Introducir codigo de color (0-15) (USAR TABS)" "Paleta" 3) ))

(DEFUN COLORP (NUM)
  (AND NUM (>= 0 NUM) (<= 0 NUM 255) ))

;-----------------------------------------------------------------
;     SET-DISPLAY
;-----------------------------------------------------------------

(DEFUN SET-DISPLAY (
   MODE NUM)
 ((SETQ MODE (VIDEO-MODE))
   ((= MODE 255) NIL)
   (REPLACA (FIRST SET-DISPLAY)
            (= (= MODE 0)  "CGA"
            (= (= MODE 7)  "MDA"
            (= (= MODE 10) "PC1+"
```

```
        (IF (== MODE 2D) "EGA" "AT&T")))))
  (REPLACA (SECOND SET-DISPLAY)
        (IF (GRAPHICS-MODE-P MODE) "Grafico" "Texto") )
  (REPLACA (THIRD SET-DISPLAY)
        (IF (OR (MEDIUM-RESOLUTION-P MODE) (== MODE 07)) "Media" "Alta") )
  (MODE-QUERY 'DISPLAY SET-DISPLAY)
  ( (EQ (FIRST (SECOND SET-DISPLAY)) "Texto")
      (SETQ NUM (IF (EQ (FIRST (FIRST SET-DISPLAY)) "MDA")
                    7
                    (IF (EQ (FIRST (THIRD SET-DISPLAY)) "Alta") 3 1)) )
    ((EQ (FIRST (FIRST SET-DISPLAY)) "CGA")
      (SETQ NUM (IF (EQ (FIRST (THIRD SET-DISPLAY)) "Alta") 6 4)) )
    (EQ (FIRST (FIRST SET-DISPLAY)) "PCjr")
      (SETQ NUM (IF (EQ (FIRST (THIRD SET-DISPLAY)) "Alta") 10 09) )
    (EQ (FIRST (FIRST SET-DISPLAY)) "AT&T")
      (SETQ NUM (IF (EQ (FIRST (THIRD SET-DISPLAY)) "Alta") 0A 09)) )
    (REPLACA (CAR EGA-DISPLAY)
          (IF (MEMBER MODE '(7 15))
              "Monocromatico"
              (IF (OR MODE 16) "Enhanced" "Color")))
    ((MODE-QUERY "DISPLAY MONITOR" EGA-DISPLAY)
      ((EQ (CAAR EGA-DISPLAY) "Monocromatico")
        (SETQ NUM 15) )
      ((EQ (FIRST (THIRD SET-DISPLAY)) "Alta")
        (SETQ (CAAR EGA-DISPLAY) "Enhanced")
          (SETQ NUM 10) )
        (SETQ NUM 14) )
      (SETQ NUM 13) )
    (RETURN) )
  (VIDEO-MODE NUM)
  (CURSOR-OFF)
  ( ((MEDIUM-RESOLUTION-P MODE)
      (MEDIUM-RESOLUTION-P))
      (REPLACA (CDR *FULL-WINDOW*) (* 2 (CADR *FULL-WINDOW*)))
      (REPLACA (CDDDR *FULL-WINDOW*) (* 2 (CADDDR *FULL-WINDOW*)))
      (SETQ NUM (LENGTH *WINDOWS*))
      (LOOP
        (ZEROP NUM))
        (DECQ NUM)
        (WINDOW-COL NUM (SUBT (* 2 (WINDOW-COL NUM))))
        (WINDOW-COLS NUM (ADD1 (* 2 (WINDOW-COLS NUM))))
        ( ((OR (> (WINDOW-COL NUM) (WINDOW-COLS NUM))
               (> (CADDDR *FULL-WINDOW*) 2))
            (WINDOW-COLS NUM (ADD1 (WINDOW-COLS NUM))) ) ) ) )
    ((MEDIUM-RESOLUTION-P)
      (REPLACA (CDR *FULL-WINDOW*) (TRUNCATE (CADR *FULL-WINDOW*) 2))
      (REPLACA (CDDDR *FULL-WINDOW*) (1-(1+(CADDDR *FULL-WINDOW*) 2)))
      (SETQ NUM (LENGTH *WINDOWS*))
      (LOOP
        ((ZEROP NUM))
        (DECQ NUM)
        (WINDOW-COL NUM (1-(1+(WINDOW-COL NUM) 2))
```



```
        (WINDOW-COLS NUM (TRUNCATE (SUBT (WINDOW-COLS NUM)) 2)) ) ) )
    (THROW *SWITCH-SCREENS* ) ) )

(SETQ SET-DISPLAY '(
 (** 0 0 "Selecciona adaptador para el monitor"
                "Adaptador"  ("MDA" "CGA" "EGA" "AT&T" "PCjr"))
 (** 0 1 "Selecciona modo de pantalla"
                "Modo"       ("Texto" "Grafico"))
 (** 0 6 "Selecciona resolucion de pantalla"
                "Resolucion" ("Media" "Alta")) ))

(SETQ EGA-DISPLAY '(
  (** 0 0 "Selecciona tipo de monitor"
                "Monitor"    ("Color" "Enhanced" "Monocromatico")) ))

;.....
;     SET-MUTE
;.....

(DEFUN SET-MUTE ()
  (MODE-QUERY *SCREEN* SET-MUTE) )

(SETQ SET-MUTE '((*Mut 0 0 "Desactiva la campana al elegir mal una opcion"
                "Desactivada" ("Si" "No"))))

;.....
;     GO-DOS
;.....

(DEFUN GO-DOS (
    (COMMAND)
  (SETQ COMMAND (PROMPT (INPUT "(((/rebut// un Comando de MS-DOS"
            (PACK* (DEFAULT-DRIVE) "\\\" (DEFAULT-PATH) "> ") "")))
  ((EQ *LINE-TERMINATOR* 27))
  (STATUS-WINDOW)
  (CLEAR-SCREEN)
  ((EQ COMMAND "")
    (MUESTRA-PROMPT "Pulse Exit para continuar")
    (CURSOR-ON)
    (MAPC 'WRITE-CONSOLE '(RESET-SCREEN))          ; Reset screen
    (EXECUTE (GETSET 'CONSPEC))
    (MAPC 'WRITE-CONSOLE '(INIT-SCREEN))           ; Initialize screen
    (CURSOR-OFF)
    (THROW 'SWITCH-SCREENS) )
  (MUESTRA-PROMPT "")
  (MAKE-WINDOW NIL)
  (SET-CURSOR 0,- (THIRD (MAKE-WINDOW)) 3) 2)
  (CURSOR-ON)
  (MAPC 'WRITE-CONSOLE '(RESET-SCREEN))          ; Reset screen
  (EXECUTE (GETSET 'CONSPEC) (PACK* "/C " COMMAND))
  (MAPC 'WRITE-CONSOLE '(INIT-SCREEN))           ; Initialize screen
  (CURSOR-OFF)
                                  .
```



```
  (TERPRI) 2)
  (SETQ *CURRENT-AREA*)
  (CONTINUE-PROMPT)
  (THROW 'SWITCH-SCREENS) )

;.....
;     DIVISION-HORIZONTAL
;.....

(DEFUN DIVISION-HORIZONTAL (
    ROW)
  (REPLACA (CAR DIVISION-HORIZONTAL) (1-(1+(WINDOW-ROWS) 2)))
  (LOOP
    ((NOT (MODE-QUERY "DIVIDIR VENTANA HORIZONTAL" DIVISION-HORIZONTAL)) NIL)
    (SETQ ROW (PARSE-INTEGER (CAAR DIVISION-HORIZONTAL)))
    ((AND ROW (== 3 ROW ) (WINDOW-ROWS) 2))
      (SETQ *NEW-WINDOW* (ADD1 *CURRENT-WINDOW*))
      (APLACA (WINDOW *CURRENT-WINDOW* *WINDOWS*)
              ((ONE) (COPY-LIST (WIN *CURRENT-WINDOW* *WINDOWS*))
                     (WINDOW (ADD1 *CURRENT-WINDOW*) *WINDOWS*)) )
      (WINDOW-ROW *CURRENT-WINDOW* (SUBT ROW))
      (WINDOW-ROW (ADD1 *CURRENT-WINDOW*)
                  (- (WINDOW-ROW (ADD1 *CURRENT-WINDOW*)) ROW))
      (WINDOW-ROWS (ADD1 *CURRENT-WINDOW*)
                   (- (WINDOW-ROWS (ADD1 *CURRENT-WINDOW*)) ROW))
      (THROW 'DIVISION-WINDOW) )
    ((ERROR-BEEP 5) ) );(La ventana ya no cabe

(SETQ DIVISION-HORIZONTAL
      '((*** 0 0 "Introducir numero de linea" "No Linea" 3)))

;.....
;     DIVISION-VERTICAL
;.....

(DEFUN DIVISION-VERTICAL (
    COL)
  (REPLACA (CAR DIVISION-VERTICAL) (1-(1+(WINDOW-COLS) 2)))
  (LOOP
    ((NOT (MODE-QUERY "DIVIDIR VENTANA VERTICAL" DIVISION-VERTICAL)) NIL)
    (SETQ COL (PARSE-INTEGER (CAAR DIVISION-VERTICAL)))
    ((AND COL (== 7 COL ) (WINDOW-COLS) 0))
      ( ((MEDIUM-RESOLUTION-P))
        ((EVEN? COL))
        ((1+(2 COL) ))
      (SETQ *NEW-WINDOW* (ADD1 *CURRENT-WINDOW*))
      (APLACA (WINDOW *CURRENT-WINDOW* *WINDOWS*)
              ((ONE) (COPY-LIST (WIN *CURRENT-WINDOW* *WINDOWS*))
                     (WINDOW (ADD1 *CURRENT-WINDOW*) *WINDOWS*)))
      (WINDOW-COLS *CURRENT-WINDOW* (SUBT COL))
      (WINDOW-COL (ADD1 *CURRENT-WINDOW*)
                  (- (WINDOW-COL (ADD1 *CURRENT-WINDOW*)) COL))
```



```
      (WINDOW-COLS (ADD1 *CURRENT-WINDOW*)
                   (- (WINDOW-COLS (ADD1 *CURRENT-WINDOW*)) COL))
      (THROW 'DIVISION-WINDOW) )
    ((ERROR-BEEP 5) ) );(La ventana ya no cabe

(SETQ DIVISION-VERTICAL
      '((*** 0 0 "Introducir numero de columna" "No columna" 3)))

;.....
;     CLOSE-WINDOW
;.....

(DEFUN CLOSE-WINDOW (
    WINDOW)
  ((OR (EQ *WINDOWS* (CDR (WINDOW-STATES)))
    (REPLACA (CAR CLOSE-WINDOW) (ADD1 *CURRENT-WINDOW*))
    (LOOP
      ((NOT (MODE-QUERY "CERRAR VENTANA" CLOSE-WINDOW)) NIL)
      (SETQ WINDOW (PARSE-INTEGER (CAAR CLOSE-WINDOW)))
      ((AND WINDOW (== 1 WINDOW (LENGTH *WINDOWS*))))
          (THROW 'CLOSE-WINDOW (SUBT WINDOW)) )
      ((ERROR-BEEP 8) ) ) );( La ventana no existe

(SETQ CLOSE-WINDOW '((*** 0 0 "Que numero de ventana" "ventana" 2)))

;.....
;     NEXT-WINDOW
;.....

(DEFUN NEXT-WINDOW ()
  (THROW 'SWITCH-WINDOW (MODULO (ADD1 *CURRENT-WINDOW*) (LENGTH *WINDOWS*))) )

;.....
;     LAST-WINDOW
;.....

(DEFUN LAST-WINDOW ()
  (THROW 'SWITCH-WINDOW (MODULO (SUBT *CURRENT-WINDOW*) (LENGTH *WINDOWS*))) )

;.....
;     EXCHANGE-WINDOW
;.....

(DEFUN EXCHANGE-WINDOW (
  WIDE ENLARGED ENVELOPED) ENCAPSULATED WRAP()
  ((SETQ MODE (VIDEO MODE))
      ((EQ MODE 255) NIL)
      (REPLACA (FIRST SET-DISPLAY) "Enhanced")
      (REPLACA (SECOND SET-DISPLAY) 1)
      (REPLACA (THIRD SET-DISPLAY) 1)
      (REPLACA (FOURTH SET-DISPLAY) 2)
      (LOOP
```

```
        (UNOT (MODE-QUERY "TRANSFERIR-INFORMACION DET-ENVIA)) NIL)
        (SETQ #MODO                    (CAR  (FIRST  DET-ENVIA)))
        (SETQ #NUMERO      (PARSE-INTEGER (CAR  (SECOND DET-ENVIA)))
        (SETQ #VENTFUENTE  (PARSE-INTEGER (CAR  (THIRD  DET-ENVIA)))
        (SETQ #VENTDESTINO (PARSE-INTEGER (CAR  (FOURTH DET-ENVIA)))
        ;Validacion
        ( (AND (SETQ #VAL 7)
               (EXISTE-VEN #VENTFUENTE) ;Si EXISTE VENTANA FUENTE
               (SETQ #VAL 8)
               (EXISTE-VEN #VENTDESTINO) ;Y EXISTE VENTANA DESTINO
                                         ;Y ES <> QUE VENT FTE
               (SETQ #VAL 9)
               (EXISTE-NUM #NUMERO #VENTFUENTE) ;Y SI EXISTE EL NUM DE REN.
                                         ;EN LA VENTANA DESEADA
                        (REALIZA-ENVIO #MODO #NUMERO
                                       #VENTFUENTE #VENTDESTINO)
                        (THROW 'SWITCH-SCREEN))
        (ERROR-MSG #VAL))
      )
   ); Del ext mode video-mode
)

(DEFUN SET-ENVIA ()
  (** 0 0
    "Ejecutar para elegir Entrada o Salida a Enviar. Use Tab para ir a Numero"
    "Enviar" ("Entrada" "Salida"))
  (** 3 0
    "Numero de entrada o salida deseado. Use <Tab> para dar ventana fuente"
    "Numero" 3)
  (** 0 0
    "Ventana que envia. Use <Tab> para ir a ventana destino"
    "Ventana fuente" 3)
  (** 0 0
   "Ventana que recibe. Use <Tab> para ir a otro campo y <enter> para ejecutar"
   "Ventana destino" 3)
                )
)

(DEFUN EXISTE-NUM (NN #VF)
   (<= 1 NN (LENGTH
                (CADR (WINDOW-STATE (SUBIT #VF))) ;El cristal activo nada es ?
                      ))
)

(DEFUN EXISTE-VEN (#VF)
   (<= 1 #VF (LENGTH *WINDOWS*))
)

(DEFUN EXISTE-VEC (#VE #VF)
   (AND (<= 1 #VE (LENGTH *WINDOWS*))
        (NOT (= #VF #VE))
   )
```



```
)

(DEFUN REALIZA-ENVIO (#MODE #N #VF #VE
                      #TEMP)
   ; Obtener entrada a enviar
   (COND ( (EQ #MODE "Entrada")
               (SETQ #VF
                     (SETF (SUBIT #N)
                           (REVERSE (SECOND (WINDOW-STATE (SUBIT #VF)))))) )
         ( (SETQ #VF
                 (SETF (SUBIT #N)
                       (REVERSE (THIRD (WINDOW-STATE (SUBIT #VF))))))) )
   ; Temp es info especifica completa del destino
   (SETQ #TEMP (WINDOW-STATE (SUBIT #VE)))
   (COND ( (EQ #MODE "Salida")
           (REPLACE-NTH (COND (#VF (SECOND (WINDOW-STATE (SUBIT #VF)))))(car
                        #TEMP 1) ; Lista, Posicion de las entradas
           ; Ahora para separar las salidas
           (REPLACE-NTH (COND (ENDL (cadr #VF))   ; Item
                              (THIRD (WINDOW-STATE (SUBIT #VF))))
                        #TEMP 2  ) ) ; Lista, Posicion de las salidas

         ( (REPLACE-NTH (COND (LIST \(
                                    #VF) (SECOND (WINDOW-STATE (SUBIT #VE))))
                        )
                        #TEMP 1) ; Lista, Posicion de las entradas
           ; Ahora para separar las salidas
           (REPLACE-NTH  (COND (#VF (THIRD (WINDOW-STATE (SUBIT #VF))))) ; Item
                         #TEMP 2)
           )
   )
   ; Cambio el cristal 0, que es el unico, por el nuevo cristal #TEMP
   ; en la ventana destino #VE
   (WINDOW-STATE (SUBIT #VE) #TEMP)
)


;-----------------------------------------------------------------------
;     AYUDA
;-----------------------------------------------------------------------

(DEFUN AYUDA ()
  (Desplegar Menu de Opciones a explicar
  (Cuando den enter, borrar pantalla y dar explicacion de la elegida
  (CURRENT-WINDOW)
  (CLEAR-SCREEN)
  (LINEA-ESTADO "AYUDA")
  (EJECUTE-OPTION AYUDA-DE *OPCIONES-AYUDA*)
  (LINEA-ESTADO)
)

(SETQ *OPCIONES-AYUDA* '(                    +
```



```
("General-Idades" . EXPLICA-MENU-GENERAL-IDADES)
("Funciones" "Tipos de Funciones" (
   ("Trascendentales" "Funciones Trascendentales" (
         ("Sen" . EXPLICA-TRASC-SEN)          ; TRASCEN.AYD
         ("Cos" . EXPLICA-TRASC-COS)
         ("Tan" . EXPLICA-TRASC-TAN)
         ("Cot" . EXPLICA-TRASC-COT)
         ("Sec" . EXPLICA-TRASC-SEC)
         ("Csc" . EXPLICA-TRASC-CSC)
         ("ASen" . EXPLICA-TRASC-ASEN)
         ("ACos" . EXPLICA-TRASC-ACOS)
         ("ATan" . EXPLICA-TRASC-ATAN)
         ("ACot" . EXPLICA-TRASC-ACOT)
         ("ASec" . EXPLICA-TRASC-ASEC)
         ("ACsc" . EXPLICA-TRASC-ACSC)
         ("Senh" . EXPLICA-TRASC-SENH)
         ("Cosh" . EXPLICA-TRASC-COSH)
         ("Tanh" . EXPLICA-TRASC-TANH)
         ("Coth" . EXPLICA-TRASC-COTH)
         ("Sech" . EXPLICA-TRASC-SECH)
         ("Csch" . EXPLICA-TRASC-CSCH)
         ("Log" . EXPLICA-LOGARITMICA-LOG )
         ("Ln" . EXPLICA-LOGARITMICA-LN )
                  )
   )
   ("Selectores" "Funciones Selectoras" (
         ("Numerador" . EXPLICA-SEL-NUMERADOR)        ; SELECTOR.AYD
         ("Denominador" . EXPLICA-SEL-DENOMINADOR)
         ("Coeficiente" . EXPLICA-SEL-COEFICIENTE)
         ("Base" . EXPLICA-SEL-BASE)
         ("Exponente" . EXPLICA-SEL-EXPONENTE)
         ("SubExp" . EXPLICA-SEL-SUBEXP)
                  )
   )
   ("Algebraicas" "Funciones Algebraicas" (
         ("Cuadrado" . EXPLICA-ALG-CUADRADO)          ; ALGEBRA.AYD
         ("Cuarta" . EXPLICA-ALG-CUARTA)
         ("Expo" . EXPLICA-ALG-EXPO)
         ("Factorial" . EXPLICA-ALG-FACTORIAL)
         ("Racionalizar" . EXPLICA-ALG-RACIONALIZA)
                  )
   )
   ("De_Calculo" "Funciones de Calculo" (
         ("Diferencia" . EXPLICA-CAL-DIFERENCIA)      ; CALCULO.AYD
         ("MiDiferencia" . EXPLICA-CAL-MIDIFERENCIA)
         ("Derivada" . EXPLICA-CAL-DERIVADA)
         ("Integral" . EXPLICA-CAL-INTEGRAL)
                  )
   )
   ("Ejecutables" "Funciones Ejecutables" (
         ("Aplica" . EXPLICA-EJE-APLICA)              ; CALCULO.AYD
         ("SubExp" . EXPLICA-EJE-SUBEXP)
```



```
                  )
   )
("Opciones del Menu" "Ayuda para la Opcion del" (
   ("Lectura" . EXPLICA-MENU-LECTURA)                 ; MENU.AYD
   ("Sesiones" "Opciones de Sesiones" (
         ("Despliega-Sesion" . EXPLICA-DESPLIEGA-SESION) ; SESION.AYD
         ("Guarda-Sesion" . EXPLICA-GUARDA-SESION)
         ("Carga-Sesion" . EXPLICA-CARGA-SESION)
         ("Modifica-Sesion" . EXPLICA-MODIFICA-SESION)
         ("Borra-Sesion" . EXPLICA-BORRA-SESION)))
   ("Ejemplos" . EXPLICA-MENU-EJEMPLOS)               ; MENU.AYD
   ("Impresion" . EXPLICA-MENU-IMPRESION)
   ("Miscelanea" "Opciones de Miscelanea" (
         ("Colores" . EXPLICA-MENU-COLORES)
         ("Monitor" . EXPLICA-MENU-MONITOR)
         ("ReiseError" . EXPLICA-MENU-ANTSACKROM)
         ("Sistema_DOS" . EXPLICA-MENU-SISTEMA-DOS)))
   ("Ventanas" "Opciones de Ventanas" (
         ("Divide Ventana" "DIVISION DE VENTANA" (
           ("Horizontal" . EXPLICA-DIVISION-HORIZONTAL)   ; VENTANA.AYD
           ("Vertical" . EXPLICA-DIVISION-VERTICAL) ))
         ("CerrarVentana" . EXPLICA-CERRAR-VENTANA)
         ("SiguienteVentana" . EXPLICA-SIGUIENTE-VENTANA)
         ("PrevVentana" . EXPLICA-PREVIA-VENTANA)
         ("EnviaVentana" . EXPLICA-ENVIA-VENTANA)
                  )
   )
   ("Ayuda" . EXPLICA-MENU-AYUDA)                     ; MENU.AYD
   ("Fin" . EXPLICA-MENU-FIN)
))
)
)

(defun BORRA_NN (NUM N #) ; Borra desde la posicion # de #str N chars
 (SETQ #STR (UNPACK #STR))
 (LOOP
   (LOOPIF (= (PACK #STR))
   (SETQ #STR (DELETE-NTH #STR #))
   (SETQ N (SUBIT N))
 )
)

(DEFUN INSERTA (#STRING1 #POS #STRING2)
 (SETQ #STRING1 (UNPACK #STRING1))
 (SETQ #STRING2 (UNPACK #STRING2))
 ( (> #POS (LENGTH #STRING1)) (PACK #STRING1))
 (LOOP
   (NULL #STRING2) (PACK #STRING1))
   (SETQ #STRING1 (INSERT-NTH (POP #STRING2) #STRING1 #POS))
   (SETQ #POS (ADD1 #POS))
 )
```

```
)

(DEFUN SUBSTITUYEN (ENTRING VIEJO NUEVO &POS)
  (loop ( SHOT (SETQ &POS (FINDSTRING VIEJO ENTRING)) ) ENTRING )
        (SETQ ENTRING (CONCAT SUB ENTRING NUEVO (LENGTH VIEJO)))
        (setq &string (INSERTA ENTRING &POS NUEVO))
  )
)

(DEFUN IMP-LINEAS (&LIST &NOMFUN &STR); multicativas
; en el caso de trigonometrías hay que substituir 'FUN' por el nombre
; de la función
  (LOOP
    (FINAL &LIST) (CLOSE-INPUT-FILE))
    (SETQ &STR (POP &LIST))
    (SETQ &STR (SUBSTITUYEN &STR "FUN-" &NOMFUN))
    (PRINC &STR)
    (TERPRI)
  )
)

(SETQ &LIST-TRIG '(SEN COS TAN COT SEC CSC ASEN ACOS ATAN ACOT ASEC ACSC))
(DEFUN Estado (el tal)
  (member el tal 'EQUAL)
)

(DEFUN IMP (ARCHIVO &OBJ &out)
  ( SHOT (OPEN-INPUT-FILE ARCHIVO))
        (PRINT 'ARCHIVO:) (PRINT ARCHIVO) (PRINT " NO EXISTENTE!") )
  ( SHOT (LISTEN ARCHIVO))
        (PRINT 'ARCHIVO:) (PRINT ARCHIVO) (PRINT " SIN DATOS!") )
  (SETQ &LIST (READ ARCHIVO))
  ( (ESTADO &OBJ &LIST-TRIG)
      (IMP-LINEAS (CADR (ASSOC 'FUN- &LIST)) &OBJ))
  (IMP-LINEAS (CADR (ASSOC &OBJ &LIST)) &OBJ)
)

(DEFUN TRUNCA (&NODO N &OBJ &LIST &LON)
  (SETQ &LIST (UNPACK &OBJ))
  (SETQ &LON (LENGTH &LIST))
  ( (> N &LON) &OBJ)
  (COND ( (EQUAL &NODO '(0)) (PACK (SUBLIST &LIST N &LON)) )
        ( (EQUAL &NODO '(00)) (PACK (SUBLIST &LIST 0 (- &LON N 1)) ) )
;       esta parte falta si la cadena es de un solo caracter debido
;       a que el 1er elemento de la lista es el elemento cero
  )
)

(DEFUN EXPLICA (&OBJ &FILE)
  (CURRENT-WINDOW)
  (CLEAR-SCREEN)
  (COND ( (FINDSTRING "TRUNC" &OBJ)
```



```
        (IMP 'TRUNCAR.AYD (TRUNCA '(2N 0 &OBJ) ) )
      ( (FINDSTRING "LOGARITMICA" &OBJ)
        (IMP 'TRUNCAR.AYD (TRUNCA '(2N 12 &OBJ)) )
      ( (FINDSTRING "SEL" &OBJ)
        (IMP 'SELECTOR.AYD (TRUNCA '(2N 6 &OBJ)) )
      ( (FINDSTRING "ALG" &OBJ)
        (IMP 'ALGEBRA.AYD (TRUNCA '(2N 6 &OBJ)) )
      ( (FINDSTRING "CAL" &OBJ)
        (IMP 'CALCULO.AYD (TRUNCA '(2N 6 &OBJ)) )
      ( (FINDSTRING "SESION" &OBJ)
        (IMP 'SESION.AYD &OBJ) )
      ( (FINDSTRING "MENU" &OBJ)
        (IMP 'MENU.AYD (TRUNCA '(2N 5 &OBJ)) )
      ( (OR (FINDSTRING "DIVISION" &OBJ)
            (FINDSTRING "VENTANA" &OBJ) )
          (IMP 'VENTANA.AYD &OBJ) )
  )
)

(SETQ OPCIONES-MENU
      '(LECTURA SESIONES ESQUEMAS IMPRESION MISCELANEA AYUDA FIN))
(SETQ OPCIONES-FUNCIONES
      '(SELECTORAS ALGEBRAICAS TRASCENDENTALES CALCULO EJECUTABLES))
(SETQ FUNCIONES-SELECTORAS
      '(NUMERADOR DENOMINADOR BASE EXPONENTE COEFICIENTE NUMEXP))
(SETQ FUNCIONES-ALGEBRAICAS
      '(CUADRADO EXPANDE SIMPE FACTORIZA RACIONALIZA))
(SETQ FUNCIONES-CALCULO
      '(DEPENDE MODIFENDE DERIVA INTEGRA))
(SETQ FUNCIONES-EJECUTABLES
     '(APLICA NUMEXP))

;-------------------------------------------------------------------------
;     QUIT-PROGRAM
;-------------------------------------------------------------------------

(DEFUN QUIT-PROGRAM ()
  (THROW 'QUIT-PROGRAM) )

(LOAD PROGRAMA\\ESCARGA\\SARIT T)

(LOAD PROGRAMA\\ESCARGA\\EXTINT T)

(LOAD PROGRAMA\\ESCARGA\\SALSE T)

(LOAD PROGRAMA\\ESCARGA\\EDIF T)

(LOAD PROGRAMA\\ESCARGA\\ETAB1 T)

(LOAD PROGRAMA\\ESCARGA\\ETAB2 T)
                                +
```



```
(LOAD PROGRAMA\\ESCARGA\\INTEXT T)

(LOAD PROGRAMA\\ESCARGA\\SESTAB T )

(LOAD PROGRAMA\\ESCARGA\\VALIDIT T)

(LOAD PROGRAMA\\VENTT.LSP)

(PROGN (WRITE-LINE "Oprima ESC para que continúe el programa") NIL)
```

```
;*****************************************************************
;                    TRANSFORMADOR DE FU-FIP
;*****************************************************************
;-----------------------------------------------------------------
; Funcion LeePrimera
;-----------------------------------------------------------------
(DEFun LEEPRIMERA (&KENTRADA &KSALIDA
  &AUXILIAR &ESCAPE &NUMERO &ESCAPAR &CAMBIO)
(LOOP
  (ASIGNACION-LEX)
  (ASIGNACION-SIN)
  (ASIGNACION-SEM)
  (SETQ &NUMERO 0 &ESCAPE () &NUMERO 0 &ESCAPAR 0)
  (IMP-PROMPT-ENTRADA)
  (LEEENTRADA)
  (SCAN)
  (TERPRI)
  (SIN &TK &ESCAPE) &CAMBIO)
  (SETQ &ENTRADA (COMANDO))     ; Se invoca al reconocedor sintactico
  (COND ( (NOT (NULL &ENTRADA)) ; Cuando la expresion de entrada es
                                ; sintacticamente correcta
          (AGREGA-ENTRADA &ENTRADA)
          (COND ((NOT &CAMBIO)
                 (SETQ &CAMBIO T)));Si hubo al menos un cambio, LeePrimera
                                   ;debera regresar T, o distinto de NIL
          (ESCRIBE-LINEA-ESTADO)
          (SETQ &SALIDA (APPLY (CAR &ENTRADA)
                               (CDR &ENTRADA)))
          ((NULL &SALIDA) ; Cuando hay errores de ejecucion
                  (PUF "ENTRADAS")
                  (ESCRIBE-LINEA-ESTADO)
                  (IMP-ERROR))
          (AGREGA-SALIDA &SALIDA)
          ((EQ (CAAR "ENTRADAS"1 &FS)&NOPESOS))
          (IMP-PROMPT-SALIDA)
          (Principal &SALIDA))
        ((IMP-ERROR)) ) ; Si hubo algun error en la entrada
  (TERPRI)
 )
)

(DEFun ESCRIBE-LINEA-ESTADO (&REN &COL)
  (SETQ &REN (REN)     ;
        &COL (COLUMN)) ;
  (LINEA-ESTADO)       ;
  (CURRENT-WINDOW)     ; Para regresar a la ventana donde estaba
                       ; se perdio al escribir la linea de estado
  (SET-CURSOR &REN &COL)
)

(DEFun ASIGNACION-SIN (&N)
  (SETQ &ERROR-SINT &N)

TRANSFORMADOR FU-FIP y LeePrimera    C-1                    APENDICE C
```

```
  &N)
)

(DEFun ASIGNACION-LEX (&N)
  (SETQ &ERROR-LEXI &N)
  &N)
)

(DEFun ASIGNACION-SEM (&N)
  (SETQ &ERROR-SEMA &N)
  &N)
)

(DEFun IMP-PROMPT-ENTRADA ()
  (PRINC (+ 1 (LENGTH *ENTRADAS*)))
  (WRITE-STRING "? : ")
)

(DEFun IMP-PROMPT-SALIDA ()
   (WRITE-STRING "@")
   (PRINC (LENGTH *ENTRADAS*))
   (WRITE-STRING " : ")
)

(DEFun AGREGA-ENTRADA (LAMBDA &REST)
   (SETQ *HUBO-CAMBIOS* T
         *ENTRADAS* (CONS &REST *ENTRADAS*))
   (UPDATE-STATE '(*ENTRADAS* *SALIDAS* *VARIABLES*
                   *HUBO-CAMBIOS* *NOMBRE-SESION*))
                       )
)

(DEFun AGREGA-SALIDA (&SAL)
  (SETQ *SALIDAS* (CONS &SAL *SALIDAS*))
  (UPDATE-STATE '(*ENTRADAS* *SALIDAS* *VARIABLES*
                  *HUBO-CAMBIOS* *NOMBRE-SESION*))
)

(DEFun AGREGA-VARIABLE (&VAR)
  (SETQ *VARIABLES* (CONS &VAR *VARIABLES*))
)

;-----------------------------------------------------------------
;       Rutinas de reconocimiento sintactico (TRANSFORMADOR FU-FIP)
;-----------------------------------------------------------------

;  COMANDO --> EXP $
;  COMANDO --> EXP ;
;  COMANDO --> EXP !
;  COMANDO --> EXP &

(DEFun COMANDO (&R &N)        .

