UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

CCITT

COMITÉ CONSULTIVO
INTERNACIONAL
TELEGRÁFICO Y TELEFÓNICO

LIBRO AMARILLO

TOMO VI - FASCÍCULO VI.8

LENGUAJE DE ALTO NIVEL DEL CCITT (CHILL)

RECOMENDACIÓN Z.200

VII ASAMBLEA PLENARIA
GINEBRA, 10-21 DE NOVIEMBRE DE 1980

Ginebra 1981

UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

CCITT

COMITÉ CONSULTIVO
INTERNACIONAL
TELEGRÁFICO Y TELEFÓNICO

LIBRO AMARILLO

CORRIGENDUM AL FASCÍCULO VI.8

LENGUAJE DE ALTO NIVEL DEL CCITT (CHILL)

Primera lista de errores de copia en la Recomendación Z.200

VII ASAMBLEA PLENARIA
GINEBRA, 10–21 DE NOVIEMBRE DE 1980

Ginebra 1982

CORRIGENDUM AL FASCÍCULO VI.8

DEL LIBRO AMARILLO

Primera lista de errores de copia en la Recomendación Z.200

Lenguaje de alto nivel del CCITT (CHILL)

1. Introducción

Este documento contiene la primera lista de correcciones de "errores de copia" presentes en la definición del CHILL, recomendación Z.200.

Un "error de copia" es un error tal que su corrección no cambia la interpretación que personas entendidas hayan podido darle a la definición.

El número (1) delante de cada corrección se refiere al hecho de que esta es la primera lista de errores de copia. Correcciones que en un futuro fuesen necesarias se incluirán en esta lista con la numeración (2), (3), etc.

2. Lista de correcciones de errores de copia

(1) En la página 14, segunda línea empezando por abajo,

- sustituya: .... si es una clase M-valuada o una clase M-derivada, ...

  por lo siguiente:
  .... si es una clase M-valuada o una clase M-derivada o una clase M-referencia ....

(1) En la página 23, apartado 3.4.6,

- añada una tercera condición estática (la cual se aplica a la sintaxis derivada para modos intervalo):

  En el caso de utilizar *BIN*, la *expresión literal entera* debe entregar un valor no negativo

(1) En la página 25, apartado 3.6.4,

- añada la siguiente condición estática:

  El modo referenciado origen debe ser parametrizable si se trata de un modo estructura

(1) En la página 38, tercera línea empezando por abajo,

- sustituya: .... declaraciones y especificaciones de parámetros formales, ...

  por lo siguiente:
  .... declaraciones y especificaciones de parámetro y de resultado, ...

(1) En la página 39,

- sustituya la línea de sintaxis (3)

  por lo siguiente:
  <paso> ::= (3)

- sustituya la línea de sintaxis (4)

  por lo siguiente:
  <pos> ::= (4)

(1) En la página 48, apartado 4.1.2,

- sustituya la primera condición estática

  por lo siguiente:

  La clase del *valor* o del *valor constante*
  debe ser compatible con el modo, y el valor entregado
  debe ser uno de los valores definidos por el *modo*.

(1) En la página 51, apartado 4.2.2,

- sustituya la primera condición dinámica

  por lo siguiente:

  Cuando se accede mediante un *nombre de identidad-loc*, éste no debe designar una localización indefinida.

(1) En la página 52, sexta línea empezando por arriba,

- sustituya: .... es el campo variable ....

  por lo siguiente: ... es el campo variable ...

(1) En la página 70, dentro de la semántica, la última línea del tercer párrafo,

- sustituya: .... véase el apartado 9.1.4).

  por lo siguiente: .... véase el apartado 9.1.3).

(1) En la página 72, apartado 5.2.5, cuarta línea del artículo número 6,

- sustituya: .... que no sea *(ELSE)*, debe ....

  por lo siguiente:
  .... que no sea ni *(ELSE)* ni
  <indiferente>, debe ....

(1) En la página 83, apartado 5.2.16, en la segunda línea de la semántica de *GETSTACK*,

- sustituya: .... apartado 7.4 ...

por lo siguiente:

.... apartado 7.9 ...

(1) En la misma página y en la primera línea de la segunda propiedad estática,

- sustituya: .... *SUCC*, es la ....

  por lo siguiente:

  .... *SUCC*, es la clase resultante ....

(1) En la página 84,

- añada una nueva condición estática entre la cuarta y la quinta (actuales):

  El argumento de *SIZE*, *localización de modo estático*, debe ser referenciable.

- En la misma página, la segunda cláusula de la sexta condición estática (trata sobre el argumento *getstack*), debe empezar como sigue:

• El modo estructura variable debe ser parametrizable y debe haber tantas expresiones como ....

(1) En la página 101, cuarta línea,

- sustituya: ... Cada etiqueta de caso define ....

  por lo siguiente:

  ... Cada lista de etiqueta de caso define ...

(1) En la página 103, en las líneas (6.1) y (8.1) de la sintaxis,

- sustituya: *<expresión>*

  por lo siguiente:

  <expresión discreta>

(1) En la página 104, duodécima línea empezando por arriba,

- sustituya: ... la acción hacer.

por lo siguiente:

... la acción hacer, o si se produce un fallo durante la ejecución del programa ejecutor asociado a la acción hacer, o se abandona la acción hacer por medio de una acción retornar o una acción parar.

(1) En la página 109, apartado 6.7, en la primera línea de la semántica,

- sustituya: Una acción llamar provoca ...

por lo siguiente:

Una accion llamar provoca o bien la llamada a un procedimiento o bien la llamada a una rutina predefinida. Una llamada a un procedimiento provoca ....

(1) En la página 110, cuarto requisito de compatibilidad dentro de las condiciones estáticas (atributo LOC),

- añada: Si la *llamada a un procedimiento* no es <u>regional</u>, la *localización* (efectiva) no debe ser <u>regional</u> (véase el apartado 8.2.2).

(1) En la página 119, décima línea empezando por abajo,

- sustituya: ... nombres de valor introducidos ....

por lo siguiente:

... nombres <u>de valor a recibir</u> introducidos ....

(1) En la página 123, apartado 7.1, primera y segunda línea,

- sustituya: ..., *región, poner en espera y elegir, recibir y elegir,* ....

por lo siguiente:

..., *región, recibir y elegir,* ...

(1) En la página 131, apartado 7.4,

- añada la siguiente condición estática:

Todos los nombres mencionados en la *lista de excepciones* deben ser diferentes.

(1) En la página 133, apartado 7.7, en la segunda línea de la semántica,

- sustituya: ... a su objeto de datos declarado ....

  por lo siguiente: ... a sus objetos de datos declarados ...

(1) En la página 135, apartado 8.2.1, segunda línea empezando por abajo,

- sustituya: ..., si y solo si se ha definido ....

  por lo siguiente: ..., si se ha definido ....

(1) En la página 137, apartado 8.2.2, quinta línea del segundo párrafo (valor),

- sustituya: Es un *contenido de localización* regional, ...

  por lo siguiente:
  Es un *contenido de localización* cuya *localización* es regional, ...

(1) En la página 139, apartado 8.4,

- añada un nuevo párrafo despues de "Acción enviar señal":

  Expresión recibir (véase el apartado 5.2.18)

Cuando un proceso ejecuta una expresión recibir, reactiva a otro proceso si y solo si el conjunto de procesos remitentes en espera, de la localización tampón especificada, no es vacío. En este caso el proceso ejecutante recibe un valor de la máxima prioridad de entre los valores de la localización tampón o de los procesos remitentes en espera. Al recibir un valor de un tampón, el proceso elimina el valor del tampón seleccionándose, para su conversión en activo, el proceso remitente en espera con el valor de máxima prioridad, de acuerdo con un algoritmo de ordenación definido en la realización. De este modo el proceso reactivado se elimina de todos los conjuntos de procesos remitentes en espera, almacenándose su valor en el tampón, con la prioridad especificada. Al recibirse directamente un valor procedente de un proceso remitente en espera, se selecciona, para su transformación en activo, el proceso en espera que tenga asociado el valor de máxima prioridad, de acuerdo con un

algoritmo de ordenación definido en la realización. De este modo el proceso reactivado se suprime del conjunto de procesos remitentes en espera, recibiéndose su valor.

(1) En la página 144, apartado 9.1.1.7, primera línea,

- sustituya: ... no es un modo compuesto, ...

  por lo siguiente: .... es un modo discreto o un modo cadena, ...

(1) En la página 151, apartado 9.1.2.5, quinta línea empezando por abajo,

- sustituya: si V es un modo estructura variable, ...

  por lo siguiente:
  si V es un modo estructura variable, ...

(1) En la página 152, apartado 9.1.2.6, tercera línea empezando por abajo,

- sustituya: ... la lista de valores de N.

  por lo siguiente: ... la lista de valores de M.

(1) En la página 184, línea número 34,

- sustituya: *END stacks-1,*

  por lo siguiente: *END stacks_1,*

(1) En la página 189,

- intercale entre las líneas 140 y 141:

```
140a DCL c column,
```

(1) En la página 190,

- sustituya las líneas 28, 29, 30, 31, 32

  por lo siguiente:

```
27a     MANIPULATE:
27b     MODULE
27c        SEIZE NODE, REMOVE, INSERT;
28         DCL NODE_A NODE := (:NULL,NULL,536:);
```

```
29          REMOVE ();
30          REMOVE ();
31          INSERT (NODE_A);
31a     END MANIPULATE;
32  END CIRCULAR_LIST;
```

(1) En la página 192,

- **sustituya el commentario en español por el original (en inglés) y una vez en el original,**

- **sustituya:** *.... lets calla through ....*

  por lo siguiente: *....lets calls through ....*

(1) En la página 193,

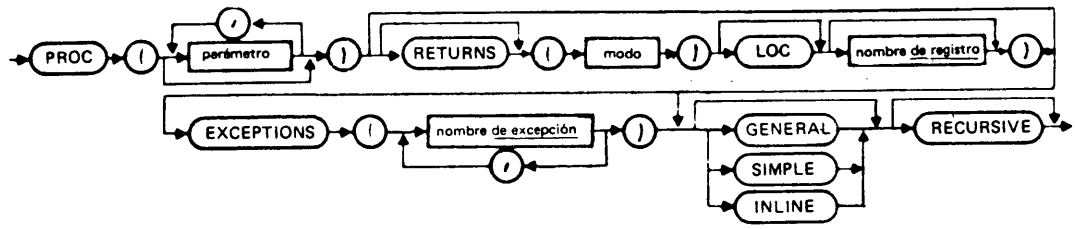
- **sustituya la línea 15**

  por lo siguiente:

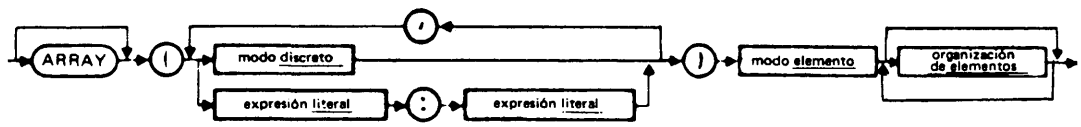
  15 *ACQUIRE, RELEASE, CONGESTED, STEP, READOUT, READY;*

(1) En la página 205,

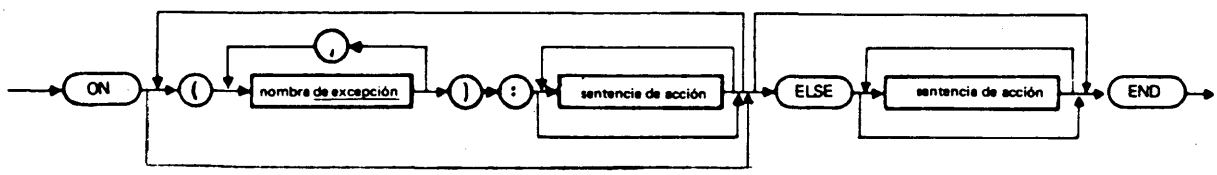
- **sustituya el diagrama de la sintaxis de PROC por lo siguiente:**



- **sustituya el diagrama de la sintaxis de ARRAY por lo siguiente:**



(1) En la página 210, sustituya el diagrama de la sintaxis de un programa ejecutor por lo siguiente:



Impreso en Suiza — ISBN 92-61-01123-3

UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

CCITT

COMITÉ CONSULTIVO
INTERNACIONAL
TELEGRÁFICO Y TELEFÓNICO

LIBRO AMARILLO

TOMO VI - FASCÍCULO VI.8

LENGUAJE DE ALTO NIVEL DEL CCITT (CHILL)

RECOMENDACIÓN Z.200

VII ASAMBLEA PLENARIA
GINEBRA, 10–21 DE NOVIEMBRE DE 1980

Ginebra 1981

ISBN 92-61-01123-3

CONTENIDO DEL LIBRO DEL CCITT EN VIGOR DESPUÉS DE LA SÉPTIMA ASAMBLEA PLENARIA (1980)

LIBRO AMARILLO

Tomo I – Actas e Informes de la Asamblea Plenaria.

Resoluciones y Ruegos.

Recomendaciones sobre:
- la organización de los trabajos del CCITT (serie A);
- los medios de expresión (serie B);
- las estadísticas generales de las telecomunicaciones (serie C).

Lista de las Comisiones de Estudio y de las Cuestiones en estudio.

Tomo II

FASCÍCULO II.1 – Principios generales de tarificación – Tasación y contabilidad en los servicios internacionales de telecomunicaciones. Recomendaciones de la serie D (Comisión III).

FASCÍCULO II.2 – Servicio telefónico internacional – Explotación. Recomendaciones E.100 a E.323 (Comisión II).

FASCÍCULO II.3 – Servicio telefónico internacional – Gestión de la red, ingeniería de tráfico. Recomendaciones E.401 a E.543 (Comisión II).

FASCÍCULO II.4 – Explotación y tarificación de los servicios de telegrafía y «de telemática».[1] Recomendaciones de la serie F (Comisión I).

Tomo III

FASCÍCULO III.1 – Características generales de las conexiones y circuitos telefónicos internacionales. Recomendaciones G.101 a G.171 (Comisiones XV, XVI, CMBD).

FASCÍCULO III.2 – Sistemas internacionales analógicos de portadoras. Características de los medios de transmisión. Recomendaciones G.211 a G.651 (Comisiones XV, CMBD).

FASCÍCULO III.3 – Redes digitales – Sistemas de transmisión y equipos de multiplexación. Recomendaciones G.701 a G.941 (Comisión XVIII).

FASCÍCULO III.4 – Transmisión en línea de señales no telefónicas – Transmisión de señales radiofónicas y de televisión. Recomendaciones de las series H y J (Comisión XV).

Tomo IV

FASCÍCULO IV.1 – Mantenimiento; consideraciones generales, sistemas internacionales de portadoras, circuitos telefónicos internacionales. Recomendaciones M.10 a M.761 (Comisión IV).

FASCÍCULO IV.2 – Mantenimiento de circuitos internacionales de telegrafía armónica y de facsímil y de circuitos internacionales arrendados. Recomendaciones M.800 a M.1235 (Comisión IV).

FASCÍCULO IV.3 – Mantenimiento de circuitos internacionales para transmisiones radiofónicas y de televisión. Recomendaciones de la serie N (Comisión IV).

FASCÍCULO IV.4 – Especificaciones de los aparatos de medida. Recomendaciones de la serie O (Comisión IV).

---

[1] El término «servicios de telemática» se utiliza provisionalmente.

Tomo V – Calidad de transmisión telefónica. Recomendaciones de la serie P (Comisión XII).

Tomo VI

FASCÍCULO VI.1 – Recomendaciones generales sobre la conmutación y la señalización telefónicas – Interfaz con el servicio marítimo. Recomendaciones Q.1 a Q.118 *bis* (Comisión XI).

FASCÍCULO VI.2 – Especificaciones de los sistemas de señalización N.^{os} 4 y 5. Recomendaciones Q.120 a Q.180 (Comisión XI).

FASCÍCULO VI.3 – Especificaciones del sistema de señalización N.° 6. Recomendaciones Q.251 a Q.300 (Comisión XI).

FASCÍCULO VI.4 – Especificaciones de los sistemas de señalización R1 y R2. Recomendaciones Q.310 a Q.490 (Comisión XI).

FASCÍCULO VI.5 – Centrales digitales de tránsito para aplicaciones nacionales e internacionales – Interfuncionamiento de los sistemas de señalización. Recomendaciones Q.501 a Q.685 (Comisión XI).

FASCÍCULO VI.6 – Especificaciones del sistema de señalización N.° 7. Recomendaciones Q.701 a Q.741 (Comisión XI).

FASCÍCULO VI.7 – Lenguaje de especificación y de descripción funcionales (LED) – Lenguaje hombre-máquina (LHM). Recomendaciones Z.101 a Z.104 y Z.311 a Z.341 (Comisión XI).

FASCÍCULO VI.8 – Lenguaje de alto nivel del CCITT (CHILL). Recomendación Z.200 (Comisión XI).

Tomo VII

FASCÍCULO VII.1 – Transmisión y conmutación telegráficas. Recomendaciones de las series R y U (Comisión IX).

FASCÍCULO VII.2 – Equipos terminales para los servicios de telegrafía y «de telemática».[1] Recomendaciones de las series S y T (Comisión VIII).

Tomo VIII

FASCÍCULO VIII.1 – Transmisión de datos por la red telefónica. Recomendaciones de la serie V (Comisión XVII).

FASCÍCULO VIII.2 – Redes de comunicación de datos; servicios y facilidades, equipos terminales e interfaces. Recomendaciones X.1 a X.29 (Comisión VII).

FASCÍCULO VIII.3 – Redes de comunicación de datos; transmisión, señalización y conmutación, aspectos de red, mantenimiento, disposiciones administrativas. Recomendaciones X.40 a X.180 (Comisión VII).

Tomo IX – Protección contra las perturbaciones. Recomendaciones de la serie K (Comisión V). Protección de las cubiertas de cable y de los postes. Recomendaciones de la serie L (Comisión VI).

Tomo X

FASCÍCULO X.1 – Términos y Definiciones.

FASCÍCULO X.2 – Índice del Libro Amarillo.

---

[1] El término «servicio de telemática» se utiliza provisionalmente.

Recomendación Z.200

LENGUAJE DE ALTO NIVEL DEL CCITT (CHILL)

(GINEBRA, 1980)

ÍNDICE

1.0 INTRODUCCIÓN

Esta Recomendación define el lenguaje de programación de alto nivel CHILL del CCITT. CHILL son las siglas inglesas de la expresión "Lenguaje de alto nivel del CCITT".

En un Manual del CCITT se proporcionará una definición alternativa del CHILL en forma matemática estricta. El Manual del CCITT denominado "Introducción al CHILL" constituye una introducción al lenguaje.

1.1 CONSIDERACIONES GENERALES

El lenguaje CHILL se ha diseñado, en principio, para la programación de centrales telefónicas SPC. Sin embargo, se considera suficientemente general para otras aplicaciones (por ejemplo, conmutación de mensajes, conmutación de paquetes, modelado, etc.).

En el diseño del CHILL se han tenido en cuenta los siguientes requerimientos (véase la Cuestión 8/XI del periodo de estudios 1977-1980):

- mejorar la fiabilidad permitiendo una amplia comprobación durante el tiempo de compilación;
- permitir la generación de un código objeto de gran eficacia;
- ser flexible y potente, con objeto de abarcar la gama de aplicaciones requerida y explotar varias modalidades de soporte físico (hardware);
- impulsar el desarrollo de la programación modular estructurada;
- aprendizaje y utilización sencillos.

El lenguaje CHILL implica la existencia de un entorno para el desarrollo del programa. Este entorno puede realizar entre otras funciones la compilación separada, facilidades de entrada/salida y de depuración. Estos aspectos no se definen en esta Recomendación.

Los programas CHILL pueden escribirse en una modalidad independiente de la máquina para la clase de máquinas que se suponga que han de utilizarse o que se hayan propuesto para su utilización en las centrales telefónicas SPC.

El propósito del CHILL no es proporcionar construcciones específicas para cada una de las aplicaciones mencionadas anteriormente, sino que más bien facilita una base general con un amplio número de posibilidades adecuadas para cada aplicación particular.

El CHILL, como lenguaje, es independiente de la máquina. Una realización particular puede, sin embargo, contener objetos de lenguaje definidos en la realización. Los programas que contengan tales objetos no serán, en general, transferibles.

El lenguaje CHILL se ha definido bajo la hipótesis de que será compilado desde el texto fuente al código objeto. No se ha diseñado específicamente para hacer factible la compilación en un solo paso ni para minimizar el tamaño del compilador.