TRANSFORMADOR FU-FIP y LeePrimera    C-2                    APENDICE C
```

```
( (NULL (SETQ &R (EXP))) &R)
( (CONS (!TERMINADOR-P &TK) (SETQ &N &TK)
                            (SCAN)
                            (CONS &N (LIST &R)))) )
 (ASIGNACION-SIN 3)
)

;  EXP --> var : EXP
;  EXP --> EXP1

(DEFun EXP (&V &R1)
  ((AND (VAR-P (SETQ &V &TK))
        (EQ (SCAN) #'(QUAL)))  (SCAN); EL =QUAL
                               ((NULL (SETQ &R1 (EXP))) &R1)
                               (AGREGA-VARIABLE &V)
                               (CONS SETQ
                                     (CONS &V (LIST &R1))))
  (CONS ((VAR-P &V) (DESCRIBATE &R1) (SETQ &TK &V))))
  (EXP1)
)

(DEFun RECREATE (&OBJ)
 (SETQ #'ENTRADAACTUAL (APPEND  (UNPACK &OBJ) #'ENTRADAACTUAL))
)

;  EXP1 --> TERM &EXP1

(DEFun EXP1 (&T1)
  ((NULL (SETQ &T1 (TERM))) &T1 )
  (&EXP1 &T1)
)

;  &EXP1 --> + TERM &EXP1
;  &EXP1 --> - TERM &EXP1
;  &EXP1 --> epsilon

(DEFun &EXP1 (&T1 &OP &T2)
  ( (AND (NOT (EQ &TK &PUNTO))
         (NOT (EQ &TK &PUNTOYCOMA))) &T1 )
  (SETQ &OP &TK)
  (SCAN)
  ((NULL (SETQ &T2 (TERM))) &T2)
  (&EXP1 (CONS &OP (LIST &T1 &T2)))
)

;  TERM --> FACTR &TERM

(DEFun TERM (&T1)
  ((NULL (SETQ &T1 (FACTR))) &T1)
  (&TERM &T1)
)
```

```
;  &TERM --> * FACTR &TERM
;  &TERM --> / FACTR &TERM
;  &TERM --> epsilon

(DEFun &TERM (&T1 &OP &T2)
  ( (OR (EQ &TK &PUNTO) (EQ &TK &VLASO))
        (SETQ &OP &TK)
        (SCAN)
        ((NULL (SETQ &T2 (FACTR))) &T2)
        (&TERM (CONS &OP (LIST &T1 &T2))) )
  ;La siguiente prueba hacia adelante es por si acaso el * es implicito
  ( (OR (EQ &TK #'NUMBER)
        (TOKENIZATE-P &TK)
        (VAR-P &TK)
        (VARIMP-P &TK)
        (DESENVAINA-P &TK) )
                        ((NULL (SETQ &T2 (FACTR))) &T2)
                        (&TERM (CONS &MPOR ((LIST &T1 &T2))))
  )
  &T1)
)


;  FACTR --> PRIMARY
;  FACTR --> PRIMARY ^ FACTR

(DEFun FACTR (&P &P1)
  ((NULL (SETQ &P (PRIMARY))) &P)
  ((EQ &TK #'CARET)
                  (SCAN)
                  ((COND ((NULL (SETQ &P1 (FACTR))) &P)
                         (((LIST '^ &P &P1)) ; AQUAS BUSCA (!' &P &P1))
                  ))
  &P
)

;  PRIMARY --> constante
;  PRIMARY --> $R1 | %E
;  PRIMARY --> var
;  PRIMARY --> varmap
;  PRIMARY --> var-implicita
;  PRIMARY --> ( PRIMARY )
;  PRIMARY --> ( EXP1 )
;  PRIMARY --> func ( LISTARGS )

(PROPERTY '(CLOSE)(ST '(LLAMADA )) '(PATTERN))
(PROPERTY '(CLOSE)(ST '(LLAMADA )) '(PATTERN))

(DEFun PRIMARY (&T &P &AL &N)
  ((OR (CONSTANT-P &TK)
       (NUMBER-P &TK))   ;Si es %E o %R
                (SETQ &P &TK)

TRANSFORMADOR FU-FIP y LeePrimera    C-4                    APENDICE C
```

```
                (SCAN)
                BC )

  ((=NB-P #TK)
         (SETQ BC #TK)
         (SCAN)
         BC)

  ((DARIMB-P #TK) (SETQ BC #TK)
                  (SCAN)
                  ;Pone en BC la entrada o salida correspondiente a la
                  ;variable implicita que dio el usuario
                  (SETQ BN (WTH (- (VAL-ENTERO (CDR (UNPACK BC))) 1 )
                                (REVERSE (APPLY (GET 'ESCOGELIST
                                                     (CAR (UNPACK BC))))))
                  ;Pregunta por terminador por si escogelist encontro ERRORes
                  ;o Salidas
                  ((TERMINADOR-P (CAR BN)) (CADR BN))
                  BN
  )
  ((=ARESP-P #TK) (SETQ BN #TK) (SCAN) BN)
  ((ES #TK #MENOS)
                  (SCAN)
                  ((NULL (SETQ BN (PRIMARY))) BN)
                  ((ENS #MENOS (LIST BN)))

  ((ES #TK #PARIZQ)
                  (SCAN)
                  ((NOT (NULL (SETQ BN1 (ESPY))))
                     ((ES #TK #PARDER)
                                 (SCAN) BN1)
                     (ASIGNACION(ISO-SIN 6)))
  ((RESERVADA-P #TK)
       (SETQ BT (NAYUSCULA #TK))
       (SCAN)
       ((ES #TK #PARIZQ)
             (SCAN)
             ((NOT (NULL (SETQ BAL (LISTAARGS))))
                ((ES #TK #PARDER)
                             (SCAN)
                             ((NULL (REVISAARGS BT BAL))
                                    (ASIGNACION(ISO-SIN 5)))
                             ((ENS BT BAL) )
                 (ASIGNACION(ISO-SIN 6)))
        )
  )
  (ASIGNACION(ISO-SIN 7)
)

; LISTAARGS --> ESP LA

(DEFUN LISTAARGS (BN1)

TRADUCTOR TU-TIP y LenPrimera    D-5                    APENDICE D
```

```
  ((NULL (SETQ BN1 (ESP1))) BN1)
  CLA (LIST BN1))
)

; LA --> , ESP1
; LA --> epsilon

(DEFUN LA (BACC BN1 BL)
  ( (NOT (ES #TK #COMA)) (REVERSE BACC) )
  (SCAN)
  ( (NULL  (SETQ BN1 (ESP1))) BN1 )
  (LA (CONS BN1 BACC))
)

(DEFUN TERMINADOR-P (#OBJ)
  (OR (ES #OBJ #SIGNOPESOS)
      (ES #OBJ #PTUCOMA)
      (ES #OBJ #NOMINACION)
      (ES #OBJ #NAMPERSAND)))

(DEFUN CONSTANTE-P (#OBJ)
   (NUMBER #OBJ))

; ES NUMSIMB-P SI #OBJ ES %I O %P?
(DEFUN NUMSIMB-P (#OBJ &LSTOBJ)
  (SETQ #LSTOBJ (UNPACK #OBJ))
  (AND (ES (CAR #LSTOBJ) #PORCENTAJE)
       (OR
          (AND (ES (CADR #LSTOBJ) 'I)
               (NULL (CDDR #LSTOBJ)))  ;SI es el %I
          (AND (ES (CADR #LSTOBJ) 'P)
               (ES (CADDR #LSTOBJ) '2)
               (NULL (CDDDR #LSTOBJ)))) ;SI es el %P?
       )
  )
)

(DEFUN VAR-P (#OBJ)
   ((AND ((IDENTIFICADOR-P #OBJ)
          (NOT (OR (RIDAMEESPECIAL-P #OBJ)
                   (RESERVADA-P #OBJ)
                   (LETRERO-P (CAR (UNPACK #OBJ))))
               )))  #OBJ)
)

(DEFUN IDENTIFICADOR-P (#OBJ)
  (AND (ATOM #OBJ)
       (ALPHA-CHAR-P (CAR (UNPACK #OBJ)))
       (NOT (RESERVADA-P #OBJ)))
  )
)

                                                                    .

TRANSFORMADOR TU-TIP y LenPrimera    D-6                    APENDICE D
```

```
(DEFUN VARIAB-P (#OBJ)
  (AND
       (OR (ES (CAR (UNPACK #OBJ)) #INTERROGACION)
           (ES (CAR (UNPACK #OBJ)) #ARROBA))
       (>= (VAL-ENTERO (CDR (UNPACK #OBJ)))
           (LENGTH *ENTRADAS*))
  )
)

(DEFUN VAL-ENTERO (#LST &N) ;LST contiene digitos decimales y VAL-ENTERO
                            ;da el valor del numero formado por esos dí-
                            ;gitos
  (SETQ &N 0)
  ((LOOP
     ( (NULL #LST) &N )
     (SETQ &N (+ (* &N 10) (NUMERICO (POP #LST))))
  )
)

(DEFUN NUMERICO(#C)
   ( (&NUMERIC-CHAR-P #C)   (- (ASCII #C) (ASCII '0')) )
)

(DEFUN FRACCION (&N &F)
  (SETQ &N 0)
  (SETQ &F 10)
  ((LOOP
      ( (OR (NULL #ENTRADAACTUAL)
            (NOT (&NUMERIC-CHAR-P (CAR #ENTRADAACTUAL)))) &N )
      (SETQ &N (+ &N (/ (NUMERICO (POP #ENTRADAACTUAL)) &F )))
      (SETQ &F (* &F 10))
  )
)


;ES PARIZQ-P SI #OBJ ES %I(I), %I(I), %I(I)
(DEFUN PARIZQ-P(#OBJ
                &LSTOBJ)
  (SETQ &LSTOBJ (UNPACK #OBJ))
  (AND (ES (CAR &LSTOBJ) #PORCENTAJE)
       (OR
          (AND (ES (CADR &LSTOBJ) 'I)
               (>= 1 (VAL-ENTERO (CDDR &LSTOBJ)) ((FACTS *ESCUERAS-ESP*)))
               (AND (ES (CADR &LSTOBJ) 'I)
                    (>= 1 (VAL-ENTERO (CDDR &LSTOBJ)) ((FACTS *ESCUERAS-INT*)))
               (AND (ES (CADR &LSTOBJ) 'E)
                    (>= 1 (VAL-ENTERO (CDDR &LSTOBJ)) ((FACTS *ESCUERAS-DEP*)))
          )
  )
)

(DEFUN RESERVADA-P (#OBJ)

TRANSFORMADOR TU-TIP y LenPrimera    D-7                    APENDICE D
```

```
  ( (NOT (NULL) (FLAST (MAYUSCULA #OBJ) 'RESERVADA))) )
)

(DEFUN MAYUSCULA (#OBJ)
  (PACK (MAPCAR 'CHAR-UPCASE (UNPACK #OBJ))))
)

(DEFUN MINUSCULA (#OBJ)
  (PACK (MAPCAR 'CHAR-DOWNCASE (UNPACK #OBJ))))
)

;-----------------------------------------------------------------------
;  Rutinas de Edicion de Linea
;-----------------------------------------------------------------------

(DEFUN LEERENTRADA ()
  (SETQ #ENTRADAACTUAL (EDITA-LINEA))
  (SETQ #LINGINICIAL (LENGTH #ENTRADAACTUAL) ))
)

(DEFUN EDITA-LINEA (#C &LINEA)
  (SETQ *PRINT-ESCAPE* NIL)   ; No imprime caracteres de escape
  (SETQ #C 0)

  ((LOOP
      ((ES (ASCII) #C) (RETORNO)
           (MAPPEARC) (CDR &LINEA)) ;CDR para que quite el RETURN

      ((ES (ASCII) #C) (BSCAPE) (SETQ &NBSU 1) (LIST (BSCAPE))

      (SETQ #C (TECLA-F (READ-BYTE)))
      ((CAE (!(EQUIVS *FUNCION*) (SETQ &LINEA ((EVAL (FUNCION #C &LINEA)))
            ((CONC ((AND (ES (ASCII) #C) #FLECHA(IZQ)
                         (NOT (NUL) &LINEA)))
                                (WRITE (ASCII) #C))
                                (BORRA-IMPRESION)
                                (SETQ &LINEA (CDR &LINEA)))
                ((ES (ASCII) #C) #FLECHA(DER) (WRITE #CAMPANA))
                ( (WRITE (ASCII) #C))(SETQ &LINEA ((CONS #C &LINEA)) )
           S)
      )
  )
)

(DEFUN MAYEARC() (&LINEA &LIN)
  ((LOOP
      ((NULL &LINEA) &LIN)
      (SETQ &LIN (CONS (ASCII) ((CAE &LINEA)) &LIN))
      (SETQ &LINEA (CDR &LINEA))
  )
)

TRANSFORMADOR TU-TIP y LenPrimera    D-8                    APENDICE D
```

```
;Esta funcion lee un byte del teclado.
;Si se presiona una tecla especial coloca a la variable
;*FUNCION* a T y regresa el codigo correspondiente,
;de acuerdo a la tabla de teclas de funciones extendidas de la IBM PC.
;Si la tecla presionada no es de funcion, la variable *FUNCION*
;se coloca a NIL y la funcion regresa el codigo ascii del caracter
;leido.
(DEFUN TECLA-F (BYTE TEMP)
  ((SETQ BYTE 255)
               (SETQ TEMP (READ-BYTE))
               (SETQ *FUNCION* T)
               TEMP)
  (SETQ *FUNCION* NIL)
  BYTE
)

(DEFUN IDENT-FUNCION (ARC #LINEA)
  ((IFQ #C 59) (WRITE 92) (APPEND *(49 37) #LINEA)) ; F2
  ((IFQ #C 60) (WRITE 91)(APPEND *(75 80 37) #LINEA)) ; F3
  ((IFQ #C 61) (WRITE (ASCII 159))(APPEND *(84 70 73) #LINEA)) ; INTEGRA
  #LINEA
)

(DEFUN BORRA-IMPRESION()
    (WRITE ^L ())
    (WRITE #FLECHA120)
)

;-----------------------------------------------------------------------
; Rutinas de reconocimiento lexico
;-----------------------------------------------------------------------

(DEFUN SCAN ()
  (SETQ #TX (NASTREATE))
)

(DEFun NASTREATE ()
    ((NULL #ENTRADAACTUAL) ()) ;Aqui es cuando es Fin de ENTRADA
    (LOOP
        (  (NOT (EQ (CAR #ENTRADAACTUAL) #ESPACIO)) )
        (POP #ENTRADAACTUAL)
    )
    ((ALPHA-CHAR-P (CAR #ENTRADAACTUAL))(IDENTIFICADOR))
    ((NUMERIC-CHAR-P (CAR #ENTRADAACTUAL)) (NUMERO))
    ((SIGNOESPECIAL-P (CAR #ENTRADAACTUAL))(POP #ENTRADAACTUAL))
    ((AND (OR
            (PORCENTAJE-P (CAR #ENTRADAACTUAL))
            (ENTARRO-P (CAR #ENTRADAACTUAL))    )
          (OR
            (NUMERIC-CHAR-P  (CADR #ENTRADAACTUAL))
            (NUMERIC-CHAR-P (CADDR #ENTRADAACTUAL)))
                                        (SAR (IMPLICITA))

TRANSFORMADOR FU-FIP y LeoFriulana    3-9                    APENDICE E
```

```
    ( (PORCENTAJE-P (CAR #ENTRADAACTUAL))
                                (RLM-ERROR 1 (QUO))

    ((OR (CAR #ENTRADAACTUAL) #ESCAPE) (POP #ENTRADAACTUAL))
    (#ESTADACCESO-LEX 1)
)

(DEFUN IDENTIFICADOR(BYTE)
  (LOOP
     ( (OR (NULL #ENTRADAACTUAL)
           (NOT (ALPHANUMERIC-P (CAR #ENTRADAACTUAL)))) (PACK (REVERSE BYTE)) )
     (SETQ BYTE (CONS (POP #ENTRADAACTUAL) BYTE))
  )
)

(DEFUN NUMERO (&N)
  (SETQ &N 0)
  (LOOP
     ( (NULL #ENTRADAACTUAL) &N )
     ( (AND (NOT (NUMERIC-CHAR-P (CAR #ENTRADAACTUAL)))
            (NOT (EQ (CAR #ENTRADAACTUAL) #PUNTO))) &N)
     ((EQ (CAR #ENTRADAACTUAL) #PUNTO)(POP #ENTRADAACTUAL))(* &N (FRACCION)))
     (SETQ &N (+ (* &N 10) (NUMBER(1) (POP #ENTRADAACTUAL))))
  )
)

(DEFUN SIGNOESPECIAL-P (#OBJ)
  ( (NOT (NULL (FLASP #OBJ '#SIGNOESPECIAL))) )
)

(DEFUN PORCENTAJE-P (#OBJ)
  (EQ #OBJ #PORCENTAJE))

(DEFUN ENTARRO-P (#OBJ)
  (OR (EQ #OBJ #INTERROGACION)
      (EQ #OBJ #ARROBA))
)

(DEFUN VAR-IMPLICITA (&N)
  (SETQ &N (LIST (POP #ENTRADAACTUAL)))
  (COND ( (MEMBER (CAR #ENTRADAACTUAL) '(E F D))
                (SETQ &N (CONS (POP #ENTRADAACTUAL) &N)) ) )
  (LOOP
     ((NOT (NUMERIC-CHAR-P (CAR #ENTRADAACTUAL))) (PACK (REVERSE &N)))
     (SETQ &N (CONS (POP #ENTRADAACTUAL) &N))
  )
)

(DEFUN NUM-SIMBOLICO ()
  (POP #ENTRADAACTUAL)
  ((EQ (CAR #ENTRADAACTUAL) '&) (POP #ENTRADAACTUAL) '%D)
  ((AND (EQ (CAR #ENTRADAACTUAL) '#)

TRANSFORMADOR FU-FIP y LeoFriulana    3-10                   APENDICE E
```

```
       (EQ (CADR #ENTRADAACTUAL) '(1)) (POP #ENTRADAACTUAL)
                                       (POP #ENTRADAACTUAL)
                                       'SP()
)

;-----------------------------------------------------------------------
;     Rutinas de manejo e impresion de mensajes de error
;-----------------------------------------------------------------------

(SETQ #LISTERR-SIN '(
                     (5 . (" Terminador esperado."))
                     (6 . (" "(" esperado."))
                     (7 . (" Identificador, dim o funcion esperado."))
                     (8 . (" Funcion no definida."))
                    )
)

(SETQ #LISTERR-LEX '(
                     (1 . (" Caracter invalido."))

                    )
)

(SETQ #LISTERR SEM
      '((1 . (" Numero o tipo invalido de argumento."))
        (2 . (" Argt y Argf dados, no son estructuralmente iguales"))
        (3 . (" La expresion no cumple con el patron dado."))
        (4 . (" Division invalida 0/0"))
        (5 . (" Division invalida 1/0"))
        (6 . (" Exponenciacion invalida 0^0"))
        (7 . (" El tercer argumento debe corresponder a &x"))
        (10 . (" No corresponde Dif() a la derivada de U con respecto a x"))
        (11 . (" La combinacion de Dif(), U no es la expresion dada"))
                )
)

(DEFUN INT-ERROR ()
  (SPACES 5)
  ((NULL #ERROR-LEX)
          ((NULL #ERROR-SINT)
              ((NULL #ERROR-SEMAN)
              (PRINT (CDR (ASSOC #ERROR-SEMAN #LISTERR-SEM)))
          )
          (PRINT (CDR (ASSOC #ERROR-SINT #LISTERR-SIN)))
  )
  (PRINT (CDR (ASSOC #ERROR-LEX #LISTERR-LEX)))
)

;    Definicion de caracteres, simbolos especiales y palabras reservadas

(SETQ #SIGNOPESOS '$)       (SETQ #OPICIONA '[;])    (SETQ #PARIZQ '[(])
(SETQ #PERCENT '[%])        (SETQ #LLAVE '{)         (SETQ #PARDER '[)])
```

```
(SETQ #ADMIRACION '!)       (SETQ #MAS '+)           (SETQ #MENOS '-)
(SETQ #POR '*)              (SETQ #SLASH '/)         (SETQ #NUMERO '#)
(SETQ #PUNTO '[.])          (SETQ #COMA '[,])        (SETQ #INTERROGACION '?)
(SETQ #ARROBA '@)           (SETQ #PORCENTAJE '%)

(SETQ #ESPACIO (ASCII 32))  (SETQ #RETORNO (ASCII 13))
(SETQ #ESCAPE  (ASCII 27))  (SETQ #CAMPANA (ASCII 7))
(SETQ #FLECHA120 (ASCII 8))

(SETQ #ENTRADAACTUAL '())

(SETQ #LISTSIGNOESPECIAL '(#PUNTOCOMA #PUNTOCOMA #PARIZQ #PARDER
                           #IGUAL #MAYORQUE #MENORQUE #MAS
                           #POR #NUMERO #SLASH #NUMERO[1] #PUNTO #COMA))

(DEFUN ESPECIALIZA (#LST)
  (SETQ #LST #LISTSIGNOESPECIAL)
  ((LOOP
     ((NULL #LST))
     (FLAG (EVAL (POP #LST)) 'SIGNOESPECIAL)
  )
)

(ESPECIALIZA)

(SETQ #LISTRESERVADAS '(DEN CON TAN COT SEC CSC
                        SINH COSH TANH COTH SECH CSCH
                        ASEN ACOS ATAN ACOT ASEC ACSC
                        ARG SIMPLIFICACION NUMERADOR
                        EXPAND DESARROLLO FACTORIZA DIVIDIR PARTFRAC RACIONALIZA
                        CUADRADO
                        SUBSIN COEFICIENTE BASE EXPONENTE PUNTO PUNTO PUNTO
                        LOG EVALUE ESCRIBE INTEGRA SUBSIN SUBCAP
                        ))

(SETQ #LISTRESERVADAS (APPEND '(#ALGUIENPIDE #PRECEPTO LA DIFFEQ) #OBJETIVOS
                              #PLICA
                              SETFLAGS FLAGS)
                      #LISTRESERVADAS))

(DEFUN DESTRUIR (#LST)
  (SETQ #LST #LISTRESERVADAS)
  (LOOP
     ((NULL #LST))
     (FLAC (POP #LST) 'RESERVADA)
  )
)

(DESTRUIR)

(SETQ #INFIJAC '(
     ; Aquellas que necesitan un solo argumento que sea EXPRESION
```

```
    ( ESP    (SEN SIN TAN EXP REC ESC
              SENH COSH TANH EXPH RECH ESCH
              ASEN ACOS ATAN ASEN ARSC ACSC
              ABS DENOMINADOR NUMERADOR
              EXPANDE EXPD FACTORIZA RACIONALIZA LN
              COEFH COEFICIENTE BASE EXPONENTE CUADRADO))
    ; Aquellas que necesitan dos argumentos EXPRESION
    ( ESPERP . (DIVIDE PARTFRAC ESERP))
    ; Aquellas que necesitan tres EXPRESIONES
    ( ESPERPESP . (DIVIDE PICO PEQUE PARTE SUMESP))
    ; Aquellas que necesitan dos o tres expresiones
    ( EAPZEXPI . EAPLICA))
    ; Aquellas que necesitan una EXPRESION y una VAR
    ( ESPVAR  . (INTEGRAL))
    ; Aquellas que necesitan una EXPRESION y una lista de VAR/INT
    ( ESPVARS . (DERIVAR))
    ; Aquellas que necesitan un lista de VARs
    ( VARS . (DEFUNCS HOSEMPHODE))
    ; Aquellas que necesitan una EXPRESION y un NUMERO
    ( ESPNUM . (LOG LOGEXP RELOGEXPO ATRIGEXPO ))
    ; La que necesita 0 numeros
    ( ESPO . (SETFLAGS))
    ; Las que no necesitan un ARG entero
    ( ARG . (FLAGS))
  )
)

(DEFUN REVISAARGS (EF VAL)
    ;Si solo requiere una ESP
    ((MEMBER EF (CDR (ASSOC ESP #'ESFUNC)))
       (EQ (LENGTH VAL) 1) )
    ;Si se requieren dos ESP
    ((MEMBER EF (CDR (ASSOC ESPESP #'ESFUNC)))
       (EQ (LENGTH VAL) 2) )
    ;Si se requieren tres ESP
    ((MEMBER EF (CDR (ASSOC ESPESPESP #'ESFUNC)))
       (EQ (LENGTH VAL) 3) )
    ;Aquellas que necesitan dos o tres expresiones  APLICA
    ((MEMBER EF (CDR (ASSOC EAPZEXPI #'ESFUNC)))
    ; Se requieren 2 o mas argumentos
       ; Si son dos
       ( ( AND (= (LENGTH VAL) 2)
               (VARESP-P (CAR VAL)) ) )
       ; Si son mas
       ( ( AND (>= (LENGTH VAL) 3)
               (VARESP-P (POP VAL)) ;Numero de regla
         (POP VAL) ; Expresion
         (POP VAL) ; Derivada de U
         (LOOP
           ((NULL VAL) )
           ((NOT (STRINGP (POP VAL))) NIL)
           ((NOT (POP VAL)) NIL)
```



```
           )
         )
       )
    ;Si se requiere una ESP y una VAR
    ((MEMBER EF (CDR (ASSOC ESPVAR #'ESFUNC)))
       ( AND  (EQ (LENGTH VAL) 2)
              (VAR-P (CADR VAL))  ) )
    ;Si se requiere una ESP y varias VARS
    ((MEMBER EF (CDR (ASSOC ESPVARS #'ESFUNC)))
       (AND (>= (LENGTH VAL) 2)
            (POP VAL)
            (LOOP
              ((NULL VAL))
              ((NOT (OR (VAR-P (CAR VAL))
                        (INTEGERP (CAR VAL)))) NIL)
              (POP VAL)
            )
       )
    )
    ;Si se requiere una lista de VARs
    ((MEMBER EF (CDR (ASSOC VARS #'ESFUNC)))
       ( AND (>= (LENGTH VAL) 2)
             (LOOP
               ((NULL VAL))
               (VAR-P (POP VAL))
             )
       )
    )
    ;Si solo se requiere una ESP y un numero que puede ser BASE
    ((MEMBER EF (CDR (ASSOC ESPNUM #'ESFUNC)))
       (AND (EQ (LENGTH VAL) 2)
            (BASE-P (CADR VAL))))
    ;Si se requiere un argumento entero
    ((MEMBER EF (CDR (ASSOC ARG #'ESFUNC)))
       (AND (EQ (LENGTH VAL) 1)
            (INTEGERP (CAR VAL))))
    ;Solo entero
    ((MEMBER EF (CDR (ASSOC ESPO #'ESFUNC)))
       (AND (EQ (LENGTH VAL) 0)
            (POP VAL)
            (LOOP
              ((NULL VAL))
              ((NOT (INTEGERP (CAR VAL))) NIL)
              (POP VAL)
            )
       )
    )
)

(DEFUN BASE-P (ROBJ)
  (OR (NUMBER ROBJ)
```



```
      (VAR-P ROBJ)
      (EQ ROBJ 'SE))
)
```

```
;*****************************************************************
;*                    TRANSFORMADOR DE FIP - FIPN                *
;*****************************************************************

(DEFUN -TRAP (EXT
   ( !MAS! (CAAR LEXT))                          ; A - B = A + (-1 * B)
                    (* -1 (CAR LEXT)) )
   (EVAL (+ (CAR LEXT) (* -1 (CADR LEXT))))
)

(DEFUN / (EXT EXT2)
   ( (AND (ZEROP EXT2)                           ; A       -1
          (ZEROP EXT1) )                         ; - + A * B
          (ASIGNACION(ERROR-ERR 4))              ; B
   (* EXT (* EXT2 -1))
)

;*****************************************************************
;*                      MODULO ARITMETICO                        *
;*****************************************************************

;-----------------------------------------------------------------
;Rutinas de suma simbolica
;-----------------------------------------------------------------

(DEFUN ADDTERMS (EXT EXT2 ZVAR1)
  ( (ATOM EXT)
     ( (INTEGERP EXT)
        ( (INTEGERP EXT2) (+ EXT EXT2) )
        ( (ZEROP EXT) EXT2)
        ( (ATOM EXT2) NIL)
        ( (NUMBER EXT2)
             (MKRAT (+ (NUMERADOR EXT2) (* EXT (DENOMINADOR EXT2)))
                    (DENOMINADOR EXT2)) )
        ( APPLY (GET '+ (CAR EXT2)) (CONS EXT (ARGLIST EXT2)) )
     )
     ( (ATOM EXT2)
           ( (ZEROP EXT2) EXT) )
     ( APPLY (GET '+ (CAR EXT2)) (CONS EXT (ARGLIST EXT2)) )
  )
  ( (ATOM EXT)
        ( (INTEGERP EXT)
             ( (ZEROP EXT) EXT )
             ( (NUMBER EXT)
                  (MKRAT (+ (NUMERADOR EXT) (* EXT (DENOMINADOR EXT)))
                         (DENOMINADOR EXT)) )
             (APPLY (GET '+ (CAR EXT)) (CONS EXT (ARGLIST EXT)))
        )
        (APPLY (GET '+ (CAR EXT)) (CONS EXT (ARGLIST EXT)))
  )
  ( (AND (NUMBER EXT)
         (NUMBER EXT2))
```



```
        ZVAR1
)

(DEFUN *TRAP LST
  ((LAMBDA (LEXT LEX2)
     (
      (SETQ LEXT (REVERSE LEXT))
      (LOOP
        (SETQ LEX2 (MERGESUM (POP LEXT) LEX2))
        ( (ATOM LEXT)       ; Termina de sumar todos los terminos
            (MKSUM LEX2) ) ;Construye la representacion FIPN de la
        )                  ;suma que consta de los terminos especificados
      )                    ;en LEX2
    ) LST
  ) ; Lambda
)

;-----------------------------------------------------------------
;Rutinas de Multiplicacion Simbolica
;-----------------------------------------------------------------

(DEFUN MULTFACTS (EXT EXT2)
  ( (ATOM EXT)
        ( (INTEGERP EXT)
                    ( (INTEGERP EXT2) (* EXT EXT2) )
                    ( (ZEROP EXT) 0 )
                    ( (ONE-P EXT) EXT2 )
                    ( (ATOM EXT2) NIL )
                    (APPLY (GET '* (CAR EXT2) )
                           (CONS EXT (ARGLIST EXT2)) )
        )
        ( (ATOM EXT2)
                  ( (ZEROP EXT2) 0 )
                  ( (ONE-P EXT2) EXT)
        )
        (APPLY (GET '* (CAR EXT2))
               (CONS EXT (ARGLIST EXT2)))
  )
  ( (ATOM EXT2)
        ( (INTEGERP EXT2)
                    ( (ZEROP EXT2) 0 )
                    ( (ONE-P EXT2) EXT )
                    (APPLY (GET '* (CAR EXT))
                           (CONS EXT2 (ARGLIST EXT)))
         )
        (APPLY (GET '* (CAR EXT))
               (CONS EXT2 (ARGLIST EXT)))
  )
  ( (APPLY #'GET '* (CAR EXT) )
                    (CONS EXT2 (ARGLIST EXT)) )
  (APPLY #'GET '* (CAR EXT2) (CONS EXT (ARGLIST EXT2)))
)
```



```
        (SETQ ZVAR1
              (RATNUM (+ (* (NUMERADOR EXT) (DENOMINADOR EXT2))
                         (* (NUMERADOR EXT2) (DENOMINADOR EXT)))
                      (* (DENOTE EXT (DENOMINADOR EXT))
                         (DENOTE EXT2 (DENOMINADOR EXT2)))
                      (EXP EXT EXT2) ) ) ; Arg 3 de RATNUM
        ZVAR1
  )
  ( (APPLY (GET '+ (CAR EXT)) (CONS EXT2 (ARGLIST EXT))) )
  (APPLY (GET '+ (CAR EXT2)) (CONS EXT (ARGLIST EXT2)))
)

(DEFUN MERGETERM (EX1 LEX)
       EX2 EX3 LEX2 LEX3 ZVAR4) ;Locales
  (SETQ LEX2 LEX1)
  (LOOP
    ( (ATOM LEX2)
        (SETQ EX2 (CCDIV EX1))
        (LOOP
          ( (ATOM LEX1) (REVERSE LEX2 (LIST EX1)) )
          (SETQ EX3 (CCDIV (CAR LEX1)))
          ( (equal EX2 EX3)
                (SETQ EX1 (ADDTERMS (COEFFICIENTE EX1)
                                    (COEFFICIENTE (POP LEX1))))
                ( (ZEROP EX1) (REVERSE LEX2 LEX1) )
                ( REVERSE LEX2 (CONS (* EX1 EX2) LEX1) )
          )
          ( (ORDERED EX2 EX3) (REVERSE LEX2 (CONS EX1 LEX1)) )
          ( PUSH (POP LEX1) LEX2 )
        )
    )
    ( (SETQ EX2 (ADDTERMS EX1 (CAR LEX2)))
                  (SETQ ZVAR4 (MERGESUM EX2 ; Arg 1 de MERGESUM
                                        (REVERSE LEX3 (CDR LEX2)) ))
                  ZVAR4
    )
    ( PUSH (POP LEX2) LEX3 )
    LEX3
  )
)

(DEFUN MERGESUM (EX1 LEX)
       LEX2 ZVAR3) ; Locales
  ((ATOM LEX1)
          ( (NUM EX1) (CDR EX1) )
          (LIST EX1) )
  ((SUM EX1) (SETQ LEX2 (CDR EX1))
          (LOOP (SETQ LEX1 (MERGETERM (POP LEX2) LEX1))
                ( (ATOM LEX2) LEX1 )
          )
  )
  (SETQ ZVAR3 (MERGETERM EX1 LEX1))
                  ^
```



```
(DEFUN MERGEFACT (EX1 LEX)
       EX2 EX3 EX4 LEX2 LEX3 ZVAR2 ZVAR5 ZVAR6) ;Locales
       (SETQ LEX2 LEX1)
  (LOOP ((ATOM LEX2) ;If no common BASE or combination
        ( (INTEGERP EX1) (CONS EX1 LEX1) )
        ( (RECIP EX1)
                ( (INTEGERP (SETQ EX3 (CAR LEX2)))
                                (CONS EX3 (CONS EX1 (CDR LEX2))) )
                (CONS EX1 LEX2)
        )
        (SETQ LEX3 NIL)
        ;If EX1 in LEX1 found , insert EX1
        (SETQ EX2 (BASE EX1))
        (LOOP;In LEX1  In the proper order
          ((ATOM LEX1) (REVERSE LEX3 (LIST EX1)))
          ((equal EX2
                  (SETQ EX4 (BASE (SETQ EX2 (CAR LEX1)))))
                  (SETQ EX2 (EXPONENTE EX2))
                  ((AND (NUMBER (SETQ EX4 (EXPONENTE EX1)))
                        (NUMBER EX2))
                        (SETQ EX1
                              (* EX2 (SETQ ZVAR2
                                           (ADDTERMS EX4 EX2))))
                        (SETQ ZVAR5
                              (MERGEFACS EX1
                                         (REVERSE LEX3 (CONS EX2 LEX1))))
                        ZVAR5
                  )
                  ((ORDERED EX4 EX2)
                                  (REVERSE LEX3 (CONS EX1 LEX2))
                  )
                  ( REVERSE LEX3 (CONS (POP LEX1) (CONS EX1 LEX1)) )
          )
          ( (AND (ORDERED EX2 EX4)
                 (NOT (INTEGERP EX2))
                 (NOT (RECIP EX2)))
                                (REVERSE LEX3 (CONS EX1 LEX1))
          )
          (POP EX2 LEX2)
          (SETQ LEX1 (CDR LEX1))
    )
    ( ( SETQ EX3 (MULTFACTS EX1 (SETQ EX2 (POP LEX2))) )
                  (SETQ ZVAR6 (MERGEFACS EX3 (REVERSE LEX3 LEX2)))
                  ZVAR6
    )
    (PUSH EX2 LEX3)
  )
)

(DEFUN MKSUM(EXT LEXT
            LEX2 ZVAR7) ;Locales LEX2
```

```
                    )
                    ((ONE-P EXT) EXT)
                    ((LIST '* EXT -1)
                    ))
               ((EXPT EXT EXZ)
          )
          ((ONE-P EXZ) EXT)
          ((ZERO-P EXZ)
               (ERROR(EXPT) 1)
          ((EQ EXT 1)
               ((SETQ EXZ (MODULO EXZ 4))
               ((EQ EXZ 2) -1)
               ((EQ EXZ 3) (- 1))
               1' EXT EXZ))
          ((ATOM EXT)
               (LIST '* EXT EXZ))
          ( (APPLY (GET 'BASE (CAR EXT))
               (CONS EXZ (ARGS> EXT))) )
          (LIST '** EXT EXZ)
     )
     ((ATOM EXT)
          ((ONE-P EXT) 1)
          ((ZERO> EXT)
               ((= EXZ 0)
                    (ERROR(ZERO>(0)-NUM 5)))
               ((OR (= EXZ 0)
                    (ERROR(NEG) 0) (LIST '** EXT EXZ)
          )
          ((ATOM EXZ)
               ((AND (EQ EXT 'E)
                    (EQ EXZ 'E)
                    (NEQNULL #'FLOATPO 7))
                                   (= (EXP 1)
                                     (* %I (SIN 1)))
               )
               (( AND (PEQNULL* #'LOGEXPO 7)
                    (NOT (OR (equal EXT #'(%PAR)
                         (= EXT 0))) )
                                   (' #'LOGAR
                                    (* EXZ (LOG EXT #'(%PAR))))
               )
               (LIST '** EXT EXZ)
          )
          ( (ATOM* (GET 'EXPONENTE (CAR EXZ))
                 (CONS EXT (ARGS> EXZ)) ) )
          ( (AND (PEQNULL #'(%PAR) 7)
                 (NOT (OR (equal EXT #'(%PAR)
                          (NUMBER EXZ)
                          (= EXT 0) 1) ))
                                   (' #'(%PAR)
                                     (* EXZ (LOG EXT #'(%PAR)))
                                   )
```



```
               )
               (LIST '** EXT EXZ)
          )
          ( (APPLY (GET 'EXPONENTE (CAR EXZ))
                 (CONS EXT (ARGS> EXZ))) )
          ( (APPLY (GET 'BASE (CAR EXT))
                 (CONS EXZ (ARGS> EXT))) )
          ( (AND (PEQNULL #'LOGEXPO 7)
                 (NOT (OR (equal EXT #'(%PAR)
                          (NUMBER EXZ)
                          (= EXT 0) 1) )
                                   (' #'(%PAR) (* EXZ (LOG EXT #'(%PAR))))
          )
          ((LIST '** EXT EXZ)
     )
)

;-----------------------------------------------------------------------
; Rutinas para obtener LOG
;-----------------------------------------------------------------------

(DEFUN LOG (EXT EXZ)
     ( (NULL EXZ) (LOG EXT #'(%PAR)) )
     ( (OR (ZEROP EXT) (ONE-P EXZ))
                    (' (LIST '*LOG EXT EXZ)) )
     ( (AND #'BASE> (ONE-P EXT)) 0 )
     ( (AND #'BASE> (equal EXT EXZ)) 1 )
     ( (AND #'BASE> (NEQNULL EXT))
                                   (= (* %I %P)) (LOG (- EXT))) )
     ( (AND (NEQNULL #'FLOATPO 7)
            (equal EXZ %E))
                                   (* 2 %I (ATAN (/ (* %I (- 1 EXT)) (+ 1 EXT)))) )
     ( (AND #'BASE>
            ((INTEGER> EXT)
            ((RATIONAL> EXZ)
            ((ZEROP (MODULO EXT EXZ)))
                                   (= 1 (LOG (EXPT(ONE) EXT EXZ) EXZ)) )
     ( (AND (NOT (EQ EXZ #'(%PAR)))
            (FONEQNULL #'LOGEXPO 2))
                                   (/ (LOG EXT #'(%PAR)) (LOG EXZ #'(%PAR))) )
     ( (ATOM EXT) (LIST '*LOG EXT EXZ) )
     ( (APPLY (GET '*LOG (CAR EXT))
              (CONS EXZ (ARGS> EXT))) )
     (LIST '*LOG EXT EXZ)
)


(DEFUN (= (EXT)
       (*LOG EXT %E))

(DEFUN (%PAR) (EXT)
  ;Hace referencia a la variable global #'(%PAR)
  ( (EXT) #'(%PAR)
```

```
                ( (ATOM LEXT)
                        ((PRODUCT EXT)
                                (CONS EXT))
                        (LIST EXT) )
                ( (PRODUCT EXT)
                        (SETQ LEXZ (CDR EXT))
                        (LOOP
                            (SETQ LEXT (MERGEFACT (POP LEXZ) LEXT))
                            ( (ATOM LEXZ) LEXT )
                            )
                        )
                (SETQ ZVARF (MERGEFACT EXT LEXT))
                ZVARF
)

(DEFUN ARGUMENTO (LST)
  (IMULL (CADR LST))(CAR LST))
  LST
)

(DEFUN *INAP LST
  ( (LAMBDA (LIST LEXZ) ; Local LEXZ
          (
            (SETQ LEXT (REVERSE LEXT))
            (LOOP
                 (SETQ LEXZ (MERGEPROD (POP LEXT) LEXZ))
                 ;When the end of LEXT is reached
                 ( (ATOM LEXT) (MEPROD LEXZ) )
            )
          )
    ) (ARGUMENTO LST)
  )
)

;-----------------------------------------------------------------------
; Utilerias para Contruccion de expresiones de suma, multiplicacion
; y Numeros Racionales
;-----------------------------------------------------------------------

(DEFUN ARGPR (EXT) ;Utiliza la variable global ARGPR
       ( (ATOM (CDDDR EXT)) (CDR EXT) )
       (LIST (CADR EXT) (CONS (CAR EXT) (CDDR EXT)))
)

(SETQ #ARGLIST !(* +))

(DEFUN ARGLIST (EXT) ;Utiliza la variable global #ARGLIST
     ( (MEMBER (POP EXT) #ARGLIST) (CONS EXT) )
      EXT
)

(DEFUN MEGUM (LEXT)
      ( (ATOM LEXT) 0 )
      ( (ATOM (CDR LEXT)) (CAR LEXT) )
      (CONS '+ LEXT)
)

(DEFUN #NPRED (LEXT)
      ( (ATOM LEXT) 1 )
      ( (ATOM (CDR LEXT)) (CAR LEXT) )
      (CONS '* LEXT)
)

(DEFUN #RDAI (EX1 EX2)
      ( (ONE-P EX2) EX1 )
      ( (ONE-P EX1) (LIST '* EX2 -1) )
      (LIST '* EX1 (LIST '* EX2 -1))
)

(DEFUN #ATSUM (EX1 EX2 EX3)
       ( (ONE-P EX3) (MERAT EX1 EX2) )
       (SETQ EX1 (#ADDT(EX1 EX1 EX3)))
       (SETQ EX2 (#ADDT(EX2 EX2 EX3)))
       (SETQ EX3 (SUB EX1 EX3) )
       (PROAT (#ADDT(EST EX1 EX3) (#ADDT(EX1 EX2 EX3)))
)

;-----------------------------------------------------------------------
;Rutinas de exponenciacion simbolica
;-----------------------------------------------------------------------

(DEFUN EXPT (EX1 EX2 EX3) ;Local ESB
        (SETQ EX3 1)
   (LOOP
      ( ( ONE-P (CDR (SETQ EX2 (DIVIDE EX2 2))))
                                    (SETQ EX3 (* EX1 EX3)) ) )
      ( (ZEROP (SETQ EX2 (CAR EX2))) EX3 )
      (SETQ EX1 (CHARRAD EX1))
   )
)

(DEFUN * (EX1 EX2)
    ((INTEGERP EX2)
        ((INTEGERP EX1)
              ((ONE-P EX1) %)
              ((ONE (ZEROP EX1)
                    (ZEROP EX2))
                              (ASIGNACION!SO-SEN A))
              ((MINUSP EX2)
                        ((ZEROP EX1)
                                    (ASIGNACION!SO-SEN 0))
                        (SETQ EX1 (EXPT EX1 (- EX2)))
                        ((MINUSP EX1)
                                    (- 0* (- EX1) -1))
```

```
        (PROG (X1 *R1))
                    (LA (ABS X1))
    )
    (LA X1)
)

;-----------------------------------------------------------------------
; Funciones de utileria
;-----------------------------------------------------------------------

; Funciones para definition de propiedades, redefinicion de BREAK, QUOTIENT

(MOVD 'MOD 'MODULO)

(DEFUN PUTPROP (X1 X2 X3)
  ( (OR (EQUAL (GET X1 X2) X3)
        (AND X3
             (EQUAL (GETD (GET X1 X2)) X3)) )
                                        X1 )
  ( (SET (GET X1 X2) X3) ; Que no haga definicion previa
         ((EQ (TYPEP X3) 'LAMBDA)
                    (PUTD (PUT X1 X2
                               (PACK (LIST X1 X2)))
                          X3)
                    X1
         )
         (PUT X1 X2 X3)
         X1
  )
)

(DEFUN PROPERTY (NAME KEY OBJECT)
  ((OR (NOT (SYMBOLP NAME))
       (NOT (SYMBOLP KEY))) ((ERROR 2)))
  (PUTPROP NAME KEY OBJECT)
)

(MOVD 'BREAK 'RUPTURA)

(DEFUN BREAK (OBJ MSG)
  ((NULL (CAR OBJ)) NIL)
  ((EQ (CAR OBJ) '+) (LESSTRAP (CDR OBJ)))
  ((EQ (CAR OBJ) '-) (GATTRAP (CDR OBJ)))
  (RUPTURA OBJ MSG)
)

(DEFUN QUOTIENT (A B)
  ( (NEGATIVE (CDR (DIVIDE A B)))
                        ( (NEGATIVE A)
                                  (ADD1 (CAR (DIVIDE A B)))
                        )
```



```
                    ( SUB1 (CAR (DIVIDE A B)) )
    )
    ( CAR (DIVIDE A B) )
)

; Funciones para reconocimiento de numeros 1, -1, negativos, nones, enteros

(DEFUN ONE-P (EXP)
   (EQUAL EXP 1)   ; Observe lo puso EQ
)

(DEFUN MINUS1-P (OBJ)
  (EQ OBJ -1)
)

(DEFUN NEGATIVE (EXP)
  (< EXP 0)
)

(DEFUN ODDP (OBJ)
  ((NOT (NUMBERP OBJ)) NIL)
  (EVENP (ADD1 OBJ))
)

(DEFUN INTEGERP (EXP)
  (OR (EVENP EXP)
      (ODDP EXP))
)

; Funciones para reconocimiento de orden

(DEFUN ORDERP (OBJ1 OBJ2)
  ((NUMBERP OBJ1)
       ((NUMBERP OBJ2)
                (< OBJ1 OBJ2))
       T)
  ((NUMBERP OBJ2) NIL)
  ((GREATERP OBJ1 OBJ2) T)
)

(DEFUN ORDERED (OBJ1 OBJ2)
  ((ATOM OBJ2)(ORDERP OBJ1 OBJ2))
  ((ATOM OBJ1) T)
  ((ORDERED (CAR OBJ1)(CAR OBJ2)) T)
  ((equal (CAR OBJ1)(CAR OBJ2))(ORDERED (CDR OBJ1)(CDR OBJ2)))
)

; Funciones para reconocimiento de multiplos positivos y negativos de un numero

(DEFUN MULTIPLE (EXP1 EXP2)
       (ZEROP (MODULO EXP1 EXP2))
)
```

```
(DEFUN PRODMULT (EX1 EX2)
       (AND (PLUSP EX1)
            (ZEROP (MOD(LD EX1 EX2))
       )
)

(DEFUN NEGMULT (EX1 EX2)
       (AND (NEGATIVE EX1)
            (ZEROP (MOD(LD EX1 EX2))
       )
)

; Funciones para reconocimiento de formas de expresiones, SUMA, producto,
; exponenciación, logaritmo

(DEFUN SUM (EX1)
       (EQ (CAR EX1) '+)
)

(DEFUN PRODUCT (EX1)
       (EQ (CAR EX1) '*)
)

(DEFUN POWER (EX1)
       (EQ (CAR EX1) '^)
)

(DEFUN LOGARITHM (EX1)
      (EQ (CAR EX1) 'LOG)
)

; Funciones de reconocimiento de formas especiales de expresiones

(DEFUN NEGTERM (EX1)
       ((TERMLIST EX1) (NEGATIVE EX1))
       ((PRODUCT EX1) (NEGATIVE (CADR EX1)))
)

(DEFUN SIMOR (EX1)
       ( AND (PLUS EX1)
             (MAPLIST (QUOTE EX1) )
)

(DEFUN MAPLIST (EX1)
       ( ( AND (PLUS EX1)
               ((TERMLIST (CADR EX1)) ( (MAPLIST (CDDR EX1))
       )
)

(DEFUN NUMBER (EX1)
       ( ( OR (TERMLIST EX1) (RECIP EX1) )
       ( ( AND (PRODUCT EX1)
```



```
                (ATOM (CDDDR EX1)) ) (REC)P (CADDR EX1)))
)

; Funciones para identificar dependencia entre variables

(DEFUN FREE (EX1 EX2)
       ( (OR (equal EX1 EX2)
             (FR (GET EX1 EX2) 'DEPENDS)) NIL )
       ( LOOP
             (ATOM EX1)
             (INCF (FREE (POP EX1) EX2)) NIL)
       )
)

; Funciones de manejo de error de "+" y "*"

(DEFUN LISTRAN (LEX1 EX1 EX2)
       (SETQ EX1 (CAR LEX1))
       (SETQ EX2 (CADR LEX1))
       ( (AND (NUMBER EX1)
              (NUMBER EX2)) (= (* (NUMERADOR EX1) (DENOMINADOR EX2))
                               (* (NUMERADOR EX2) (DENOMINADOR EX1)))
       )
)

(DEFUN QUITRAN (LEX1)
       ((LISTRAN (REVERSE) LEX1))
)

(DEFUN SIMPL (LOP EX1)
       ( (SYMBOLP EX1) (LIST LOP EX1) )
       ( (APPLY (GET LOP (LAB EX1))
                (MAPLIST EX1)) ; (SIGUIENTE argumento de APPLY
       (LIST LOP EX1)
)

; Funciones aritméticas primitivas

(DEFUN MENOR (EX1 EX2)
       ( (equal EX1 EX2) EX1 )
       ( (AND (NUMBER EX1)
              (NUMBER EX2)) ( (= EX1 EX2) EX1)
                            EX2)
       (LIST 'MENOR EX1 EX2)
)

(DEFUN LCM (EX1 EX2)
       ( ABS (* EX2 (/ EX1 (GCD EX1 EX2))) )
)

(DEFUN SUSTRAE (LEX1)
       (CONS '(0) LEX1)
```

```
)

; funciones para la obtencion de valores absolutos

(DEFUN ENT (EXT)
   (AND (INTEGERP EXT)
        (NOT (ZEROP EXT)))
)

(DEFUN ABS (EXT)
  ( CNUMBER EXT) ((> EXT 0) EXT) (- EXT) )
  ( CATOM EXT)
          ( (SET EXT 'NOMABS) EXT )
          ( (EQ EXT 0NCIAT) 0PRINT )
          ( (EQ EXT 0NCIERO) 0PZERO )
          (LIST 'ABS EXT)
  )
  ( (SET (CAR EXT) 'NOMABS) EXT )
  ( (SET (CAR EXT) 'DEFUN)
          (APPLY (CAR EXT) (CONS (ABS (CADR EXT))))  )
  ( (TREE EXT %)
          (SIMPU 'ABS EXT) )
  (LIST 'ABS EXT)
)

(DEFUN ABSPROD (EXT EXZ EXS EXA LEXT)
   )
      ( (PRODUCT (SETQ LEXT EXS))
                                  (POP LEXT) )
      (SETQ LEXT (CONS LEXT))
   )
   (LOOP
      ( (ATOM LEXT)
                  (* (ABS EXZ)
                     (ABS EXZ)) )
      ( (AND
           (NOT (SYMBOLP (DEFQ EXA (POP LEXT))))
           (equal EXT (CADR EXA))
           (OR (SET (CAR EXA) 'DEFUN)
               (AND (POWER EXA)
                    (ONE (CADDR EXA)))
            ))                              (* EXZ
                                               EXA
                                               (ABS (/ EXS EXA)))
      )
   )
)
```

```
;+                          MODULO DE TERMINADORES
;

;El terminador ; es el terminador "automatico" que evalua EXP,
;asigna a la variable implicita correspondiente dicha evaluacion e
;imprime el resultado.
(DEFUN \; (LAMBDA (MEXP)
   (SETQ #TRACE NIL)
   (EVALUA-EXP MEXP #PTUSEDNA)
         ))

;El terminador $, es el terminador "mudo" que evalua EXP y asigna
;a la variable implicita correspondiente dicha evaluacion sin
;imprimirla.
(DEFUN $ (LAMBDA (MEXP)
   (SETQ #TRACE NIL)
   (EVALUA-EXP MEXP #$(SNOPRESS)
         ))
)

;El terminador ! es el terminador "explicito" que describe el
;procedimiento para realizar la evaluacion de EXP.
(DEFUN ! (LAMBDA (MEXP #MES)
   (SETQ #TRACE T)
   (SETQ #MES (EVALUA-EXP MEXP #ADMIRACION))
   (SETQ #STACK-TRACE (DEPURA-STACK #STACK-TRACE))
   (INF-PROCESSO)
   (SETQ #STACK-TRACE)
   #MES
         )
)

;El terminador &, es el terminador "dirigido" que permite al usuario
;dirigir el proceso para llegar a la evaluacion de EXP.
(DEFUN & (LAMBDA (MEXP)
   (SETQ #TRACE NIL)
   (EVALUA-EXP MEXP #AMPERSAND)
         )
)

(DEFUN (GUALES (EXT EXZ ACC)
   ((OR (NULL EXT) (NULL EXZ)) ACC)
   ((ATOM EXT EXZ) (AND ACC (EQUAL EXT EXZ)))
   (AND ((GUALES (CAR EXT) (CAR EXZ) ACC)
         ((GUALES (CDR EXT) (CDR EXZ) ACC))
)

(DEFUN DEPURA-STACK (ST-INICIAL A ST-FINAL)
  (SETQ ST-INICIAL (ELIMINA-ENTRADAS-IGUALES ST-INICIAL))
  (LOOP
```



```
   ((NULL ST-INICIAL)
         (REVERSE ST-FINAL))
   (SETQ N (POP ST-INICIAL))
   (COND ( (AND (FIXNUMP N)
                (EQUALES (CADDR N)
                         (CADDR N)) T) )
         ( (FIXNUMP N) (PUSH N ST-FINAL) )
         ( (PUSH N ST-FINAL) )
   )
  )
)