Con objeto de lograr seguridad sin una pérdida inaceptable de eficacia, la mayoría de las comprobaciones pueden hacerse en forma estática. Unas pocas reglas del lenguaje pueden comprobarse solamente en la ejecución. La violación de esta regla provoca una excepción de ejecución. Sin embargo, la generación de comprobaciones de ejecución para estas comprobaciones es facultativa, a menos que se especifique un ejecutor de excepciones definido por el programador.

1.2 EXAMEN DEL LENGUAJE

Un programa CHILL consta esencialmente de tres partes:

- descripción de objetos de datos;
- descripción de acciones que han de efectuarse con los objetos de datos;
- descripción de la estructura del programa.

Los objetos de datos se describen mediante sentencias de datos (sentencias de declaración y de definición). Las acciones se describen mediante sentencias de acción y la estructura del programa se determina mediante sentencias de estructuración del programa.

Los objetos de datos manipulables del CHILL son: valores y localizaciones en las que pueden almacenarse los valores. Las acciones definen las operaciones que han de efectuarse sobre los objetos de datos y el orden en que los valores se almacenan y se recuperan de las localizaciones. La estructura del programa determina el tiempo de vida y visibilidad de los objetos de datos.

El CHILL permite una amplia comprobación estática de la utilización de objetos de datos en un contexto dado.

En los puntos siguientes se facilita un resumen de los diversos conceptos relativos al CHILL. Cada punto es una introducción a un capítulo de igual título que describe el concepto en detalle.

1.3 MODOS Y CLASES

Los objetos de datos manipulables del CHILL son: valores y localizaciones en las que pueden almacenarse los valores.

Una localización tiene asociado un modo. El modo de una localización define el conjunto de valores que pueden residir en dicha localización y otras propiedades asociadas con la localización y los valores que puede contener. (Obsérvese que no todas las propiedades de una localización están determinadas solamente por su modo.) Son propiedades de las posiciones: tamaño, estructura interna, característica de 'sólo lectura', referenciabilidad, etc. Son propiedades de los valores: representación interna, ordenación, operaciones aplicables, etc.

Un valor tiene asociada una clase. La clase de un valor determina los modos de las localizaciones que puede contener dicho valor.

El CHILL proporciona las siguientes categorías de modo:

modos discretos	modos entero, carácter, booleano, conjunto (simbólico) e intervalos correspondientes;
modos conjuntistas	conjuntos de elementos de algún modo discreto;
modos de referencia	referencias ligadas, referencias libres y descriptores utilizados como referencias a localizaciones;
modos compuestos	modos cadena, matriz y estructura;

modos procedimiento	procedimientos considerados como objetos de datos manipulables;
modos ejemplo	identificaciones para los procesos;
modos de sincronización	modos suceso y tampón para los procesos de sincronización y comunicación.

El CHILL proporciona designaciones para un conjunto de modos normalizados. Los modos definidos en el programa pueden introducirse mediante definiciones de modo. Algunas construcciones del lenguaje tienen un modo dinámico. Un modo dinámico es un modo del cual algunas propiedades solamente pueden determinarse en forma dinámica. Los modos dinámicos son siempre modos parametrizados con parámetros determinados en la ejecución. Un modo no dinámico, se denomina modo estático. Un modo designado explícitamente en un programa CHILL es siempre estático.

Ni los modos dinámicos ni las clases tienen una designación en CHILL. Se introducen solamente en el metalenguaje para describir condiciones estáticas y dinámicas del contexto.

1.4 LOCALIZACIONES Y SUS ACCESOS

Las localizaciones son lugares (abstractos) donde pueden almacenarse valores o de donde pueden obtenerse valores. Con objeto de almacenar o extraer un valor, debe de accederse a la localización.

Las sentencias de declaración definen los nombres que deben utilizarse para acceder a una localización.

Estos son:

1. declaraciones de localización;
2. declaraciones identidad-loc;
3. declaraciones referidas a una base.

La primera crea localizaciones y establece nombres de acceso para las localizaciones nuevas. Las dos segundas establecen nuevos nombres de acceso para las localizaciones creadas en cualquier otra parte.

Además de las declaraciones de localización pueden crearse nuevas localizaciones mediante la rutina predefinida *GETSTACK* que proporcionará un valor de referencia (véase más adelante) a la localización así creada.

Una localización puede ser referenciable. Esto significa que existe para la localización un valor referencia correspondiente. Este valor referencia se obtiene como resultado de la operación de referenciación aplicada a la localización referenciable. Desrreferenciando un valor referencia, se obtiene la localización referida. El CHILL exige que ciertas localizaciones sean siempre referenciables, pero para otras localizaciones existe la libertad de decidir en la realización si serán o no referenciables. La referenciabilidad debe ser una propiedad de las localizaciones, determinable estáticamente.

Una localización puede ser de sólo lectura, lo que implica que solamente se puede acceder a la misma para obtener un valor y no para almacenar en ella un nuevo valor (con excepción de la inicialización).

Una localización puede ser compuesta, lo que implica que tiene sublocalizaciones a las que puede accederse por separado. Una sublocalización no es necesariamente referenciable. Una localización que contenga al menos una sublocalización de sólo lectura se dice que tiene la propiedad de sólo lectura. Los métodos de acceso que entregan sublocalizaciones (o subvalores) son formar una subcadena, indizar y segmentar para cadenas y matrices y seleccionar para las estructuras.

Una localización tiene asociado un modo. Si este modo es dinámico, la localización se denomina localización de modo dinámico. (Obsérvese que la palabra dinámico se utiliza solamente en relación con el modo; la localización no es dinámica en el sentido de que varía en el momento de la ejecución, sino que sus propiedades no pueden determinarse completamente en forma estática.)

Las propiedades siguientes de una localización, aunque se determinan estáticamente, no forman parte del modo:

referenciabilidad: si existe o no un valor de referencia para la localización;

clase de almacenamiento: si se ha atribuido o no estáticamente;

regionalidad: si se ha declarado o no que la localización está contenida en una región.

1.5 VALORES Y SUS OPERACIONES

Los valores son los objetos básicos sobre los que se definen las operaciones específicas. Un valor tiene la forma de valor definido (CHILL) o valor indefinido (en sentido CHILL). La utilización de un valor indefinido en contextos específicos, provoca una situación indefinida (en sentido CHILL) considerándose incorrecto el programa.

El CHILL permite utilizar las localizaciones en contextos en los que se requieren valores. En este caso, para obtener el valor contenido se accede a la localización.

Un valor tiene asociada una clase. Los valores fuertes son valores que además de su clase tienen asociado un modo. En ese caso el valor es siempre uno de los valores definidos por el modo. La clase se utiliza para verificación de compatibilidad y el modo para describir las propiedades del valor. Algunos contextos requieren conocer esas propiedades, siendo entonces necesario un valor fuerte.

Un valor puede ser un literal, en cuyo caso designa un valor discreto independiente de la realización, conocido en el momento de la compilación. Un valor puede ser constante, en cuyo caso siempre entrega el mismo valor, es decir, es necesario evaluarlo sólo una vez. Se supone que los valores literal y constante se evalúan antes de la ejecución y no pueden generar una excepción de ejecución. Un valor puede ser regional, en cuyo caso se refiere, de alguna manera, a localizaciones regionales. Un valor puede ser compuesto, es decir, contiene subvalores.

Las sentencias de definición de sinónimos establecen nuevos nombres que designan valores constantes.

1.6 ACCIONES

Las acciones constituyen la parte algorítmica de un programa CHILL.

La acción de asignación almacena un valor (calculado) en una o más localizaciones. La llamada a un procedimiento invoca un procedimiento, la llamada a una rutina predefinida invoca una rutina predefinida (una rutina predefinida es una rutina cuya definición no se ha escrito en CHILL y que tiene un mecanismo más general de transferencia de parámetros y del resultado). Se utilizan las acciones retornar y resultado para el retorno de una llamada a un procedimiento o para establecer el resultado de la misma.

Para controlar el desarrollo secuencial de acciones, CHILL proporciona el siguiente flujo de acciones de control:

acción si	para una ramificación bilateral;
acción caso	para una bifurcación múltiple. La selección de la rama puede basarse en varios valores, es similar a un cuadro de decisión;
acción hacer	para una iteración o parentización;
acción salir	para abandonar de manera estructurada una acción parentizada;
acción causar	para causar una excepción específica;
acción ir	para la transferencia incondicional a un punto etiquetado del programa.

Las sentencias de acción y de datos pueden agruparse para formar un módulo o un bloque principio-fin, que forma una acción (compuesta).

Para controlar los desarrollos simultáneos de acciones, el CHILL proporciona las acciones arrancar, parar, poner en espera, continuar, enviar, poner en espera y elegir, recibir y elegir o la evaluación de una expresión recibir.

1.7 ESTRUCTURA DEL PROGRAMA

Las sentencias de estructuración del programa son el bloque principio-fin, módulo, procedimiento, proceso y región. Las sentencias de estructuración del programa proporcionan los medios de controlar el tiempo de vida de las localizaciones y la visibilidad de los nombres.

El tiempo de vida de una localización es el tiempo durante el cual una localización existe dentro del programa. Las localizaciones pueden declararse explícitamente (en una declaración de localización) o generarse (llamada a rutina predefinida *GETSTACK*), o pueden declararse implícitamente o generarse como resultado de la utilización de construcciones del lenguaje.

Se dice que un nombre es visible en un cierto punto del programa, si puede utilizarse en dicho punto. El alcance de un nombre comprende todos los puntos en los que es visible, es decir donde el objeto designado se identifica por ese nombre.

Los bloques principio-fin determinan la visibilidad de los nombres y el tiempo de vida de las localizaciones.

Se han previsto módulos para restringir la visibilidad de los nombres con objeto de protegerlos contra utilización no autorizada. Mediante las sentencias de visibilidad es posible ejercer un control sobre la visibilidad de los nombres en varias partes del programa.

Un procedimiento es un subprograma (posiblemente parametrizado) que puede invocarse (llamarse) en diferentes lugares dentro de un programa. Puede dar un valor (procedimiento que proporciona valor) o una localización (procedimiento que proporciona localización) o no entregar ningún resultado. En este último caso sólo puede llamarse al procedimiento mediante una acción de llamada a procedimiento.

Los procesos y regiones proporcionan los medios mediante los cuales puede conseguirse una estructura de ejecuciones simultáneas.

Un programa CHILL completo es una lista de módulos o regiones que se considera englobada en una definición (imaginaria) de proceso. Este proceso externo es iniciado por el sistema bajo cuyo control se ejecuta el programa.

1.8 EJECUCIÓN SIMULTÁNEA

El CHILL permite la ejecución simultánea de unidades de programa. La unidad de ejecución simultánea es el proceso. La acción arrancar provoca la creación de un nuevo proceso de la definición de proceso indicada. Se considera entonces que el proceso se ejecuta simultáneamente con el proceso de arranque. El CHILL permite la activación simultánea de uno o más procesos con la misma o diferente definición. La acción parar ejecutada por un proceso provoca su terminación.

Un proceso está siempre en uno de los dos estados siguientes: puede estar activo o en espera. La transición de activo a espera se denomina puesta en espera del proceso. La transición del estado en espera al estado activo se denomina reactivación del proceso. La ejecución de acciones poner en espera sobre sucesos, o de acciones recibir sobre tampones o señales, o de acciones enviar sobre tampones pueden poner en espera el proceso que las ejecuta. La ejecución de una acción continuar sobre sucesos, o de acciones enviar sobre tampones o señales, o de acciones recibir sobre tampones, puede volver nuevamente activo un proceso en espera.

Los tampones y sucesos son localizaciones de uso restringido. Las operaciones enviar, recibir y recibir y elegir se definen en los tampones, las operaciones poner en espera, poner en espera y elegir y continuar se definen sobre sucesos. Los tampones constituyen un modo de sincronizar y transmitir información entre procesos. Los sucesos se utilizan solamente para sincronización. Las señales se definen mediante sentencias de definición de señales. Designan funciones para componer y descomponer listas de valores transmitidas entre procesos. Las acciones enviar y las acciones recibir y elegir permiten la comunicación de una lista de valores y la sincronización.

Una región es una clase especial de módulo. Se utiliza para permitir un acceso mutuamente exclusivo a las estructuras de datos compartidas por varios procesos.

1.9 PROPIEDADES SEMÁNTICAS GENERALES

Las condiciones semánticas (no independientes del contexto) del CHILL son las condiciones de compatibilidad de modos y clases (comprobación de modo) y las condiciones de visibilidad (comprobación de alcance). Las reglas de comprobación de modo determinan cómo pueden utilizarse los nombres y las reglas de comprobación de alcance determinan dónde pueden utilizarse los nombres.

Las reglas de comprobación de modo se formulan en términos de requisitos de compatibilidad entre modos, entre clases y entre modos y clases. Los requisitos de compatibilidad entre modos y clases y entre clases entre sí, se definen en términos de relaciones de equivalencia entre modos. Si aparecen modos dinámicos, la comprobación de modos es parcialmente dinámica.

Las reglas de alcance definen la visibilidad de nombres, que está determinada por la estructura del programa y por sentencias explícitas de visibilidad. Las sentencias explícitas de visibilidad determinan el alcance de los nombres allí mencionados y también de los posibles nombres implicados de los nombres mencionados.

Hay un lugar donde están definidos o declarados los nombres introducidos en un programa. Este lugar se denomina la ocurrencia de definición del nombre. Los lugares donde el nombre se utiliza se denominan ocurrencias de aplicación del nombre. Las reglas de unión de nombres asocian una ocurrencia de definición única con cada ocurrencia de aplicación de un nombre.

1.10 TRATAMIENTO DE LAS EXCEPCIONES

Las condiciones semánticas dinámicas del CHILL son aquellas (no independientes del contexto) que, en general, no pueden determinarse estáticamente. (Se deja para la realización decidir o no la generación de un código para verificar las condiciones dinámicas durante la ejecución.) La violación de una regla semántica dinámica provoca una excepción de ejecución.

Las excepciones también pueden deberse a la ejecución de una acción causar, o condicionalmente, a la ejecución de una acción predicar. Cuando aparece una excepción en un punto dado de un programa, se transfiere el control al ejecutor asociado a dicha excepción si es especificable (es decir si la excepción tiene nombre) y si está especificado. Es posible determinar estáticamente si un ejecutor está o no especificado para una excepción, en un punto dado. Si no se ha especificado un ejecutor explícito, el control puede transferirse a un ejecutor de excepciones definido en la realización.

La mayoría de las excepciones tienen nombre. Este nombre puede ser un nombre de excepción definido en CHILL, un nombre de excepción definido en la realización o un nombre de excepción definido por programa. Obsérvese que cuando se especifica un ejecutor para un nombre de excepción definido en CHILL, debe comprobarse la condición dinámica asociada.

1.11 OPCIONES DE REALIZACIÓN

El CHILL permite utilizar modos enteros definidos en la realización, rutinas predefinidas definidas en la realización, definiciones de procesos definidas en la realización y ejecutores de excepciones definidos en la realización.

Un modo entero definido en la realización debe designarse mediante un nombre de modo definido en la realización. Se considera que debe definirse este nombre, mediante la sentencia de definición de neomodo que no está especificada en CHILL. Dentro del cuadro de reglas sintácticas y semánticas del CHILL, está permitida la extensión de las operaciones aritméticas existentes definidas en CHILL, a los modos enteros definidos en la realización. Como ejemplos de modos enteros definidos en la realización pueden citarse los enteros largos y los enteros cortos.

Una rutina predefinida es un procedimiento cuya definición no está especificada en CHILL y que tiene un mecanismo más general de transferencia de parámetros y de transmisión de resultados que los procedimientos CHILL.

Un nombre de proceso predefinido es un nombre de proceso cuya definición no se ha especificado en CHILL. Un proceso CHILL puede cooperar con los procesos definidos en la realización o iniciar tales procesos.

Un ejecutor de excepciones definido en la realización, es un ejecutor añadido a la definición del proceso (imaginario) más externo. Si este ejecutor recibe el control después de la ocurrencia de una excepción, la realización puede decidir qué acciones deben cumplirse.

2.0 PRELIMINARES

2.1 EL METALENGUAJE

La descripción CHILL consta de dos partes:

- la descripción de la sintaxis independiente del contexto;
- la descripción de las condiciones semánticas.

2.1.1 DESCRIPCIÓN SINTÁCTICA INDEPENDIENTE DEL CONTEXTO

La sintaxis independiente del contexto se describe utilizando una extensión de la forma de Backus-Naur. Las categorías sintácticas se indican mediante una o más palabras españolas escritas en cursiva y encerradas entre paréntesis angulares (< y >). Este indicador se denomina símbolo no terminal. Para cada símbolo no terminal se da una regla de producción en la sección sintáctica apropiada. Una regla de producción para un símbolo no terminal consta del símbolo no terminal a la izquierda del símbolo *::=* y una o más construcciones consistentes en producciones no terminales y/o terminales al lado derecho. Dichas construcciones se separan mediante una barra vertical (|) y designan diferentes producciones posibles para el símbolo no terminal.

A veces el símbolo no terminal incluye una parte subrayada. Esta parte subrayada no forma parte de la descripción independiente del contexto, sino que define una subcategoría semántica (véase el apartado 2.1.2).

Los elementos sintácticos pueden agruparse conjuntamente utilizando llaves ({ y }). La repetición de los grupos encerrados en llaves se indica por un asterisco (*) o un signo más (+). Un asterisco indica que el grupo es facultativo y puede repetirse ulteriormente cualquier número de veces; un más significa que el grupo debe estar presente y puede repetirse ulteriormente cualquier número de veces. Por ejemplo $\{A\}^*$ representa cualquier secuencia de *A*, con inclusión de cero, mientras que $\{A\}^+$ representa cualquier secuencia de al menos un *A*. Si los elementos sintácticos se agrupan utilizando corchetes ([y]) el grupo es facultativo.

Se distingue entre sintaxis estricta, para la que las condiciones semánticas se facilitan directamente, y sintaxis derivada. Se considera que la sintaxis derivada es una extensión de la sintaxis estricta y los aspectos semánticos de la sintaxis derivada se explican indirectamente en términos de la sintaxis estricta asociada.

Debe observarse que se ha elegido la descripción de sintaxis independiente del contexto para facilitar la descripción semántica en este documento y no para facilitar un algoritmo específico de análisis (por ejemplo, se han introducido ciertas ambigüedades independientes del contexto con fines aclaratorios).

2.1.2 DESCRIPCIÓN SEMÁNTICA

Para cada categoría sintáctica (símbolo no terminal) se facilita la descripción semántica en los apartados semántica, propiedades estáticas, propiedades dinámicas, condiciones estáticas y condiciones dinámicas.

El apartado semántica describe los conceptos designados por las categorías sintácticas (es decir, su significado y comportamiento).

El apartado propiedades estáticas define propiedades semánticas de la categoría sintáctica determinables estáticamente. Se utilizan estas propiedades en la formulación de las condiciones estáticas y/o dinámicas en los apartados donde se utiliza la categoría sintáctica.

El apartado propiedades dinámicas define, cuando corresponda, las propiedades de la categoría sintáctica que se conocen solamente en forma dinámica.

El apartado condiciones estáticas describe condiciones comprobables estáticamente y dependientes del contexto que deben satisfacerse cuando se utiliza la categoría sintáctica. En la sintaxis se expresan algunas condiciones estáticas, mediante una parte subrayada del símbolo no terminal (véase el apartado 2.1.1). Esta utilización requiere que el no terminal pertenezca a una subcategoría semántica específica. Por ejemplo, *<expresión booleana>* es idéntico a *<expresión>* en sentido independiente del contexto, pero semánticamente requiere que *expresión* pertenezca a la clase booleana. La parte subrayada se utiliza a veces en el texto como adjetivo calificativo del no terminal. Por ejemplo, la frase "la *expresión* es constante" es idéntica a "la *expresión* es una *expresión constante*".

El apartado condiciones dinámicas describe las condiciones dependientes del contexto que deben satisfacerse durante la ejecución. En algunos casos las condiciones son estáticas única y exclusivamente si no intervienen modos dinámicos. En estos casos la condición se menciona en el apartado condiciones estáticas y se hace referencia a ella en el apartado condiciones dinámicas.

En la descripción semántica los no terminales se escriben en cursiva sin paréntesis angulares para indicar los objetos sintácticos.

2.1.3 EJEMPLOS

Para la mayor parte de los apartados de sintaxis hay un apartado de ejemplos que proporciona uno o más ejemplos de las categorías sintácticas definidas. Estos ejemplos se han extraído de un conjunto de ejemplos de programas contenido en el apéndice D. Las referencias indican por medio de qué reglas sintácticas se obtiene cada ejemplo y de qué ejemplo se toma.

Ejemplo *6.20 (d+5)/5 (1.2)* indica un ejemplo de la cadena terminal *(d+5)/5*, obtenido mediante la regla *(1.2)* del apartado sintaxis correspondiente, tomada del ejemplo de programa N.º *6* línea *20*.