(DEFUN TA-ESTA (OBJ LST)
  ((ATOM LST) NIL)
  ((FUNCALL 'IGUALES OBJ (CAR LST) T) LST)
  (TA-ESTA OBJ (CDR LST)))

(DEFUN ELIMINA-ENTRADAS-IGUALES (LST)
  ((ATOM LST) LST)
  ((TA-ESTA (CAR LST) (CDR LST))
     (ELIMINA-ENTRADAS-IGUALES (CDR LST)))
  (ELIMINA-AUX LST LST)
)

(DEFUN ELIMINA-AUX (LST TAIL)
  ((ATOM (CDR TAIL)) LST)
  ((TA-ESTA (CADR TAIL) (CDDR TAIL))
     (RPLACD TAIL (CDDR TAIL))
     (ELIMINA-AUX LST TAIL))
  (ELIMINA-AUX LST (CDR TAIL))
)

(SETQ EJECUTABLES '(APLICA SUBSEP))

(DEFUN FLAT ( EXP )
   ((NULL EXP) ())
   ((ATOM EXP) (LIST EXP))
   (APPEND (FLAT (CAR EXP)) (FLAT (CDR EXP)))
)

(DEFUN UN-NIVEL (OBJ LST)
  (SETQ LST (FLAT OBJ))
  (LOOP
     ( (NULL LST) T)
     ( (MEMBER (POP LST) EJECUTABLES) () ) )
)

(DEFUN EVALUA-EXP (EXP F-TERMINA LOCAL AUX &RSAPL)  ;EXP en FIP sin terminador
  ((AND (UN-NIVEL EXP)
        (OR (EQ F-TERMINA #$(SNOPRESS))
            (EQ F-TERMINA #PTUSEDNA)
            (EQ F-TERMINA #ADMIRACION)))
```



```
          #(NOT (EQ (SETQ AUX (FUAL EXP)) T)) AUX)
          #(NOT (EQ (SETQ AUX (APPLY (CAR EXP) (CDR EXP))) T)) AUX)
          ( (IDENTITY T) EXP )
  )
  ((AND (UN-NIVEL EXP)
        (EQ F-TERMINA #ANDVERSAND)) EXP)
  ((EQ (CAR EXP) 'SUBSEP)
       (FUNCALL 'SUBSEP (EVALUA-EXP (CADR  EXP) F-TERMINA)
                        (EVALUA-EXP (CADDR EXP) F-TERMINA)
                        (EVALUA-EXP (CADDDR EXP) F-TERMINA)))
  ((EQ (CAR EXP) 'APLICA)
       (SETQ &RSAPL (MAPCAR 'EVALUA-EXP
                            (CDR EXP)
                            (LISTA-TERMINA (CDR EXP) F-TERMINA)))
       (APPLY (CDR EXP) &RSAPL))
;       (FUNCALL 'APLICA (EVALUA-EXP (CADR  EXP) F-TERMINA)
;                        (EVALUA-EXP (CADDR EXP) F-TERMINA)))
  (SETQ LOCAL (MAPCAR 'EVALUA-EXP
                      (CDR EXP)
                      (LISTA-TERMINA (CDR EXP) F-TERMINA)))
  ((EQ F-TERMINA #AMPERSAND) (APPEND (LIST (CAR EXP)) LOCAL))
  (APPLY (CAR EXP) LOCAL)
)

(DEFUN LISTA-TERMINA (EXP F-TERMINA LST)
  (LOOP
     ((NULL EXP) LST)
     (POP EXP)
     (SETQ LST (CONS F-TERMINA LST))
  )
)

;Las formulas de explicitacion tienen la forma (:= (exp) (exp))
;                                                 |car cadr  caddr
(PROPERTY 'APLICACION '= 'VALANDA (VALOR #CERO)
  (FUNCALL 'SUBSEP
           #CERO
           (CADR  (ATU 1 (VALOR 1) "DIMENSION") EXP"))
           (CADDR (ATU 1 (VALOR 1) "DIMENSION") EXP"))
                  )
)

;Forma de las formulas de derivacion  (:= (DERIVA exp   X) (exp))
;                                      |car cadr          caddr
;                                                 (exp)  (VAR X)
;
(PROPERTY 'APLICACION 'DERIVA 'VALANDA (VALOR #CERO)
  (FUNCALL 'SUBSEP
           #CERO
           (CADR  (ATU 1 (VALOR 1) "DIMENSION") EXP"))
           (CADDR (ATU 1 (VALOR 1) "DIMENSION") EXP"))  )
)
```



```
)

;Forma de las formulas de integracion (:= (INTEGRA exp   X) (exp))
;                                      |car  cadr          caddr
;                                                 (exp)  (VAR X)
(PROPERTY 'APLICACION '= 'VALANDA (VALOR #CERO)
  (FUNCALL 'SUBSEP
           #CERO
           (CADR  (ATU 1 (- (VALOR 1)) "DIMENSION") EXP"))
           (CADDR (ATU 1 (- (VALOR 1)) "DIMENSION") EXP"))
                  )
)

(DEFUN EQUALS (EXP1 EXP2
  #TRACE #PARAMETRO (VALOR #CERO) #DESDE #HASTA #NUMERO
  #RASTREAR #LAPSO1 #LAPSO2 #RESULTADO
  ((ZERO? 1- EXP1 EXP2)))
  ((SETQ #PARAMETRO  (VALOR #CERO) #DESDE (VALOR #CERO)
         #HASTA (VALOR #CERO) #NUMERO 0
         #RASTREAR NIL #LAPSO1 NIL #LAPSO2 NIL #TRACE NIL)
   ((ZERO? 1- EXP1 EXP2)))
  ((SETQ #PARAMETRO 'A  (VALOR #CERO) #DESDE 'B  (VALOR #CERO) #NUMERO 'X
         #RASTREAR NIL #LAPSO1 NIL #LAPSO2 NIL #TRACE NIL)
   ((ZERO? 1- EXP1 EXP2)))
)

;Aplicar(Sbr, <exp> DifVAR1 VAR1 VVAR1 VAR2 VVAR2 ... VARK VVARK), donde X que
;                    es el numero de regla
;Ej: es (Sbr exp ... ETC)
(DEFUN APLICA (&LAMBDA #LST ; Regresar su valor a las variables ajustadas
  ( (OR (EQ (LENGTH #LST) 0)
        (AND (EQ (CADR (UNPACK (CAR #LST))) '=)
             (! EVAL ENTERO (CDDR (UNPACK (CAR #LST)))  5) ) )
        (APPLY (GET 'APLICACION
                    (CADR (UNPACK (CAR #LST))))
               ((LIST
                   (VAL ENTERO (CDDR (UNPACK (CAR #LST))))
                   (CADR #LST)))
                )
  ( (AND (EQ (CADR (UNPACK (CAR #LST))) '=)
         (= (VAL ENTERO (CDDR (UNPACK (CAR #LST))))  5) )
            (APPLY (GET 'APLICA2
                        (CADR (UNPACK (CAR #LST))))
                   ((LIST #LST)) )
  (APLICA2 (CDR #LST) T)
))

;Aplicar(Sbr, <exp> DifVAR1 VAR1 VVAR1 VAR2 VVAR2 ... VARK VVARK), donde X que
;                    es el numero de regla
;Ej: es (Sbr exp ... ETC)
;Forma de las formulas de integracion (:= (INTEGRA exp   X) (exp))
;                                      |car  cadr          caddr
;                                                 (exp)  (VAR X)
```

```
(PROPERTY 'APLICAR *) '(&LAMBDA &REST
                                    &AUX
                                    &EXPR
                                    &DIFU
                                    &RESULA
                                    &STACK
                                    &VAR1
                                    &VVAR1)
  (SETQ &AUX  (VAL-ENTERO (CDDR (UNPACK (POP &LST)))))
  (SETQ &EXPR (POP &LST))
  (SETQ &DIFU (POP &LST))
  ;(VAR1 VVAR1 VAR2 VVAR2 ... Vdda Vvdda), donde X puede ser E,F,I
  (IMOT (EQUALS &DIFU
                (DIFI (CADR &LST) E)))
        (ASIGNACION(ND-SEN '(E)
         (PRINC (DIFU no corresponde a la derivada de U con respecto a E)
        ()))
  (VAR1 VVAR1 VAR2 VVAR2 ... VARN VVARN)
  (SETQ &VVAR1 (CAR &LST) &VVAR1 (CADR &LST))
  (SETQ &RESULA (NTH (- &AUX 1) *ESQUEMAS-INT*)) ;EL CAR DEL CADR ES LA EXPRES
  (SETQ &ACC (CADR (CADR &RESULA)))
  (LOOP
     ((NULL &LST))
     (SETQ &ACC (SYSUN &ACC (CAR &LST) (CADR &LST) ) )
     (PUSH (POP &LST) &LSTALN)
     (PUSH (POP &LST) &LSTALN)
  )
  (SETQ &LST (REVERSE &LSTALN))
  ((NOT (EQUALS &EXPR (* &ACC &DIFU)))
        (ASIGNACION(ID-SEN '(
        ()))
  (SETQ &ACC (CADDR (NTH (- &AUX 1) *ESQUEMAS-INT*)))
  (LOOP
     ((NULL &LST))
     (SETQ &ACC (SYSUN &ACC (CAR &LST) (CADR &LST) ) )
     (POP &LST)(POP &LST)
  )
  &ACC
                          )
)

(SETQ #STACK-TRACE &LL)

(DEFUN POST-FIJO (&OBJ &PF)
   (PACK (APPEND (UNPACK &OBJ)
                 (UNPACK " ")
                 (UNPACK &PF)))
)

(DEFUN TRACE-FUN LST ; LST = (NOMFUN (&OPT ARG2 . . . &RGN))
  ((NOT #TRACE))
  (SETQ LST (PUSH (POST-FIJO (POP LST) "(&) LST))
  (PUSH LST #STACK-TRACE)
)

(DEFUN TRACE-RET (&NOMFUN &RESULTADO)
  ((NOT (#TRACE) (&RESULTADO)
  (SETQ &NOMFUN (POST-FIJO &NOMFUN "OUT"))
  (PUSH (LIST &NOMFUN &RESULTADO) #STACK-TRACE)
  &RESULTADO
)

(DEFUN INVIERTE (LST AUX)
   (SETQ AUX (LIST (POP LST)))
   (PUSH (POP LST) AUX)
   (SETQ AUX (REVERSE AUX))
   (PUSH (POP AUX) LST)
   (PUSH (POP AUX) LST)
)

(DEFUN TRACE-PTO LST ; LST=(NOMPTO [&NOT ARG2 . . . ARGN])
  ((NOT #TRACE))
  ((OR (((FINDSTRING "@" (CAR LST))
       ((FINDSTRING "$" (CAR LST)))
            (SETQ LST (INVIERTE LST))))
  (PUSH LST #STACK-TRACE)
)

(SETQ #LST-TRACE '(
  (DERIVA_DX . ()Se llego a la funcion para derivar, con los siguientes
                 |argumentos: |))
  (DERIVA_OUT . ()El derivada obtenida fue |))
  (INTEGRA_IN . ()Se llego a la funcion para integrar, con los siguientes
                 |argumentos: |))
  (INTEGRA_OUT . ()La integral obtenida fue |))
  (FORMULA_D . ()Se utilizo la siguiente formula de derivacion |))
  (FORMULA_I . ()Se utilizo la siguiente formula para integrar |))
  (TRANSF1 . ()Se transformo la expresion |))
  (TRANSF2 . ()Se transformo la expresion |))
 )
)

(DEFUN IMP-LLEGA ()
 ; Imprimir el letrero correspondiente a la llegada (CAR #DATO)
  ((IMP-STRING (CDR (ASSOC (CAR #DATO) #LST-TRACE)))
 ; Imprimir los  argumentos
  (IMP-EXPS (CADR #DATO))
)

(DEFUN IMP-SALE ()
 ; Imprimir el letrero correspondiente a la salida (CAR #DATO)
  ((IMP-STRING (CDR (ASSOC (CAR #DATO) #LST-TRACE)))
 ; Imprimir los argumentos que regresa la funcion
  (IMP-EXPS (LIST (CADR #DATO)))
```

```
)

(DEFUN (IMPRIME-STACK (LIST)
  (LOOP
    (CHECK LIST)
    (PRINT (POP LIST))
  )
)

;Imprime el proceso registrado en el #STACK-TRACE
(DEFUN (MP-PROCESO (&OPTIONAL &BASES)
  (SETQ #STACK-TRACE (REVERSE #STACK-TRACE))
  (LOOP
    (CHECK-NULL #STACK-TRACE))
    (SETQ #DATO (POP #STACK-TRACE))
    (COND
      ;------ TRACE-FUN REGISTRO (NOMFUN_IN (ARG1 ARG2 ... ARGN)) ------
       ((FIND-STRING "_IN" (CAR #DATO))
                      (MP-LISTA))
      ;------ TRACE-RET REGISTRO (NOMFUN_OUT REGRESO) ----------------
       ((FIND-STRING "_OUT" (CAR #DATO))
                      (MP-SALE)  )
      ;------ TRACE-VAR REGISTRO ------------------------------------
      ;              (SETQ VIEW (VAR1 ASOC1) (VAR2 ASOC2) (VARN ASOCN))
       ((FIND-STRING "=>" (CAR #DATO))
         (PRINC [Se tiene para integrar la expresion: ])
         (MP-EXPS (LIST (POP #DATO)))
         (ESPERA)
         (SETQ #BASES (CDR #DATO))
         ; Imprime el antecedente enviado
         (COND ( #BASES
                 (PRINC [Realizando la(s) asociacion(es): ])(ESPERA) ))
         ((LOOP
            (CHECK-NULL #BASES))
            (PRINC (CAAR #BASES))
            (PRINT [ con la expresion: ])
            (ESPERA)
            (MP-EXPS (CDAR #BASES))
            (COND ( (LET #VALUES (ESPERA) ) )
            (POP #BASES)
         )
         (ESPERA)
         ; Imprime el retorno correspondiente a la llegada a una
         ; formula de integracion
         (MP-STRING (CDR (ASSOC '(FORMULA_I) #LIST-TRACE)))
         (ESPERA)
         ; Imprime la clave y su formula
         (MP-FORMULA (POP #DATO))
         ((ERROR)) ; Para no quitar su puesto a la integral

      )
```



```
; TRACE-PTO REGISTRO (VAR1 VAR2 (VAR1 ASOC1) (VAR2 ASOC2) (VARN ASOCN))
      ((FIND-STRING "=>" (CAR #DATO))
        ; Imprime el antecedente enviado
        (PRINC [Se tiene para obtener la expresion: ])
        (MP-EXPS (LIST (POP #DATO)))
        (ESPERA)
        (SETQ #BASES (CDR #DATO))
        (PRINC [Realizando la(s) asociacion(es): ])(ESPERA)
        ((LOOP
           ((NULL #BASES))
           (PRINC [con ])(PRINC (CAAR #BASES))(PRINC ] ))
           (PRINT [a expresion ])
           (MP-EXPS (CDAR #BASES))
           (COND ( (CDR #BASES) (ESPERA) ) )
           (POP #BASES)
        )
        (ESPERA)
        ; Imprime el retorno correspondiente a la llegada a una
        ; formula de derivacion
        (MP-STRING (CDR (ASSOC '(FORMULA_D) #LIST-TRACE)))
        (ESPERA)
        ; Imprime la clave y su formula
        (MP-FORMULA (POP #DATO))

      )

      ;------ TRACE-PTO REGISTRO (TRANSFN (CHAR VARS))
      ((FIND-STRING "=TRANS=" (CAR #DATO))
       ;Imprime el tipo de transformacion realizada dependiendo de e
       ((MP-STRING (CDR (ASSOC (CAR #DATO) #LIST-TRACE)))
       (ESPERA)
       ; Imprime la expresion antes
       ((MP-EXPS ((LIST (CDADR #DATO)))
       (ESPERA)
       (PRINC [ A la siguiente expresion equivalente ])
       (ESPERA)
       ((MP-EXPS (CDADR #DATO))
      )
     );COND
    (ESPERA)
  );LOOP
)

(DEFUN (ESPERA (&OPTIONAL MSG &C (CLR)
  (SETQ &R (READ) &C (COLUMN))
  (((>= &R (CURSOR-POSITION-WINDOW)))
     (REGISTRA-EVENT "Teclee <RETURN> para continuar")
     ((LOOP
       (SETQ TECLA (READ-BYTE))
       ((OR TECLA (NULL TECLA)) (SETQ-TECLA))
                  (CLEAR-SCREEN)
                  (SETQ-TECLA)
```

```
                (CURRENT-WINDOW)
                (SET-CURSOR BM BC)
                (TERPRI A)
                (SET-CURSOR 1+ (SUB1 (WINDOW-ROWS)) 3) 0)
            )
        )
    )
    (TERPRI)
)

;Formas de las formulas de derivacion 1-: (DERIVA exp    X)   (exp1)
;                                         (cat (add'    (add0' X )   (add0')
;                                                (cadr  (add0' X )
(DEFUN INP-FORMULA (PREY BMAT PCONC BC)
  (COND ((!=NOSTRING "0" PREY)
         (SETQ BMAT (CADR
                    (NTH 1- (VAL-ENTERO (CDDR (UNPACK PREY))) 1)
                        "ESQUEMAS-DER")))
         (SETQ PCONC (CADDR
                    (NTH 1- (VAL-ENTERO (CDDR (UNPACK PREY))) 1)
                        "ESQUEMAS-DER"))) )
        ((!=NOSTRING "0)^ PREY)
         (SETQ BMAT (CADR
                    (NTH 1- (SOL-ENTERO (CDDR (UNPACK PREY))) 1)
                        "ESQUEMAS-EXP")))
         (SETQ PCONC (CADDR
                    (NTH 1- (VAL-ENTERO (CDDR (UNPACK PREY))) 1)
                        "ESQUEMAS-EXP")))
        ) )
  (SETQ BC (PRINTEXP BMAT))
  (COND ((MEMBER "INTEGRA"
                 (FLAT BMAT))
         (POK-CURSOR (SUB1 (ROW)) BC)))
  (PRINC ) --> ())
  (PRINTEXP PCONC)
  (ESPERA)
)

(DEFUN INP-STRING (ALST-STR BM)
  (LOOP
     (PRINC (POP ALST-STR))
     (COND ((!=NIL BM)(SETQ BM (SUB1 BM))))
     (!=0 (ZEROP BM)
          (NULL ALST-STR)))
     (TERPRI)
  )
)

(DEFUN INP-EXPS (ALST-STR)
  (LOOP
     (SPACES 2)
     (PRINTEXP (CAR ALST-STR))
```



```
     (POP ALST-STR)
     ((NULL ALST-STR))
     (PRINC #CONA)
  )
)
```

```
        )
      )
    )
)

(DEFUN SUBSEP (VALANDOA &REST ARGS
                        &AUX? &OPTIONAL)
  ; &AUX? guarda la lista de asociaciones, la calcula EMPATE
  ((ATOM ARGS)
           (SETQ &AUX? (CONS (CONS ARGS ARGS) &AUX?))
           (SUBLIS &AUX? ARGS))
  (SETQ &OPTIONAL T)
  ((NULL (EMPATE/& ARGS) ARGS))
                        (SETQ &OPTIONAL (APPEND &OPTIONAL ARGS))
                        ((AND (EMPATEACERO &AUX?)
                              &OPTIONAL)
                              (SUBLIS &AUX? &OPTIONAL))
                        (ASIGNACION=0 T))
  )
  ((AND (EMPATEACERO &AUX?)
        &OPTIONAL)
                        (SUBLIS &AUX? &OPTIONAL))
  (ASIGNACION=0 T))
    )
)

(DEFUN EMPATE (&AUX? ARGS)
   ; &AUX? viene de SUBSEP
   ((NOT (EQUAL (CAR ARGS)
                (CAR &AUX?))) (ASIGNACION=0 T))
   (SETQ ARGS)
   (SETQ &AUX?)
   (LOOP
      ((NULL ARGS) &AUX?)
      ((AND ( (ATOM (CAR ARGS))
                        (SETQ &AUX? (CONS (CONS (CAR ARGS) (CAR &AUX?)) &AUX?))
                                  &AUX?) ) )

            ( (SETQ ARGS (APPEND (CDAR ARGS) (CDR ARGS)) (CDR &AUX?)) )
      )
      ((NOT ARGS) &AUX?)
   )
)

(DEFUN EMPATE &(CONSTANTE) (VALANDOA &AUX? ARGS)
;3! &Aux? es a algebraicamente debe empatar con &'n haciendo nil
;3! ARGS es a algebraicamente debe empatar con c^& haciendo nil
NIL)

;-----------------------------------------------------------------
; Rutinas de manejo de variables de control para multiplicidad algebraica
;-----------------------------------------------------------------



;Las siguientes funciones verifican que la variable de control correspondiente
;sea multiplo de 2, 3 o 5, dependiendo del tipo de la expresion que integra, si
;numerica, si es una suma o si no es suma.

(DEFUN NUMNUM (EXP)
    ((INTEGERP EXP) (MULTIPLE NUMNUM 2))
    ((SUM EXP) (MULTIPLE NUMNUM 5))
    (MULTIPLE NUMNUM 3)
)

(DEFUN DENNUM (EXP)
   ((INTEGERP EXP) (MULTIPLE NUMDEN 2))
   ((SUM EXP) (MULTIPLE NUMDEN 5))
   (MULTIPLE NUMDEN 3)
)

(DEFUN DENDEN (EXP)
    ((INTEGERP EXP) (MULTIPLE NUMDEN 2))
    ((SUM EXP) (MULTIPLE NUMDEN 5))
    (MULTIPLE NUMDEN 3)
)

(DEFUN NUMDEN (EXP)
    ((INTEGERP EXP) (MULTIPLE DENNUM 2))
    ((SUM EXP) (MULTIPLE DENNUM 5))
    (MULTIPLE DENNUM 3)
)

(DEFUN DEFBAS (EXP)
   ((NUMBER EXP) (MULTIPLE BASEBAS 2))
   ((SUM EXP) (MULTIPLE BASEBAS 5))
   (MULTIPLE BASEBAS 3)
)

(DEFUN BASEBAS
  ( (NUMBER EXP) (MULTIPLE BASEBAS 2) )
  ( (SUM EXP) (MULTIPLE BASEBAS 5) )
  (MULTIPLE BASEBAS 3)
)

;-----------------------------------------------------------------
; Rutinas de manejo de variables algebraicas
;-----------------------------------------------------------------

(DEFUN DIVISOR (EXP EXP) BASEBAS BASEBAS (EXP)
   ((ATOM EXP) (EXP)
      (SETQ EXP (MULTIPLE (* EXP EXP) (EXP)) (EXP))
      ((ATOM (CDR) (EXP)
   )
)

(DEFUN EXPAND (EXP) BASEBAS BASEBAS NUMNUM DENDEN NUMDEN DENNUM


```

```
;*****************************************************************
;*                        MODULO ALGEBRAICO                      *
;*****************************************************************
;
;-----------------------------------------------------------------
; Rutinas de extension de subexpresiones
;-----------------------------------------------------------------

(PROPERTY 'NUMERADOR '* '(LAMBDA (EX1 EX2)
                  ((MEGGDET EX2) 1)
                   EX1
          ))

(PROPERTY 'NUMERADOR '+ '( LAMBDA (EX1 EX2)
                  (* (NUMERADOR EX1) (NUMERADOR EX2))
          ))

(DEFUN NUMERADOR ((EX) &PUN(PTC &ALINALN)
       ( (ATOM EX1) EX1)
       ( (APPLY (GET 'NUMERADOR (CAR EX)) (ARGEX EX)) )
         EX1)

(PROPERTY 'DENOMINADOR '* '(LAMBDA (EX1 EX2)
                  ( (RECCDET EX2) (* EX1 ( EX2)) )
                  ) ))

(PROPERTY 'DENOMINADOR '+ '(LAMBDA (EX1 EX2)
                  (* (DENOMINADOR EX1) (DENOMINADOR EX2))))

(DEFUN DENOMINADOR (EX1 &PARAMS) &OPTIONAL)
       ( (ATOM EX1) 1)
       ( (APPLY (GET 'DENOMINADOR (CAR EX1)) (ARGEX EX1)))
       ))

(DEFUN CDR(X (EX1)
  ((INTEGERP EX1) 1)
  ((PRODUCT EX1)
          ( ( INTEGERP (CADR EX1) )
                ( CRECIP (CADDR EX1)) (MAPPEND (CDDDR EX1)) )
                ( MAPPEND (CDDR EX1) )
          )
          ( (RECIP (CADR EX1)) (MAPPEND (CDDR EX1)) )
          EX1 )
  ((RECIP EX1) 1)
  EX1)

(DEFUN CDR*I(CUENTE (EX1) ; SAKIT
  ((INTEGERP EX1) EX1)
  ((PRODUCT EX1)
         ( ((INTEGERP (CADR EX1))
                ( (RECIP (CADDR EX1)) (LIST '* (CADR EX1) (CADDR EX1)))
                (CADR EX1)
         )
```



```
          ( (RECIP (CADR EX1)) (CADR EX1))
          ) )
  ((RECIP EX1) EX1)
  ))

(DEFUN BASE (EX1)
       ( (POWER EX1) ( (RECIP EX1) EX1)
                     (CADR EX1)
       )
       EX1)

(DEFUN EXPONENTE (EX1)
       ( (POWER EX1)
                     ( (RECIP EX1) 1)
                     (CADDR EX1) )
       ))

(DEFUN CUADRADO (EX1 LEX1 LEX2 LEX3) ; Local LEX1, LEX2, LEX3
   ((NUM EX1)
          (SETQ LEX1 (ADVANCE (CDR EX1)))
          (LOOP
              (SETQ LEX2 (MERGESUM (* (SETQ EX1 (POP LEX1)) 2) LEX2))
              ((ATOM (SETQ LEX3 LEX1)) (MERGUM LEX2))
              (SETQ EX1 (* 2 EX1))
              (LOOP
                  (SETQ LEX2 (MERGESUM (* EX1 (POP LEX3)) LEX2))
                  ((ATOM LEX3))
                  )
          )
   )
   (* EX1 EX1)
)

(DEFUN EMPAQUETADO (&ALIST &ELEM &LIST &ALIST &NOMATCH)
   ;EMPAQUETADO ES VERDADERO CUANDO &ALIST ES UNA LISTA
   ;DE ASOCIACIONES CONSISTENTE
   ;DE LO CONTRARIO ES FALSA
   (SETQ &LIST &ALIST)
   (LOOP
     ((NULL (CDR &LIST)) ; Termina cuando queda solo un dotpair
     ((DEBUT!?? &NOMATCH) ))
     (SETQ &ALIST (CDR &LIST))
     (SETQ &ELEM (POP &LIST))
     (LOOP
        ((NULL &ALIST) T)
        (DEBUT!?? &NOMATCH) ))
        (TRUE ((EQUAL (CAR &ELEM)
                      (CAR (CAR &ALIST))) ((EQUAL (CDR &ELEM)
                                                  (CDR (POP &ALIST))))
                                          (SETQ &NOMATCH T)
              )
              ((POP &ALIST))
```

```
                    @EXPBAS)
  (SETQ #TRGEXPO -2)
  (SETQ #PUNTEXPO 6)
  (SETQ #NUMNUM (SETQ #DENDEN (SETQ #BASEXP (SETQ #EXPBAS 50))))
  (SETQ #NUMDEN 3)
  (EVAL EXT)
)

(DEFUN EDEXP (EXP1 EXP2
  #TRIG #PUNTEXPO #NUMNUM #DENDEN #DENNUM #NUMDEN
  #BASEXP #EXPBAS #LOGEXPO #TRGEXPO)
  ((ZEROP )- EXP1 EXP2)) (PRINT '[Son expresiones equivalentes-]) )
  (SETQ #PUNTEXPO  0  #NUMNUM 0  #DENDEN 2  #DENNUM 0  #NUMDEN 0
        #BASEXP -30  #EXPBAS 30 #LOGEXPO 0 #TRGEXPO 0)
  ((ZEROP (- EXP1 EXP2)) (PRINT '[Son expresiones equivalentes-]) )
  (SETQ #PUNTEXPO  0  #NUMNUM 30  #DENDEN 30   #DENNUM 30  #NUMDEN 0
        #BASEXP -30  #EXPBAS 30  #LOGEXPO 30  #TRGEXPO 30)
  ((ZEROP (- EXP1 EXP2)) (PRINT '[Son expresiones equivalentes-]) )
  (PRINT '[NO son expresiones equivalentes-])
)

(DEFUN TRGEXPO (EXP1 #TRGEXPO #NUMNUM #DENDEN #DENNUM #PUNTEXPO)
  (SETQ #PUNTEXPO 6)
  (SETQ #DENNUM -30)
  (SETQ #NUMNUM (SETQ #DENDEN 30))
  (EVAL EXT)
)

(DEFUN LOGEXPO (EXT #LOGEXPO)
  (EVAL EXT)
)