2.1.4 REGLAS DE UNIÓN DEL METALENGUAJE

A veces la descripción semántica menciona nombres especiales CHILL (véase el apéndice C). Estos nombres especiales se utilizan siempre con su significado CHILL y en consecuencia no están influidos por las reglas de unión de un programa CHILL efectivo.

2.2 VOCABULARIO

Los programas se representan utilizando el alfabeto N.º 5 del CCITT descrito en la Recomendación V.3 (véase el apéndice A.1). Es posible representar cualquier programa CHILL utilizando un conjunto mínimo de caracteres que es un subconjunto del código básico del alfabeto N.º 5 del CCITT (véase el apéndice A.2).

Los elementos del léxico CHILL son:

- símbolos especiales
- nombres
- literales

El apéndice B contiene una lista de los símbolos especiales.

Los nombres se forman de acuerdo con la siguiente sintaxis:

sintaxis:

<nombre> ::= *(1)*
<letra> { *<letra>* | *<dígito>* |_}* *(1.1)*

El símbolo de subrayado (_) forma parte del nombre, es decir, el nombre *LIFE_TIME* es diferente del nombre *LIFETIME*. En caso de que se disponga de un alfabeto con letras minúsculas, éstas pueden utilizarse dentro de los nombres. Las letras mayúsculas y minúsculas son diferentes, es decir, *Estado* y *estado* son dos nombres diferentes.

El lenguaje tiene varios nombres especiales con significados predeterminados, (véase el apéndice C). Algunos de ellos están reservados, es decir, no pueden utilizarse para otros fines a menos que se hayan liberado explícitamente mediante un directivo de liberación.

En el caso en que se utilice un alfabeto con letras mayúsculas y minúsculas, los nombres especiales pueden estar representados totalmente con mayúsculas o totalmente con minúsculas. Los nombres reservados solamente lo están dentro de la representación elegida (por ejemplo, si se eligen las minúsculas, *row* estará reservado, y *ROW* no lo estará).

2.3 UTILIZACIÓN DE ESPACIOS

Los espacios pueden utilizarse para delimitar los elementos léxicos de un programa. Los elementos léxicos se terminan en el primer carácter que no puede ser parte del elemento lexical. Por ejemplo, *IFBTHEN* será considerado un *nombre* y no el comienzo de la acción *IF B THEN*, *//** se interpretará como un símbolo de concatenación (//) seguido de un asterisco (*) y no como símbolo de dividir (/) seguido de un paréntesis de iniciación de comentario (/*). Los espacios contiguos tienen el mismo efecto delimitador que un solo espacio.

2.4 COMENTARIOS

sintaxis:

<comentario> ::= *(1)*
/ <cadena de caracteres> */* *(1.1)*

<cadena de caracteres> ::= *(2)*
{*<carácter>*}* *(2.1)*

semántica: Un *comentario* facilita información al lector de un programa. No tiene influencia sobre la semántica del programa.

propiedades estáticas: Puede insertarse un *comentario* en todos los lugares donde estén permitidos los espacios como delimitadores.

condiciones estáticas: La *cadena de caracteres* no debe contener la secuencia especial: asterisco barra (*/).

ejemplos: *4.1 /* from collected algorithms from CACM nr.93 */* *(1.1)*

2.5 CARACTERES DE FORMATO

Los caracteres de formato BS (retroceso), CR (retroceso del carro), FF (página siguiente), HT (tabulación horizontal), LF (cambio de renglón), y VT (tabulación vertical) del alfabeto N.º 5 del CCITT (posiciones FE_0 a FE_5) no se mencionan en la descripción sintáctica independiente del contexto CHILL. Sin embargo, en una realización de programas CHILL pueden utilizarse estos caracteres de formato. Cuando se emplean, tienen el mismo efecto delimitador que el espacio. No deben utilizarse dentro de elementos léxicos.

2.6 DIRECTIVOS DE COMPILACIÓN

sintaxis:

<cláusula directivo> ::= *(1)*
<> *<directivo>* {,*<directivo>*}* [<>] *(1.1)*

<directivo> ::= *(2)*
<directivo CHILL> *(2.1)*
| *<directivo de realización>* *(2.2)*

<directivo CHILL> ::= *(3)*
<directivo de liberación> *(3.1)*

<directivo de liberación> ::= *(4)*
FREE(<lista de nombres reservados>) *(4.1)*

<lista de nombres> ::= *(5)*
<nombre> {,*<nombre>*}* *(5.1)*

semántica: Una cláusula directivo facilita información al compilador. Con excepción del directivo de liberación, esta información se especifica en un formato definido en la realización.

Un directivo de realización no debe influir la semántica del programa, es decir un programa con directivos de realización es correcto, en sentido CHILL, única y exclusivamente si es correcto sin estos directivos.

Un directivo de liberación se aplica a una unidad de compilación. Liberará los nombres reservados especificados en la *lista de nombres reservados* de forma que puedan redefinirse en la unidad de compilación.

propiedades estáticas: Una *cláusula directivo* puede insertarse en todos los lugares donde se permiten los espacios. Tiene el mismo efecto delimitador que un espacio. Los nombres utilizados en una misma *cláusula directivo* siguen un sistema de unión de nombres definido en la realización que no influye las reglas de unión de los nombres del CHILL (véase el apartado 9.2.8).

condiciones estáticas: El símbolo facultativo de fin de directivo (<>) sólo puede omitirse si se lo coloca inmediatamente antes de un punto y coma (es decir, la *cláusula directivo* se termina con el primer signo <> o un punto y coma. Sin embargo el punto y coma no pertenece a la *cláusula directivo*. En consecuencia, un *directivo* no puede contener el símbolo <> ni el punto y coma, a menos que estén colocados entre paréntesis, véase más adelante.) Si aparecen paréntesis en un *directivo de realización*, deben equilibrarse de manera adecuada, y si aparecen dentro de un paréntesis, un punto y coma o un símbolo de fin de directivo éstos no constituyen el punto final del *directivo*.

ejemplos:

```
15.1    <> FREE (STEP)          (1.1)
15.1    FREE (STEP)             (4.1)
```

3.0 MODOS Y CLASES

3.1 CONSIDERACIONES GENERALES

Una localización tiene asociado un modo; un valor tiene asociada una clase. El modo asociado a una localización define el conjunto de valores que dicha localización puede contener, los métodos de acceso a la localización y las operaciones permitidas con los valores. La clase asociada a un valor constituye una forma de determinar los modos de las localizaciones que pueden contener el valor. Algunos valores son fuertes. Un valor fuerte tiene asociado una clase y un modo. Este modo es siempre compatible con la clase del valor y el valor es uno de los valores definidos por el modo. Se requieren valores fuertes en aquellos contextos de valor en los que se necesita información de modo.

3.1.1 MODOS

El CHILL tiene modos estáticos (es decir, modos de los que todas las propiedades pueden determinarse estáticamente) y modos dinámicos (es decir, modos algunas de cuyas propiedades solamente se conocen en el momento de la ejecución). Los modos dinámicos son siempre modos parametrizados con parámetros de ejecución.

Los modos estáticos se designan en el programa mediante producciones terminales de la categoría sintáctica *modo*.

Los modos dinámicos no tienen designaciones en CHILL. Sin embargo, en este documento se introducen designaciones virtuales para designar los modos dinámicos. Estas designaciones virtuales irán precedidas del signo *&*; así, por ejemplo *& VM(i)*, indica un modo dinámico parametrizado con el parámetro de ejecución i.

Adicionalmente, se introducen en algunos lugares designaciones virtuales para los modos estáticos. Esto se hace para aquellos modos que no están o no pueden designarse explícitamente en el texto del programa, sino que se introducen virtualmente mediante determinadas construcciones del lenguaje. Estos modos se designan también mediante designaciones virtuales precedidas del signo *&*.

3.1.2 CLASES

En CHILL no existe designación para las clases.

Existen las siguientes especies de clases y cualquier valor en un programa CHILL tendrá una clase de una de esas especies:

- Para cualquier modo M existe la clase M-valuada. Todos los valores de esta clase y sólo estos son fuertes y el modo asociado al valor es M.
- Para cualquier modo M de novedad nula (véase el apartado 9.1.1.1), existe la clase M-derivada.
- Para todo modo M existe la clase M-referencia.
- Clase nula.
- Clase general.

Las últimas dos clases son constantes, es decir no dependen del modo M. Se dice que una clase es dinámica única y exclusivamente si es una clase M-valuada o una clase M-derivada, donde M es un modo dinámico.

3.1.3 PROPIEDADES DE LOS MODOS Y DE LAS CLASES Y SUS RELACIONES

En el capítulo 9 se definen todas las propiedades fundamentales de los modos y, clases y sus relaciones. Se enumeran a continuación dichas propiedades y relaciones:

1. Un modo M tiene una novedad.

2. Un modo M puede ser de sólo lectura.

3. Un modo M puede tener la propiedad de sólo lectura.

4. Un modo M puede tener la propiedad de referenciación.

5. Un modo M puede tener la propiedad de marcación parametrizada.

6. Un modo M puede tener la propiedad de sincronización.

7. Un modo M puede ser definido por un modo N.

8. Un modo M puede ser de lectura compatible con un modo N (asimétrico).

9. Un modo M puede ser compatible con una clase C (en este caso, se dice que C es compatible con M).

10. Una clase C puede tener un modo raíz.

11. Una clase C puede ser compatible con una clase D (simétrica).

12. Dada una lista de clases compatibles, existe la clase resultante.

Para cada modo se definen propiedades específicas en el apartado correspondiente. Se dice que una propiedad es hereditaria si, cuando se verifica para un modo específico, también se verifica para todos los modos definidos por ese modo. En consecuencia, las propiedades hereditarias no se definirán explícitamente para los nombres de modo. Cada propiedad que se verifica para un modo también se verifica para el modo precedido por la palabra clave *READ* (excepto en algunos casos en que interviene la propiedad de sólo lectura: estos casos se indican explícitamente). En consecuencia, las propiedades no se definirán explícitamente en los modos precedidos por *READ*.

3.2 DEFINICIONES DE MODOS

3.2.1 CONSIDERACIONES GENERALES

sintaxis:

<definición de modo> :: = (*1*)
  <lista de nombres> = *<modo definidor>* (*1.1*)

<modo definidor> :: = (*2*)
  <modo> (*2.1*)

sintaxis derivada: Una *definición de modo* cuya *lista de nombres* consta de más de un *nombre*, se deriva de varias definiciones de modo, una para cada nombre, separadas por comas, con el mismo *modo definidor*.

Por ejemplo, *NEWMODE DOLLAR, POUND = INT*, se deriva de *NEWMODE DOLLAR = INT, POUND = INT*.

semántica: Las definiciones de modo definen uno o más nombres que han de ser nombre de modo es decir nombres que designan modos. Las definiciones de modo aparecen dentro de las sentencias de definición de neomodo y de sínmodo. La diferencia entre un neomodo y un sínmodo radica en el tratamiento que les reservan los algoritmos de equivalencia de modo (véase el apartado 9.1). Por definición, se transfieren al nombre de modo definido, todas las propiedades hereditarias del modo definidor. Las definiciones de modo pueden ser (mutuamente) recursivas.

propiedades estáticas: Un nombre de modo puede ser uno de los nombres de modo definidos en el lenguaje *INT*, *BOOL*, *CHAR*, *PTR*, *INSTANCE*, *EVENT* o un nombre definido en una *definición de modo*.

Un nombre de modo que no pertenezca a la lista de nombres de modo definidos en el lenguaje tiene un único modo definidor, que es el modo designado por el *modo definidor* en la *definición de modo* que lo define.

Un conjunto de definiciones recursivas es un conjunto de definiciones de modo o definiciones de sinónimo (véase el apartado 5.1) tales que el *modo definidor* en cada *definición de modo* o *valor constante* o *modo* en cada *definición de sinónimo* es, o contiene directamente, un nombre de modo o un nombre de sinónimo o un nombre de elemento de conjunto definido por una definición del conjunto.

Un conjunto de definiciones recursivas de modo es un conjunto de definiciones recursivas que tienen solamente definiciones de modo (cualquier conjunto de definiciones recursivas debe ser un conjunto de definiciones recursivas de modo, véase el apartado 5.1).

Un modo que sea o contenga un nombre de modo definido en un conjunto de definiciones recursivas de modo se dice que designa un modo recursivo. Un camino, en un conjunto de definiciones recursivas de modo, es una lista de nombres de modo cada uno de los cuales está indizado con un marcador tal que:

- todos los nombres del camino tienen una definición diferente;
- para cada nombre, su sucesor es, u ocurre directamente, en su modo definidor (el sucesor del último nombre es el primer nombre);
- el marcador indica unívocamente la posición del nombre en el modo definidor de su predecesor (el predecesor del primer nombre es el último nombre).

Ejemplo: *NEWMODE M = STRUCT (i M, n REF M)* contiene dos caminos: $\{M_i\}$ y $\{M_n\}$.

Un camino es seguro única y exclusivamente si por lo menos uno de sus nombres está contenido en un *modo de referencia* o en un *modo descriptor* o en un *modo procedimiento* en el lugar señalado. El modo debe ser o estar contenido en el *modo definidor* del predecesor del *nombre de modo*.

condiciones estáticas: Para cualquier conjunto de definiciones recursivas de modo, todos los caminos deben ser seguros (el primer camino del ejemplo anterior no es seguro).

ejemplos:

```
1.12    operand-mode = INT                          (1.1)
3.3     complex = STRUCT (re, im INT)               (1.1)
```

3.2.2 *DEFINICIONES DE SÍNMODOS*

sintaxis:

<sentencia de definición de sínmodo> :: = (1)
SYNMODE <definición de modo> {, *<definición de modo>*}*; (1.1)

semántica: Las sentencias de definición de sínmodo definen nombres que designan modos que son sinónimos con su modo definidor. En el apartado 9.1 se explica el tratamiento preciso de los nombres definidos en una definición de sínmodo.

propiedades estáticas: Se dice que un *nombre* es un nombre de sínmodo única y exclusivamente si se define en una *definición de modo* en una *sentencia de definición de sínmodo*. Se dice que un nombre de sínmodo es sinónimo de un modo dado (o recíprocamente se dice que el modo dado es sinónimo con el nombre de sínmodo) única y exclusivamente si:

- o el modo dado es el *modo definidor* del nombre de sínmodo;
- o el *modo definidor* del nombre de sínmodo es por sí mismo un nombre de sínmodo, sinónimo del modo dado.

ejemplos:

6.3 SYNMODE mes = SET (jan, feb, mar, apr, may, jun,
jul, aug, sep, oct, nov, dec) (1.1)

3.2.3 *DEFINICIONES DE NEOMODO*

sintaxis:

<sentencia de definición de neomodo> :: = (1)
NEWMODE <definición de modo> {, *<definición de modo>*}*; (1.1)

semántica: Las sentencias de definición de neomodo, definen nombres que designan modos que no son sinónimos con el modo definidor. Los valores definidos por un neomodo, son los definidos por el modo definidor. En el apartado 9.1 se define el tratamiento preciso de los nombres definidos en una sentencia de definición de neomodo.

propiedades estáticas: Se dice que un *nombre* es un nombre de neomodo, única y exclusivamente si se define en una *definición de modo* en una *sentencia de definición de neomodo*.

Si el *modo definidor* es un modo intervalo entonces, junto con el nombre de neomodo definido, se introduce un nuevo nombre virtual designado por *&nombre_progenitor* que indica el modo progenitor del nombre de neomodo. Los valores definidos por este modo progenitor virtual son los valores del modo progenitor del modo intervalo definidor. Los límites superior e inferior del modo progenitor virtual son los del modo progenitor del *modo intervalo* definidor.

Si el *modo definidor* es un modo cadena, se introducen los nuevos modos virtuales: *&nombre(i)* para cada *i* mayor que la longitud de la cadena del nombre de neomodo donde *&nombre*, designa el neomodo introducido. Esta *i* designa la longitud de la cadena del modo virtual. La propiedad hereditaria, cadena de bit o de carácter, del nombre de neomodo se transfiere a los modos virtuales.

ejemplos:

```
11.4   NEWMODE línea = INT (1:8);                          (1.1)
11.10  NEWMODE cuadro = ARRAY (línea) ARRAY (columna)
         cuadrado.                                         (1.1)
```

3.3 CLASIFICACIÓN DE MODOS

sintaxis:

```
<modo> :: =                                                (1)
     <modo no compuesto>                                   (1.1)
   | <modo compuesto>                                      (1.2)

<modo no compuesto> :: =                                   (2)
     <modo discreto>                                       (2.1)
   | <modo conjuntista>                                    (2.2)
   | <modo de referencia>                                  (2.3)
   | <modo procedimiento>                                  (2.4)
   | <modo ejemplo>                                        (2.5)
   | <modo de sincronización>                              (2.6)
```

semántica: En un programa CHILL, los modos se designan mediante producciones terminales de la categoría sintáctica *modo*. A lo largo de este capítulo se definirán las propiedades específicas de los diferentes modos. Se definen las relaciones de igualdad *(=)* y desigualdad *(/=)* para el conjunto de valores de cualquier modo dado (véase el apartado 5.3).

propiedades estáticas: Un *modo* tiene un tamaño que es el valor entregado por *SIZE (M)* donde *M* es un nombre virtual de sínmodo, que es sinónimo de *modo*.

3.4 MODOS DISCRETOS

3.4.1 CONSIDERACIONES GENERALES

sintaxis:

```
<modo discreto> :: =                                       (1)
     <modo entero>                                         (1.1)
   | <modo booleano>                                       (1.2)
   | <modo carácter>                                       (1.3)
   | <modo conjunto>                                       (1.4)
   | <modo intervalo>                                      (1.5)
```

semántica: Los modos discretos definen conjuntos y subconjuntos de valores bien ordenados. Todos los modos discretos que no son modos intervalo, pueden ser modos progenitores de modos intervalo (véase el apartado 3.4.6). Todos los modos discretos definen un límite superior y un límite inferior que designan, respectivamente, los valores máximo y mínimo.

3.4.2 *MODOS ENTEROS*

sintaxis:

```
<modo entero> :: =                                          (1)
    [READ] INT                                              (1.1)
  | [READ] BIN                                              (1.2)
  | [READ] <nombre de modo entero>                          (1.3)
```

sintaxis derivada: BIN es la sitaxis derivada de INT.

semántica: Un modo entero define un conjunto de valores enteros, con signo comprendidos entre límites definidos en la realización, sobre los cuales se definen las operaciones de ordenación y aritméticas usuales (véase el apartado 5.3.2). Una realización puede definir otros modos enteros con límites diferentes (por ejemplo *LONG_INT, SHORT_INT*, ...) que pueden también utilizarse como modos progenitores de intervalos (véase el apartado 11.2).

propiedades estáticas: Un modo entero tiene las siguientes propiedades hereditarias:

- El límite superior y el límite inferior de un modo entero son los literales que designan respectivamente los valores máximos y mínimos definidos por el modo entero.
- El número de valores de un modo entero se define en la realización.

ejemplos:

```
1.4  INT                                                    (1.1)
```

3.4.3 *MODOS BOOLEANOS*

sintaxis:

```
<modo booleano> ::=                                         (1)
    [READ] BOOL                                             (1.1)
  | [READ] <nombre de modo booleano>                        (1.2)
```

semántica: Un modo booleano define los valores lógicos de verdad *(TRUE* y *FALSE)* con las operaciones booleanas usuales (véase el apartado 5.3.2). *TRUE* es mayor que *FALSE*.

propiedades estáticas: Un modo booleano tiene las siguientes propiedades hereditarias:

- El límite superior de un modo booleano es *TRUE*; su límite inferior es *FALSE*.
- El número de valores definido por un modo booleano, es 2.

ejemplos:

```
5.4  BOOL                                                   (1.1)
```

3.4.4 *MODOS CARÁCTER*

sintaxis:

<modo carácter> ::= *(1)*
[*READ*] *CHAR* *(1.1)*
| [*READ*] *<nombre de modo carácter>* *(1.2)*

semántica: Un modo carácter define valores carácter tales como se describen en la versión de referencia del alfabeto internacional N.º 5 del CCITT (Recomendación V3, véase el apéndice A1). Este alfabeto define también la ordenación de los caracteres.

propiedades estáticas: Un modo carácter tiene las siguientes propiedades hereditarias:

- El límite superior y el límite inferior de un modo carácter son los literales de cadena de carácter de longitud 1 que designan, respectivamente, los valores máximo y mínimo definidos por *CHAR*.
- El número de valores definido por un modo carácter es 128.

ejemplos:

8.4 CHAR *(1.1)*

3.4.5 *MODOS CONJUNTO*

sintaxis:

<modo conjunto> ::= *(1)*
[*READ*] *SET (<lista de conjunto>)* *(1.1)*
| [*READ*] *<nombre de modo conjunto>* *(1.2)*

<lista de conjunto> ::= *(2)*
<lista de conjunto numerada> *(2.1)*
| *<lista de conjunto no numerada>* *(2.2)*

<lista de conjunto numerada> ::= *(3)*
<elemento de conjunto numerado> {,*<elemento de conjunto numerado>*}* *(3.1)*

<elemento de conjunto numerado> ::= *(4)*
<nombre> = *<expresión literal entera>* *(4.1)*

<lista de conjunto no numerada> ::= *(5)*
<elemento de conjunto>{, *<elemento de conjunto>*}* *(5.1)*

<elemento de conjunto> ::= *(6)*
<nombre> *(6.1)*
| *<valor innominado>* *(6.2)*

| *<valor innominado> ::=* *(7)*
* *(7.1)*

semántica: Un modo conjunto define un conjunto de valores nominados o innominados. Los valores nominados se designan por los nombres en la *lista de conjunto*; los valores innominados son los restantes valores. La representación interna de los valores nominados es el valor entero asociado con el valor nominado (véase más adelante). Esta representación define también la ordenación de los valores.

propiedades estáticas: Un modo conjunto tiene las siguientes propiedades hereditarias:

- Un modo conjunto tiene un conjunto de nombres de elemento de conjunto que es el conjunto de nombres de elemento de su *lista de conjunto*.

  Cada nombre de elemento de conjunto de un modo conjunto, tiene un valor (representación) entero asociado que, en caso de *una lista de conjunto numerada*, es el valor entregado por la *expresión literal entera* en el *elemento de conjunto numerado* en el que aparece el nombre del elemento de conjunto. En los demás casos, es uno de los valores 0,1,2 ... etc., de acuerdo con su posición en la *lista de conjunto no numerada*. Por ejemplo: *SET (*, A, *, B, *)*. A tiene asociado un valor de representación 1 y B un valor de representación 3.

- Un modo conjunto tiene un límite superior y un límite inferior que son sus nombres de elemento de conjunto que designan, respectivamente, los valores nominados máximo y mínimo.

- El número de valores de un modo conjunto, en caso de una *lista de conjunto numerada*, es el máximo de los valores asociados a los nombres de elemento de conjunto más 1; en cualquier otro caso, el número es igual al número de ocurrencias del *elemento de conjunto* en la *lista de conjunto no numerada*.

- Un modo de conjunto es un modo de conjunto con huecos única y exclusivamente si, el número de ocurrencias de *nombre* en la *lista de conjunto* es inferior al número de valores del modo conjunto.

condiciones estáticas: Cada *expresión literal entera* de la *lista de conjunto* debe entregar un valor entero no negativo diferente, en el sentido de que para dos expresiones cualesquiera *e1* y *e2*: *NUM (e1)* y *NUM (e2)*, entregan resultados diferentes.

Un modo conjunto define por lo menos un valor nominado.

ejemplos:

```
11.5 . SET (occupied, free)                    (1.1)
 6.4   month                                   (1.2)
```

3.4.6 *MODOS INTERVALO*

sintaxis:

```
<modo intervalo> ::=                                              (1)
     [READ] <nombre de modo discreto> <intervalo literal>         (1.1)
    |[READ]  RANGE (<intervalo literal>)                          (1.2)
    |[READ]  BIN   (<expresión literal entera>)                   (1.3)
    |[READ] <nombre de modo intervalo>                            (1.4)

<intervalo literal> ::=                                           (2)
     <límite inferior>: <límite superior>                         (2.1)

<límite inferior> ::=                                             (3)
     <expresión literal discreta>                                 (3.1)

<límite superior> ::=                                             (4)
     <expresión literal discreta>                                 (4.1)
```

sintaxis derivada: La notación *BIN (n)* se deriva de *INT (0 : 2^n -1)*.
Por ejemplo *BIN (2 + 1)* corresponde a *INT (0 : 7)*.

semántica: Un modo intervalo define el conjunto de valores comprendido entre los límites especificados por el *intervalo literal* (con inclusión de ambos). El intervalo se toma de un modo progenitor específico que determina las operaciones con los valores de intervalo y la ordenación de éstos.

propiedades estáticas: Un modo intervalo, tiene la siguiente propiedad (no hereditaria): posee un modo progenitor único, definido como sigue:

- Si el modo intervalo es de la forma:

  <nombre de modo discreto> (<intervalo literal>) entonces, si el *nombre de modo discreto* no es un modo intervalo, el modo progenitor es el *nombre de modo discreto*; en cualquier otro caso es el modo progenitor del *nombre de modo discreto*.

- Si el modo intervalo es de la forma:

  RANGE (<intervalo literal>) entonces el modo progenitor es el modo raíz de la clase resultante de las clases del *límite superior* y del *límite inferior* del *intervalo literal*.

- Si el modo intervalo es un *nombre de sinmodo* su modo progenitor es el del modo definidor del *nombre de sinmodo*.

- Si el modo intervalo es un *nombre de neomodo*, su modo progenitor es el modo progenitor introducido virtualmente (véase el apartado 3.2.3).

Un modo intervalo tiene las siguientes propiedades hereditarias:

- Un modo intervalo tiene un límite inferior y un límite superior que son los literales que designan los valores entregados por *límite superior* y *límite inferior*, respectivamente del *intervalo literal*.

- El número de valores de un modo intervalo, es el valor entregado por *NUM (U) - NUM (L) + 1*, donde U y L designan el límite superior y el límite inferior del modo intervalo, respectivamente.

- Se dice que un modo intervalo es un modo intervalo con huecos, única y exclusivamente si su modo progenitor es un modo conjunto con huecos y en el intervalo especificado en el modo intervalo hay un valor innominado.

condiciones estáticas: Las clases del *límite superior* y *límite inferior*, deben ser compatibles entre sí y con el *nombre de modo discreto*, si está especificado.

El *límite inferior* debe entregar un valor menor o igual que el valor proporcionado por el *límite superior* y ambos valores deben estar en la gama definida por el *nombre de modo discreto*, si se ha especificado.

ejemplos:

9.4	*INT (2:max)*	*(1.1)*
11.11	*line*	*(1.4)*
9.4	*2:max*	*(2.1)*

3.5 MODOS CONJUNTISTAS

sintaxis:

<modo conjuntista> ::=	*(1)*
[READ] POWERSET <modo primitivo>	*(1.1)*
| [READ] <nombre de modo conjuntista>	*(1.2)*
<modo primitivo> ::=	*(2)*
<modo discreto>	*(2.1)*

semántica: Un modo conjuntista define valores que son conjuntos de valores de su modo primitivo. Los modos conjuntistas se extienden sobre todos los subconjuntos del modo primitivo. Con los valores conjuntistas se definen los operadores teóricos de conjuntos usuales (véase el apartado 5.3).

propiedades estáticas: Un modo conjuntista tiene la siguiente propiedad hereditaria:

- Es el único modo primitivo para el cual el modo se designa mediante *modo primitivo*.

ejemplos:

8.4	*POWERSET CHAR*	*(1.1)*
9.4	*POWERSET INT (2: max)*	*(1.1)*
9.6	*number_list*	*(1.2)*

3.6 MODOS DE REFERENCIA

3.6.1 CONSIDERACIONES GENERALES

sintaxis:

<modo de referencia> ::= *(1)*
<modo de referencia ligada> *(1.1)*
| *<modo de referencia libre>* *(1.2)*
| *<modo descriptor>* *(1.3)*

semántica: Un modo de referencia define referencias (direcciones o descripciones) a localizaciones referenciables. Por definición las referencias ligadas corresponden a localizaciones de un modo estático dado; las referencias libres pueden corresponder a localizaciones de cualquier modo estático, los descriptores corresponden a localizaciones de un modo dinámico.

La operación de desreferenciación se define sobre valores de referencia (véanse los apartados 4.2.3, 4.2.4 y 4.2.15) y entregan la localización referenciada.

Dos valores de referencia son iguales única y exclusivamente si se refieren ambos a la misma localización o si no se refieren a localización alguna (por ejemplo son el valor *NULL*).

3.6.2 MODOS DE REFERENCIA LIGADA

sintaxis:

<modo de referencia ligada> ::= *(1)*
[*READ*] *REF <modo referenciado>* *(1.1)*
| [*READ*] *<nombre de modo de referencia ligada>* *(1.2)*

<modo referenciado> ::= *(2)*
<modo> *(2.1)*

semántica: Las referencias ligadas definen valores de referencia a localizaciones del modo referenciado especificado.

propiedades estáticas: Un modo de referencia ligada tiene la siguiente propiedad hereditaria:

- Posee un único modo referenciado que es el modo designado por *modo referenciado*.

ejemplos:

10.38 REF cell *(1.1)*

3.6.3 MODO DE REFERENCIA LIBRE

sintaxis:

<modo de referencia libre> ::= *(1)*
[*READ*] *PTR* *(1.1)*
| [*READ*] *<nombre de modo de referencia libre>* *(1.2)*

semántica: Un modo de referencia libre define valores relativos a localizaciones de cualquier modo estático.

ejemplos:

19.5 PTR *(1.1)*

3.6.4 MODOS DESCRIPTORES

sintaxis:

<modo descriptor> ::= *(1)*
[*READ*] *ROW <modo cadena>* *(1.1)*
| [*READ*] *ROW <modo matriz>* *(1.2)*
| [*READ*] *ROW <nombre de modo estructura variable>* *(1.3)*
| [*READ*] *<nombre de modo descriptor>* *(1.4)*

semántica: Un modo descriptor define valores de referencia relativos a localizaciones de un modo dinámico (que son localizaciones de algún modo parametrizado con parámetros desconocido estáticamente).

Un valor descriptor puede referirse a:

localizaciones cadena con longitud desconocida estáticamente,

localizaciones matriz con un límite superior desconocido estáticamente,

localizaciones estructura parametrizada con parámetros desconocidos estáticamente.

propiedades estáticas: Un modo descriptor tiene la siguiente propiedad hereditaria:

- Tiene un modo referenciado origen, que es el *modo cadena*, el *modo matriz*, o el nombre *de modo estructura variable*, respectivamente.

ejemplos:

8.6 ROW CHAR (max) *(1.1)*

3.7 MODOS PROCEDIMIENTO

sintaxis:

<modo procedimiento> ::= *(1)*
[*READ*] *PROC* ([*<lista de parámetros>*]) [*<especificación de resultado>*]
[*EXCEPTIONS* (*<lista de excepciones>*)] [*RECURSIVE*] *(1.1)*
| [*READ*] *<nombre de modo procedimiento>* *(1.2)*

<lista de parámetros> ::= *(2)*
<especificación de parámetro> {*<especificación de parámetro>*}* *(2.1)*

<especificación de parámetro> ::= *(3)*
<modo> [*<atributo de parámetro>*] [*<nombre de registro>*] *(3.1)*

<atributo de parámetro> ::= *(4)*
IN | *OUT* | *INOUT* | *LOC* *(4.1)*

<especificación de resultado> ::= (5)
[*RETURNS*] (*<modo>* [*LOC*] [*<nombre de registro>*]) (5.1)

<lista de excepciones> ::= (6)
<nombre de excepción>{, *<nombre de excepción>*}* (6.1)

<nombre de excepción> ::= (7)
<nombre> (7.1)

sintaxis derivada: Una *especificación de resultado* sin la palabra clave facultativa *RETURNS* es una sintaxis derivada para la *especificación de resultado* con *RETURNS*.

semántica: Un modo procedimiento define valores (generales) de procedimiento, es decir, el objeto designado por los nombres de procedimiento general, que son nombres definidos en las sentencias de definición de procedimiento o en las sentencias de definición de entrada. Los valores procedimiento indican elementos de código en un contexto dinámico. Los modos procedimiento permiten manipular dinámicamente un procedimiento. Por ejemplo, pasarlo en forma de parámetro a otros procedimientos, enviarlo en forma de valor de mensaje a una memoria intermedia, almacenarlo en una localización, etc.

Los valores procedimiento pueden ser llamados (véase el apartado 6.7).

Dos valores procedimiento son iguales única y exclusivamente si ambos designan el mismo procedimiento en el mismo contexto dinámico o si no designan procedimiento alguno (por ejemplo son el valor *NULL*).

propiedades estáticas: Un modo procedimiento tiene las siguientes propiedades hereditarias:

- Contiene una lista de especificaciones de parámetro cada una de las cuales consta de un modo, posiblemente un atributo de parámetro y/o un nombre de registro. Las especificaciones de parámetro se definen mediante la *lista de parámetros*.
- Tiene una especificación de resultado facultativa que consta de un modo, un atributo *LOC* facultativo y/o nombre de registro. La especificación de resultado, se define por la *especificación de resultado*.
- Tiene un conjunto, posiblemente vacío, de nombres de excepción, que son los mencionados en la *lista de excepciones*.
- Tiene una recursividad que es recursiva si se especifica *RECURSIVE*; en cualquier otro caso, una especificación subsidiaria definida en la realización recursiva o no recursiva.

condiciones estáticas: Todos los nombres mencionados en la *lista de excepciones*, deben ser distintos.

Sólo si se especifica *LOC* en la *especificación de parámetro*, o en la *especificación de resultado*, el *modo* en ella puede tener la propiedad de sincronización.

3.8 MODO EJEMPLO

sintaxis:

```
<modo ejemplo> ::=                                          (1)
      [READ] INSTANCE                                       (1.1)
    | [READ] <nombre de modo ejemplo>                       (1.2)
```

semántica: Un modo ejemplo, define valores que identifican unívocamente los procesos. La creación de un nuevo proceso (véanse los apartados 5.2.17 y 8.1), proporciona un valor ejemplo único como identificación del proceso creado.

Dos valores ejemplos son iguales, única y exclusivamente si identifican el mismo proceso, o no identifican ningún proceso (es decir, son el valor *NULL*).

ejemplos:

15.29 INSTANCE (1.1)

3.9 MODOS DE SINCRONIZACIÓN

3.9.1 CONSIDERACIONES GENERALES

sintaxis:

```
<modo de sincronización> ::=                                (1)
      <modo suceso>                                         (1.1)
    | <modo tampón>                                         (1.2)
```

semántica: Las localizaciones de modo de sincronización permiten la sincronización y la comunicación entre procesos (véase el capítulo 8). No existe en CHILL ninguna expresión que designe un valor definido por un modo de sincronización. En consecuencia, no hay operaciones definidas sobre los valores.

3.9.2 MODOS SUCESO

sintaxis:

```
<modo suceso> ::=                                           (1)
      [READ] EVENT [(<longitud de suceso>)]                 (1.1)
    | [READ] <nombre de modo suceso>                        (1.2)

<longitud de suceso> ::=                                    (2)
      <expresión literal entera>                            (2.1)
```

semántica: Las localizaciones de modo suceso proporcionan los medios para la sincronización entre procesos. Las operaciones definidas en las localizaciones de modo suceso son la acción continuar, la acción poner en espera y la acción poner en espera y elegir, descritas en los apartados 6.15, 6.16 y 6.17 respectivamente.

propiedades estáticas: Un modo suceso tiene la siguiente propiedad hereditaria:

- Tiene asociada, eventualmente, una longitud de suceso que es el valor proporcionado por *NUM (longitud de suceso)*.

condiciones estáticas: La *longitud de suceso,* debe entregar un valor positivo.

ejemplos:

14.10 EVENT. *(1.1)*

3.9.3 MODOS TAMPÓN

sintaxis:

<modo tampón> ::= *(1)*
[READ] BUFFER [(<longitud de tampón>)] *(1.1)*
<modo elemento tampón>
| [READ] <nombre de modo tampón> *(1.2)*

<longitud de tampón> ::= *(2)*
<expresión literal entera> *(2.1)*

<modo elemento tampón> ::= *(3)*
<modo> *(3.1)*

Nota: La sintaxis indicada anteriormente, es sintácticamente ambigua en relación con la sintaxis de los modos matriz. Se aplica la siguiente interpretación subsidiaria: si la palabra clave *BUFFER,* va inmediatamente seguida de una apertura de paréntesis, se considera que el texto inmediatamente siguiente es el comienzo de la indicación facultativa *longitud de tampón* y no pertenece al *modo elemento tampón.*

semántica: Las localizaciones de modo tampón permiten la sincronización y comunicación entre procesos. Las operaciones definidas en las localizaciones tampón, son la acción enviar, la acción recibir y elegir, y la expresión recibir, descritas en los apartados 6.18, 6.19 y 5.2.18, respectivamente.

propiedades estáticas: Un modo tampón tiene asociadas las siguientes propiedades hereditarias:

- Tiene una longitud de tampón facultativa, que es el valor suministrado por *NUM (longitud de tampón).*
- Tiene un modo elemento tampón, que es el modo designado por *modo elemento tampón.*

condiciones estáticas: La *longitud de tampón,* debe suministrar un valor no negativo.

El *modo elemento tampón* no debe poseer la propiedad de sincronización.

ejemplos:

16.28 BUFFER (1) USER_MESSAGES *(1.1)*
16.32 USER_BUFFERS *(1.2)*

3.10 MODOS COMPUESTOS

3.10.1 CONSIDERACIONES GENERALES

sintaxis:

```
<modo compuesto> ::=                                    (1)
     <modo cadena>                                      (1.1)
    |<modo matriz>                                      (1.2)
    |<modo estructura>                                  (1.3)
```

semántica: Las localizaciones y valores compuestos, tienen sublocalizaciones y subvalores que pueden ser accesibles u obtenibles, respectivamente (véanse los apartados 4.2.5-9, 4.2.13-14 y 5.2.6-12).

3.10.2 MODOS CADENA

sintaxis:

```
<modo cadena> ::=                                       (1)
     [READ] <tipo de cadena> (<longitud de cadena>)     (1.1)
    |<modo cadena parametrizado>                        (1.2)
    |[READ] <nombre de modo cadena>                     (1.3)

<modo cadena parametrizado> ::=                         (2)
     [READ] <nombre de modo cadena origen> (<longitud de
                                            cadena>)    (2.1)
    |[READ] <nombre de modo cadena parametrizado>        (2.2)

<nombre de modo cadena origen> ::=                      (3)
     <nombre de modo cadena>                            (3.1)

<tipo de cadena>                                        (4)
      CHAR                                              (4.1)
    | BIT                                               (4.2)

<longitud de cadena>                                    (5)
     <expresión literal entera>                         (5.1)
```

semántica: Un modo cadena define, valores cadena de bits o de caracteres, cuya longitud está indicada o implicada en el modo cadena.

Los valores cadena de un modo cadena, están bien ordenados. La ordenación de los valores cadena de caracteres responden al orden lexicográfico definido por el alfabeto N.º 5 del CCITT. Para los valores cadena de bits, la ordenación lexicográfica es tal que el bit 1 es mayor que el bit 0.

En los valores cadena se define el operador concatenación. En los valores cadena de bits, se definen los operadores lógicos usuales (véase el apartado 5.3).

propiedades estáticas: Un modo cadena tiene las siguientes propiedades hereditarias:

- Es un modo cadena de bits o un modo cadena de caracteres, según que el *tipo de cadena* especifique *BIT* o *CHAR* o según que el *nombre de modo cadena origen* sea un modo cadena de bits o de caracteres.

- Tiene una *longitud de cadena*, que es el valor suministrado por *NUM (longitud de cadena)*.

condiciones estáticas: La *longitud de cadena*, debe proporcionar un valor no negativo.

El valor proporcionado por la *longitud de cadena*, contenido directamente en un *modo cadena parametrizado* debe ser menor o igual que la longitud de cadena del *nombre de modo cadena origen*.

ejemplos:

7.45 CHAR (20) *(1.1)*

3.10.3 MODOS MATRIZ

sintaxis:

<modo matriz> ::= *(1)*
[READ] [ARRAY] (<modo índice> {,<modo índice>})* *(1.1)*
*<modo elemento> { <organización de elementos>}**
|<modo matriz parametrizado> *(1.2)*
|[READ] <nombre de modo matriz> *(1.3)*

<modo matriz parametrizado> ::= *(2)*
[READ] <nombre de modo matriz origen> (<índice superior>) *(2.1)*
|[READ] <nombre de modo matriz parametrizado> *(2.2)*

<nombre de modo matriz origen> ::= *(3)*
<nombre de modo matriz> *(3.1)*

<modo índice> ::= *(4)*
<modo discreto> *(4.1)*
|<intervalo literal> *(4.2)*

<índice superior> ::= *(5)*
<expresión literal> *(5.1)*

<modo elemento> ::= *(6)*
<modo> *(6.1)*

sintaxis derivada: La palabra clave *ARRAY*, es facultativa. Un *modo matriz* (que no sea un *nombre de modo matriz*, ni un *modo matriz parametrizado*) sin la palabra clave *ARRAY*, se deriva del *modo matriz* que contenga la palabra clave *ARRAY*.