;Paquete Opcional de Factorizacion y comun denominador

(DEFUN CONTENT (LEX1 LEX2)
  ((ATOM LEX1)
        (CATOM LEX2) NIL)
        ((AND (DEMON (CAR LEX2))
              (NEGATIVE #DENNUM)
              (DENNUM (* (CAR LEX2) -1)))
                                    (CONS (POP LEX2)
                                          (CONTENT NIL LEX2)))
        )
        (CONTENT1 (BASE (CAR LEX1)) LEX1 LEX2)
  )
  (CONTENT1 (BASE (CAR LEX1)) LEX1 LEX2)
)

(DEFUN CONTENT1 (EX1 LEX1 LEX2 EX2)
     ((ATOM LEX2) (CONTENT LEX2 LEX1))
     ((SETQ EX2 (BASE (CAR LEX2)))
     ((equal EX1 EX2)
```



```
          (SETQ EX1 (MEMOR (EXPONENTE (POP LEX1))
                           (EXPONENTE (POP LEX2))))
          ((EQ (CAR EX1) 'MEMOR) (CONTENT LEX1 LEX2))
          (SETQ EX2 (* EX2 EX1))
          ((DEMON EX2)
                ((LAND (NEGATIVE #DENNUM)
                       (DENNUM (* EX2 -1)))
                                 (CONS EX2 (CONTENT LEX1 LEX2)))
                (CONTENT LEX1 LEX2)
          )
          ((AND (NEGATIVE #NUMNUM)
                (NUMNUM EX2))
                          (CONS EX2 (CONTENT LEX1 LEX2)))
          )
          (CONTENT LEX1 LEX2)
     )
     ((ORDERED EX1 EX2)
          (SETQ EX1 (POP LEX1))
          ((AND (DEMON EX1)
                (NEGATIVE #DENNUM)
                (DENNUM (* EX1 -1)))
                                 (CONS EX1 (CONTENT1 EX2 LEX2 LEX1)))
          (CONTENT1 EX2 (EX2 LEX1))
     (SETQ EX2 (POP LEX2))
     ((AND (DEMON EX2)
           (NEGATIVE #DENNUM)
           (DENNUM (* EX2 -1)))
                          (CONS EX2 (CONTENT1 EX1 LEX1 LEX2)))
     (CONTENT1 EX1 LEX1 LEX2)
)

(DEFUN CANCEL (LEX1 LEX2)
       ((ATOM LEX1) LEX2)
       (CANCEL1 (BASE (CAR LEX1)) LEX1 LEX2)
)

(DEFUN CANCEL1 (EX1 LEX1 LEX2 EX2)
   ((ATOM LEX2)
          (CONS (* EX1 (- (EXPONENTE (POP LEX1))))
                (CANCEL LEX1 LEX2))
   )
   ((equal EX1 (SETQ EX2 (BASE (CAR LEX2))))
          (SETQ EX1 (EXPO (LIST '*
                                (EXPONENTE (POP LEX2))
                                (EXPONENTE (POP LEX1)))))
          ((ZEROP EX1) (CANCEL LEX1 LEX2))
          (CONS (* EX2 EX1)
                (CANCEL LEX1 LEX2))
   )
   ((ORDERED EX1 EX2)
          (CONS (* EX1
                   (- (EXPONENTE (POP LEX1))))
```

```
                    (CANCEL LEN1 LEN2))
  )
  (CONS (POP LEN2)
        (CANCEL? EXT LEN1 LEN2))
))

(DEFUN FACTOR (LEN1 LEN2 LEN3 LEN4 EXT)
  (SETQ EXT T)
  (SETQ LEN2 LEN1)
  (SETQ LEN4 LEN2)
  (
    ((INTEGERP (CAR LEN1))
        ((INTEGERP (CAR LEN2))
                    (
                      (AND (NEGATIVE #NUMNUM)
                           (NUMNUM (CAR LEN1))
                                (SETQ EXT (NOT EXT)) (CDR LEN2)))
                                (AND (NEGATIVE (CAR LEN1))
                                     (NEGATIVE (CAR LEN2)))
                                          (SETQ EXT (- EXT)))
                    )
                    )
                    (SETQ LEN1 (CDR LEN1))
                    (SETQ LEN2 (CDR LEN2))
        )
        (SETQ LEN1 (CDR LEN1))
  )
  ((INTEGERP (CAR LEN2)) (SETQ LEN2 (CDR LEN2)))
  (
  (
    ((ONEP (CAR LEN1))
           ((ONEP (CAR LEN2))
               (
                ((AND (NEGATIVE #DENNUM)
                      (DENNUM (CADR (CAR LEN1)))
                       (SETQ EXT
                            (* EXT
                               (* (LCM (CADR (CAR LEN1))
                                       (CADR (CAR LEN2)))
                                  -1)))
                )
                )
                (SETQ LEN1 (CDR LEN1))
                (SETQ LEN2 (CDR LEN2))
           )
           (
              ((AND (NEGATIVE #DENNUM)
                    (DENNUM (CADR (CAR LEN1))))
                                  (SETQ EXT (* EXT (CAR LEN1))))
           )
           (SETQ LEN1 (CDR LEN1))
    )

                                                                  ((ONEP (CAR LEN2))
                                                                   (
                                                                     ((AND (NEGATIVE #DENNUM)
                                                                           (DENNUM (CADR (CAR LEN2))))
                                                                                                 (SETQ EXT (* EXT (CAR LEN2))))
                                                                     )
                                                                     )
                                                                     (SETQ LEN2 (CDR LEN2))
                                                                   )
                                                                 )
                                                                 (SETQ LEN1 (CONTENT LEN1 LEN2))
                                                                 ((AND (ONE-P EXT) (ATOM LEN1)) NIL)
                                                                 (#NPROD (#MERGEPROD EXT (#MERGEPROD (* (*TRAP (CONS (* EXT -1)
                                                                                                                     (CANCEL LEN1 LEN3)))
                                                                                                        (*TRAP (CONS (* EXT -1)
                                                                                                                     (CANCEL LEN1 LEN4))))
                                                                                                    LEN1)))
)

(DEFUN EXPR (EXT
             #TRIGEXPO #PWREXPO #NUMNUM #DENDEN #DENNUM #NUMDEN #BASEXP #EXPBAS)
  (SETQ #TRIGEXPO 2
        #PWREXPO 0
        #NUMNUM  1
        #DENDEN  1
        #EXPBAS  1
        #DENNUM -1
        #BASEXP -1
        #NUMDEN 0)
  (EVAL EXT)
)

(DEFUN FACTORIZE (EXT
     EXT #DENNUM #BASEXP #EXPBAS #PWREXPO #NUMDEN #NUMNUM #DENDEN) ; Locals
  (LOAD '(IDENTITY F:FACTORIZE)
        (PRINT 'FACTORIZE_EXT:)
        (PRINT EXT)))
  (SETQ #DENNUM -1
        #BASEXP -1
        #EXPBAS  0
        #PWREXPO  0
        #NUMDEN   0
        #NUMNUM  -1
        #DENDEN  -1)
  ( ((#PRODUCT EXT) (POP EXT))
    (SETQ EXT (CONS EXT)) )
  (SETQ EXT T)
  (LOOP
  ;   (PRINT EXT:)(PRINT EXT)
  ;   (PRINT EXT:)(PRINT EXT)
      (SETQ EXT (* EXT (- (EXPT (BASE (CAR EXT)))
                          ((NTH (#EXPONENTS (POP EXT)))
```

```
                    )
                )
            )
            (ATOM EXP1 EXP2)
        )
    )

(DEFUN CONJ (EXP1 EXP2 EXP3 )Locales EXP2 EXP3
    (SETQ EXP (EXPANDE EXP1))
    (
     (CLEAR EXP1) (POP EXP1)
     (SETQ EXP1 (CONS EXP1))
    )
    (SETQ EXP2 (SETQ EXP3 1))
    (LOOP
        (
          ((FREE (CAR EXP)  3)) (SETQ EXP2 (* EXP2 (POP EXP1))))
          (SETQ EXP3 (* EXP3 (POP EXP)))
        )
        ((ATOM EXP1) (- EXP2 EXP3))
    )
)

(DEFUN RACIONALIZA (EXP1 EXP2)  ;Local EXP
    (/ (EXPANDE (* NUMERADOR (SETQ EXP1 (EXPD EXP1)))
                (SETQ EXP2 (CONJ (SETQ EXP1 (DENOMINADOR EXP1))))
       (EXPANDE (* EXP1 EXP2)))
)

(DEFUN DERIVADA (EXP
                 EXP2 EXP3 LEXP) ; Locales
    ; Flags de PRUEBA la variable INDET
    ;(FREE EXP INDET))
    (((PRODUCT EXP1 (SETQ LEXP (CDR EXP1))) (SETQ LEXP (CONS EXP1)))
    (LOOP
        ((NOT (FREE (SETQ EXP3 (POP LEXP)) INDET))
                        ((AND (equal (BASE EXP3) INDET)
                              (NUMBER (SETQ EXP2 (EXPONENTE EXP3)))
                              (FREE (SETQ EXP (/ EXP EXP2)) INDET))
                                                    (CONS EXP2 EXP))
                        )
    )
)

;PRIMITIVA(EXP,INDET) regresa NIL si EXP no es un termino o una suma de terminos,
;cada uno de los cuales es libre de INDET o es una potencia numerica de
;INDET, tal vez multiplicada por un coeficiente. De esta manera regresa
;(integrar,coeff) para el termino que tiene la potencia mas alta de INDET
;en EXP

(DEFUN PRIMITIVA (EXP INDET EXP2 EXP3)
    ((FREE (SETQ EXP (EXPANDE EXP)) INDET) (CONS 0 EXP))

MODULO ALGEBRAICO                    F-9                    APENDICE F
```

```
    (((SUM EXP) (POP EXP)) (SETQ EXP (CONS EXP)))
    (LOOP
      ( (NULL (SETQ EXP2 (REDUCE' (POP EXP)))) NIL)
      (
        ( (ATOM EXP2) )
        ( (NULL EXP) (SETQ EXP3 EXP2) )
        ( (= (CAR EXP2) (CAR EXP)) )
        ( (> (CAR EXP2) (CAR EXP)) (SETQ EXP3 EXP2) )
        ( SETQ EXP (CONS (POP EXP) (= EXP2 (CDR EXP))) )
      )
      ( (FREE EXP INDET) EXP)
    )
)

;POLY regresa (quotient,remainder) de EXP1/EXP2 con respecto a INDET

(DEFUN POLY (EXP1 EXP2 INDET
             EXP3 EXP4 EXP5 EXP6 EXP7 EXP8)
       ( (FREE EXP2 INDET) (CONS (/ EXP1 EXP2) 0) )
       (SETQ EXP3 (DENOMINADOR (SETQ EXP1 (FACTORIZA EXP1))))
       (SETQ EXP1 (NUMERADOR EXP1))
       (SETQ EXP5 (DENOMINADOR (SETQ EXP2 (FACTORIZA EXP2))))
       (
        (SETQ EXP5 (PRUEBA (SETQ EXP2 (NUMERADOR EXP2)) INDET))
                (SETQ EXP6 0)
                (LOOP
                   ((NULL (SETQ EXP7 (PRUEBA EXP INDET)))
                                                (CONS (/ EXP1 EXP) 0))
                   ((NEGATIVE (SETQ EXP8 (- (CAR EXP) (CAR EXP5))))
                                                (CONS (/ (* EXP6 EXP4) EXP2)
                                                      (/ EXP1 EXP2)) )
                   (SETQ EXP2 (* EXP3 (CDR EXP2)))
                   (SETQ EXP6 (+ (* EXP6 (CDR EXP5))
                                 (* (^ INDET EXP8) (CDR EXP7))))
                   (SETQ EXP1 (EXPD (- (* EXP1 (CDR EXP5))
                                       (* (^ INDET EXP8) (CDR EXP7) EXP2))))
                )
       )
       ((CONS (/ EXP1 EXP2) 0))
)

(DEFUN PARTF (EXP1 EXP2 INDET)
     (CAR (POLY EXP1 EXP2 INDET))
)

(DEFUN RESTO (EXP1 EXP2 INDET)
     (CDR (POLY EXP1 EXP2 INDET))
)

;MCD regresa GCD(EXP1,EXP2), COFACTOR(EXP1), COFACTOR(EXP2) con respecto a INDET
(DEFUN MCD1 (EXP1 EXP2 INDET EXP3 EXP4 EXP5 EXP6 EXP7)
     (SETQ EXP6 EXP2)
                          ^

MODULO ALGEBRAICO                    F-10                   APENDICE F
```

```
      (SETQ EXP4 (SETQ EXP5 0))
      (LOOP
        ((ERROR (CDR (SETQ EXP7 (POLY EXP1 EXP2 INDET)))) (LIST EXP2 EXP5 EXP4))
        (SETQ EXP1 EXP2)
        (SETQ EXP2 (CDR EXP7))
        (SETQ EXP3 (- EXP4 (* (SETQ EXP4 EXP3) (CAR EXP7))))
        (SETQ EXP5 (- EXP6 (* (SETQ EXP6 EXP5) (CAR EXP7))))
      )
)

;PARTFRAC regresa la expancion de la fraccion parcial de EXP relativa a INDET
(DEFUN PARTFRAC (EXP INDET EXP2)
     (- (CAR (SETQ EXP (POLY (NUMERADOR (SETQ EXP (FACTORIZA EXP1)))
                             (SETQ EXP2 (DENOMINADOR EXP1))
                             INDET
                      )
          )
        )
        (PARF (CDR EXP) EXP2))
)

(DEFUN PARF (EXP1 EXP2 EXP3)
    (((ERROR (SETQ EXP1 (CDR (POLY EXP1 (SETQ EXP2 (FACTORIZA EXP2)) INDET)))) 0)
    ((PRIMLIST EXP2)
                (((FREE (CAR (SETQ EXP3 (MCD1 (/ EXP2 (CADR EXP2))
                                              (CADR EXP2)
                                              INDET
                                              )))
                        INDET)
                    (+ (PARF (/ (* EXP1 (CADR EXP3)) (CAR EXP3))
                             (CADR EXP2))
                       (PARF (/ (* EXP1 (CAADR EXP3)) (CAR EXP3))
                             (/ EXP2 (CADR EXP2))))
                )
                (/ (PARFRAC 0)
    )
    (+ (/ (CDR (SETQ EXP3 (POLY EXP1 (BASE EXP2) INDET)))
          EXP2)
       (PARF (CAR EXP3)
             (/ EXP (BASE EXP2)))
    )
)

;POLY regresa el polinomio gcd de EXP1 y EXP2 con respecto a INDET
(DEFUN POLY (EXP1 EXP2 INDET)
     (CAR (POLY EXP1 EXP2 INDET 0))
)

;DIVINT regresa EXP1 convertido gcd del numerador y denominador con
;respecto a INDET

(DEFUN DIVINT (EXP1 INDET EXP2)

MODULO ALGEBRAICO                    F-11                   APENDICE F
```

```
    (((FREE (SETQ EXP2 (NUMERADOR (FACTORIZA
                                    (POLY (NUMERADOR (SETQ EXP1
                                                           (FACTORIZA EXP1)))
                                          (DENOMINADOR EXP1)
                                          INDET)
                                    )
                       )
            )
            INDET)
           (EVAL EXP1))
     (EVAL (FACTORIZA (/ (CAR (POLY (NUMERADOR EXP1) EXP2 INDET))
                         (CAR (POLY (DENOMINADOR EXP1) EXP2 INDET))))))
)

MODULO ALGEBRAICO                    F-12                   APENDICE F
```

```
;-----------------------------------------------------------------------
;                          MODULO DE CALCULO
;-----------------------------------------------------------------------
;-----------------------------------------------------------------------
;   Funciones para marcar dependencia de variables e|: DEPENDE(x_y_z)
;-----------------------------------------------------------------------

(DEFUN DEPENDE! (EX1 LEXZ)
   (LOOP
      ( ATOM LEXZ) )
      (PUT EX1 (POP LEXZ) 'DEPENDE)
   )
)

(DEFUN DEPENDE LST  ; (DEPENDE (x 0 Y Z)) SIGNIFICA QUE U(X,Y,Z)
   (DEPEND! (CAR LST) (CDR LST))
)

(DEFUN UNDEPEND! (EX1 LEXZ)
    (LOOP
       ( ATOM LEXZ))
       (REMPROP EX1 (POP LEXZ))
    )
)

(DEFUN NODEPEND! LST
  (UNDEPEND! (CAR LST) (CDR LST))
)

(DEFUN ZERONBREAK (&REAK #RK#)
   ((NOT ZERONBREAK)
                 (THROW 'ZERONBREAK))
   (&REAK &REAK #RK#)
)

;-----------------------------------------------------------------------
;    Funciones de Derivacion Simbolica
;-----------------------------------------------------------------------

(DEFUN DIFNEN2 (LEX1)
   ;Flujo de DIF1 : INDET
   ((ATOM LEX1)
               (ADJOIN INDET1))
   ((INTEGERP (CAR LEX1))
                     (ADJOIN (POP LEX1) (DIFNEN2 LEX1)))
   ((EQUAL (CAR LEX1) INDET)
            (( OR (ATOM (CDR LEX1))
                (NOT (INTEGERP (CADR LEX1))) )
                              (ADJOIN INDET (ADJOIN 2 (CDR LEX1))))
            )
            (ADJOIN INDET (ADJOIN (+ 1 (CADR LEX1)) (CDDR LEX1)))
   )
```

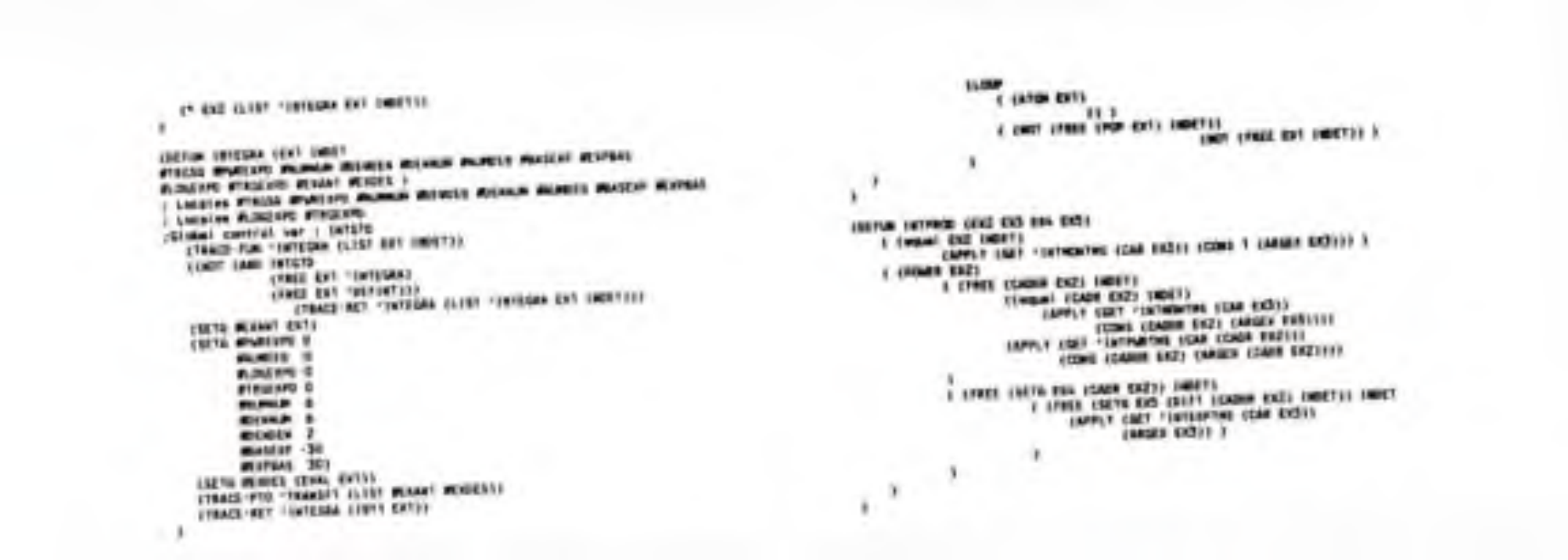


```
   ((ORDERED INDET (CAR LEX1))
                        (ADJOIN INDET LEX1))
   (ADJOIN (POP LEX1) (DIFNEN2 LEX1))
)

(DEFUN DIF1 (EX1 INDET)
   ;(FREE EX1 INDET) 0;
   ((equal EX1 INDET) 1)
   ((ATOM EX1)
            (LIST 'DERIVA EX1 INDET))
   ((EQ (CAR EX1) 'DERIVA)
            (ADJOIN 'DERIVA (ADJOIN (CADR EX1) (DIFNEN2 (CDDR EX1)))))
   ( (APPLY (GET 'DIF1 (CAR EX1)) (ARGZA EX1)) )
   (LIST 'DERIVA EX1 INDET)
)

(DEFUN DIFN (EX1 LEX1
             EX2 EX3) ;Locales EX2 _ EX3
   (LOOP
      ((ATOM LEX1) EX1)
      (SETQ EX2 (POP LEX1))
      (
         (((INTEGERP (CAR LEX1))
                  ((PLUSP (SETQ EX3 (POP LEX1)))
                           (LOOP
                              (SETQ EX1 (DIF1 EX1 EX2))
                              ((ZEROP (SETQ EX3 (- EX3 1)))
                                                        EX1)
                           )
                  )
                  (LOOP
                      ( (PLUSP (SETQ EX3 (+ EX3 1))) EX1)
                      (SETQ EX1 (INTEGRA EX1 EX2))
                  )
         )
         (SETQ EX1 (DIF1 EX1 EX2))
      )
 )
)

(DEFUN DERIVA LEX1
   (TRACE-FUN 'DERIVA LEX1)
   (TRACE-RET 'DERIVA (DIFN (POP LEX1) LEX1))
)

;-----------------------------------------------------------------------
; Funciones de Integracion Simbolica
;-----------------------------------------------------------------------

(DEFUN INT3 ()
   ; Global var @@A(,F+1,)@ & Fluid vars INDET from INTEGRA and EX1 from DIF1
   ((equal EX1 INDET)          ;
```

```
                (TRACE-FILE "$X3.EX5
                                ;LIST U EX5)
                                ;LIST M T))
                        (* @MAKE 1" EX5 2)))
    (APPLY (GET 'INTEGRA (LIST EX5))
           (ARGS EXT))
)

(DEFUN DRAW1V (LEXT)
  ;Fluid vars, from INTEGRA & UNIT : (WIDE, EX1, EX2, EX3,EX4, EX5
  (TRACE-FUN 'DRAW1V LEXT)
  ( (NULL LEXT)
                (TRACE #01 'DRAW1V (LIST)) )
  ( (equal (CDR LEXT) EX4 (POP LEXT)) (WIDE)
                (TRACE-RET 'DRAW1V (DRAW1V LEXT)))
  (SETQ EX5 (FACTORIZA (LIST */ EX1 EX4)))
  ( (DRAWP (SETQ EX3 (LIST EX4 (WIDE)))
                                        (SETQ EX2 (* EX2 EX4))
                                        (SETQ EX1 EX5)
                (TRACE)-#01 'DRAW1V    (DRAW1V LEXT)))
  ( (MAKE (SETQ EX3 (CONS (/ EX5 EX3)) (WIDE))
                (TRACE #01 'DRAW1V    (* @MAKE EX5 1" EX5 2)))
  ( (SAM EX4)
                (TRACE #01 'DRAW1V    (DRAW1V LEXT)) )
  ( (DRAWP (SETQ EX3 (LIST (CADR EX4) (WIDE))))
        ( (DRAWP (SETQ EX3 (LIST (CADR EX4) (WIDE))))
                        (TRACE-RET 'DRAW1V    (DRAW1V LEXT)))
        ( (MAKE (SETQ EX5 (SAMO (/ EX5 EX3))) (WIDE))
            ( (SETQ EX) (APPLY (GET 'INTEGRA (CAR EX4))
                                (LIST (CADR EX4) (WIDE))))
                        (TRACE-MSG (WIDE) (CADDR EX4))
                ; "---  Condition  Action ---<
                        (TRACE) RET 'DRAW1V
                                (* EX3
                                   (LIST(R EX5-
                                           (WIDE
                                           (CADDR EX4))))
                                                )
                (TRACE-RET 'DRAW1V (DRAW1V LEXT))
        )
        (TRACE-RET 'DRAW1V (DRAW1V LEXT))
  )
  (LIST? (FIND FULL (WIDE))
     (LIST? (SETQ EX5 (SETQ // EX5 EX5))) (WIDE))
                ( (SETQ EX5 (APPLY (GET 'INTEGRA (CAR EX4))
                                   (SAMO EX4)))
                        (TRACE-MSG (WIDE) (CADR EX4))
                        (TRACE-RET 'DRAW1V  (* EX3
                                               (LIST(R EX5
                                                       (WIDE
                                                       (CADR EX4)
                                                ) ) )
```



```
                )
                (TRACE RET 'DRAW1V (DRAW1V LEXT))
        )
        (TRACE-RET 'DRAW1V (DRAW1V LEXT))
  )
  (TRACE-RET 'DRAW1V (DRAW1V LEXT))
)

(DEFUN TRACE-MSG (X Y) BRAVO)
    (SETQ BRAVO (POP @STACK:TRACE))
    (PUSH (LIST BRAVO X Y) @STACK:TRACE)
)

(DEFUN SYSUB (EX1 EX2 EX3)
  (SYAL (SUB EX1 EX2 EX3))
)

(DEFUN SUB (EX1 EX2 EX5)
  ( (equal EX1 EX2) EX5 )
  ( (ATOM EX1) EX1 )
  (SUB5 (SUB (POP EX1) EX2 EX3)
        (SUB EX1 EX2 EX3))
)

(DEFUN UNIT (Y)
  ;Fluid vars, from INTEGRA & UNIT : (WIDE, EX1
  (PROGUNIT EX1)
                        (DRAW1V (COS EX1))
  (LOOP EX1 (WIDE)
                (TRACE-FILE "$X1 EX1
                                        (LIST A EX1))
                (* EX1 (WIDE))
  (RETU)
)

(DEFUN UNIT (EX1 EX2 EX3 EX4 EX5 EX6) EX1 EX2 EX3 EX4 EX5)
  ; Local : EX2 EX3 EX4 EX5
  (SETQ EX2 Y)
  (LOOP EX3 (LENGTH
                (* EX2 EX3))
  (SETQ EX4 EX1)
  (SETQ #(EX2 EX3 EX4 EX5
        #(EX2 EX3 EX4 EX5
        #(EX2 EX3 EX4 EX5
        #(EX2 EX3 EX4 EX5
        #(EX2 EX3 EX4 EX5
        #(EX2 EX3 EX4 EX5
        #(EX2 EX3 EX4 EX5
        #(EX2 EX3 EX4 EX5
  (SETQ EX5 (FACT EX1))
  (TRACE-FILE "$X2 (LIST EX4 EX5))
  (SETQ EX1 (DIV EX1 EX5))
  (LOOP (AND (NOT
                (* EX2 EX5))
```



```
(DEFUN RATIONAL (EXT (MOST)
  ( (OR (ATOM EXT)
        (FREE EXT (MOST)) )
  ((POWER EXT)
          (AND (RATIONAL (BASE EXT) (MOST)
               (INTEGERP (EXPONENT) EXT)))
  )
  (LOR (PRODUCT EXT)
       (SUM EXT))
              (POP EXT)
              (LOOP
                  ( (ATOM EXT) )
                  ( (NOT (RATIONAL (POP EXT) (MOST)) )
                                                 )) )
              )
  )
)

(DEFUN MULTIPLYFAC (EXT (SET)
  ((PRODUCT EXT)
              (POP EXT)
```


MODULO DE DERIVACION E INTEGRACION D-6 APENDICE D

```
;*****************************************************************
;                    MODULO DE CODIFICACION DE TABLAS
;*****************************************************************
;-------- -- ----------------------------------------------------
;
;-------- ------- ---- -- -- ---- ---- ---- ---- ---- ---- ----

(PROPERTY '+ '* '(LAMBDA (EX1 EX2)
                 (COND (NEGATIVE #NUMVALUE)
                       (NEGATIVE #DIVNUM))
                       ( CONS'P EX1) (FACTOR LEX1) )
                       ( (PRODUCT EX2) (FACTOR LEX1 (CDR EX1)) )
                       ( FACTOR LEX2 (LIST EX1) )
                 )
          )
)

(PROPERTY '+ '* '(LAMBDA (EX1 EX2 EX3)
                 (COND (NEGATIVE #NUMVALUE)
                       (NEGATIVE #DIVNUM))
                             (SETQ EX2 (LIST '* EX2 EX3))
                             ( CONS'P EX1) (FACTOR (LIST EX2)) )
                             ( (PRODUCT EX1) (FACTOR (LIST EX2) (CDR EX1)) )
                             ( FACTOR (LIST EX3) (LIST EX1) )
                 )
          )
)