La notación de modo índice *<intervalo literal>*, se deriva del modo discreto *RANGE (<intervalo literal>)*. Un *modo matriz* que contenga más de un modo índice (lo que designa una matriz "multidimensional"), es la sintaxis derivada de un *modo matriz* que tiene un *modo elemento*, que a su vez es un *modo matriz*. Por ejemplo:

ARRAY (1:20, 1:10) INT

se deriva de

ARRAY (RANGE (1:20)) ARRAY (RANGE (1:10)) INT

La ocurrencia de una *organización de elementos* está permitida sólo si se utiliza esta sintaxis derivada. El número de ocurrencias de *organización de elementos* debe ser menor o igual que el número de ocurrencias del *modo índice*. En este caso la *organización de elementos* situada más a la izquierda, se asocia con el *modo elemento* más interno, etc.

semántica: Un modo matriz define valores compuestos que son listas de valores definidos por su modo elemento. Mediante la especificación de la *organización de elementos* puede controlarse la organización física de una localización o un valor matricial (véase el apartado 3.10.6). Dos valores matriciales son iguales, única y exclusivamente si todos los valores de los elementos correspondientes son iguales.

propiedades estáticas: Un modo matriz tiene las siguientes propiedades hereditarias:

- Tiene un modo índice que es el modo discreto designado por *modo índice* si no es un *modo matriz parametrizado*, en cualquier otro caso, el modo índice es el modo intervalo construido como sigue:

  &nombre (límite inferior: límite superior)

  donde *&nombre*, es un nombre virtual de sínmodo, sinónimo del modo índice del *nombre de modo matriz origen*, *límite inferior* es el límite inferior del modo índice del *nombre de modo matriz origen* y *límite superior* es el *índice superior*.

- Tiene un límite superior y un límite inferior, que son los límites superior e inferior de su modo índice, respectivamente.

- Tiene un modo elemento que es o *M* o *READ M*, donde *M* es el *modo elemento* o el modo elemento del *nombre de modo matriz origen*, respectivamente. El modo elemento, será *READ M*, única y exclusivamente si, *M* no es un modo de sólo lectura y el *modo matriz* es un modo de sólo lectura.

- Tiene una organización de elementos, la cual, si se trata de un *modo matriz parametrizado*, es la organización de elementos de su *nombre de modo matriz origen* en cualquier otro caso es la *organización de elemento* especificada, o la realización subsidiaria que a su vez es o *PACK* o *NOPACK*.

- Es un modo de correspondencia, única y exclusivamente si se especifica la *organización de los elementos* y es un *paso*.

- Tiene un número de elementos, que es el valor proporcionado por:

  NUM (límite superior) - NUM (límite inferior) + 1.

condiciones estáticas: La clase de *índice superior* debe ser compatible con el modo índice del *nombre de modo matriz origen* y el valor entregado por éste debe estar en el intervalo definido por dicho modo índice.

El *modo índice*, no debe ser un modo conjunto con huecos ni un modo intervalo con huecos.

ejemplos:

```
5.30   ARRAY (1:16) STRUCT (c4, c2, c1 BOOL)                    (1)
11.10  ARRAY (line) ARRAY (column) square                       (1.1)
11.15  board                                                    (1.3)
```

3.10.4 MODOS ESTRUCTURA

sintaxis:

```
<modo estructura> ::=                                           (1)
     <modo estructura fascicular>                               (1.1)
    |<modo estructura escalonada>                               (1.2)
    |<modo estructura parametrizada>                            (1.3)
    |[READ] <nombre de modo estructura>                         (1.4)

<modo estructura fascicular> ::=                                (2)
     [READ] STRUCT (<campos>{,<campos>}*)                       (2.1)

<campos> ::=                                                    (3)
     <campos fijos>                                             (3.1)
    |<campos alternativos>                                      (3.2)

<campos fijos> ::=                                              (4)
     <lista de nombres> <modo> [<organización de campo>]        (4.1)

<campos alternativos> ::=                                       (5)
      CASE [<marcadores>] OF
     <alternativa variable>{,<alternativa variable>}*
     [ELSE [<campos variables>{,<campos variables>}*]] ESAC     (5.1)

<alternativa variable> ::=                                      (6)
    [<especificación de etiqueta de caso>]
   :[<campos variables>{,<campos variables}*]                   (6.1)

<marcadores> ::=                                                (7)
     <nombre de campo marcador> {,<nombre de campo marcador>}*  (7.1)

<campos variables> ::=                                          (8)
     <lista de nombres> <modo> [<organización de campo>]        (8.1)

<modo estructura parametrizada>                                 (9)
     [READ] <nombre de modo estructura variable origen>
    (<lista de expresiones literales>)                          (9.1)
    |[READ] <nombre de modo estructura parametrizada>           (9.2)

<nombre de modo estructura variable origen>::=                  (10)
     <nombre de modo estructura variable>                       (10.1)

<lista de expresiones literales> ::=                            (11)
     <expresión literal>{,<expresión literal>}*                 (11.1)
```

sintaxis derivada: La sintaxis derivada de un *modo estructura fascicular*, es el *modo estructura escalonada*. Esto se explica en el apartado 3.10.5.

La ocurrencia de *campos fijos* o *campos variables*, en los que la *lista de nombres*, contenga más de un *nombre*, es la sintaxis derivada para

varias ocurrencias de *campos fijos* o *campos variables* con un *nombre* respectivamente, cada uno con el *modo* especificado y una *organización de campo* facultativa. En el caso de *organización de campo*, esta *organización* no será *pos*. Por ejemplo:

STRUCT (I, J BOOL PACK)

se deriva de:

STRUCT (I BOOL PACK, J BOOL PACK).

semántica: Los modos estructura definen valores compuestos consistentes en una lista de valores, seleccionables por un nombre de componente. Cada valor se define mediante un modo asociado al nombre del componente. Los valores estructura pueden residir en localizaciones estructura (compuesta), en las que el nombre del componente sirve de acceso a la sublocalización. Los componentes de un valor o localización estructura se denominan campos y sus nombres, nombres de campo.

Hay estructuras fijas, estructuras variables y estructuras parametrizadas.

Las estructuras fijas constan solamente de campos fijos, es decir campos que están siempre presentes y que son accesibles sin ninguna comprobación dinámica.

Las estructuras variables tienen campos variables, es decir campos que no están siempre presentes. Para las estructuras variables marcadas, la presencia de estos campos solamente es conocida en el momento de la ejecución por medio de uno o varios valores de ciertos campos fijos asociados denominados campos marcadores. Las estructuras variables sin marcadores, no tienen tales campos marcadores. Puesto que la composición de una estructura variable puede cambiar durante la ejecución, la dimensión de la localización correspondiente a una estructura variable se basa la mayor parte del conjunto (caso más desfavorable) de alternativas variables.

Una estructura parametrizada se determina a partir de un modo estructura variable para el cual se ha especificado estáticamente la elección de alternativas variables, mediante expresiones literales. La composición es fija desde el punto de creación de la estructura parametrizada y no puede cambiar durante la fase de ejecución. Los campos marcadores, si existen, son de sólo lectura y se inicializan automáticamente con los valores especificados. Para una localización estructura parametrizada, puede asignarse una cuantía precisa de almacenamiento en el punto de declaración o generación. Obsérvese que también existen modos estructura parametrizada dinámica (virtual). En el apartado 3.11.4 se define su semántica.

Puede controlarse la organización de un valor o una localización estructura, mediante una especificación de organización de campos (véase el apartado 3.10.6).

Dos valores estructura son iguales, única y exclusivamente si lo son sus valores componentes correspondientes. Sin embargo, si uno o ambos valores estructura son valores estructura variables sin marcador, el resultado de la comparación se define en la realización.

propiedades estáticas:

Consideraciones generales:

Un modo estructura tiene las siguientes propiedades hereditarias:

- Un modo estructura es un modo estructura fija única y exclusivamente si se lo designa mediante un *modo estructura fascicular* (o *escalonada*) que no contiene una situación *campos alternativos*.

- Un modo estructura es un modo estructura variable única y exclusivamente si se lo designa mediante un *modo estructura fascicular* (o *escalonada*) y contiene al menos una situación *campos alternativos*.

- Un modo estructura es un modo estructura parametrizada única y exclusivamente si se le designa mediante un *modo estructura parametrizada*.

- Un modo estructura tiene un conjunto de nombres de campo. Más abajo se determina este conjunto para los distintos casos. Se dice que un nombre es un nombre de campo única y exclusivamente si está definido en una *lista de nombres* en *campos fijos* o en *campos variables* en un *modo estructura*. Cada nombre de campo de un modo estructura dado tiene un único modo campo a él asociado, que es *M* o *READ M*, donde *M* es el *modo* asociado al nombre de campo. El modo de campo será *READ M* si el *modo* asociado no es un modo sólo lectura y es, o bien el nombre de campo marcador de un modo estructura parametrizada (véase más abajo), o bien el *modo estructura* es un modo sólo lectura. Un nombre de campo de un modo estructura dado tiene una organización de campo única a él asociada que es la *organización de campo* que sigue al nombre de campo, si existe, de lo contrario la organización de campo subsidiaria, que es o bien *PACK* o bien *NOPACK*. Un nombre de campo es referenciable (por lenguaje) única y exclusivamente si su organización de campo es *NOPACK*.

- Un *modo estructura* designa un modo correspondencia única y exclusivamente si sus nombres de campo tienen una organización de campo que es *pos*.

Estructuras fijas:

Un modo estructura fija tiene las siguientes propiedades hereditarias:

- Tiene un conjunto de nombres de campo que es el conjunto de nombres definidos por cualquier *lista de nombres* en *campos fijos*. Esos nombres de campo son nombres de campo fijo.

Estructuras variables:

Un modo estructura variable tiene las siguientes propiedades hereditarias:

- Tiene un conjunto de nombres de campo, que es la unión del conjunto de nombres definido por cualquier *lista de nombres* en *campos fijos* con el conjunto de nombres definido por cualquier *lista de nombres* en *campos alternativos*. Los nombres de campo, definidos por una

lista de nombres en *campos fijos,* son los nombres de campo fijo del modo estructura variable y los restantes nombres de campo, son los nombres de campo variable.

Un nombre de campo de un modo estructura variable, es un nombre de campo marcador, única y exclusivamente si aparece en cualquier *marcador* de los *campos alternativos.* Los *campos alternativos* que no tengan especificados *marcadores,* son *campos alternativos* sin marcadores. Los nombres de campo variable, definidos por cualquier *lista de nombres* en *campos variables* de *campos alternativos* sin marcadores, son nombres de campo variable sin marcador. Los restantes nombres de campo variable, son nombres de campo variable con marcador.

- Un modo estructura variable, es un modo estructura variable sin marcadores, única y exclusivamente si todas las ocurrencias de sus *campos alternativos* son sin marcador. En cualquier otro caso, es un modo estructura variable con marcadores.

- Un modo estructura variable, es un modo estructura variable parametrizado, única y exclusivamente si es un modo estructura variable con marcadores o un modo estructura variable sin marcadores, en el cual, para cada una de las ocurrencias de *campos alternativos,* se facilita una *especificación de la etiqueta de caso,* para todas las ocurrencias de *alternativa variable* de la misma.

- Un modo estructura variable parametrizable, tiene asociada una lista de clases determinada como sigue:

  - si se trata de un modo estructura variable con marcadores, la lista de clases M_i-valuada, donde M_i son los modos de los nombres de campo marcador en el orden en que se definen en *campos fijos;*

  - si se trata de un modo estructura variable sin marcadores, la lista está formada por las listas individuales resultantes de las clases de cada uno de los *campos alternativos,* concatenados en el orden en que ocurren estos *campos alternativos.* La lista resultante de las clases de ocurrencia de *campos alternativos,* es la lista de clases resultante de la lista de ocurrencias de la *especificación de la etiqueta de caso* de la misma (véase el apartado 9.1.13).

estructuras parametrizadas: Un modo estructura parametrizada, tiene las siguientes propiedades hereditarias:

- Tiene un modo estructura variable origen, que es el modo designado mediante el *nombre de modo estructura variable origen.*

- Es un modo estructura parametrizada con marcadores única y exclusivamente si su modo estructura variable origen es un modo estructura variable con marcadores, de lo contrario el modo estructura parametrizada es sin marcadores.

- Tiene un conjunto de nombres de campo, que es la unión del conjunto de nombres de campo fijo del nombre de modo de estructura variable origen con el conjunto de aquellos nombres de campo variable del nombre de modo estructura variable origen definidos en las

ocurrencias *alternativa variable* seleccionadas por la lista de valores definidos en *lista de expresiones literales*.

El conjunto de nombres de campo marcador de un *modo estructura parametrizada*, es el conjunto de nombres de campo marcador de su modo estructura variable origen.

- Un modo estructura parametrizada, tiene una lista de valores asociada definida en *lista de expresiones literales*.

condiciones estáticas:

Consideraciones generales:

Todos los nombres de campo de un modo estructura deben ser distintos.

Si un campo cualquiera tiene una organización que es *pos*, todos los campos deben tener una organización de campo que sea *pos*.

Estructuras variables:

Un nombre de campo marcador, debe ser un nombre de campo fijo que debe estar definido textualmente con antelación a todas las ocurrencias de *campos alternativos* en cuyos *marcadores* se mencione. (En consecuencia un campo marcador precede todos los campos variables que dependen del mismo). El modo de un nombre de campo marcador, debe ser un modo discreto.

En un modo estructura variable, las ocurrencias de *campos alternativos*, deben ser o todas con marcadores o todas sin marcadores. En el caso de *campos alternativos* sin marcadores debe omitirse la *especificación de etiqueta de caso* en todas las ocurrencias de *alternativas variables* conjuntamente o debe especificarse para todas las ocurrencias de *alternativas variables*.

Si se da la *especificación de etiqueta de caso*, para uno cualquiera de los *campos alternativos* correspondientes a un modo estructura variable sin marcadores, todos sus *campos alternativos*, deben tener la *especificación de etiqueta de caso*.

Para los *campos alternativos*, deben cumplirse las condiciones de selección de caso (véase el apartado 9.1.3) y deben asegurarse los mismos requisitos de completud, coherencia y compatibilidad que para la acción de caso (véase el apartado 6.4). Cada uno de los nombres de campo marcador de *marcadores* (si existen) sirve como selector de caso con las clases M-valuadas, donde M es el modo del nombre del campo marcador. En el caso de campos alternativos sin marcadores se ignoran las comprobaciones relativas al selector de caso.

Para un modo estructura variable parametrizada, ninguna de las clases de su lista de clases asociada puede ser la clase general. (Esta condición se cumple automáticamente con el modo estructura variable con marcadores.)

Estructuras parametrizadas:

El *nombre de modo estructura variable origen*, debe ser parametrizable.

Debe haber tantas expresiones literales en la *lista de expresiones literales*, como clases en la lista de clases del *nombre de modo estructura*

variable origen. La clase de cada expresión literal, debe ser compatible con la clase correspondiente (en posición) de la lista de clases. Si la última clase es una clase M-valuada, el valor entregado por la expresión literal, debe ser uno de los valores definidos por M.

ejemplos:

```
3.3   STRUCT (re, im INT)                                        (2.1)
11.5  STRUCT (status SET (occupied, free),
             CASE status OF
              (occupied): p piece,
              (free):
              ESAC)                                              (2.1)
2.5   fraction                                                   (1.4)
11.5  status SET (occupied, free)                                (4.1)
11.6  status                                                     (7.1)
11.7  p piece                                                    (8.1)
```

3.10.5 NOTACIÓN DE LA ESTRUCTURA ESCALONADA

sintaxis derivada:

```
<modo estructura escalonada> ::=                                 (1)
     1 [<especificación de la matriz>]
     [READ]{,<campos de escalón (2)>}+                           (1.1)

<campos de escalón (n)> ::=                                      (2)
     <campos fijos de escalón (n)>
     <campos alternativos de escalón (n)>                        (2.1)

<campos fijos de escalón (n)> ::=                                (3)
   n <lista de nombres> <modo> [<organización de campo>]         (3.1)
  |n <lista de nombres> [<especificación de la matriz>]
     [READ] [<organización de campo>]{,campos de escalón (n+1)>}+ (3.2)

<campos alternativos de escalón (n)> ::=                         (4)
      CASE [<marcadores>] OF
     <alternativa de escalón (n)>{,<alternativa de escalón (n)>}+
     [ELSE[<campos variables de escalón (n)>
     {,<campos variables de escalón (n)>}*]]
      ESAC                                                       (4.1)

<alternativa de escalón (n)> ::=                                 (5)
    [<especificación de etiqueta de caso>
     {,<especificación de etiqueta de caso>}*]
     :[<campos variables de escalón (n)>
    {,<campos variables de escalón (n)>}*]                       (5.1)

<campos variables de escalón (n)> ::=                           (6)
   n <lista de nombres> <modo> [<organización de campo>          (6.1)
  |n <lista de nombres> [<especificación de la matriz>]
     [READ][<organización de campo>]{,<campos de escalón (n+1)>}+ (6.2)

<especificación de la matriz> ::=                                (7)
     [READ][ARRAY] (<modo índice>{,<modo índice>}*)
     {<organización de elemento>}*                               (7.1)
```

Nota. La descripción anterior de una notación de números de escalón para las estructuras implica una extensión del método de descripción sintáctica explicado en el capítulo 2. La sintaxis se define recursivamente utilizando como parámetro el número de escalón *(n)*.

semántica: La sintaxis derivada de un *modo estructura fascicular* único, es el *modo estructura escalonada*.

Se considera que la notación fascicular es la sintaxis estricta, dándose en función de ésta todas las características semánticas, propiedades y condiciones (véase el apartado 3.10.4).

Si una estructura contiene campos que por su parte, son estructuras o matrices de estructuras, se forma una jerarquía de estructuras, pudiéndose asociar con cada campo un número de nivel.

ejemplo:

```
SYNMODE M = STRUCT (B BOOL,
                    S ARRAY (1:10) STRUCT (T INT, U BOOL));
```

La estructura en conjunto tiene nivel 1, B y S tienen nivel 2, T y U tienen nivel 3. En vez de escribir modos estructura jerarquizados, es admisible en el *modo estructura escalonada*, escribir el número de nivel delante del nombre.

ejemplos:

```
SYNMODE M = 1, 2 B BOOL,
               2 S ARRAY (1:10),
               3 T INT,
               3 U BOOL.
```

En las definiciones de modo y definiciones de sinónimo con un modo, no existe ningún nombre asociado al primer nivel. La asociación se produce en la declaración o en el punto de especificación de los parámetros formales. En estos lugares deberá colocarse el nombre del primer nivel a continuación de la posición de nivel 1.

ejemplos:

```
DCL 1 A,
    2 B BOOL,
    2 S ARRAY (1:10),
    3 T INT,
    3 U BOOL.
```

Cuando existan declaraciones y especificaciones de parámetros formales, atributos e inicializaciones, deben especificarse al final de la posición del nivel-1.

ejemplo:

```
P: PROC (1 X INOUT,
         2 B BOOL,
         2 C INT).
```

Si dentro de un *modo estructura escalonada*, se especifica una matriz de estructuras, se proporciona la especificación de la matriz inmediatamente después del indicador del nivel.

condiciones estáticas: No deben mezclarse las notaciones fasciculares y escalonadas.

ejemplos:

```
19.9  DCL 1   BASED (P),
              2 I INFO POS (0,8:31),
              2 PREV PTR POS (1,0 :15),
              2 NEXT PTR POS (1,16:31)                    (1.1)
```

3.10.6 *DESCRIPCIÓN DE LA CONFIGURACIÓN DE LOS MODOS MATRIZ Y LOS MODOS ESTRUCTURA*

sintaxis:

```
<organización de elemento> ::=                            (1)
      PACK | NOPACK | <paso>                              (1.1)

<organización de campo> ::=                               (2)
      PACK | NOPACK | <pos>                               (2.1)

<paso>                                                    (3)
      STEP (<pos> [,<longitud del paso> [,<tamaño de la
                                          configuración>]] )  (3.1)

<pos>                                                     (4)
      POS (<palabra>, <bit inicial>, <longitud>)          (4.1)
     |POS (<palabra> [,<bit inicial> [:<bit final>]] )    (4.2)

<tamaño de la configuración> ::=                          (5)
     <expresión literal entera>                           (5.1)

<palabra> ::=                                             (6)
     <expresión literal entera>                           (6.1)

<longitud del paso> ::=                                   (7)
     <expresión literal entera>                           (7.1)

<bit inicial> ::=                                         (8)
     <expresión literal entera>                           (8.1)

<bit final> ::=                                           (9)
     <expresión literal entera>                           (9.1)

<longitud> ::=                                            (10)
     <expresión literal entera>                           (10.1)
```

semántica: Es posible controlar la organización de una matriz o de una estructura proporcionando información de empaquetamiento o de correspondencia en su modo. La información de empaquetamiento, puede ser *PACK* o *NOPACK*. La información de correspondencia es, o un ***paso*** en caso de modos matriz, o ***pos*** en caso de campos de modos estructura. La ausencia de ***organización de campo*** u ***organización de matriz,*** en un modo matriz o estructura se interpretará siempre como información de empaquetamiento, es decir como *PACK* o *NOPACK*.

Si se especifica *PACK* para elementos de una matriz o un campo de una estructura, esto implica la utilización óptima de espacio de memoria para los elementos de la matriz o los campos de la estructura, mientras que *NOPACK* implica la optimización del tiempo de acceso a los elementos de la matriz o a los campos de la estructura. *NOPACK* implica también la referenciabilidad (por lenguaje).

La información *PACK, NOPACK,* se aplica sólo para un nivel, es decir se aplica a los elementos de la matriz o a los campos de la estructura, no a los posibles componentes del elemento de la matriz o del campo de la estructura. La información de organización, va siempre asociada al modo más próximo al cual puede aplicarse y que no tenga ya una organización establecida. Por ejemplo si el empaquetamiento subsidiario es *NOPACK:*

STRUCT (F ARRAY (0:1) M PACK)

es equivalente a:

STRUCT (F ARRAY (0:1) M PACK NOPACK)

Es también posible controlar la organización precisa de un objeto compuesto, especificando la información de posición para sus componentes en el modo. Esta información de posición se proporciona de las siguientes formas:

- Para los modos matriz, la información de posición se facilita conjuntamente para todos los elementos, en forma de un ***paso*** siguiendo al modo matriz.

- Para los modos estructura, la información de posición se proporciona individualmente para cada campo en forma de una ***pos*** siguiendo al modo del campo.

La posición precisa de un componente C, por ejemplo un elemento o el campo de un objeto se proporciona mediante las tres constantes siguientes: W_c, B_c y L_c donde:

W_c es la distancia, en palabras, entre la primera palabra ocupada (quizá parcialmente) por C, con relación a la primera palabra ocupada (quizá parcialmente) por el objeto del que C es una componente.

B_c es la distancia, en bits, entre el primer bit ocupado por C con relación al bit situado más a la izquierda de la primera palabra ocupada (quizá parcialmente) por C.

L_c es el número de bits ocupados por C.

La información de posición proporcionada para los componentes del objeto, determina el posicionado preciso de estos componentes, si el objeto es entero (es decir, no es una componente de otro objeto). Sin embargo, si el objeto no es entero, el posicionado preciso de la componente, depende del posicionado preciso del objeto mismo.

Un *paso* especificado para los elementos de una matriz es una notación abreviada para la enumeración explícita de la *pos* de cada elemento individual. Desde un punto de vista informal, *pos* y *longitud de paso,* especifican una "configuración de posición" para los elementos que encajan completamente en las primeras palabras *de la configuración tamaño,* suponiendo que la matriz es entera. La posición del primer elemento se determina por *pos,* la posición de los elementos subsiguientes, que encajan completamente en las primeras palabras *de la configuración tamaño,* es tal que la distancia en bits entre los primeros bits ocupados de los elementos sucesivos es igual a la *longitud del paso.* La configuración de posición especificada de esta forma, se repite con la frecuencia necesaria para las unidades subsiguientes de palabras *de la configuración tamaño.*

Pos:

Dado un objeto O de un modo correspondencia, en el que se especifica, para una componente C de dicho objeto, una *pos* de la forma:

POS (<número de palabra> <bit de arranque> <longitud>)

La posición precisa de cada componente C, se determina como sigue:

- Si el objeto O (del que C es una componente) es entero, entonces:

  W_C es *NUM (número de palabra),*

  B_C es *NUM (bit de arranque)* y

  L_C es *NUM (longitud).*

- Si el objeto O (del cual C es una componente) no es entero, entonces:

  W_C es *NUM (número de palabra) + (NUM) (bit de arranque) + B_O)/WIDTH,*

  B_C se designa por: *(NUM (bit de arranque) + B_O) MOD WIDTH,* y

  L_C se designa por: *NUM (longitud)* donde *WIDTH* es el número de bits en una palabra.

Paso:

Sea el elemento i:

- el de menor índice, si i = 0,

- el de índice sucesor del índice del elemento n donde n = i-1, en cualquier otro caso.

Sea i_O el número de elementos que preceden al elemento O en su esquema de posicionamiento. Dado un atributo de paso de la forma

STEP (<pos>, <longitud del paso>, <tamaño de la configuración>),

la *pos* de un elemento individual, con relación al comienzo del esquema de posicionamiento del elemento 0, se determina como sigue:

la *pos* del elemento i es:

*POS (NUM (tamaño de la configuración) * ((i + i_0) / DENS) + $RBPOS_i$/WIDTH, $RBPOS_i$ MOD WIDTH, <longitud>)*

para 1 $\leq$ i $\leq$ "número de elementos"

donde $RBPOS_i$ es

*NUM (número de la palabra) * WIDTH + NUM (bit de arranque) + NUM (longitud del paso) * ((i + i_0) MOD DENS),*

y *DENS* es

(NUM *(tamaño de la configuración) * WIDTH)/NUM (longitud del paso),*

y *WIDTH* es el número de bits en una palabra.

Sentencias subsidiarias

La notación:

POS (<número de palabra>,<bit de arranque>:<bit final>)

es semánticamente equivalente a :

POS (<número de palabra>,<bit de arranque>, NUM (bit final) - NUM (bit de arranque) + 1)

La notación:

POS (<número de palabra>,<bit de arranque>)

es semánticamente equivalente a:

POS (<número de palabra>,<bit de arranque>, BSIZE)

donde *BSIZE* es el número mínimo de bits que debe ocupar la componente para la cual se ha especificado *pos*.

La notación:

POS (<número de palabra>)

es semánticamente equivalente a:

*POS (<número de palabra>, 0, WSIZE * WIDTH)*

donde *WSIZE* es el tamaño del modo de la componente para la cual se ha especificado *pos*.

La notación:

STEP (<pos>, <longitud del paso>)

es semánticamente equivalente a:

STEP (<pos>, <longitud del paso>, PSIZE)

donde *PSIZE* es el mínimo entero tal que,

*PSIZE * WIDTH ≥ NUM (longitud del paso).*

La notación:

STEP (<pos>)

es semánticamente equivalente a:

STEP (<pos>), SSIZE),

donde *SSIZE* es la *<longitud>* especificada en *pos* o deducible de *pos* mediante las reglas anteriores.

propiedades estáticas: La organización de elementos del modo determina, para cualquier localización de un modo matriz de correspondencia, la referenciabilidad (por lenguaje) de sus sublocalizaciones (con inclusión de submatrices, segmentos de matrices) como sigue:

- o bien todas sublocalizaciones son referenciables (por lenguaje), o ninguna de ellas lo es;
- si la organización de elementos es *NOPACK*, todas las sublocalizaciones son referenciables (por lenguaje).

Para una localización de un modo estructura de correspondencia, la referenciabilidad del campo de la estructura seleccionado por un nombre de campo, se determina mediante la organización de campo del nombre de campo como sigue:

- el nombre de campo es referenciable (por lenguaje) si la organización de campo es *NOPACK*.

condiciones estáticas: Si el modo elemento de un modo matriz dado, o el modo de campo de un nombre de campo de un modo estructura dado, es él mismo un modo matriz o estructura, debe ser un modo de correspondencia si el modo matriz o estructura dado es de correspondencia y no será un modo de correspondencia en cualquier otro caso.

Cada una de las ocurrencias de *expresión literal entera*, debe suministrar un valor no negativo. Además, *longitud*, *longitud del paso* y *tamaño de la configuración*, deben entregar un valor no nulo y el *bit de arranque* y el *bit final*, deben proporcionar un valor inferior a *WIDTH*, donde *WIDTH* es el número de bits de una palabra (definido en la realización). Además, el *bit de arranque* debe proporcionar un valor no superior al suministrado por el *bit final*.

Para cada nombre de campo de un modo estructura de correspondencia la *longitud* de su *organización de campo*, no debe ser inferior al número mínimo de bits que ocupa el campo.

Para cada modo matriz de correspondencia, la *longitud* en la *pos* de su *organización de elementos,* no será inferior al número mínimo necesario para la ocupación por los elementos. Además, para cualquier organización de elementos, deben cumplirse las siguientes condiciones:

- *NUM (longitud del paso)* $\geq$ *NUM (longitud)*

- *(NUM (tamaño configuración) * WIDTH) MOD NUM (longitud del paso)* $\geq$ *NUM (número de palabra) * WIDTH + NUM (bit de arranque)*

Consistencia y viabilidad

Consistencia:

No podrá especificarse ninguna componente de una matriz u objeto de estructura de modo que ocupe bits ocupados por otro componente del mismo objeto, salvo en el caso de dos nombres de campo variable definidos en la misma ocurrencia de ***campos alternativos***. Sin embargo, en este último caso, los nombres de campos variables, pueden no estar definidos en la misma ***alternativa variable*** ni a continuación de *ELSE*.

Viabilidad:

En el lenguaje no se definen requisitos de viabilidad salvo el que se deduce de la regla que expresa que la referenciabilidad de una sublocalización de cualquier localización (referenciable o no), se determina mediante la organización (de elemento o de campo) solamente, lo cual es una propiedad del modo de la localización. Esto impone algunas restricciones en la representación de componentes que, a su vez, tienen componentes referenciables.

ejemplos:

17.5 PACK *(1.1)*
19.11 POS (1,0:15) *(4.2)*

3.11 MODOS DINÁMICOS

3.11.1 CONSIDERACIONES GENERALES

Un modo dinámico, es un modo tal que algunas de sus propiedades se conocen solamente en el momento de la ejecución. Los modos dinámicos son siempre modos parametrizados con uno o más parámetros de ejecución. Los modos dinámicos no tienen designación en CHILL. Sin embargo, en este documento, se introducen designaciones virtuales, a fines de descripción. Estas designaciones virtuales van precedidas del símbolo (&), para distinguirlas de las notaciones reales que pueden aparecer en el texto de un programa CHILL.

3.11.2 MODOS CADENA DINÁMICOS

designación virtual: *&<nombre de modo cadena origen> (<expresión entera>)*

semántica: Un modo cadena dinámico es un modo cadena parametrizado con una longitud estáticamente desconocida. La longitud de la cadena dinámica es el valor suministrado por la *expresión entera*.

propiedades estáticas:

- El modo cadena dinámico es un modo cadena de bits (de caracteres). única y exclusivamente si el *nombre de modo cadena origen* es un modo cadena de bits (de caracteres).

propiedades dinámicas:

- Un modo cadena dinámico, tiene una longitud dinámica, que es el valor suministrado por *NUM (expresión entera)*.

3.11.3 MODOS MATRIZ DINÁMICOS

designación virtual: *&<modo matriz dinámico> (<expresión discreta>)*

semántica: Un modo matriz dinámico, es un modo matriz parametrizado con un límite superior desconocido estáticamente. Se conocen estáticamente el límite inferior, el modo índice y el modo elemento. El límite superior dinámico es el valor suministrado por la *expresión discreta*.

propiedades estáticas:

- Un modo matriz dinámico tiene asociados un modo índice, un modo elemento, una organización de elementos y un límite inferior, que son respectivamente, el modo índice, el modo elemento, la organización de elementos y el límite inferior del *nombre de modo matriz origen*.

propiedades dinámicas:

- Un modo matriz dinámico, tiene un límite superior dinámico, que es el valor suministrado por *expresión discreta*, y un número de elementos dinámicos que es el valor entregado por
  NUM (límite superior) - NUM (límite inferior) + 1.

3.11.4 MODOS ESTRUCTURA PARAMETRIZADA DINÁMICA

designación virtual: *&<nombre de modo estructura variable origen>*
(<lista de expresiones>)

semántica: Un modo estructura parametrizada dinámica, es un modo estructura parametrizada, con parámetros desconocidos estáticamente. La composición del modo estructura sólo puede determinarse dinámicamente mediante la lista de valores proporcionada por *lista de expresiones*.

propiedades estáticas:

- Un modo estructura parametrizada dinámica tiene asociado un único modo estructura variable origen, que es modo designado por el *nombre de modo estructura variable origen*.

- Un modo estructura parametrizada dinámica está con marcadores única y exclusivamente si su modo estructura variable origen es un modo estructura variable con marcadores, de lo contrario es sin marcadores.

- El conjunto de nombres de campo (nombres de campo fijo, nombres de campo marcador, nombres de campo variable) de un modo estructura parametrizada dinámica, es el conjunto de nombres de campo (nombres de campo fijo, nombres de campo marcador, nombres de campo variable) de su modo estructura variable origen.

propiedades dinámicas:

- Un modo estructura parametrizada dinámica, tiene asociada una lista de valores, que es la lista de valores proporcionada por las expresiones de la *lista de expresiones*.

4.0 LOCALIZACIONES Y SUS ACCESOS

4.1 DECLARACIONES

4.1.1. CONSIDERACIONES GENERALES

sintaxis:

```
<sentencia de declaración> ::=                                   (1)
     DCL <declaración> {,<declaración>}*;                        (1.1)

<declaración> ::=                                                (2)
     <declaración de localización>                               (2.1)
   | <declaración de identidad-loc>                              (2.2)
   | <declaración referida a una base>                           (2.3)
```

semántica: Una sentencia de declaración declara que uno o más nombres constituyen un acceso a una localización.

ejemplos:

```
6.9     DCL j INT := julian_day_number,
            d, m, y INT;                                         (1.1)
6.10    d, m y INT                                               (2.1)
11.34   starting_square LOC := b(m.lin_1)(m.col_1)               (2.2)
```

4.1.2 DECLARACIONES DE LOCALIZACIÓN

sintaxis:

```
<declaración de localización> ::=                                (1)
     <lista de nombres> <modo> [STATIC] [<inicialización>]       (1.1)

<inicialización> ::=                                             (2)
     <inicialización ligada al dominio>                          (2.1)
   | <inicialización ligada al tiempo de vida>                   (2.2)

<inicialización ligada al dominio> ::=                           (3)
     <símbolo asignación> <valor> [<ejecutor>]                   (3.1)

<inicialización ligada al tiempo de vida> ::=                    (4)
     INIT <símbolo asignación> <valor constante>                 (4.1)
```

semántica: Una declaración de localización crea tantas localizaciones como nombres especificados en la *lista de nombres*.

Con la *inicialización ligada al dominio*, se evalúa el *valor* cada vez que se produce la entrada del dominio en el que está situada la declaración (véase el apartado 7.2) asignándose a las localizaciones el valor proporcionado. Antes de evaluar el *valor*, la localización contiene un valor indefinido (excepto cuando se haya especificado un modo con la propiedad de marcador parametrizado o la de sincronización; véase más adelante).

Con la *inicialización ligada al tiempo de vida*, se asigna a la, o las, localización(es) el valor proporcionado por *valor constante*, sólo una vez, al comienzo de la duración de la localización (véanse los apartados 7.2 y 7.9).

Especificar que no hay *inicialización*, es semánticamente equivalente a la especificación de una *inicialización ligada al tiempo de vida* con el *valor indefinido* (véase el apartado 5.3). Sin embargo, si el *modo* tiene la propiedad marcador parametrizado, las sublocalizaciones de campo marcador de la localización se inicializan con el valor correspondiente del modo estructura parametrizada asociado.

El significado de *valor indefinido*, como inicialización de una localización de sincronización es que las sublocalizaciones suceso y/o tampón creadas, se inicializan automáticamente en el estado "vacío", es decir, ni se asocian procesos demorados al suceso o al tampón ni hay mensajes en el tampón.

En el apartado 7.9 y en el capítulo 10 se encuentran las semánticas de *STATIC* y *ejecutor*, respectivamente.

propiedades estáticas: Los nombres declarados en una declaración de localización son nombres de localización. El modo asociado a un nombre de localización, es el *modo* especificado en la *declaración de localización*. Un nombre de localización es referenciable (por lenguaje).

condiciones estáticas: La clase del *valor* o del *valor constante*, debe ser compatible con el *modo*.

Si el *modo* tiene la propiedad de sólo lectura, debe especificarse la *inicialización*. Si el *modo* tiene la propiedad de sincronización no debe especificarse la *inicialización ligada al dominio*.

condiciones dinámicas: En caso de *inicialización ligada al dominio*, se aplican las condiciones de asignación del *valor* con respecto al modo (véase el apartado 6.2).

ejemplos:
```
5.8     k2, x, w, t, s, r BOOL                    (1.1)
6.9     := julian_day_number                      (3.1)
8.4     INIT := ['A':'Z']                         (4.1)
```

4.1.3 DECLARACIONES DE IDENTIDAD-LOC

sintaxis:
```
<declaración de identidad-loc> ::=                (1)
        <lista de nombres> <modo> LOC <símbolo de asignación>
        <localización de modo estático> [<ejecutor>]   (1.1)
```

semántica: Una declaración de identidad-loc crea tantos accesos a la localización de modo estático especificada, como nombres hay especificados en la *lista de nombres*.

Si la *localización de modo estático* se evalúa dinámicamente, se efectúa esta evaluación cada vez que se produce la entrada del dominio en el que está situada la declaración de identidad-loc.

En este caso, un nombre declarado designa una localización indefinida antes de la primera evaluación durante el tiempo de vida del acceso indicado por el nombre declarado (véanse los apartados 7.2 y 7.9).

propiedades estáticas: Los nombres declarados en una declaración de identidad-loc, son nombres de identidad-loc. El modo asociado a un nombre de identidad-loc, es el *modo* especificado en la *declaración de identidad-loc*. Un nombre de identidad-loc es referenciable (por lenguaje) única y exclusivamente si lo es la *localización de modo estático* especificada.

condiciones estáticas: El *modo* especificado debe ser de lectura compatible con el modo de la *localización de modo estático*.

ejemplos: *11.34 starting square LOC := b(m.lin_1)(m.col_1)* *(1.1)*

4.1.4 DECLARACIONES REFERIDAS A UNA BASE

sintaxis: *<declaración referida a una base> ::=* *(1)*
<lista de nombres> <modo> BASED
[(<nombre de localización de referencia libre o ligada>)] *(1.1)*

sintaxis derivada: La sintaxis derivada de una sentencia de definición de sínmodo es una *declaración referida a una base* sin un *nombre de localización referencia libre o ligada*. Por ejemplo:
DCL 1 INT BASED;
se deriva de:
SYNMODE I = INT;
Los nombres declarados son nombres de sínmodo, sinónimos del *modo* especificado.

semántica: Una *declaración referida a una base* (con un *nombre de localización de referencia libre o ligada*) especifica tantos accesos como nombres hay en la *lista de nombres*. Los nombres declarados en una declaración referida a una base constituyen una forma alternativa de acceder a una localización, desrreferenciando un valor de referencia. Este valor de referencia está contenido en la localización especificada por el *nombre de localización referencia libre o ligada*. Esta operación de desrreferenciación se efectúa única y exclusivamente cuando se establece un acceso mediante un nombre basado declarado.

propiedades estáticas: Los nombres declarados en una *declaración referida a una base* son nombres basados. El modo asociado a un nombre basado, es el *modo* especificado en la *declaración referida a una base*. Un nombre basado es referenciable (por lenguaje).

condiciones estáticas: Si el modo del *nombre de localización referencia libre o ligada* es un modo referencia ligada, el *modo* especificado debe ser de lectura compatible con el modo referenciado del modo del *nombre de localización de referencia libre o ligada*.

ejemplos: *19.9* *1 X BASED (P),*
2 I INFO POS(0,8:31),
2 PREV PTR POS(1,0:15),
2 NEXT PTR POS(1,16:31) *(1.1)*

4.2 LOCALIZACIONES

4.2.1 CONSIDERACIONES GENERALES

sintaxis:

<localización> ::= *(1)*
<localización de modo estático> *(1.1)*
| *<localización de modo dinámico>* *(1.2)*

<localización de modo estático> ::= *(2)*
<nombre de acceso> *(2.1)*
| *<referencia ligada desreferenciada>* *(2.2)*
| *<referencia libre desreferenciada>* *(2.3)*
| *<elemento cadena>* *(2.4)*
| *<sub cadena>* *(2.5)*
| *<elemento matriz>* *(2.6)*
| *<sub matriz>* *(2.7)*
| *<campo de estructura>* *(2.8)*
| *<llamada procedimiento que produce una localización>* *(2.9)*
| *<llamada a rutina predefinida que produce una localización>* *(2.10)*
| *<conversión de localización>* *(2.11)*

<localización de modo dinámico> ::= *(3)*
<segmento de cadena> *(3.1)*
| *<segmento de matriz>* *(3.2)*
| *<descriptor desreferenciado>* *(3.3)*

semántica: Una localización es un objeto que puede contener valores. Una localización puede designarse ya sea mediante una *localización de modo estático*, es decir, su modo es determinable estáticamente o mediante una *localización de modo dinámico*, cuando parte de la información de modo solamente puede obtenerse dinámicamente. Se ha de acceder a las localizaciones para almacenar u obtener un valor.

propiedades estáticas: Una *localización de modo estático* tiene asociado un modo estático. A cada *localización de modo dinámico* se asociará un modo dinámico virtual, solamente con fines descriptivos (véase el apartado 3.1). En el caso de localizaciones de modo dinámico, las comprobaciones de compatibilidad requeridas sólo pueden efectuarse completamente en la ejecución; el fallo de la verificación de la parte dinámica provocará una excepción *RANGEFAIL* o *TAGFAIL*.