(PROPERTY '* '+ '(LAMBDA (EX1 LEX1 LEX2 LEX3)
      (COR (PLUSP #NUMVALUE)
           (PLUSP #DIVNUM))
                ((PRODUCT EX1)
                         (SETQ LEX1 (CDR EX1))
                         (LOOP
                             (SETQ EX1 (POP LEX1))
                             (
                               ((ZEROP EX1)
                                   ((AND (PLUSP #DIVNUM)
                                         (ZEROP(1+ EX1 )))
                                                        (PUSH EX1 LEX2))
                                   (PUSH EX1 LEX3) )
                               ((AND (PLUSP #NUMVALUE)
                                     (NUMVALUE EX1)) (PUSH EX1 LEX2))
                               (PUSH EX1 LEX3)
                             )
                             ((ATOM LEX1))
                         )
                         (PROGLETACT (MESUM (DIVISION (MAPCAR (REVERSE LEX2))
                                                              LEX4
                                                              #NUMVALUE
                                                              #DIVNUM)
                                     )
```



```
                                                    (REVERSE LEX5))
                         )
                         ((ZEROP EX1)
                               ((AND (PLUSP #DIVNUM)
                                     (ZEROP(1+ EX1 )))
                                                        (MESUM (DIVISION EX1 LEX4 #NUMVALUE #DIVNUM)))
                               ((AND (PLUSP #NUMVALUE)
                                     (NUMVALUE EX1))
                                                        (MESUM (DIVISION EX1 LEX4 #NUMVALUE #DIVNUM)))
                         )
                )
          ) (ERROR)
)

(PROPERTY '* '/ '(LAMBDA (EX1 EX2 EX3 LEX1 LEX2 LEX3)
    ( (AND (NEGATIVE #DIVNUM)
           (ZEROP EX2)
           (equal (BASE EX1 EX2)) (* EX2 (- (EXPONENTE EX1) EX3)) )
    ( (AND (NEGATIVE #NUMVALUE)
           (ZEROP EX3)
           (equal (EXPONENTE EX1) EX3)) (* (* (BASE EX1) EX2) EX3) )
    ((MEMBER P EX3)
             ( (NATEQUAL EX2)
                    ( (NATEQUAL EX1)
                       (SETQ EX5 (CDR EX1 EX2))
                       (LOOP # EX3) nil)
                       (SETQ EX1 (ADDITION EX1 EX3))
                       (SETQ EX2 (SUBTRACT EX2 EX3))
                       (LOOP P EX2) EX1)
                       (SETQ EX2 (LIST '* EX2 -1))
                       (LOOP P EX1) EX2)
                       (LIST '* EX1 EX2)
                    )
                    ( (ZERO EX1)
                        (LIST '* (* EX2 (CADR EX1)) -1)
                    )
             )
             ( (SUM EX2)
                 ( LOOP (PLUSP #BOARD[N]
                        (PLUSP #NUMB[N])
                            (SETQ EX2 (CDR EX2))
                            ( (PRODUCT EX1)
                                (SETQ LEX1 (CDR EX1))
                                (LOOP
                                   (SETQ EX1 (POP LEX1))
                                   (
                                    ((ZEROP EX1)
                                       (SETQ EX3 (* EX1 -1))
                                       ((AND (PLUSP #DIVISOR)
                                             (ZEROP(1+ EX3))
                                               (SETQ LEX2
                                                  (CONS EX3
```

```
(DEFUN MCOSA(F)(N)
     (* (+  1 (* -1 MR(D)(1) (* (+ 3)) MS(D)(F)))        MR(L)) ) (* 8 -1)))
(DEFUN MCOS(F)(N)
     (* (+ -1 (* -1 MR(D)(1) (* (+ 3) (* -1 MS(D)(F))) MR(L)) ) (* 8 -1)))

(DEFUN CONS( (EX) EX2)
    ( (+ EX) 0) (COSP( (- EX))))
    (
      ( (+ EX) 2)
          (SETQ EX) (- EX)
                    (* 2
                      (QUOT)(ex) (NUMERADOR EX)
                                 (* 2 (DENOMINADOR EX))) ) ) ) ) )
    )
    (
      ( (+ EX) 1)
                (SETQ EX) (- 2 EX)) )
    )
    ( (EVENP (DENOMINADOR EX))
            ( (+ EX) MR(L)
                  (- (* (* (+ 1 (COSP) (* 2 EX))) MR(L)
                       MR(L)) ) )
          (* (* (+ 1 (COSP) (* 2 EX))) MR(L)
            MR(L))
    )
    ( SETQ EX)
         (DIVIDE (NUMERADOR EX)
                 (SETQ EX2 (DENOMINADOR EX))) )
    ( (ODD (EVENP (POP EX)))
                (- (COSP)) EX) EX2)) )
    (COSP)) EX) EX2)
)

(DEFUN COSP)) (EX) EX2)
    ( (ZERO) EX) 1)
    ( (= EX2 (+ EX) EX)))
          (- (COSP)) (- EX2 EX)) EX2)) )
    ( (EQ EX2 2) MR(L) )
    ( (EQ EX2 3)
                ( (MC-0 EX) MCOS(F)(N) )
                MCOS2P(DN) )
    ( (EQ EX2 5)
                ( (MC) P EX) MCOS(F)(N)) )
                ( (EQ EX) 2) MCOS2P(DN)) )
                ( (EQ EX) 4) MCOSA(F)(N)) )
                MCOS2P(DN)5 )
    ))(QT (CONS (/ (* SP)) EX) EX2))
)

(DEFUN TAN (EX)
    ( (NEGLT EX)
              (- (TAN (- EX))) )
```



```
    ( (POSMAL) MR(SIMP) 2)
              (/ (SEN EX)) (COS EX)) )
    ((COMP) 'TAN EX)
)

(DEFUN COT (EX)
    ( (NEGLT EX)
              (- (COT (- EX))) )
    ( (POSMAL) MR(SIMP) 2)
              (/ (COS EX)) (SEN EX)) )
    ((COMP) 'COT EX)
)

(DEFUN SEC (EX)
    ( (POSMAL) MR(SIMP) 2)
              (* (COS) EX)) -1) )
    ((COMP) 'SEC EX)
)

(DEFUN CSC (EX)
    ( (POSMAL) MR(SIMP) 2)
              (* (SEN EX)) -1) )
    ((COMP) 'CSC EX)
)

;*****************************************************************
; Funciones T/igonometricas Inversas
;*****************************************

(DEFUN ASEN (EX)
    ( (NOT MR(ARCH) (COMP) 'ASEN EX) )
    ( SETQ EX) (TRIGSMP EX) -2) )
    ( (NEGLT EX)) (- (ASEN (- EX))) )
    ( (ZERO) EX) 0 )
    ( (equal EX) MR(L)) MP(ON) )
    ( (equal EX) MR(M)2) MP(ON) )
    ( (equal EX) MR(3)M2) MP(ON) )
    ( (MC-M EX) MP(ON) )
    ( (POSMAL) MR(SIMP) T)
              (* (- 5))
                 (LM (+ (* (- 1 T (* EX) 2))
                        MR(L))
                     (* 5) EX))))
    )
    ( (NEGMAL) MR(SIMP) T))
              (ATAN (* EX)
                    (* (- 1 (* EX) 2))
                       MR(N)(R(L))))
    )
    ((COMP) 'ASEN EX)
)
```

```
(DEFUN ACOS (EXP)
    (- #P(1.0)
       (ASIN EXP)))
)

(DEFUN ATAN (EXP)
    ( (ZEROP EXP) (SIMPLE 'ATAN EXP) )
    ( SETQ EXP (TRUNCATE EXP (ABS EXP)) )
    ( (equal EXP #P(0.0)) (- #P(1.0)) )
    ( (MINUSP EXP) (- (ATAN (- EXP))) )
    ( (ZEROP EXP) 0 )
    ( (equal EXP #P(1.0)) #P(1.0) )
    ( (ONE-P EXP) #P(1.0) )
    ( (equal EXP #P(0.5)) #P(1.0) )
    ( (equal EXP #P(1.0)) #P(1.0) )
    ( (POSMULT #P(0.0)
               (* #P(1.0)
                  2)
               (LN (/ (+ 1) (+ 1) EXP)
                      (- 1) EXP))))
    )
    ((POSMULT #P(0.0)
              (ATAN (* EXP
                       (* (+ 1 (* EXP EXP))
                          #P(1.0)))))
    )
    ((SIMPLE 'ATAN EXP)
)

(DEFUN ACOT (EXP)
    (ATAN (/ 1 EXP))
)

(DEFUN ASEC (EXP)
    (- #P(1.0)
       (ATAN (/ 1 EXP)))
)

(DEFUN ACSC (EXP)
    (ASIN (/ 1 EXP))
)

;-----------------------------------------------------------------
; Definición de funciones trigonométricas
;-----------------------------------------------------------------

(DEFUN SINH (EXP EXP)                                  ;          A    -A
  ((ZEROP EXP) EXP)                                    ;         e  - e
  ((MINUS-P EXP) (- (SINH (- EXP))))                   ; sinh A = -------
  ((POSMULT #P(0.0)                                    ;            2
        (SETQ EXP (* 2) EXP))
        (* (- EXP (/ 1 EXP)) #P(0.5)))
)

(DEFUN TANH (EXP)                                      ;          sinh A
  ((MINUS-P EXP) (- (TANH (- EXP))))                   ; tanh A = ------
  ((POSMULT #P(0.0)                                    ;          cosh A
        (SETQ EXP (* 2) EXP)) (/ (SINH EXP) (COSH EXP))))
  ((ZEROP EXP) EXP)
  ((SIMPLE 'TANH EXP)
)

(DEFUN COTH (EXP)                                      ;          cosh A
  ((MINUS-P EXP) (- (COTH (- EXP))))                   ; coth A = ------
  ((POSMULT #P(0.0)                                    ;          sinh A
        (SETQ EXP (* 2) EXP)) (/ (COSH EXP) (SINH EXP))))
  ((SIMPLE 'COTH EXP)
)

(DEFUN SECH (EXP)                                      ;            1
  ((POSMULT #P(0.0)                                    ; sech A = ------
        (SETQ EXP (* 2) EXP)) (* (COSH EXP) (/)))      ;          cosh A
  ((SIMPLE 'SECH EXP)
)

(DEFUN CSCH (EXP)                                      ;            1
  ((POSMULT #P(0.0)                                    ; csch A = ------
        (SETQ EXP (* 2) EXP)) (* (SINH EXP) (/)))      ;          sinh A
  ((SIMPLE 'CSCH EXP)
)

;-----------------------------------------------------------------
; Identidades trigonométricas para suma de angulos (n+1)
; y angulos multiples n-0
;-----------------------------------------------------------------

(DEFUN MULTIANGLE (EXP EXP EXP EXP)
  ((ZEROP EXP) EXP)
  ((AND (INTEGERP EXP)
        (PLUSP EXP))
                (MULTIANGLET EXP EXP EXP))
)

(DEFUN MULTIANGLET (EXP EXP EXP)
  ; filled var from MULTIANGLE: EXP
  ((ONE-P EXP) EXP)
  (MULTIANGLET (- EXP 1) (+ (* EXP EXP) EXP) EXP)
)
```

```
)

(PROPERTY :* '*SA '(LAMBDA (EX1 EX2)
   ( (equal (CADR EX1) EX2)
             ( (EQ (CAR EX1) 'COS) (SIN EX2) )          ; SEN A COS A = SEN A
             ( (EQ (CAR EX1) 'COT) 1 )                  ; TAN A COT A = 1
   )
          ) ;Lambda
)

(PROPERTY :* 'COT '(LAMBDA (EX1 EX2)
   ( (EQ (CAR EX1) 'SEN)
             ( (equal (CADR EX1) EX2) (COS EX2) ) )  ; COT A SEN A = COS A
            ); Lambda
)

(PROPERTY :* 'SEC '(LAMBDA (EX1 EX2)
   ( (NOT (ATOM EX1))
            ( (equal (CADR EX1) EX2)
                 ( (EQ (SETQ EX1 (CAR EX1)) 'COS) 1 ) ; SEC A COS A = 1
                 ( (EQ EX1 'SEN) (TAN EX2) )          ; SEC A SEN A = TAN A
                 ( (EQ EX1 'COT ) (CSC EX2) )         ; SEC A COT A = CSC A
            )
   )
            ) ; Lambda
)

(PROPERTY :* 'CSC '(LAMBDA (EX1 EX2)
  ( (NOT (ATOM EX1))
     ( (equal (CADR EX1) EX2)
                 ( (EQ (SETQ EX1 (CAR EX1)) 'SEN) 1 ) ; CSC A SEN A = 1
                 ( (EQ EX1 'COS) (SEC EX2) )          ; CSC A COS A = SEC A
                 ( (EQ EX1 'TAN) (SEC EX2) )          ; CSC A TAN A = SEC A
     )
  )
            ) ;Lambda
)

;-------------------------------------------------------------------
;   Identidades de combinacion de funciones trigonometricas hiperbolicas
;-------------------------------------------------------------------

(PROPERTY :* 'SENH '(LAMBDA (EX1 EX2)      ; (3) A.48
 ((FORMAT #'DECIMAL 5)                     ;                 1          1
   (((EQ (CAR EX1) 'SENH)                  ; senh A senh B = - cosh(A+B) - cosh(A-B)
        (SETQ EX1 (CADR EX1))              ;                 2          2
        (- (* @HALF (COSH (- EX1 EX2)))
           (* @HALF (COSH (+ EX1 EX2))))
   )
   ((EQ (CAR EX1) 'COSH)
     ((SETQ EX1 (CADR EX1))                ;                 1          1
     (+ (* @HALF (COSH (+ EX1 EX2))) ;senh A cosh B = - cosh(A+B) + cosh(A-B)
```



```
        (* @HALF (SENH (- EX2 EX1))))))                2          2
   )
 )
            ) ;Lambda
)

(PROPERTY :* 'COSH '(LAMBDA (EX1 EX2)  ; (3) A.47
  ((FORMAT #'DECIMAL 5)                ;                 1          1
   (((EQ (CAR EX1) 'COSH)  ; cosh A cosh B = - cosh(A+B) + cosh(A-B)
        (SETQ EX1 (CADR EX1)) ;                 2          2
        (+ (* @HALF (COSH (+ EX1 EX2)))
           (* @HALF (COSH (- EX1 EX2))))
   )
   (
            ) ; Lambda
)

(PROPERTY :* 'TANH '(LAMBDA (EX1 EX2)                          ;
  ((equal (CADR EX1) EX2)                                      ;
                 (((EQ (CAR EX1) 'COSH) (SENH EX2)) ;tanh A cosh A = senh A
                 (((EQ (CAR EX1) 'COTH) 1))         ;tanh A coth A = 1
            )
)

(PROPERTY :* 'COTH '(LAMBDA (EX1 EX2)                          ;
   (((EQ (CAR EX1) 'SENH)                                      ;
         ( (equal (CADR EX1) EX2) (COSH EX2) ) )   ; coth A senh A = cosh A
           )
)

(PROPERTY :* 'SECH '(LAMBDA (EX1 EX2)                          ;
  ((equal (CADR EX1) EX2)                                      ;
                 (((SETQ EX1 (CAR EX1)) 'COSH) 1)   ; sech A cosh A = 1
                 (((EQ EX1 'SENH) (TANH EX2))       ; sech A senh A = tanh A
                 (((EQ EX1 'COTH) (CSCH EX2)))      ; sech A coth A = csch A
                 )
)

(PROPERTY :* 'CSCH '(LAMBDA (EX1 EX2)                          ;
  ((equal (CADR EX1) EX2)                                      ;
                 (((SETQ EX1 (CAR EX1)) 'SENH) 1)   ; csch A senh A = 1
                 (((EQ EX1 'COTH) (SECH EX2)))      ; csch A cosh A = coth A
                 (((EQ EX1 'TANH) (SECH EX2)))      ; csch A tanh A = sech A
                 )
)

;-------------------------------------------------------------------
;  Identidades de combinacion de funciones trigonometricas segun el
;  valor del angulo
;-------------------------------------------------------------------
```

```
(PROPERTY 'SEN *= '(LAMBDA (EX1 EX2)     ; (5) 5.34
    ( (NEGRAL #TRIGEXPO 5)               ; sen(A+B) = sen A cos B + cos A sen B
              (+ (* (SEN EX1) (COS EX2))
                 (* (COS EX1) (SEN EX2))) )
            )
)

(PROPERTY 'COS *= '(LAMBDA (EX1 EX2)     ; (5) 5.35
    ( (NEGRAL #TRIGEXPO 5)               ; cos(A+B) = cos A cos B - sen A sen B
              (- (* (COS EX1) (COS EX2))
                 (* (SEN EX1) (SEN EX2))) )
            )
)

(PROPERTY 'SENH *= '(LAMBDA (EX1 EX2)    ; (5) 6.20
  ( (NEGRAL #TRIGEXPO 5)                 ; senh(A+B) = senh A cosh B + cosh A senh B
          (+ (* (SENH EX1) (COSH EX2))
             (* (COSH EX1) (SENH EX2))) )
            )
)

(PROPERTY 'COSH *= '(LAMBDA (EX1 EX2)    ; (5) 6.21
   ((NEGRAL #TRIGEXPO 5)                 ; cosh(A+B) = cosh A cosh B + senh A senh B
          (+ (* (COSH EX1) (COSH EX2))
             (* (SENH EX1) (SENH EX2))) )
            )
)

(PROPERTY 'SEN *= '(LAMBDA (EX1 EX2)
    ( (NEGRAL #TRIGEXPO 5)
              (MULTANGLE EX1
                         (* 2 (COS EX2))
                         (SEN EX2)
                         0) )
            )
)

(PROPERTY 'COS *= '(LAMBDA (EX1 EX2)
    ( (NEGRAL #TRIGEXPO 5)
              (MULTANGLE EX1
                         (* 2 (COS EX2))
                         (COS EX2)
                         1) )
            )
)

(PROPERTY 'SENH *= '(LAMBDA (EX1 EX2)
  ( (NEGRAL #TRIGEXPO 5)
        (MULTANGLE EX1 (* 2 (COSH EX2)) (SENH EX2) 0) )
            )
)
```



```
(PROPERTY 'COSH *= '(LAMBDA (EX1 EX2)
  ( (NEGRAL #TRIGEXPO 5)
        (MULTANGLE EX1 (* 2 (COSH EX2)) (COSH EX2) 1) )
            )
)

;-----------------------------------------------------------------------
;  Identidades de combinacion de funciones trigonometricas
;-----------------------------------------------------------------------

(PROPERTY '* *SEN *(LAMBDA (EX1 EX2)
    ( (POSNULL #TRIGEXPO 0)
              ( (EQ (CAR EX1) 'SEN)
                    (SETQ EX1 (CADR EX1))
                    ;                    1                    1
                    ; sen A sen B =  - cos(A-B) + cos(A+B)
                    ;                    2                    2
                    (- (* #NULL (COS (- EX1 EX2)))
                       (* #NULL (COS (+ EX1 EX2))))
                    )
              ;                    1                    1
              ; sen A cos B = + sen(A+B) + + sen(A-B)
              ;                    2                    2
              ( (EQ (CAR EX1) 'COS)
                    (SETQ EX1 (CADR EX1))
                    (+ (* #NULL (SEN (+ EX1 EX2)))
                       (* #NULL (SEN (- EX1 EX2))))
                    )
    ); psnull()
            ) ;Lambda
)

(PROPERTY '* *COS *(LAMBDA (EX1 EX2)
    ( (POSNULL #TRIGEXPO 5)
              ;                    1                    1
              ; cos A sen B = - sen(A+B) + - sen(A-B)
              ;                    2                    2
              ((EQ (CAR EX1) 'SEN)
                    (SETQ EX1 (CADR EX1))
                    (+ (* #NULL (SEN (+ EX1 EX2)))
                       (* #NULL (SEN (- EX1 EX2))))
                    )
              ;                    1                    1
              ; cos A cos B = + cos(A-B) + + cos(A+B)
              ;                    2                    2
              ((EQ (CAR EX1) 'COS)
                    (SETQ EX1 (CADR EX1))
                    (+ (* #NULL (COS (- EX1 EX2)))
                       (* #NULL (COS (+ EX1 EX2))))
                    )
    ); psnull()
            ); Lambda
```

```
(PROPERTY 'SEN 'ASEN '(DEFUN (X)                    ; DEF (ASEN X) = A
(PROPERTY 'ACOS 'ACOS '(DEFUN (X)                   ; DEF (ACOS X) = A
(PROPERTY 'TAN 'ATAN '(DEFUN (X)                    ; DEF (ATAN X) = A
(PROPERTY 'COT 'ACOT '(DEFUN (X)                    ; DEF (ACOT X) = A
(PROPERTY 'CSC 'ACSC '(DEFUN (X)                    ; DEF (ACSC X) = A
(PROPERTY 'SEC 'ASEC '(DEFUN (X)                    ; DEF (ASEC X) = A

(PROPERTY 'COS 'ASIN '(LAMBDA (EXT)                 ;                    2 1/2
    (* (- 1 (* EXT EXT))                            ; COS (ASIN X) = (1 - X )
       (RECIP)
              )
)

(PROPERTY 'SEN 'ATAN '(LAMBDA (EXT)                 )
    (* EXT
       (                    # 1/2
        (* (+ 1 (* EXT EXT)) (RECIP(SQRT)))         ; SEN (ATAN X) = X (1 + X )
              )
)

(PROPERTY 'COS 'ATAN '(LAMBDA (EXT)                 ;                    2 1/2
    (* (+ 1 (* EXT EXT))                            ; COS (ATAN X) = (1 + X )
       (RECIP(SQRT))                                )
              )
)

(PROPERTY 'ASEN 'SEN '(LAMBDA (EXT)                 )
   ( ((EQUAL(TYP (PROP(X) EXT ))                    ; ASEN (SEN X) = X
              )
)

(PROPERTY 'ASEN 'COS '(LAMBDA (EXT)                 )
   ( ((EQUAL(TYP (PROP(X) (- (PROP(2 EXT)) )        ;                  (1)
              )                                     ; ASEN (COS X) = --- - X
)                                                   ;                  (2)

(PROPERTY 'ATAN 'TAN '(LAMBDA (EXT)                 )
   ( ((EQUAL(TYP (PROP(X) EXT )                     ; ATAN (TAN X) = X
              )
)

(PROPERTY 'ATAN 'COT '(LAMBDA (EXT)                 ;                  (1)
   ( ((EQUAL(TYP (PROP(X) (- (PROP(2 EXT)) )        ; ATAN (COT X) = --- - X
              )                                     ;                  (2)
)

;-----------------------------------------------------------------------
;
;
;-----------------------------------------------------------------------

(PUT 'SEC 'ABS 'SEC)                                ; ABS(SEC X) = SEC X
(PUT 'COS 'ABS 'COS)                                ; ABS(COS X) = COS X
```



```
(PUT 'SECH 'ABS 'SECH)                              ; ABS(SECH X) = SECH X
(PUT 'COSH 'ABS 'COSH)                              ; ABS(COSH X) = COSH X

;-----------------------------------------------------------------------
; ATOMO BASE
;-----------------------------------------------------------------------

(PROPERTY 'BASE '* '(LAMBDA (EXT EXT EXT) #NUMMUM_ARGUMENT
                                        ; Compiler MACRO#LIM #DEFAULT
    (((NOT(ATOM EXT)
        (
         (SETQ EXT (CONS '* (CONS EXT (CDR EXT))))
         (SETQ EXT (LIST '* EXT EXT))
        )
        (
         (NUMBER? # EXT)
                  ((OR (NEGATIVE #DEFAULT)
                       (NEGATIVE #NUMMUM)) (SETQ #NUMMUM #DEFAULT)
                                           (SETQ #DEFAULT #DEFAULT)
                                           (SETQ EXT (CONS EXT EXT))
                                           ( (SUB EXT) #1L )
                                           (* EXT -1)
                  )
         )
         ( (PLUS? #DEFAULT)
                  (SETQ #NUMMUM 1)
                  (SETQ #DEFAULT 1)
                  ( (PLUS? EXT)
                           ((MULTIPLE #DEFAULT 2) (EXT EXT EXT) )
                  ((MULTIPLE #DEFAULT 3) (* (EXPT EXT (- #DEFAULT 1)))
         )
    )
  )
)

(PROPERTY 'BASE '^ '(LAMBDA (EXT EXT EXT)
                    ((AND (PLUS? #EXPONENT)
                          (EQUAL? EXT))
                                 (* (^ EXT EXT)
                                    (^ EXT EXT)))
          )
)

(PROPERTY 'BASE '* '(LAMBDA (EXT EXT EXT)           ;
                    ( (OR ((NEGATIVE? EXT) #DEFAULT)    ;   A 1     A 2
                          (* EXT (* EXT EXT)) )         ; (A )   * A
          )
)

(PROPERTY 'BASE 'ABS '(LAMBDA (EXT EXT)             ;
    ( ((EVEN? EXT) (* EXT EXT) )                    ;
              )                                     ;
```

```
)

(PROPERTY 'BASE 'SEN '(LAMBDA (EX1 EX2)
   ( (INTEGERP EX1)
          ( (AND (PLUSP EX1)
                 (POSMULT #TRIGEXPO 3))
                 (* (* (SEN EX2)
                       (- EX1 2))
                    (- #HALF
                       (* (COS (* 2 EX2)) #HALF))))
          )
          ( (AND (= EX1 -1)
                 (POSMULT #TRIGEXPO 3))
                 (* (* (SEN EX2) (- EX1))
                    -1) )
          ( (AND (NEGATIVE EX1)
                 (NEGMULT #TRIGEXPO 2))
                 (* (COS EX2)
                    (- EX1)) )
          ( (AND (PLUSP EX1)
                 (NEGATIVE #TRIGSEN))
                 (* (* (- 1 (* (COS EX2)
                               2))
                       (QUOTIENT EX1 2))
                    (* (SEN EX2)
                       (REMAINDER EX1 2)))
          )
          ( (AND (= EX1 -1)
                 (NEGATIVE #TRIGSEN))
                 (* (* (* (- 1 (* (COS EX2) 2))
                          (QUOTIENT (SETQ EX1 (- EX1))
                                    2))
                       (* (SEN EX2)
                          (REMAINDER EX1 2)))
                    -1)
          )
   ) ( INTEGERP-S
               ) ( LAMBDA
)

(PROPERTY 'BASE 'COS '(LAMBDA (EX1 EX2)
   ( (INTEGERP EX1)
          ( (AND (PLUSP EX1)
                 (POSMULT #TRIGEXPO 3))
                 (* (* (COS EX2)
                       (- EX1 2))
                    (+ #HALF (* (COS (* 2 EX2)) #HALF)))
          )
          ( (AND (= EX1 -1)
                 (POSMULT #TRIGEXPO 3))
                 (* (* (SEN EX2) (- EX1))
                    -1)
```

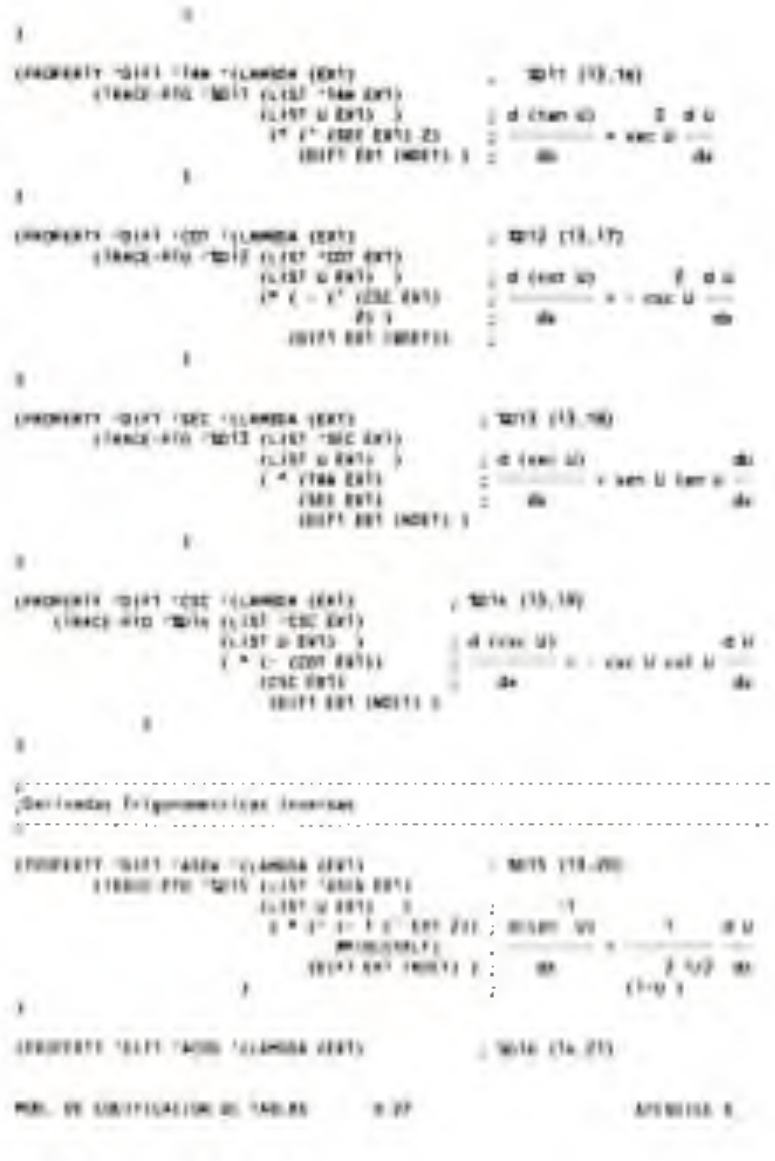


```
          )
          ( (AND (NEGATIVE EX1)
                 (NEGMULT #TRIGEXPO 2))
                 (* (SEC EX2)
                    (- EX1))
          )
          ( (AND (PLUSP EX1)
                 (PLUSP #TRIGSEC))
                 (* (* (- 1 (* (SEN EX2) 2))
                       (QUOTIENT EX1 2))
                    (* (COS EX2) (REMAINDER EX1 2)))
          )
          ( (AND (= EX1 -1)
                 (PLUSP #TRIGSEC))
                 (* (* (* (- 1 (* (SEN EX2) 2))
                          (QUOTIENT (SETQ EX1 (- EX1)) 2))
                       (* (COS EX2) (REMAINDER EX1 2)))
                    -1)
          )
   ) ( INTEGER
               ) ( LAMBDA
)

(PROPERTY 'BASE 'TAN '(LAMBDA (EX1 EX2)
   ( (NEGATIVE EX1)
          ( (NEGMULT #TRIGEXPO 2)
                                 (* ( (SEN EX2) (- EX1)) )
   )
               ) (LAMBDA
)

(PROPERTY 'BASE 'COT '(LAMBDA (EX1 EX2)
   ( (NEGATIVE EX1)
          ( (NEGMULT #TRIGEXPO 2)
                                 (* (TAN EX2) (- EX1)) )
   )
               )(LAMBDA
)

(PROPERTY 'BASE 'SEC '(LAMBDA (EX1 EX2)
   ( (NEGATIVE EX1)
          ( (NEGMULT #TRIGEXPO 2)
                                 (* (COS EX2) (- EX1)) )
   )
               ) (LAMBDA
)

(PROPERTY 'BASE 'CSC '(LAMBDA (EX1 EX2)
   ( (NEGATIVE EX1)
          ( (NEGMULT #TRIGEXPO 2)
                                 (* (SEN EX2) (- EX1)) )
   )
               ) (LAMBDA
```

```
)

(PROPERTY *BASE* 'SENO '(LAMBDA (EXT EXZ)
  ((INTEGERP EXT)
      ((AND (PLUSP EXT)
            (PODWAL? #'EQUEXPO 3)
                 (* (* (SENO EXZ) (- EXT 2))
                    (- (* (COS (* 2 EXZ)) #HALF) #HALF))
      )
      ((AND (= EXT -1)
            (PODWAL? #'EQUEXPO 3))
                 (* (* (SENO EXZ) (- EXT)) -1)
      )
      ((AND (NEGATIVE EXT)
            (NEQWAL? #'EQUEXPO 2))
                 (* (COS EXZ) (- EXT))
      )
      ((AND (PLUSP EXT)
            (NEGATIVE #'EQUEQU))
                 (* (* (- (- (COS EXZ) 2) 1) (QUOTIENT EXT 2))
                    (* (SENO EXZ) (MODULO EXT 2)))
      )
      ((AND (= EXT -1)
            (PLUSP #'EQUEQU))
                 (SETQ EXT (- EXT))
                 (* (* (- (- (COS EXZ) 2) 1) (QUOTIENT EXT 2))
                    (* (SENO EXZ) (MODULO EXT 2)))
                 -1)
      )
   )
                  ) ; Lambda
)

(PROPERTY 'BASE 'COS# '(LAMBDA (EXT EXZ)
   ((INTEGERP EXT)
         ((AND (PLUSP EXT)
               (PODWAL? #'EQUEXPO 3)
                    (* (* (COS# EXZ)
                          (- EXT 2))
                       (+ #HALF
                          (* (COS# (* 2 EXZ)) #HALF)))
         )
         ((AND (= EXT -1)
               (PODWAL? #'EQUEXPO 3)
                    (* (* (COS# EXZ) (- EXT)) -1))
         ((AND (NEGATIVE EXT)
               (NEQWAL? #'EQUEXPO 2)
                    (* (SENO EXZ) (- EXT)))
         )
         ((AND (PLUSP EXT)
               (PLUSP #'EQUEQU))
                    (* (* (+ 1 (* (SENO EXZ) 2)) (QUOTIENT EXT 2))
                       (* (COS# EXZ) (MODULO EXT 2)))
         )
         ((AND (= EXT -1)
               (PLUSP #'EQUEQU))
                    (SETQ EXT (- EXT))
                    (* (* (+ 1 (* (SENO EXZ) 2)) (QUOTIENT EXT 2))
                       (* (COS# EXZ) (MODULO EXT 2)))
                    -1)
         )
      )
                  ) ; Lambda
)

(PROPERTY 'BASE 'TAN# '(LAMBDA (EXT EXZ)
   ((AND (NUMBER EXT)
         (= EXT 0))
              ((WEIGHULT #'EQUEXPO 2) (* (COS# EXZ) (- EXT))))
   )
                  ); Lambda
)

(PROPERTY 'BASE 'COT# '(LAMBDA (EXT EXZ)
   ((AND (NUMBER EXT)
         (= EXT 0))
              ((WEIGHULT #'EQUEXPO 2) (* (TAN# EXZ) (- EXT))))
   )
                  ); Lambda
)

(PROPERTY 'BASE 'SEC# '(LAMBDA (EXT EXZ)
    ((AND (NUMBER EXT)
          (= EXT 0))
              ((WEIGHULT #'EQUEXPO 2) (* (COS# EXZ) (- EXT))))
    )
                  ) ; Lambda
)

(PROPERTY 'BASE 'CSC# '(LAMBDA (EXT EXZ)
    ((AND (NUMBER EXT)
          (= EXT 0))
              ((WEIGHULT #'EQUEXPO 2) (* (SENO EXZ) (- EXT))))
    )
                  ) ;Lambda
)

;-------------------------------------------------------------------------
; ATUMO EXPONENTE
;-------------------------------------------------------------------------

(PROPERTY 'EXPONENTE '+= '(LAMBDA (EXT EXZ EXC)
                  ((AND (PLUSP #BASE#)
                        (BASE# EXT))
```

```
(SETQ PRIMOS '(2 3 5 7 11 13 17 ???))

(DEFUN ROOT (EX1 EX2 EX3 LEX1 EX4 EX5 EX6 EX7) ;Limit: EX7
  (LOOP (
         ((ZERO? (COR (SETQ EX7 (1- EX7)) (AND (SETQ EX8 (1+ EX8)) (= (MAX EX8 EX9) EX9))))
                  (SETQ EX6 (AND EX7))
                  (  (NOT (= (SETQ EX5 (+ EX5 1)) EX6))
                                          (SETQ EX8 0)
                                          (SETQ EX9 02 )
         )
         (SETQ EX3 0)
         ((NOT (= (MOD (EX1) (AND EX7)))
                                          (SETQ EX8 1)
         )
         (LISTOM LEX1)
              (SETQ EX7 (- EX8 1))
              (SETQ LEX1 EX8)
              (LOOP
                (SETQ EX9 (* LEX1 EX7))
                ( (NOT (= (SETQ EX5 (QUOTIENT )+ EX6 (* EX7 LEX1 EX5))
                                          (* EX8 EX5)))
                          LEX1)) )
                (SETQ LEX1 EX9)
              )
              ( (EX (* LEX1 EX5) EX6)
                   (SETQ EX6 (* EX4 LEX1))
                                          (SETQ EX6 1)
              )
              (SETQ EX8 1)
         )
        )
        ((ZERO-P EX8)
                 (SETQ EX1 (/ EX1 (* EX6 EX8)))
                 (SETQ EX4 (* EX4 EX2))
                 ((ZERO-P EX7) EX4)
                 (* EX4 (LIST -* EX1 (/ EX2 EX3)))
        )
    )
)


;-----------------------------------------------------------------------
;  Leyes logarítmicas
;-----------------------------------------------------------------------

(PROPERTY 'LOG '* 'LAMBDA                          ;
    (EXT EX2 EX3)                                  ; log (M N) = log M + log N
    ((PROGN,F #LOGICKPD 5)                         ;     a       a     a
          (+ (LOG EX2 EX1)
             (LOG EX3 EX1)))
          ); Lambda
)

;-----------------------------------------------------------------------
;  Leyes logarítmicas
;-----------------------------------------------------------------------

(PROPERTY 'LOG '* 'LAMBDA (EX1 EX2 EX3)
   (SETQ EX4 (MOD EX2))
   ( (OR
        (AND (equal (SETQ EX5 (ABS EX1)) EX1)
             (equal EX5 EX2))
        (ZERO? (ZEROP (= EX7 EX3))
             (ZEROP (= EX8 EX4)))
                                  (= EX3 EX4))
   )
            ); Lambda
)

(PROPERTY 'ABS '* 'LAMBDA (EX1 EX2)                ; abs(A) abs(B) = abs(A B)
   ( (INTEGERP EX7)
               (* (ABS EX1)
                  (ABS EX2)) )
   ( (STRNGLP EX1)
               (ABSPROD EX1 EX1 EX2) )
   ( (LET (CAR EX1) '(EX1)
```

```
                FABSPROG (CADR EXT) EXT EXZ) )
    (* LABS EXT) (ABS EXZ))
            ) ;LAMBDA
)

(PROPERTY 'ABS '* '(LAMBDA (EXT EXZ)
    ( (INTEGERP EXZ)                                ;       0
                ( (EVENP EXZ) (* EXT EXZ) )         ; (abs x)
                (* (ABS EXT) EXZ)                   ;
      )
    ( (ABS (CDR (ATOM EXT)))
          (GET EXT 'NONNEGZ))
                        (* EXT EXZ) )
            ) ;LAMBDA
)


; ABS.ALG Absolute Value Package

(PROPERTY '(0) 'NONNEGZ T )
(PROPERTY '(1) 'NONNEGZ T )
(PROPERTY '(EXPT 'NONNEGZ T )
(PROPERTY '(EXPEVEN 'NONNEGZ T )
(PROPERTY '(ABS 'NONNEGZ T )
(PROPERTY '(COS) 'NONNEGZ T )
(PROPERTY '(SEC) 'NONNEGZ T )

; CDR13 significa que la funcion es una funcion impar de un argumento
; regresa valores que caen en el primero y tercer cuadrantes.

(PROPERTY '(SIN) 'CDR13 T )
(PROPERTY '(SIN) 'CDR13 T )
(PROPERTY '(TAN) 'CDR13 T )
(PROPERTY '(SINH) 'CDR13 T )
(PROPERTY '(ATAN) 'CDR13 T )
(PROPERTY '(ASIN) 'CDR13 T )
(PROPERTY '(SIGN) 'CDR13 T )

;.....
;
;.....

(COND ((NOT (INTEGERP #TRIGX)) (SETQ #TRIGX 0)) )
(COND ((NOT (NUMBER #TRIGX FLAG1)) (PUSH #TRIGX FLAG1)) )
(COND ((NOT (NUMBER #LNABS FLAG1)) (PUSH #LNABS FLAG1)) )

(DEFUN COMPRCX (EXT)
   ( (IDENTITY #PRACX) EXT )
)

(DEFUN COMPRCX (EXT)
   ( (IDENTITY #PRATAU) (- #PIOAZ EXT) )
```



```
((FREE EX2 (ABS)) ;Cuando la base es constante (NO Fix))
  ((PROG (SETQ EX4 (DIFF EX3 (ABS))) (ABS)
      (COND ((ONE-P EX4)
             (TRACE-PTO '$215 (LIST '* EX2 EX3)
                              (LIST '$ EX2)
                              (LIST '% EX3)))
            ((TRACE-PTO '$115 (LIST '* EX2 EX4)
                        (LIST '$ EX2)
                        (LIST '% EX3)
                        (LIST |Se completa el Integrando | EX4))))
       ( / (/ (LIST '* EX2 EX3) EX4)
          (LAMBS EX2) )
      )
     ((PROG (SETQ EX5 (DIFF EX4 (ABS))
            (ABS)
            (PRINT 'SEGUNDA ITERACION)
            (PRINT 'EX5)(PRINT EX5)
                   (* (' ) ) ( (* AMBIGUEDAD
                                  SQ)
                               EX5)
                            (LAMBS EX2))
                         #NULF)
                      (' EX2
                         I- EX3 ) (* #NULF (' EX4 Z)) EX5)))
                      (EQU (* ( EX4)
                           (* ) (* AMBIGUEDAD
                                    (LAMBS EX2))
                              EX5)
                           #NULF)
                        )
                     )
                  )
      )
     )
          ) ) Lambda
)

;-------------------------------------------------------------------------
; Integrales de Funciones Trigonometricas
;-------------------------------------------------------------------------

(PROPERTY 'INTEGRA 'SEN '(LAMBDA (EX1 EX2) ( Local EX2
  ((FREE (SETQ EX2 (DIFF EX1 (NOW))) (NOW))    ; /
     (COND ((ONE-P EX2)                        ;  |
            (TRACE-PTO $15 (LIST 'SEN EX1)     ;  | sen U dU = -cos U dU
                           (LIST 'U EX1)))     ;  |
           ((TRACE-PTO $15 (LIST 'SEN EX1)     ; /
                           (LIST 'U EX1)
                           (LIST | Se completa el Integrando |
                                 EX2))))
      (* -1 (/ (COS EX1) EX2)))
          )
)

(PROPERTY 'INTEGRA 'COS '(LAMBDA (EX1 EX2) (Local EX2
  ((FREE (SETQ EX2 (DIFF EX1 (NOW))) (NOW))
     (COND ((ONE-P EX2)                        ;  /
            (TRACE-PTO $16 (LIST 'COS EX1)     ;  |
                           (LIST '# EX1)))     ;  | cos U dU= sen U
           ((TRACE-PTO $16 (LIST 'COS EX1)     ;  |
                           (LIST '# EX1)       ;  |
                           (LIST 'U EX1)       ; /
                           (LIST | Se completa el Integrando |
                                 EX2))))
                                    (/ (SEN EX1) EX2)))
          );Lambda
)

(PROPERTY 'INTEGRA 'ASEN '(LAMBDA (EX2 EX3)
  ((FREE (SETQ EX3 (DIFF EX2 (NOW))) (NOW))    ;  /                          1/2
    (COND ((ONE-P EX3)                         ;  |
           (TRACE-PTO $17 (LIST 'ASEN EX2)     ;  | asen U dU=U*asen(1-u^2)
                          (LIST 'U EX2)))      ;  |
          ((TRACE-PTO $17 (LIST 'ASEN EX2)     ; /
                          (LIST 'U EX2)
                          (LIST | Se completa el Integrando |
                                EX3))))
     (/ (- (* EX1 EX2)
           (* I- 1 (* EX2 Z))
           #NULF))
      EX3))
   );Lambda
)

(PROPERTY 'INTEGRA 'ATAN '(LAMBDA (EX2 EX3)
  ((PROG (SETQ EX3 (DIFF EX2 (NOW))) (NOW))
     (COND ((ONE-P EX3)
            (TRACE-PTO $18                     ;  /                          U*ln(U^2+1)
                          (LIST 'ATAN EX2)     ;  |                          ----------- +
                          (LIST U EX2)))       ;  | atanU dU= U atan U-        2
           ((TRACE-PTO $18                     ;  |
                          (LIST 'ATAN EX2)     ; /
                          (LIST U EX2)         ;
                          (LIST | Se completa el Integrando |
                                EX3))))
      (/ (- (* EX1 EX2)
            (* #NULF
               (LAMBS (+ (* EX2 Z) 1))))
         EX3))
  );Lambda
)

(PROPERTY 'INTEGRA 'LOG '(LAMBDA (EX2 EX3)
  ((PROG (SETQ
```

```
        (* (DIFF EXT)                    ; ---------- + cosh U ---
           (DIFF EXT (NEXT)))            ;   dx           dx
              )
)

(PROPERTY 'SIN1 'DIFF 'LAMBDA (EXT)     ; SIN(U) (15.59)
   (TRACE-FYO 'SIN2 (LIST 'DIFF EXT)
              (LIST U EXT))             ; d (cosh U)              d U
        (* (SIN4 EXT)                   ; ---------- = - sinh U * ---
           (DIFF EXT (NEXT)))           ;   dx                    dx
              )
)

(PROPERTY 'SIN1 'TANH 'LAMBDA (EXT)     ; SIN(U) (15.59)
   (TRACE-FYO 'SIN2 (LIST 'TANH EXT)
              (LIST U EXT))             ; d (sinh U)              -2 d U
        (* (- (SECH EXT) 2)             ; ---------- = sech U * ----
           (DIFF EXT (NEXT)))           ;   dx                    dx
              )
)

(PROPERTY 'SIN1 'COTH 'LAMBDA (EXT)     ; SIN(U) (15.34)
   (TRACE-FYO 'SIN2 (LIST 'COTH EXT)
              (LIST U EXT))             ; d (coth U)              -2 d U
        (* (- 1 (CSCH EXT) 2))          ; ---------- = - csch U * ----
           (DIFF EXT (NEXT)))           ;   dx                    dx
              )
)

(PROPERTY 'DIFF 'SECH 'LAMBDA (EXT)     ; SIN(U) (15.33)
   (TRACE-FYO 'SIN2 (LIST 'SECH EXT)
              (LIST U EXT))             ; d (sech U)                    dU
        (* (- (SECH EXT))               ; ---------- = - sech U tanh U * --
           (TANH EXT)                   ;   dx                          dx
           (DIFF EXT (NEXT)))
              )
)

(PROPERTY 'DIFF 'CSCH 'LAMBDA (EXT)     ; SIN(U) (15.56)
   (TRACE-FYO 'SIN2 (LIST 'CSCH EXT)
              (LIST U EXT))             ; d (csch U)                    d U
        (* (- (CSCH EXT))               ; ---------- = - csch U coth U * ---
           (COTH EXT)                   ;   dx                          dx
           (DIFF EXT (NEXT)))
              )
)

;-----------------------------------------------------------------------
; Integrales Elementales
;-----------------------------------------------------------------------

(PROPERTY 'INTEGRAL 'LAMBDA (EXT EXZ)   ; /           /       /
```



```
   (TRACE-FYO 'SIN2 (LIST 'INTEGRAL EXT EXZ)   ; |           |       |
              (SETQ EXT (DIFF EXT EXZ))        ; | d v dx = | U dx - | V dx
              (SETQ EXT (SIMP EXT))            ; |           |       |
                    (PROP EXT)                 ; /           /       /
                    (CLOSE
                       (SETQ EXT (+ EXT (INT (PROP EXT))))
                       (SIMP EXT)
                             EXT)
                    )
              )
              (I EXT (INT EXT EXZ))
           )
)

(PROPERTY 'INTEGRAL '* 'LAMBDA (EX1 EX2)
    ( (MAPLIST (EX1 EX2) ) )
    (TRACE-FYO 'SIN
               EX1
               (LIST 'U EX2)
               (LIST 'DV EX2))
    ( (INTPROD EX1 EX2) )
    (POP #STACK-TRACE)
    (TRACE-FYO 'SIN
               EX1
               (LIST 'U EX2)
               (LIST 'DV EX2))
    ((INTPROD EX1 EX2))
    (POP #STACK-TRACE)
    ()
  ))LAMBDA
)

(PROPERTY 'INTEGRAL '* '(LAMBDA (EX1 EX2 EX3 EX4)
;LOCAL EX5 EX6
;Alcholic #BALI=1/2, #MINUSBALI=-1/2
_Field var INPUT from INTEGRA
   ((FREE EX3 (NEXT) ; Cuando el exponente es constante (NO FIJO)
      (Simpal EX2 (NEXT)
        ((ZEROP (SETQ EX5 (+ EX3 1)))
           (TRACE-FYO '$ln
                      (LIST '* EX2 -1)
                      (LIST 'U EX2))
           (LOGAR EX2)
        )
        (TRACE-FYO 'SIN (LIST '* EX2 (- EX3 1))
                        (LIST 'U EX2)
                        (LIST '# (- EX3 1)))
        (/ (* EX2 EX5) EX5)
      )
      (APPLY (GET 'INTPWR (LOG EX2))
             (ARGS EX2) )
   )
```

```
; "REGRESIVO.ESC"

(DEFUN "C:REGRESIVO" ()
  (SETQ
    ;E1
    C:=> (TAN V1 1/ (SEN V1)(COS V1))
    ;E2
    C:= (EXP V1 1/ 3 (RAD V1))
    ;E3
    C:=> (ATAN V1 1/ (COS V1 (SEN V1)))
    ;E4
    C:=> (ASIN V1 1/ 3 (COS V1))
    ;E5
    C:= (ACOS V1 1/ 3 (SEN V1))
    ;E6
    C:=> (+ (* (SEN V1 2)
               (* (COS V1 2)  3)
    ;E7
    C:=> (- (* (ASC V1 2)
               (* (TAN V1 2)  3)
    ;E8
    C:=> (- (* (ESC V1 2)
               (* (EXP V1 2)  3)
    ;E9
    C:=> (SEN (+ A B))
         (+ (* (SEN A) (COS B)) (* (COS A) (SEN B))))
    ;E10
    C:=> (SEN (- A B))
         (- (* (SEN A) (COS B)) (* (COS A) (SEN B))))
    ;E11
    C:=> (COS (+ A B)) (- (* (COS A) (COS B)) (* (SEN A) (SEN B))))
    ;E12
    C:=> (COS (- A B)) (+ (* (COS A) (COS B)) (* (SEN A) (SEN B))))
    ;E13
    C:=> (TAN (+ A B))
         (/ (+ (TAN A) (TAN B))
            (- 1 (* (TAN A) (TAN B)))))
    ;E14
    C:=> (TAN (- A B))
         (/ (- (TAN A) (TAN B))
            (+ 1 (* (TAN A) (TAN B)))))
    ;E15
    C:=> (EXP (* A B))
         (/ (- (* (EXP A) (EXP B))
            (+ (EXP A) (EXP B))))
    ;E16
    C:=> (EXP (- A B))
         (/ (+ (* (EXP A) (EXP B))
            (- (EXP A) (EXP B))))
    ;E17
    C:=> (SEN (* 2 A))
         (* (* 2 (SEN A))
            (COS A)))
```

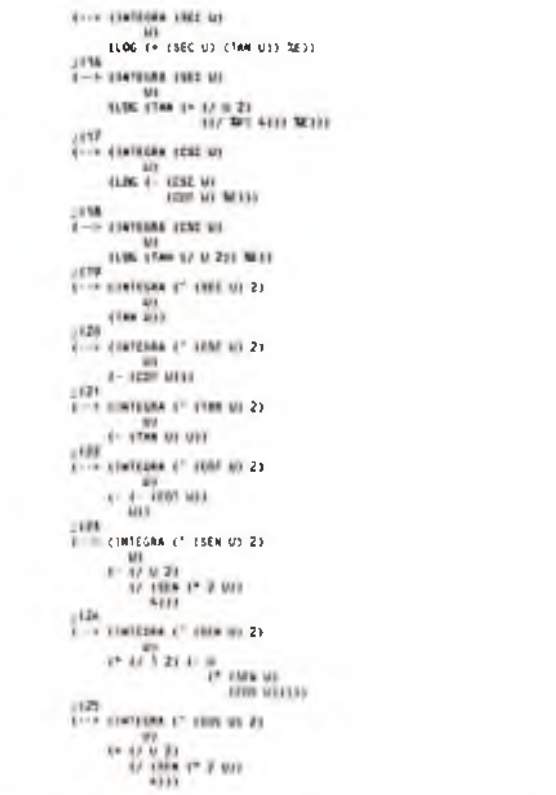


```
    ;E18
    C:=> (COS (* 2 A))
         (- (* (COS A) (COS B))
            (* (SEN A) (SEN B))))
    ;E19
    C:=> (COS (* 2 A))
         (- 1 (* 2 (* (SEN A) (SEN B)))))
    ;E20
    C:=> (COS (* 2 A))
         (- (* 2 (* (COS A) (COS B)) 1))
    ;E21
    C:=> (TAN (* 2 A))
         (/ (* 2 (TAN A))
            (- 1 (* (TAN A) (TAN B))))
    ;E22
    C:=> (SEN (/ A 2))
         (- (/ 1 (- 1 (COS A))) (/ 1 2)))
    ;E23
    C:=> (COS (/ A 2))
         (+ (/ 1 (+ 1 (COS A))) (/ 1 2)))
    ;E24
    C:=> (COS (/ A 2))
         (* (/ (+ 1 (COS A)) 2) (/ 1 2)))
    ;E25
    C:=> (COS (/ A 2))
         (- (/ (+ 1 (COS A)) 2) (/ 1 2)))
    ;E26
    C:=> (TAN (/ A 2))
         (* (/ 1 (- 1 (COS A))) (+ 1 (COS A))) (/ 1 2)))
    ;E27
    C:=> (TAN (/ A 2))
         (- (/ (/ 1 (- 1 (COS A))) (+ 1 (COS A))) (/ 1 2))))
    ;E28
    C:=> (+ (SEN A) (SEN B))
         (* (* 2 (SEN (/ (+ A B) 2))) (COS (/ (- A B) 2))))
    ;E29
    C:=> (- (SEN A) (SEN B))
         (* (* 2 (COS (/ (+ A B) 2))) (SEN (/ (- A B) 2))))
    ;E30
    C:=> (+ (COS A) (COS B))
         (* (* 2 (COS (/ (+ A B) 2))) (COS (/ (A B) 2))))
    ;E31
    C:=> (- (COS A) (COS B))
         (* (* 2 (SEN (/ (+ A B) 2))) (SEN (/ (- (B A) 2)))))
    ;E32
    C:=> (* (SEN A) (SEN B))
         (* (/ 1 2) (- (COS (- A B)) (COS (+ A B)))))
    ;E33
    C:=> (* (COS A) (COS B))
         (* (/ 1 2) (+ (COS (- A B)) (COS (+ A B)))))
    ;E34
    C:=> (* (SEN A) (COS B))
```

```
            (* (/ 1 2) (+ (SEN (- A B)) (SEN (* A B))))))
)
(defun *DIAGRAMAS-DER* ()
  ;01
  (--> (DERIVA (* C X) X) C)
  ;02
  (--> (DERIVA (* C (^ X N)) X) (* (* C N) (^ X (- N 1))))
  ;03
  (--> (DERIVA (+ U V) X) (+ (DERIVA U X) (DERIVA V X)))
  ;04
  (--> (DERIVA (* C U) X) (* C (DERIVA U X)))
  ;05
  (--> (DERIVA (* U V) X)
       (+ (* U (DERIVA V X)) (* V (DERIVA U X))))
  ;06
  (--> (DERIVA (* (* U V) W) X) (+ (+ (* (* U V) (DERIVA W X))
                                      (* (* U W) (DERIVA V X)))
                                   (* (* V W) (DERIVA U X))))
  ;07
  (--> (DERIVA (/ U V) X) (/ (- (* V (DERIVA U X))
                                (* U (DERIVA V X)))
                             (^ V 2)))
  ;08
  (--> (DERIVA (^ U W) X) (* (* W (^ U (- W 1))) (DERIVA U X)))
  ;09
  (--> (DERIVA (SEN U) X) (* (COS U) (DERIVA U X)))
  ;010
  (--> (DERIVA (COS U) X) (* (- (SEN U)) (DERIVA U X)))
  ;011
  (--> (DERIVA (TAN U) X) (* (^ (SEC U) 2) (DERIVA U X)))
  ;012
  (--> (DERIVA (COT U) X) (* (- (^ (CSC U) 2)) (DERIVA U X)))
  ;013
  (--> (DERIVA (SEC U) X) (* (* (SEC U) (TAN U)) (DERIVA U X)))
  ;014
  (--> (DERIVA (CSC U) X) (* (- (* (CSC U) (COT U))) (DERIVA U X)))
  ;015
  (--> (DERIVA (ASEN U) X) (* (/ 1 (^ (- 1 (^ U 2)) (/ 1 2)))
                              (DERIVA U X)))
  ;016
  (--> (DERIVA (ACOS U) X) (* (/ (- 1) (^ (- 1 (^ U 2)) (/ 1 2)))
                              (DERIVA U X)))
  ;017
  (--> (DERIVA (ATAN U) X) (* (/ 1 (+ 1 (^ U 2))) (DERIVA U X)))
  ;018
  (--> (DERIVA (ACOT U) X)
       (* (/ (- 1) (+ 1 (^ U 2))) (DERIVA U X)))
  ;019
  (--> (DERIVA (ASEC U) X)
       (* (/ 1 (* (ABS U)
                  (^ (- (^ U 2) 1) (/ 1 2))))
```



```
          (DERIVA U X)))
  ;020
  (--> (DERIVA (ACSC U) X)
       (* (/ (- 1) (* (ABS U)
                      (^ (- (^ U 2) 1) (/ 1 2))))
          (DERIVA U X)))
  ;021
  (--> (DERIVA (SENH U)
               X)
       (* (COSH U) (DERIVA U X)))
  ;022
  (--> (DERIVA (COSH U)
               X)
       (* (SENH U) (DERIVA U X)))
  ;023
  (--> (DERIVA (TANH U)
               X)
       (* (^ (SECH U) 2) (DERIVA U X)))
  ;024
  (--> (DERIVA (COTH U)
               X)
       (* (- (^ (CSCH U) 2)) (DERIVA U X)))
  ;025
  (--> (DERIVA (SECH U)
               X)
       (* (* (- (SECH U)) (TANH U)) (DERIVA U X)))
  ;026
  (--> (DERIVA (CSCH U)
               X)
       (* (* (- (CSCH U)) (COT U)) (DERIVA U X)))
  ;027
  (--> (DERIVA (^ (SEN U) M)
               X)
       (* (* M (^ (SEN U) (- M 1)))
          (COS U)
          (DERIVA U X)))
  ;028
  (--> (DERIVA (^ (COS U) M)
               X)
       (* (* M (^ (COS U) (- M 1)))
          (- (SEN U))
          (DERIVA U X)))
  ;029
  (--> (DERIVA (^ (TAN U) M)
               X)
       (* (* M (^ (TAN U) (- M 1)))
          (^ (SEC U) 2)
          (DERIVA U X)))
  ;030
  (--> (DERIVA (^ (COT U) M)
               X)
       (* (* M (^ (COT U) (- M 1)))
```