4.2.2 NOMBRES DE ACCESO

sintaxis: *<nombre de acceso> ::=* *(1)*
<nombre de localización> *(1.1)*
| *<nombre de identidad-loc>* *(1.2)*
| *<nombre basado>* *(1.3)*
| *<nombre de numeración de localización>* *(1.4)*
| *<nombre de localización do-with>* *(1.5)*

semántica: Un nombre de acceso es un acceso a una localización.

Un nombre de acceso será uno de los siguientes:

- un nombre de localización, es decir un nombre declarado explícitamente en una *declaración de localización* o declarado implícitamente en un *parámetro formal* sin el atributo *LOC*;
- un nombre de identidad-loc, es decir un nombre declarado explícitamente en una *declaración de identidad-loc* o implícitamente en un *parámetro formal* con el atributo *LOC*;
- un nombre basado, es decir un nombre declarado en una *declaración referida a una base*;
- un nombre de enumeración de localización, es decir un *contador de bucle* en una *enumeración de localización*;
- un nombre de localización do-with, es decir un nombre de campo utilizado como acceso directo en una *acción hacer* con una *componente de simultaneidad*.

Si la localización designada por un *nombre de localización do-with* es un campo variable de una localización estructura variable sin marcadores, la semántica se define en la realización.

propiedades estáticas: El modo asociado a un *nombre de acceso* es el modo de *nombre de localización*, *nombre de identidad-loc*, *nombre basado*, *nombre de enumeración de localización*, *nombre de localización do-with*, respectivamente.

Un *nombre de acceso* es referenciable (por lenguaje) única y exclusivamente si es un *nombre de localización*, un *nombre de identidad-loc referenciable*, un *nombre basado*, un *nombre de enumeración de localización referenciable* o un *nombre de localización do-with referenciable*.

condiciones dinámicas: Un *nombre de identidad-loc* no debe designar una localización indefinida.

Cuando se accede mediante un *nombre basado*, se aplican las mismas condiciones dinámicas que cuando se desrreferencia el *nombre de localización de referencia libre o ligada*, en la *declaración basada* asociada (véanse los apartados 4.2.3 y 4.2.4).

El acceso mediante un *nombre de localización do-with* origina una excepción *TAGFAIL* si la localización designada es el campo variable de:

- una localización de modo estructura variable con marcadores, cuando los valores del campo marcador asociado indican que el campo no existe;
- una localización de modo estructura parametrizada dinámica y cuya lista de valores asociada indica que el campo no existe.

ejemplos:

4.11	*a*	*(1.1)*
11.38	*starting*	*(1.2)*
19.14	*X*	*(1.3)*
15.25	*EACH*	*(1.4)*
5.11	*c1*	*(1.5)*

4.2.3 REFERENCIAS LIGADAS DESREFERENCIADAS

sintaxis: *<referencia ligada desreferenciada> ::=* *(1)*
<expresión de referencia ligada> -> [<nombre de modo>] *(1.1)*

semántica: La localización obtenida desreferenciando un valor de referencia ligada es la referenciada por el valor de referencia ligada.

propiedades estáticas: El modo asociado a la *referencia ligada desreferenciada* es el *nombre de modo*, si se ha especificado alguno; en cualquier otro caso es el modo referenciado del modo de la *expresión de referencia ligada*. Una *referencia ligada desreferenciada* es referenciable (por lenguaje).

condiciones estáticas: La *expresión de referencia ligada* debe ser fuerte. Si se especifica el *nombre de modo* facultativo, debe ser de lectura compatible con el modo referenciado del modo de la *expresión de referencia ligada*.

condiciones dinámicas: El tiempo de vida de una localización referenciada no debe haber finalizado.

Se produce la excepción *EMPTY* si la *expresion de referencia ligada* suministra el valor *NULL*.

ejemplos: *10.49 p->* *(1.1)*

4.2.4 REFERENCIAS LIBRES DESREFERENCIADAS

sintaxis: *<referencia libre desreferenciada> ::=* *(1)*
<expresión de referencia libre> -> <nombre de modo> *(1.1)*

semántica: La localización obtenida desreferenciando un valor de referencia libre, es la referenciada por dicho valor.

propiedades estáticas: El modo asociado a una *referencia libre desreferenciada* es el *nombre de modo*. Una *referencia libre desreferenciada* es referenciable (por lenguaje).

condiciones estáticas: La *expresión de referencia libre* debe ser fuerte.

condiciones dinámicas: La *expresión de referencia libre* no debe entregar un valor obtenido referenciando una localización no referenciable (véase el apartado 5.2.13). El tiempo de vida de una localización referenciable no debe haber finalizado.

Se produce la excepción *EMPTY* si la *expresion de referencia libre* suministra el valor *NULL*.

Se produce la excepción *MODEFAIL* si el *nombre de modo* no es de lectura compatible con el modo de la posición referenciada.

4.2.5 *ELEMENTOS CADENA*

sintaxis:

<elemento cadena> ::= (1)
<localización de cadena> (<posición>) (1.1)

sintaxis derivada: La sintaxis derivada de una subcadena de longitud 1 es un *elemento cadena* (véase el apartado 4.2.6). Por ejemplo:
<localización cadena> (<posición>)
se deriva de:
<localización cadena> (<posición> UP 1)

ejemplos: *18.16 string->(i)* (1.1)

4.2.6 *SUBCADENAS*

sintaxis:

<subcadena> ::= (1)
<localización cadena> (<elemento izquierda> : <elemento derecha>) (1.1)
| *<localización cadena> (<posición> UP <longitud de la cadena>)* (1.2)

<elemento izquierda> ::= (2)
<expresión literal entera> (2.1)

<elemento derecha> ::= (3)
<expresión literal entera> (3.1)

<posición> ::= (4)
<expresión entera> (4.1)

semántica: Una subcadena proporciona una localización cadena que es una subcadena de la localización cadena especificada.

propiedades estáticas: El modo asociado a una *subcadena* es un modo cadena parametrizado construido como sigue:
&nombre (longitud de la subcadena)
donde *&nombre* es un nombre de sínmodo virtual, sinónimo del modo (posiblemente dinámico) de la *localización cadena* y donde *longitud de la subcadena* es, o bien *longitud de la cadena*, o *NUM(elemento derecha) - NUM(elemento izquierda) + 1*

condiciones estáticas: El *elemento izquierda*, *elemento derecha* y la *longitud* de la cadena deben suministrar valores enteros no negativos tales que se mantengan las siguientes relaciones:

1. $NUM(\text{ elemento izquierda }) \leq NUM(\text{ elemento derecha })$

2. $NUM(\text{ elemento derecha }) \leq L - 1$

3. $1 \leq NUM(\text{ longitud de cadena }) \leq L$

donde *L* es la longitud de cadena de *localización cadena*. (Si la *localización cadena* es una *localización de modo dinámico*, las relaciones 2 y 3 sólo pueden comprobarse en el momento de la ejecución; véase más adelante).

condiciones dinámicas: Se produce la excepción *RANGEFAIL* si no se cumple una cualquiera de las relaciones 2 ó 3 anteriores en el caso de una *localizacion cadena* de modo dinámico o si se satisfacen alguna de las relaciones siguientes:

1. $NUM(\text{ posición }) < 0$

2. $NUM(\text{ posición }) + NUM(\text{ longitud de cadena }) > L$

donde *L* es la longitud de la cadena del modo de la *localización cadena*.

ejemplos: *18.23 string->(scanstart UP 10)* *(1.2)*

4.2.7 ELEMENTOS MATRICIALES

sintaxis: *<elemento matricial> ::=* *(1)*
<localización matriz> (<lista de expresiones>) *(1.1)*

<lista de expresiones> ::= *(2)*
<expresión> {, *<expresión>*}* *(2.1)*

sintaxis derivada: La notación: *(<lista de expresiones>)* es la sintaxis derivada de:
(<expresión>) {*(<expresión>)*}*
en la que hay tantas expresiones entre paréntesis como figuren en la *lista de expresiones*. En consecuencia un *elemento matricial* tiene sólo una expresión (índice) en estricto sentido sintáctico.

semántica: Un elemento matricial proporciona una (sub)localización que es un elemento de la localización matriz especificada.

propiedades estáticas: El modo asociado a un *elemento matricial* es el modo elemento del modo de la *localización matriz*.

Un *elemento matricial* es referenciable (en lenguaje) si la organización de elementos del modo de la *localizacion matriz* es *NOPACK*.

condiciones estáticas: La clase de *expresión* debe ser compatible con el modo índice del modo de la *localización matriz*.

condiciones dinámicas: Se produce una excepción *RANGEFAIL* si se cumple alguna de las relaciones siguientes:

1. *expresión* < *L*

2. *expresión* > *U*

donde *L* y *U* son, respectivamente el límite inferior y el límite superior (posiblemente dinámicos) del modo de la *localización matriz*.

ejemplos: *11.34 b(m.lin_1)(m.col_2)* *(1.1)*

4.2.8 SUBMATRICES

sintaxis:

<submatriz> ::= *(1)*
<localización matriz> (<elemento inferior> :
<elemento superior>) *(1.1)*
| *<localización matriz>*
(<expresión entera> UP <longitud de matriz>) *(1.2)*

<elemento inferior> ::= *(2)*
<expresión literal> *(2.1)*

<elemento superior> ::= *(3)*
<expresión literal> *(3.1)*

<longitud de matriz> ::= *(4)*
<expresión literal entera> *(4.1)*

semántica: Una submatriz proporciona una localización de (sub)matriz, que es la parte de la localización matriz especificada indicada por el *elemento inferior* y el *elemento superior*, o la *expresión entera* y la *longitud de matriz*. El límite inferior de la submatriz es igual al límite inferior de la matriz especificada y el límite superior se determina a partir de las expresiones especificadas.

propiedades estáticas: El modo asociado a una *submatriz* es un modo matriz parametrizado, definido como sigue:
&nombre(índice superior),
donde *&nombre* es un nombre de sínmodo virtual, sinónimo del modo (posiblemente dinámico) de la *localización matriz* y el *índice superior* es *L + longitud matriz - 1*, donde *L* es el límite inferior del modo matriz de la *localización matriz* o *lit*,

donde *lit* es un literal cuya clase es compatible con las clases de *elemento inferior* y *elemento superior*, tal que:
NUM(lit) = NUM(L) + NUM(elemento superior) - NUM(elemento inferior).
Una *submatriz* es referenciable (por lenguaje) si la organización de los elementos del modo de la *localización matriz* es *NOPACK*.

condiciones estáticas Las clases de *elemento inferior* y *elemento superior*, o *expresión entera* y *longitud de matriz*, deben ser compatibles con el modo índice del modo de la *localización matriz*.

El *elemento inferior*, el *elemento superior* y la *longitud de matriz* deben proporcionar valores tales que se cumplan las siguientes relaciones:

1. $L \leq$ *elemento inferior* $\leq$ *elemento superior*

2. $1 \leq$ *longitud de matriz*

3. *elemento superior* $\leq U$

4. *longitud de matriz* $\leq U - L + 1$

donde L y U son, respectivamente, los límites inferior y superior del modo de la *localización matriz*. (Si la *localización matriz* es una *localización de modo dinámico*, las relaciones 3 y 4 sólo pueden comprobarse en la ejecución, véase más adelante.)

condiciones dinámicas: Se produce la excepción *RANGEFAIL* si una cualquiera de las relaciones 3 y 4 anteriores no se cumple para una *localización matriz* de modo dinámico o si se satisface una de las relaciones siguientes:

1. $L >$ *expresión entera*

2. *expresión entera* + *longitud matriz* $- 1 > U$

donde L y U son el límite inferior y superior del modo matriz de la *localización matriz*, respectivamente.

4.2.9 *CAMPOS DE ESTRUCTURA*

sintaxis: <*campo de estructura*> ::= (1)
<*localizacion estructura*> . <*nombre de campo*> (1.1)

semántica: Un campo de estructura proporciona una (sub)localización que es un campo de la localización estructura especificada.

Si la *localización estructura* tiene un modo estructura variable sin marcadores, y el *nombre de campo* es un nombre de campo variable, la semántica se define en la realización.

propiedades estáticas: El modo del *campo de estructura* es el modo del *nombre de campo*. Un *campo de estructura* es referenciable (por lenguaje) si lo es el *nombre de campo*.

condiciones estáticas: El *nombre de campo* debe ser un nombre del conjunto de nombres de campo del modo de la *localización estructura*.

condiciones dinámicas: Se produce una excepción *TAGFAIL* si la *localización estructura* designa:

- una localización de modo estructura variable con marcadores y el (los) valor(es) de campo marcador asociado(s), indica(n) que el campo no existe;
- una localización de modo estructura dinámica parametrizada, y la lista de valores asociada indica que el campo no existe.

ejemplos: *10.52 last->.info* *(1.1)*

4.2.10 LLAMADAS A PROCEDIMIENTO QUE PRODUCEN LOCALIZACIÓN

sintaxis: *<llamada a procedimiento que produce una localización> ::=* *(1)*
<llamada a procedimiento que produce una localización> *(1.1)*

semántica: Como resultado de una llamada a procedimiento que produce una localización se proporciona una localización.

propiedades estáticas: El modo asociado a una *llamada a procedimiento que produce una localización* es el modo de la especificación del resultado de la *llamada a procedimiento que produce una localización*.

condiciones dinámicas: La *llamada a procedimiento que produce una localización* no debe entregar una localización indefinida y el tiempo de vida de la localización entregada no debe haber finalizado.

4.2.11 LLAMADAS A RUTINA PREDEFINIDA QUE PRODUCEN LOCALIZACIÓN

sintaxis: *<llamada a rutina predefinida que produce una localización> ::=* *(1)*
<llamada a rutina predefinida que produce una localización de realización> *(1.1)*

semántica: Como resultado de una llamada a rutina predefinida que produce una localización de realización se facilita una localización.

propiedades estáticas: El modo asociado a la *llamada a rutina predefinida que produce una localización* es el modo de la especificación del resultado de la *llamada a rutina predefinida que produce una localización de realización*.

condiciones dinámicas: La *llamada a rutina predefinida que produce una localización de realización* no debe suministrar una localización indefinida y el tiempo de vida de la localización entregada no debe haber finalizado.

4.2.12 CONVERSIONES DE LOCALIZACIÓN

sintaxis: *<conversión de localización> ::=* *(1)*
<nombre de modo>(<localización de modo estático>) *(1.1)*

semántica: Una conversión de localización deroga y suplanta las reglas de compatibilidad y verificación de modos del CHILL. Asocia explícitamente un modo a la localización de modo estático especificada.

La semántica dinámica precisa de una conversión de localización se define en la realización.

propiedades estáticas: El modo de una *conversión de localización* es el *nombre de modo*.

condiciones estáticas: La *localización de modo estático* debe ser referenciable.

Debe cumplirse la siguiente relación:
SIZE(nombre de modo) = SIZE(localización de modo estático)

4.2.13 SEGMENTOS DE CADENA

sintaxis:

<segmento de cadena> ::= *(1)*
<localización cadena>(<principio> : <fin>) *(1.1)*

<principio> ::= *(2)*
<expresión entera> *(2.1)*

<fin> ::= *(3)*
<expresión entera> *(3.1)*

Nota: Si el *principio* y *fin* son una *expresión literal entera*, la construcción sintáctica es ambigua y se interpretará como una *subcadena*.

semántica: Un segmento de cadena proporciona una localización cadena de modo dinámico, es decir, una cadena con una longitud desconocida estáticamente.

propiedades estáticas: El modo dinámico asociado a un *segmento de cadena* es un modo cadena parametrizada dinámica, construido de la misma forma que el de una *subcadena* (véase el apartado 4.2.6), pero con un parámetro dinámico *longitud de cadena* formado por:
NUM(fin) - NUM(principio) + 1

condiciones dinámicas: Se produce la excepción *RANGEFAIL* si se verifica alguna de las relaciones siguientes:

1. *NUM(principio) > NUM(fin)*

2. *NUM(principio) < 0*

3. *NUM(fin) ≥ L*

donde *L* es la longitud (posiblemente dinámica) del modo cadena de la *localización cadena*.

ejemplos: *18.26 blanks(count:9)* *(1.1)*

4.2.14 SEGMENTOS DE MATRIZ

sintaxis: *<segmento de matriz> ::=* *(1)*
<localización matriz>(<primero> : <último>) *(1.1)*

<primero> ::= *(2)*
<expresión> *(2.1)*

<último> ::= *(3)*
<expresión> *(3.1)*

Nota: Si *primero* y *último* son una *expresión literal*, la construcción sintáctica es ambigua y será interpretada como submatriz.

semántica: Un segmento de matriz proporciona una localización matriz de modo dinámico, es decir, una matriz con un límite superior desconocido estáticamente.

propiedades estáticas: El modo dinámico asociado a un *segmento de matriz* es un modo matriz dinámica parametrizada de la misma forma que para una *submatriz* (véase el apartado 4.2.8), pero con un parámetro dinámico *índice superior* constituido por *exp*, donde *exp* es una expresión de clase compatible con las clases de *primero* y *último*, y tal que:
NUM(exp) = NUM(L) + NUM(último) - NUM(primero)
donde *L* es el límite inferior del modo de la *localización matriz*.

Un *segmento de matriz* es referenciable (por lenguaje) si la configuración del modo de la *localización matriz* es *NOPACK*.

condiciones estáticas: Las clases de *primero* y *último* deben ser compatibles con el modo índice del modo de la *localización matriz*.

condiciones dinámicas: Se produce la excepción *RANGEFAIL* si se verifica alguna de las relaciones siguientes:

1. *primero* > *último*

2. *primero* < *L*

3. *último* > *U*

donde *L* y *U* designan, respectivamente, el límite inferior y el límite superior (posiblemente dinámico) del modo de la *localización matriz*.

ejemplos: *17.27 res(0:count-1)* *(1.1)*

4.2.15 DESCRIPTORES DESREFERENCIADOS

sintaxis: *<descriptor desreferenciado> ::=* *(1)*
<expresión descriptor> -> *(1.1)*

semántica: Un descriptor desreferenciado proporciona la localización de modo dinámico referenciada por el valor descriptor.

propiedades estáticas: El modo dinámico asociado a un *descriptor desreferenciado* se construye como sigue:
&nombre de modo origen(<parámetro> {,<parámetro>})*
donde *nombre de modo origen* es un nombre de sínmodo virtual, sinónimo del modo origen referenciado del modo de la *expresión descriptor* (fuerte) y donde los parámetros son, dependiendo del modo origen referenciado:

- la longitud dinámica, en el caso de un modo cadena;
- el límite superior dinámico, en el caso de un modo matriz;
- la lista de valores asociada con el modo de la localización estructura parametrizada, en el caso de un modo estructura variable. Un *descriptor desreferenciado* es referenciable (por lenguaje).

condiciones estáticas: La *expresión descriptor* debe ser fuerte.

condiciones dinámicas: El tiempo de vida de la localización referenciada no debe haber finalizado.

Se produce la excepción *EMPTY* si la *expresión descriptor* entrega *NULL*.

ejemplos: *8.10 input->* *(1.1)*

5.0 *VALORES Y SUS OPERACIONES*

5.1 *DEFINICIONES DE SINÓNIMOS*

sintaxis:

<sentencia de definición de sinónimo> ::= (1)
SYN <definición de sinónimo> {, <definición de sinónimo>};* (1.1)

<definición de sinónimo> ::= (2)
<lista de nombres> [<modo>] = <valor constante> (2.1)

sintaxis derivada: Una *definición de sinónimo*, en la que *lista de nombres* consta de más de un nombre, se deriva de varias ocurrencias de *definición de sinónimo*, una por cada nombre con el mismo *valor constante* y *modo*, si existe. Por ejemplo *SYN I,J = 3;* se deriva de *SYN I = 3, J = 3;*

semántica: Una definición de sinónimo define un nombre que designa el valor constante especificado.

propiedades estáticas: El nombre definido en una definición de sinónimo es un nombre de sinónimo.