```
(DEFUN PRINTSTRING (#X#1)
  (LOOP (STREAMP #X#1)
        (DECROP #X#1)
                (PRINC #X#1))
  (COND ((NOTIGEAP #X#1)
         ((NOTIGEAP #X#1#))
                (PRINTPRINT (NUMERADOR #X#1) (DENOMINADOR #X#1) #X#1#))
  (PRINTDEAP (LIST #SLASH# (NUMERADOR #X#1) (DENOMINADOR #X#1))
             #X#P #L#P))
)

(DEFUN PRINTPAR (#L#1 #X#2)
  (COND (AND (EVAL (GET '#X#P #X#1))
             (> #L#P (EVAL (GET '#X#P #X#1))))
        (AND (EVAL (GET '#L#P #X#1))
             (NOT (> #X#P (EVAL (GET '#L#P #X#1))))))
                                        (PRINC #X#2)
  )
)

(DEFUN PRINTDELTA (#L#1)
  (LOOP
    ( (NULL #L#T) )
    (SPACES 1)
    (PRINC (ASCII 235))
    (PRINC (NOMBRELLA (POP #L#T)))
    ( (NULL #L#T) )
    (COND ((EQUAL (CAR #L#T))
           (PRINC #SUBSTR#)
           (PRINC (POP #L#T)))
    )
  )
)

;#X#1 es el oper-redu- #L#1 tiene los operandos de #X#1
(DEFUN PRINTPER (#X#1 #L#1 #X#2 #X#3)
 ((ATOM #L#1) ((PRINTFUNC #X#1 #L#1)))
 ((SETQ #X#2 (PROP #L#1)))
 ((ATOM #L#1)
    ((SET '#X#P# #X#1)
     (PRINC #L#1)
     (PRINTSPC #L#T (GET '#P#) #L#T))
     (LOOP (NOT (> (LEN #X#2)
                   (GET '#L#P (CAR #C#T))))
           (PRINT #X#A-(2))
           (PRINTPAR #X#2)
           (PRINC #X#A#2#))
     (PRINTDEAP #X#P (GET '#X#P #X#1)))
    ((SET '#L#P #X#1)
     (PRINTLEN #X#2 0 (GET '#L#P #X#1))
     (PRINTS #X#1 (GET '#L#P #X#1))
     (PRINC #L#T)))
```

```
    (PRINTFUNC #X#1 (CONS #X#2 #L#T#))
((AND (GET '#X#P #X#1)
      (GET '#L#P #X#1))
       (PRINTDEAP #X#2 0 (GET '#L#P #X#1))
       (LOOP
          (SETQ #X#3 (POP #L#T#))
          (((AND (EQ #X#T '+)
                 (NOT (AND (OR (SYMBOLP #X#2)
                               (AND (NOT (ATOM #X#2))
                                    (SET '#X#P (CAR #X#2))
                                    (NOT (< (GET '#X#P (CAR #X#2))
                                            (GET '#L#P #P#K#))))
                               (PPAR (CAR #X#2) (*#1#C #X#2)))
                           (NOT (ATOM #X#3))
                           (SET '#L#P (CAR #X#3))
                           (OR (> (GET '#X#P #P#K#)
                                  (GET '#L#P (CAR #X#3)))
                               (LPAR (CAR #X#3) (SECOND #X#T)))))))
            (SPACES 1))
           (PRINTSPC #L#T (GET '#L#P #X#1))
           (PRINC #X#T)
           (PRINTSPC #X#T (GET '#X#P #X#1))
           (SETQ #X#2 #X#3)
           ((ATOM #L#T))
           (PRINTDEAP #X#2 (GET '#X#P #X#1) (GET '#L#P #X#1)))
       (PRINTDEAP #X#2 (GET '#X#P #X#1)))
(PRINTFUNC #X#1 (CONS #X#2 #L#T#))
)

(DEFUN PRINTPOINT (#X#1 #L#1 #P#1#1 #X#3 #X#4 #L#5 #(#P#L#6#)
  ( (NEGATIVE #P#1#T)
                (SETQ #X#3 0)
                (SETQ #X#4 (ABS #X#T))
                (SETQ #X#5 #X#2)
                (LOOP
                   ( (> #X#4 #X#5) )
                   (SETQ #X#5 (+ #X#5 1))
                   (SETQ #X#5 (* #X#5 *#1#AC BASE*))
                )
                (LOOP
                   (SETQ #X#2 (/ #1#3 1))
                   (SETQ #X#4 (* #L#+ *#1#AC BASE*))
                   ( (OR (< #X#4 #X#3)
                         (> #X#4 #X#5) ) )
                )
                ( (> (ABS #X#3) 4)
                                (PRINTPOINT #X#T #X#2 (- #P#1#T)) )
                (
                  (INITIATIVE #T#1)
                                (SETQ #X#4 (- #X#4)))
                )
                (SETQ #P#1#T (- #P#1#T))
```

```
;;;*****************************************************************
;;;
;;;                    REGLAS DE CONVERSION FIPA-FU
;;;
;;;*****************************************************************

(SETQ #FIAB 3)
(SETQ #INDENT 2)

(PROPERTY '#MAP '\; 0)
(PROPERTY '#MAP '+ 25)
(PROPERTY '#MAP '* 130)
(PROPERTY '#MAP '* 120)
(PROPERTY '#MAP '/ 120)
(PROPERTY '#MAP '+ 100)
(PROPERTY '#MAP '- 100)
(PROPERTY '#MAP '_ 125)

(PROPERTY '#LAP '\; 160)
(PROPERTY '#LAP '[1] 200)
(PROPERTY '#LAP '+ 180)
(PROPERTY '#LAP '* 140)
(PROPERTY '#LAP '* 120)
(PROPERTY '#LAP '/ 120)
(PROPERTY '#LAP '+ 100)
(PROPERTY '#LAP '- 100)
(PROPERTY '#LAP '_ 125)

(PROPERTY '#MAP 'NEW #'#EFIX)
(PROPERTY '#MAP 'ERR #'#EFIX)
(PROPERTY '#MAP 'TAN #'#EFIX)
(PROPERTY '#MAP 'EXC #'#EFIX)
(PROPERTY '#MAP 'SEC #'#EFIX)
(PROPERTY '#MAP 'COT #'#EFIX)
(PROPERTY '#MAP 'SENH #'#EFIX)
(PROPERTY '#MAP 'COSH #'#EFIX)
(PROPERTY '#MAP 'TANH #'#EFIX)
(PROPERTY '#MAP 'EXCH #'#EFIX)
(PROPERTY '#MAP 'SECH #'#EFIX)
(PROPERTY '#MAP 'COTH #'#EFIX)

(PROPERTY '#MAP 'ASEN #'#EFIX)
(PROPERTY '#MAP 'ACOS #'#EFIX)
(PROPERTY '#MAP 'ATAN #'#EFIX)
(PROPERTY '#MAP 'ACSC #'#EFIX)
(PROPERTY '#MAP 'ASEC #'#EFIX)
(PROPERTY '#MAP 'ACOT #'#EFIX)

(PROPERTY '#MAP 'ABS #'#EFIX)
(PROPERTY '#MAP 'LN  #'#EFIX)

(SETQ #INFSLASH T)

(DEFUN PRINTEXP (#EXP)
```



```
                #MAP #LAP #LEVEL)
  ( (LISTP #EXP)
      (SETQ #MAP  T)) )
  ( (LISTP #LAP)
      (SETQ #LAP  T)) )
  ;Si la expresion no atomica o numerica ej: de la forma (* a (* b 1))
  ( (OR (ATOM #EXP)
        (NUMBERP #EXP))
                    (PRINTATOM #EXP) )
  (SETQ #LEVEL #EXP)
  (SETQ #EXP (POP #LEVEL))
  ((ATOM #EXP)
       ((AND (GET 'PRINTEXP #EXP)
             (APPLY (GET 'PRINTEXP #EXP)
                    (LIST #LEVEL))) )
       (PRINTFAB #EXP #PAR((D)
       (PRINTOPER #EXP #LEVEL)
       (PRINTFAB #EXP #PAR(E))
   )
)

(DEFUN LFAB (#EXP #EXZ)
  ((AND (NOT (ATOM #EXZ))
        (SET '#LAP #EXT))
                    ((AND (SET '#MAP (CAR #EXZ))
                          (> (SET '#LAP #EXT)
                             (SET '#MAP (CAR #EXZ))))
                     ((LFAB (CAR #EXZ) (CADR #EXZ))
   )
)

(DEFUN RFAB (#EXP #EXZ)
  ((ATOMOLP #EXZ))
  ((OR (ATOM #EXZ)
       (NOT (SET '#MAP #EXT)))
                              ))
  ((OR (NOT (SET '#LAP (CAR #EXZ)))
       (> (SET '#MAP #EXT)
          (SET '#LAP (CAR #EXZ))))
                              (RFAB (CAR #EXZ) (CADR #EXZ)))
)

(DEFUN ORDEN  (#LST #OR) ; Retorna el orden o grado de la derivacion
  (SETQ #OR 0)
  ((DO
    ((NULL #LST) #OR)
    ((ONE ((INTEGERP (CAR #LST))
                  (SETQ #OR (+ #OR 1) (POP #LST) T ))))
         ((SETQ #OR (MAX1 #OR)))(POP #LST))
    )
   )
)
```

```
                ( (= @R++ @EX3)
                        (PRINTEXP (LIST !* 'READ-BASE* @EX3)
                                  @RBP @LBP) )
                (PRINTEXP (LIST @PUSH (READ! @EX1 @EX2)
                                (LIST !* 'READ-BASE* @EX3) )
                          @RBP @LBP)
        )
        ((NEGATIVE @EX1)
            ( (= @LBP 130)
                  (PRINC @PAR[28])
                  (PRINC @NEWOS)
                  (PRINTPOINT (- @EX1) @EX2 @POINT)
                  (PRINC @PAR[29])
            )
            (PRINC @NEWOS)
            (PRINTPOINT (- @EX1) @EX2 @POINT)
        )
        (SETQ @EX1 (DIVIDE @EX1 @EX2))
        (PRINC (CAR @EX1))
        ((ZEROP (CADR @EX1)))
        (PRINC ".")
        (LOOP
           ((ZEROP @POINT))
           ((ZEROP (CADR @EX1)))
           (SETQ @EX1 ( DIVIDE (* (CADR @EX1)
                                  'READ-BASE*)
                               @EX2 ))
           (SETQ @POINT (- @POINT 1))
           (
             ( (= (CAR @EX1) 10)
                   (PRINC (CAR @EX1))
             )
             (PRINC (ASCII (+ (CAR @EX1) 55 ) ) )
           )
        )
)

;Cuando el operador es una funcion, es decir, un operador prefija.
(DEFUN PRINTFUNC (@LST @LEXT)
  ;Escribir el nombre de la funcion
  (PRINC @EXT)
  (SPACES 1)
  (PRINT(LIST @LEXT) ;Entre parentesis la lista de sus argumentos
)

(DEFUN PRINTSPC (@EX1 @EX2)
   ((OR (= @EX2 120)
        (NOT (ONE-P ((LENGTH @EX1)))))
                                  (SPACES 1))
)

(DEFUN PRINTLIST (@LDES)
```



```
  (PRINC @PAR[28])
  (LOOP
     ((ATOM @LDES))
     (PRINTEXP (POP @LDES))
     ((ATOM @LDES))
     (PRINC @COMA)
     (SPACES 1)
  )
  (PRINC @PAR[29])
)

(PROPERTY 'PRINTEXP '* '(LAMBDA (@LEXT @EX1 @EX2 @EX3 @NUMAUX @DENAUX)
  (SETQ @NUMAUX 0)
  (SETQ @DENAUX 1)
  (SETQ @EX1 (COEFICIENTE (SETQ @EX2 (CDR! @PUSH @LEXT))))
  (
    ( ( AND (NOT (INTEGERP @PUSH))
            (ONE-P (DEN (NUMERADOR @EX1))) )
                              (SETQ @EX1 (NUMERADOR @EX1))
                              (SETQ @EX2 (* @EX1 @EX2))
    )
    (SETQ @EX2 (QUOTIENT @EX2))
  )
  (
    ( (INTEGERP @EX1)
                  (SETQ @EX3 (DENOMINADOR @EX2))
                  (SETQ @EX2 (NUMERADOR @EX2))
    )
    (SETQ @EX3 1)
  )
  ( (ONE-P @EX3)
      ( (INTEGERP @EX1)
          ((MINUS-P @EX1)
              ((PRODUCT @EX2)
                   (PRINTPAR @PUSH @PAR[28])
                   (PRINTOPER @PUSH ((COMA (LIST @NEWOS) (CADR @EX2))
                                    (CDDR @EX2) ) )
                   (PRINTPAR @PUSH @PAR[29])
                   )
              )
              (PRINTEXP (LIST @NEWOS (EX2) @RBP @LBP)
              )
          )
      )
      (PRINTPAR @PUSH @PAR[28])
      ((PRODUCT @EX2)
              (PRINTOPER @PUSH (COMA @EX1 (CDR @EX2)))
              (PRINTPAR @PUSH @PAR[29])
              )
      )
      (PRINTOPER @PUSH (LIST @EX1 @EX2))
      (PRINTPAR @PUSH @PAR[29])
```

```
        T
    )
    (PRINTEXP (LIST #%LAMB (* #%x1 #%x2) #%x3) #%BUF #%AP)
    T
  ; ¿Esta parentesis es de Lambda
)

(PROPERTY 'PRINTEXP '* '(LAMBDA (#%EXP #%BUF #%AP)
    (SETQ #%BUF (CAR #%BUF))
    ((AND (NEQUAL #%EXP (SETQ #%X2 (CADR #%EXP)))
          (NUMBERP #%X2))
          ( (MEMBER(1-P #%X2)
                    (PRINTEXP (LIST #%LAMB 1 #%X1)
                              #%BUF #%AP ) T )
            (PRINTEXP (LIST #%LAMB 1 (LIST '* #%X1 '- #%X2))
                      #%BUF #%AP)
            T
    )
  ) ;Lambda
)

(PROPERTY 'PRINTEXP '* '(LAMBDA (#%EXP #%BUF #%AP)
     (SETQ #%X1 (*(P1) #%EXP))
     ((AND (NOT(ZERO* (SETQ #%X2 (CADR #%EXP))) #%IMPLASH)
           ((MEMBER(-P #%X2)
               (PRINTEXP (LIST '/ 1 #%X1) #%BUF #%AP) T)
           (PRINTEXP (LIST '/ 1 (LIST '* #%X1 (- #%X2))) #%BUF #%AP)
            T)
))

(PROPERTY 'PRINTEXP '- '(LAMBDA (#%EXP)
  ((ATOM (CDR #%EXP))
     ((= #%BP 100)
        (PRIN1 #%"(")
        (PRINC #%"-")
        (PRINTEXP (CAR #%EXP) 132)
        (PRINC #%"+(N) T)
     (PRIN1 #%"-")
     (PRINTEXP (CAR #%EXP) 130) T)))

(PROPERTY 'PRINTEXP 'LOG '(LAMBDA (#%EXP)
    ( (= 10 (CADR #%EXP)) %%)
           (PRINTPAR "LN #%"(")
           (PRINTOPER '(LN (LIST (CAR #%EXP)))
           (PRINTPAR "LN #%"+Dec)
           T)
    ((=Real (CADR #%EXP)) #%(CBAS)
                    (PRINTPAR "LOG #%"(")
                    (PRINTOPER '(LOG (LIST (CAR #%EXP))))
                    (PRINTPAR "LOG #%"(") T)
                    )
)
```



```
(PROPERTY 'PRINTEXP 'INTEGRAL '(LAMBDA (#%EXP #%BUF #%AP)
 (SETQ #%X (CDR #%EXP) #%C (CDR #%X))
 ((AND ((EQ #%(CDR #%X)) #%C) (NUMBERP #%(CAR #%X)))
                              (PRINTPAR)
                              (PRINTEXP
                              (SET-CURSOR (+ #%X 2) #%))))
 (PRIN1 (ASCII 244))
 (TAB-CURSOR (ADD1 #%X) #%C)
 (PRIN1 (ASCII 179))
    (SPACES 1)
    ((AND ((ATOM (CAR #%EXP)) (PRINTEXP (CDR #%EXP))))
          (
            (PRINC #%"d(X)")
            (PRINTEXP (CAR #%EXP))
            (PRINC #%"d(X)")) ))
    (SPACES 1)
    (PRINTDELTA (CDR #%EXP))
((SETQ #%BUF (CDR #%X))
(TAB-CURSOR (+ #%X 2) #%C)
(PRIN1 (ASCII 245))
#%BUF
                    );;Lambda
)

(PROPERTY 'PRINTEXP 'DERIVE '(LAMBDA (#%EXP)
    ( Alt-235 genera &{}la x'{} x )
    (PRIN1 (ASCII 235))
    ((AND ((= (ORDEN (CDR #%EXP))) 1)
                  (PRIN1 #%(DERIV))
                  (PRINC (ORDEN (CDR #%EXP))))))
    ((AND ((ATOM (CAR #%EXP)) (PRINTEXP (CAR #%EXP))))
          ((PRINC #%"d(X)")
           (PRINTEXP (CAR #%EXP))
           (PRINC #%"d(X)") ))
    (SPACES 1)
    (PRINC #%(LAMB))
    (PRINTDELTA (CDR #%EXP))
                    )
)

(PROPERTY 'PRINTEXP '+ '(LAMBDA (#%EXP #%AP)
  (PRINTPAR #%A1 #%AP)(P1)
  (PRINTEXP (POP #%EXP) 0 (QE' '#%EXP #%A1))
  ((AND
     ((A1(1) 1)
     (
       ( (NEQUAL (SETQ #%X1 (CAR #%EXP)))
                    (PRIN1( #%(MINUS)
                    (SETQ #%X1 (- #%X1)) )
       (PRINC #%A1)
     )
     ((A1(1) 1)
     ((ATOM #%EXP))
```

```
    (PRINTEXP HEX) (SET 'BREAP BMAS) (SET 'ALAP BMAS))
  )
 (PRINTEXP BIN) (SET 'BREAP BMAS))
 (PRINTPAR BMAS BPARDER)
 )
                    )
)

(DEFUN POS-CURSOR (X Y)
  ((>= X Y)  (WINDOW-ROWS)
             )))(SET-CURSOR X Y))
  (TERPRI Z); para ir(mpar al ultimo renglon
  (TERPRI)   ; para recorrer un renglon todo
  (SET-CURSOR (- (WINDOW-ROWS) 5) Y)
)
```

APENDICE J

FORMA RAPIDA DE FUNCIONAMIENTO DEL SISTEMA

J1. Contenido del disco del Sistema.
J2. Requerimientos de máquina e inicialización.

J3. El sistema
- **J3.1** Para elegir las opciones que se ven en el Menú
- **J3.2** Utilidad de las opciones que se presentan
- **J3.3** Tipo de expresiones permitidas en el sistema
- **J3.4** Funciones disponibles

J1. El sistema esta grabado en un disco, que contiene lo siguiente:

a) Sistema Operativo MS-DOS
   COMMAND.COM
   IBMBIOS.COM (oculto)
   IBMDOS.COM (oculto)

b) Intérprete de Mu-LISP
   MULISP.COM

c) Sistema de manipulación simbólica
   DEMOS.COM

d) Archivos requeridos por DEMOS durante ejecución
   MENU.AYD
   CALCULO.AYD
   SELECTOR.AYD
   VENTANA.AYD
   SESION.AYD
   ALGEBRA.AYD
   TRASCEN.AYD
   IGUALDAD.LSP
   DERIVADA.LSP
   INTEGRAL.LSP

e) Otros archivos requeridos
   KEYBSP.COM (configuración del teclado en español)
   AUTOEXEC.BAT (iniciación automática del sistema)

J2. Para trabajar con el sistema se requiere:

a) PC compatible con IBM-PC que tenga al menos 512 K de memoria RAM y un manejador de disco.

b) El disco que contiene la información descrita en 1.

c) Colocar el disco en el manejador A y prender la máquina. El sistema comenzará automáticamente presentando la pantalla inicial del sistema.

J3. El sistema

J3.1 Para elegir las opciones que se ven en el Menú

Por medio de la barra de espacio se ilumina la opción deseada del menú y se presiona <ENTER> (Retorno de carro). Otra manera es teclear la inicial de la opción deseada. (L-Lectura, S-Sesiones, E-Esquemas, I-Impresion, M-Miscelanea, V-Ventanas, A-Ayuda y F-Fin). De manera similar en las opciones que se presentan en los submenués.

J3.2 Utilidad de las opciones que se presentan

a) Lectura: Lleva el cursor a la ventana activa y aparece el prompt para teclear una entrada. Entonces se pueden empezar a teclear expresiones.

b) Sesiones: Presenta el submenú para realizar operaciones sobre sesiones, tales como Desplegar, Cargar, Guardar, Borrar y Modificar sesiones.

c) Esquemas: Presenta el submenú para desplegar en la ventana activa las tablas de Equivalencias, Derivadas o Integrales. Las fórmulas desplegadas aquí son usadas en la función APLICA.

d) Impresión: Despliega una sesión en la impresora.

e) Miscelánea: Permite configurar Colores en la pantalla, elegir tipo de Monitor, prender o apagar la campana que marca errores, salir a ejecutar comandos del sistema operativo.

f) Ventanas: Presenta el submenú para realizar operaciones sobre ventanas, tales como Dividir, Cerrar, Moverse de Ventana y Enviar entradas o salidas de sesiones entre dos ventanas.

g) Ayuda: Permite desplegar en la ventana activa un breve resumen de ayuda acerca de las funciones permitidas en el sistema y de las opciones de los menúes.

h) Fin: Termina de ejecutar el sistema.

J3.3 Tipo de expresiones permitidas en el sistema

Cuando se elige la opción de Lectura en la primer pantalla que presenta el sistema pueden empezar a teclearse expresiones. Para ello hay que tener en consideración lo siguiente.

a) Toda expresión debe ser finalizada con uno de los siguientes terminadores: **;**. **$**, **!**, **&**. Los cuales tendrán efectos diferentes sobre la manera en que se evaluará la expresión introducida. Obsérvense los ejemplos que se presentan en los siguientes incisos. Todos llevan al final un terminador.

b) Pueden realizarse asignaciones utilizando el signo =, como se muestra a continuación:

```
A = 5 ;
EXP = A + B ;
```

c) Los operadores válidos en el sistema son: +, -, *, / y ^. Con ellos pueden formarse expresiones tales como:

```
A+B-C;
C/D^2;
```

d) También es permitido utilizar paréntesis para indicar prioridad en la evaluación de las operaciones.

```
A * (B+C^2)&
C / (B^2 - 4 A C);
```

e) Hay disponibles funciones trigonométricas y logaritmicas, como por ejemplo:

```
SEN (A + B);
SEN (A) * COS (B);
LN (A) &
```

Como puede observarse, los argumentos de las funciones trigonométricas (SEN, COS, TAN, COT, SEC, CSC) deben ir entre paréntesis, así como también los de las funciones trigonométricas inversas (ASEN, ACOS, ATAN, ACOT, ASEC, ACSC) e hiperbólicas (SENH, COSH, TANH, COTH, SECH, CSCH).

Es un caso diferente la función LOG, a la cual debe indicársele la base del logaritmo de la siguiente forma:

```
LOG (A, %E);
```

que es lo mismo que:

```
LN(A)$
```

J3.4 Funciones disponibles

Dentro de una expresión también puede ser usado otro tipo de funciones, de manera semejante a las funciones trigonométricas; pero que, según la función, pueden llevar varios argumentos. El número y tipo de argumentos que debe llevar cada una de ellas se describe en el capitulo 3 detalladamente. Por lo pronto tenemos algunos ejemplos:

Las funciones NUMERADOR, DENOMINADOR, COEFICIENTE, BASE, y EXPONENTE, sirven para extraer tales partes de una expresión, por ejemplo:

```
NUMERADOR((A+B)/2);        (con lo cual obtendríamos A+B)
DENOMINADOR(A+B+C);        (" " "  " "   "        "   1)
COEFICIENTE(3 X);          (" " "  " "   "        " 3)
BASE(A^2);                 (" " "  " "   "        " A)
EXPONENTE(A+B);            (" " "  " "   "        " 1)
```

Con la función SUBEXP se puede obtener una parte de una expresión; pero a ésta es necesario darle más argumentos, en donde se le dice, además de la expresión, la forma que tiene y que parte de ella queremos, por ejemplo:

La expresión A SEN(3 B^2)/ B^2 tiene la forma X/Y, o con más detalle podríamos decir que tiene la forma X Y/ Z. Como se puede dar a la función SUBEXP en el segundo argumento cualquiera de esas formas vamos a suponer que queremos obtener de nuestra expresión la parte que dice SEN(3 B^2). Eso se escribiría como:

```
SUBEXP(A SEN(3 B^2)/ B^2, X Y / Z, Y);
```

y obtendríamos: SEN (3 B^2)

Así como estas funciones existen otras, las cuales se clasifican, enumeran y describen en el capítulo 3, que es importante revisar para escribir expresiones correctas y permitidas en el sistema.

Cuando una expresión no observa las reglas, en lugar de que se despliegue la salida correspondiente a la expresión tecleada, se despliega un mensaje que indica que hubo un error. La relación de este tipo de mensajes se puede consultar en el Apéndice K.

APENDICE K

Durante la ejecución del sistema aparecen algunos mensajes cuando sucede algún error. Aparecen en dos lugares:

a) En la ventana, donde debe aparecer el prompt de salida y el resultado, si es que se introduce una entrada inválida. Estos errores se listan a continuación y se da una idea de la causa que pudo haber originado el error.

Caracter invalido: Se ha introducido un caracter no válido en la expresión.

Terminador Esperado : Se tecleo la expresión sin terminador.

")" Esperado : El número de paréntesis está desequilibrado.

Identificador, cte o funcion esperado : Posiblemente se teclaron dos operadores juntos, o no se abrió un paréntesis.

Función no definida : Posiblemente falta indicar una multiplicación o tal vez se tecleo mal el nombre de la función.

Numero o tipo invalido de argumentos: Los argumentos dados a una función en la expresión no corresponden en número o en tipo a lo especificado para esa función.

Arg1 y Arg2 dados, no son estrucuralmente iguales : La estructura dada para la expresión de la que se desea obtener una parte no corresponde correctamente.

Division invalida 0/0 : En alguna operación aritmética se provoca una división entre cero.

Division invalida 1/0 : En alguna operación aritmética se provoca una división entre cero.

Exponenciacion Invalida 0^0 : La operación 0^0 no está definida.

El tercer argumento debe corresponder a dU : El argumento donde debe especificarse la derivada de U no corresponde a la derivada de la expresión dada como U.

b) En el área de mensajes, cuando al pedir una opción o introducir algunos datos ha incurrido un error. O bien, a veces se despliegan letreros que indican que el sistema está realizando alguna acción, éstos últimos no son mensajes de error.

Elija una Opcion : Indica que el sistema está listo para que el usuario teclee una opcion

Selecciona .....: Indica que el sistema espera que el usuario elija alguna de las opciones que se presentan en el área de opciones en

ese momento.

Rango Invalido : Indica que tal rango de entradas para borrar no existe.

Numero de ventana invalido : La ventana dada no existe.

BIBLIOGRAFIA

[1] Manual de Mu-Lisp 87

[2] Manual de Mu Math 83

[3] Manual de Tablas y Formulas Matematicas (SHAUM)

[4] Principles of Compiler Design (Ullman)

[5] Proceso de Construcción de un Programa de Manipulación Simbólica, y Aplicación al Análisis Numérico. (Feliu Davino Sagols Troncoso) Tesis de Licenciatura.

El jurado designado por la Sección de Computación del Departamento de Ingeniería Eléctrica del Centro de Investigación y de Estudios Avanzados del I.P.N. aprobó ésta tesis el 30 de junio de 1989.

Dra Shirley Bromberg Silverstein

Dr Josef Kolar Sabor

M.en C. Oscar Olmedo Aguirre

AUTOR NAVARRO VILLICAÑA, Z.

TITULO UN SISTEMA DE MANIPULACION SIMBOLICA INTERACTIVO

CLASIF. XM 89.12 RGTRO. BI 11,223

NOMBRE DEL LECTOR	FECHA PREST.	FECHA DEVOL.
Carlota Tamayo	23-9-93	28-Sept-93
Dr. Guillermo Morales	1-12-93	2 Dic