Cuando se especifica un *modo*, la clase de un nombre de sinónimo es la clase M-valuada en la que M es el *modo*; en cualquier otro caso, es la clase del *valor constante*.

Un nombre de sinónimo es indefinido, única y exclusivamente si el *valor constante* es un *valor indefinido* (véase el apartado 5.3.1).

Un nombre de sinónimo es literal, única y exclusivamente si el *valor constante* es una *expresión literal*.

condiciones estáticas: Si se especifica un *modo*, éste debe ser compatible con la clase del *valor constante* y el valor proporcionado por el *valor constante* debe ser uno de los valores definidos por el *modo*.

Las definiciones de sinónimos no deben ser recursivas ni recíprocamente recursivas a través de otras definiciones de sinónimos o de modo, es decir, ningún conjunto de definiciones recursivas puede contener definiciones de sinónimos (véase el apartado 3.2.1).

ejemplos:

```
1.14    SYN neutral_for_add = 0,                    (1.1)
            neutral_for_mult = 1;
2.17    neutral_for_add_fraction = [0,1]            (2.1)
```

5.2 *VALOR PRIMITIVO*

5.2.1 *CONSIDERACIONES GENERALES*

sintaxis:

<valor primitivo> (1)

<contenido de localización> (1.1)
| *<nombre de valor>* (1.2)
| *<literal>* (1.3)
| *<tupla>* (1.4)
| *<valor elemento cadena>* (1.5)
| *<valor subcadena>* (1.6)
| *<valor segmento de cadena>* (1.7)
| *<valor elemento matricial>* (1.8)
| *<valor submatriz>* (1.9)
| *<valor segmento de matriz>* (1.10)
| *<valor campo de estructura>* (1.11)
| *<localización referenciada>* (1.12)
| *<conversión de expresión>* (1.13)
| *<llamada a procedimiento que proporciona valor>* (1.14)
| *<llamada a rutina predefinida que proporciona valor>* (1.15)
| *<expresión arrancar>* (1.16)
| *<expresión recibir>* (1.17)
| *<operador cero-ádico>* (1.18)

semántica: Un valor primitivo es el componente básico de una expresión. Algunos valores primitivos (contenido de localización correspondiente a un modo dinámico, algunas tuplas, valores segmentos de matriz, valores segmentos de cadena) tienen una clase dinámica es decir una clase basada en un modo dinámico. Las verificaciones de compatibilidad para tales valores primitivos sólo pueden completarse en la ejecución. Los fallos de estas verificaciones provocarán la excepción *TAGFAIL* o *RANGEFAIL*.

propiedades estáticas: La clase del *valor primitivo* es la clase del *contenido de localización*, del *nombre de valor*, etc., respectivamente.

Un *valor primitivo* es un *valor primitivo constante*, única y exclusivamente si se trata de un *nombre de valor constante*, *literal*, *tupla constante*, *localización referenciada constante*, *conversión de expresión constante*, o *llamada a rutina predefinida que proporciona valor constante*.

Un *valor primitivo* es un *valor primitivo literal*, única y exclusivamente si es un *nombre de valor literal*, un *literal discreto* o una *llamada a rutina predefinida que proporciona valor literal*.

5.2.2 *CONTENIDOS DE LOCALIZACIÓN*

sintaxis:

<contenidos de localización> ::= (1)
<localización> (1.1)

semántica: El contenido de localización proporciona el valor almacenado en la localización especificada. Para obtener dicho valor almacenado debe accederse a la localización.

propiedades estáticas: La clase de *contenidos de localización* es la clase M-valuada donde M es el modo (posiblemente dinámico) de la *localización*.

condiciones estáticas: El modo de la *localización* no debe poseer la propiedad de sincronización.

condiciones dinámicas: El valor proporcionado no debe ser indefinido (véase el apartado 5.3.1).

ejemplos:

3.6 c2.im (1.1)

5.2.3 *NOMBRES DE VALOR*

sintaxis:

<nombre de valor> ::= (1)
<nombre de sinónimo> (1.1)
| *<nombre de enumeración de valores>* (1.2)
| *<nombre de valor do-with>* (1.3)
| *<nombre de valor recibir>* (1.4)

semántica: Un nombre de valor suministra un valor.

Un nombre de valor es uno de los siguientes:

- un nombre de sinónimo, es decir un nombre definido en una *sentencia de definición de sinónimo*;
- un *contador de bucle*, en una *enumeración de valores*;
- un nombre de valor do-with, es decir un nombre de campo introducido como nombre de valor en una *acción hacer* con una *componente de simultaneidad*.

. un nombre de valor recibir, es decir un nombre introducido en una *acción recibir y elegir*.

propiedades estáticas: La clase de un *nombre de valor* es la clase del *nombre de sinónimo*, del *nombre de enumeración de valores* del *nombre de valor do-with*, del *nombre de valor recibir*, respectivamente.

Un *nombre de valor*, es constante (literal), única y exclusivamente si es un *nombre de sinónimo*, (*nombre de sinónimo literal*)

condiciones estáticas: El *nombre de sinónimo* no debe ser indefinido.

condiciones dinámicas: La evaluación de un *nombre de valor do-with*, produce una excepción *TAGFAIL*, si el valor designado es un campo variable de:

. un valor modo estructura variable con marcadores, cuyo(s) campo(s) marcador(es) asociado indica(n) que el campo designado no existe;

. un valor de modo estructura dinámica parametrizada cuya lista de valores asociada indica que el campo designado no existe.

ejemplos:

10.9	*max*	*(1.1)*
8.8	*i*	*(1.2)*
15.45	*THIS_COUNTER*	*(1.4)*

5.2.4 *LITERALES*

5.2.4.1 *Condiciones generales*

sintaxis: *<literal> ::=* *(1)*

<literal entero> *(1.1)*
| *<literal booleano>* *(1.2)*
| *<literal de conjunto>* *(1.3)*
| *<literal de vacío>* *(1.4)*
| *<literal de procedimiento>* *(1.5)*
| *<literal de cadena de caracteres>* *(1.6)*
| *<literal de cadena de bits>* *(1.7)*

semántica: Un literal proporciona un valor constante conocido en el momento de la compilación.

propiedades estáticas: La clase del *literal*, es la clase del *literal entero*, *literal booleano*, ... etc., respectivamente. Un *literal* es discreto, si es un *literal entero*, un *literal booleano*, un *literal de conjunto*, un *literal de cadena de caracteres* de longitud 1 ó un *literal de cadena de bits* de longitud 1.

5.2.4.2 *Literales enteros*

sintaxis:

<literal entero> ::= (1)

<literal entero decimal> (1.1)

| *<literal entero binario>* (1.2)

| *<literal entero octal>* (1.3)

| *<literal entero hexadecimal>* (1.4)

<literal entero decimal> ::= (2)

[*D'*] {*<dígito>* | _}$^{+}$ (2.1)

<literal entero binario> ::= (3)

B' { *0*| *1* | _}$^{+}$ (3.1)

<literal entero octal> ::= (4)

O' { *0*| *1* | *2*| *3*| *4*| *5*| *6*| *7*| _}$^{+}$ (4.1)

<literal entero hexadecimal> ::= (5)

H' {*<dígito hexadecimal>* | _}$^{+}$ (5.1)

<dígito> ::= (6)

0 | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* (6.1)

<dígito hexadecimal> ::= (7)

<dígito> | *A* | *B* | *C* | *D* | *E* | *F* (7.1)

semántica: Un literal entero suministra un valor entero. Se han previsto la notación usual decimal así como la binaria, octal, hexadecimal y decimal explícita. El símbolo de subrayado (_) no es significativo, es decir sirve sólo a fines de lectura y no influye en el valor designado.

propiedades estáticas: La clase de un *literal entero*, es la clase *INT*-derivada.

condiciones estáticas: La cadena que sigue al apóstrofo (') y totalidad del *literal entero*, no pueden constar solamente de símbolos subrayados.

ejemplos:

6.11 *1_721_119* (1.1)

--- *D'1_721_119* (1.1)

--- *B'101011_110100* (1.2)

--- *O'53_64* (1.3)

--- *H'AF4* (1.4)

5.2.4.3 *Literales booleanos*

sintaxis:

<literal booleano> ::= (1)
FALSE | TRUE (1.1)

semántica: Un literal booleano proporciona un valor booleano.

propiedades estáticas: La clase de un *literal booleano* es la clase *BOOL*-derivada.

ejemplos:

5.46 FALSE (1.1)

5.2.4.4 *Literales de conjunto*

sintaxis:

<literal de conjunto> ::= (1)
<nombre de elemento del conjunto> (1.1)

semántica: Un literal de conjunto proporciona un valor de conjunto. Un literal de conjunto se define como nombre en un modo conjunto.

propiedades estáticas: La clase de un *literal de conjunto*, es la clase M-derivada donde M es el modo conjunto (en un contexto dado) que tiene el *nombre de elemento del conjunto* especificado como nombre de elemento del conjunto.

ejemplos:

6.51 dec (1.1)
11.89 king (1.1)

5.2.4.5 *Literal de vacío*

sintaxis:

<literal de vacío> ::= (1)
NULL (1.1)

semántica: El literal de vacío entrega ya sea, el valor de referencia vacía, es decir un valor que no se refiere a una localización, el valor de procedimiento vacío, esto es un valor que no indica un procedimiento o el valor ejemplo vacío que es un valor que no identifica un proceso.

propiedades estáticas: La clase del *literal de vacío* es la clase nula.

ejemplos:

10.40 NULL *(1.1)*

5.2.4.6 *Literales de procedimiento*

sintaxis:

<literal de procedimiento> ::= *(1)*
<nombre de procedimiento general> *(1.1)*

semántica: Un literal de procedimiento proporciona un valor de procedimiento general. Un literal de procedimiento se define como nombre en una definición de procedimiento o de entrada (véase el apartado 7.4).

propiedades estáticas: La clase del *literal de procedimiento*, es la clase M-derivada, en la que M es el modo del *nombre de procedimiento general*.

5.2.4.7 *Literales de cadena de caracteres*

sintaxis:

<literal de cadena de caracteres> ::= *(1)*
' {<carácter distinto del apóstrofo> | <apóstrofo>}'* *(1.1)*
| C' {<dígito hexadecimal> <dígito hexadecimal>}'* *(1.2)*

<carácter> ::= *(2)*
<letra> *(2.1)*
| <dígito> *(2.2)*
| <símbolo> *(2.3)*
| <espacio> *(2.4)*

<letra> ::= *(3)*
A | B | C | D | E | F | G | H | I | J | K | L | M *(3.1)*
| N | O | P | Q | R | S | T | U | V | W | X | Y | Z *(3.2)*

<símbolo> ::= *(4)*
*_ | ' | (|) | * | + | , | - | .| / | : | ; | < | = | >* *(4.1)*

<espacio> ::= *(5)*
SP *(5.1)*

<apóstrofo> ::= *(6)*
'' *(6.1)*

Nota: *SP* representa el carácter "espacio"; véase el Apéndice A1.

semántica: Un literal de cadena de caracteres proporciona un valor cadena de caracteres que puede tener longitud 0. Un literal de cadena de caracteres de longitud 1 puede servir como valor carácter. Para representar el carácter apóstrofo (') dentro de un literal de cadena de caracteres, debe escribirse dos veces (''). Los caracteres mencionados anteriormente constituyen el conjunto mínimo de caracteres susceptibles de impresión que debe proporcionarse. Toda realización deberá admitir cualquier carácter imprimible en un literal de cadena de caracteres que pertenezca al alfabeto N.º 5 del CCITT (véase el Apéndice A1). Además de la representación imprimible puede emplearse la representación hexadecimal. Cada pareja de dígitos hexadecimales designa al valor de carácter cuya representación corresponde al número hexadecimal dado (véase el Apéndice A1).

propiedades estáticas: La longitud de un *literal de cadena de caracteres* es, o el número de *caracteres distintos del apóstrofo* y ocurrencias de *apóstrofo* o el número de pares de *dígitos hexadecimales*.

La clase de un *literal de cadena de caracteres* es la clase *CHAR(n)*-derivada, donde *n* es la longitud del *literal de cadena de caracteres*.

ejemplos:

8.18 'A-B<ZAA9K''' *(1.1)*
8.18 '' *(6.1)*

5.2.4.8 *Literales de cadena de bits*

sintaxis:

<literal de cadena bits> ::= *(1)*
<literal binario de cadena de bits> *(1.1)*
<literal octal de cadena de bits> *(1.2)*
<literal hexadecimal de cadena de bits> *(1.3)*

<literal binario de cadena de bits> ::= *(2)*
B' {0 | 1 | _ }'* *(2.1)*

<literal octal de cadena de bits> ::= *(3)*
0' {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | _}'* *(3.1)*

<literal hexadecimal de cadena de bits> ::= *(4)*
H' {<dígito hexadecimal> | _}'* *(4.1)*

semántica: Un literal de cadena de bits, proporciona un valor cadena de bits que puede tener longitud 0. Pueden emplearse notaciones binaria, octal o hexadecimal. El símbolo de subrayado (_) no es significativo y sirve solamente para facilitar la lectura, no influyendo en el valor indicado.

propiedades estáticas: La longitud de un *literal de cadena de bits*, es o el número de ocurrencias *0* y *1* después de *B'* o el tres veces el número de ocurrencias de *0, 1, 2, 3, 4, 5, 6 y 7* después de *O'* o cuatro veces el número de ocurrencias de *dígito hexadecimal* después de *H'*. La clase de un *literal de cadena de bits* es la clase *BIT(n)*-derivada donde *n* es la longitud del *literal de cadena de bits*.

ejemplos:

--- *B'101011_110100'* (1.1)
--- *O'53_64'* (1.2)
--- *H'AF4'* (1.3)

5.2.5 *TUPLAS*

sintaxis:

<tupla> ::= (1)
[*<nombre de modo>*] *(:*
{*<tupla conjuntista>* | *<tupla matricial>* | *<tupla de estructura>*}*:)* (1.1)

<tupla conjuntista> ::= (2)
[{*<expresión>* | *<intervalo>*}
{, {*<expresión>* | *<intervalo>*}}*] (2.1)

<intervalo> ::= (3)
<expresión> : *<expresión>* (3.1)

<tupla matricial> ::= (4)
<tupla matricial no etiquetada> (4.1)
| *<tupla matricial etiquetada>* (4.2)

<tupla matricial no etiquetada> ::= (5)
<valor> {,*<valor>*}* (5.1)

<tupla matricial etiquetada> ::= (6)
<lista de etiquetas de caso> : *<valor>*
{,*<lista de etiquetas de caso>* : *<valor>*}* (6.1)

<tupla de estructura> ::= (7)
<tupla de estructura no etiquetada> (7.1)
| *<tupla de estructura etiquetada>* (7.2)

<tupla de estructura no etiquetada> ::= (8)
<valor> {,*<valor>*}* (8.1)

<tupla de estructura etiquetada> ::= (9)
<lista de nombres de campo> : *<valor>*
{, *<lista de nombres de campo>* : *<valor>*}* (9.1)

<lista de nombres de campo> ::= (10)
.*<nombre de campo>* {, .*<nombre de campo>*}* (10.1)

sintaxis derivada: La sintaxis derivada de (: y :) son los corchetes [y] de apertura y cierre de la tupla, respectivamente. Esto no se indica en la sintaxis para evitar confusión con la utilización de corchetes como metasímbolos.

semántica: Una tupla entrega un valor conjuntista, un valor matricial o un valor estructura.

Si se trata de un valor conjuntista, consta de una lista de expresiones y/o intervalos, que designan los valores miembros del valor conjuntista. Un intervalo designa aquellos valores comprendidos en el mismo o son uno de los valores proporcionados por las expresiones del intervalo. Si la segunda expresión proporciona un valor inferior al entregado por la primera, se dice que el intervalo es vacío, es decir no designa valores. La tupla conjuntista, puede designar el valor conjuntista vacío.

Si se trata de un valor matricial, es una lista de valores (posiblemente etiquetada) de los elementos de la matriz; en una tupla matricial no etiquetada, los valores de los elementos se ordenan según su índice creciente, en una tupla matricial etiquetada, se dan los valores de los elementos cuyos índices se especifican en la lista de etiquetas de caso que etiqueta al valor. Esta puede utilizarse como notación abreviada para tuplas matriciales grandes, en las que hay muchos valores comunes. La etiqueta *ELSE*, designa todos los valores índices no mencionados explícitamente, la etiqueta * designa todos los valores índice (por más detalles, véase el punto 9.1.4).

Si se trata de un valor estructura, es un conjunto de valores (posiblemente etiquetado) de los campos de la estructura. En la tupla de estructura no etiquetada, se indican los valores de los campos en el mismo orden en que están especificados en el modo estructura asociado. En la tupla de estructura etiquetada, se dan los valores de los campos cuyos nombres están especificados en la lista de nombres de campo del valor.

No está definido el orden de evaluación de las expresiones y valores de una tupla, por lo que puede considerarse que se evalúan en un orden mixto.

propiedades estáticas: La clase de una *tupla*, es la clase M-valuada donde M es el *nombre de modo*, si se ha especificado. En cualquier otro caso, M depende del contexto en el que aparece la *tupla* de acuerdo con las normas siguientes:

- si la *tupla* es el *valor* o el *valor constante*, de una *inicialización* en una *declaración de localización*, entonces M es el *modo* de la *declaración de localización*;
- si la *tupla* es el *valor* del segundo miembro de una *acción de asignación simple*, entonces M es el modo (posiblemente dinámico) de la *localización* del primer miembro;

(3857)

- si la *tupla* es el *valor* constante de una *definición de sinónimo*, con un *modo* especificado, M es dicho *modo*;

- si la *tupla* es un *parámetro efectivo* de una *llamada a procedimiento*, M es el modo de la especificación de parámetro correspondiente;

- si la *tupla* es el *valor* en una *acción retornar* o una *acción resultado*, M es el modo resultado del nombre de procedimiento de la *acción resultado* o *acción retornar* (véase el apartado 6.8);

- si la *tupla* es un *valor* de una *acción enviar*, entonces es el modo asociado especificado en la definición de señal del *nombre de señal* o el modo elemento tampón del modo de la *localización tampón*;

- si la *tupla* es una *expresión* de una *tupla matricial*, M es el modo elemento del modo de la *tupla matricial*;

- si la *tupla* es una *expresión* de una *tupla de estructura no etiquetada* o de una *tupla de estructura etiquetada* cuya *lista de nombres de campo* asociada, contiene solamente un *nombre de campo*, M es el modo de campo de la *tupla de estructura* para la que se especifica la tupla.

Una *tupla* es constante, única y exclusivamente si cada *valor* o *expresión* de la misma lo es.

condiciones estáticas: El *nombre de modo* facultativo sólo puede suprimirse en los contextos especificados anteriormente. Según se especifique una *tupla conjuntista, matricial,* o *de estructura,* deberán cumplirse los siguientes requerimientos de compatibilidad:

a. *tupla conjuntista*

1. El modo de la *tupla* debe ser un modo conjuntista.

2. La clase de cada *expresión*, debe ser compatible con el modo primitivo del modo de la *tupla*.

3. El valor proporcionado por cada *expresión*, debe ser uno de los valores definidos por dicho modo primitivo.

b. *tupla matricial*

1. El modo de la *tupla*, debe ser un modo matriz.

2. La clase de cada *valor*, debe ser compatible con el modo elemento del modo de la *tupla*.

En el caso de una *tupla matricial no etiquetada*:

3. Deben producirse tantas ocurrencias del *valor* como número de elementos tenga el modo matriz de la *tupla*.

En el caso de una *tupla matricial etiquetada*:

4. Deben cumplirse las condiciones de selección según caso para la lista de ocurrencias de la *lista de etiquetas de caso* (véase el apartado 9.1.3). La clase resultante debe ser compatible con el modo índice del modo de la *tupla*.

5. El valor proporcionado por cada *expresión literal* de cada *lista de etiquetas de caso* y los valores definidos por cada *nombre de modo* de cada *lista de etiquetas de caso*, deben estar comprendidos entre los límites inferior y superior (ambos incluidos) del modo de la *tupla*.

6. En una *tupla matricial no etiquetada*, por lo menos una ocurrencia de *valor* debe ser una *expresión*; en una *tupla matricial etiquetada*, al menos una ocurrencia de *valor* que siga a una *lista de etiquetas de caso* que no sea (*ELSE*), debe ser una *expresión* (véase el apartado 5.3.1).

7. Para una *tupla* (matricial) constante en la que el modo elemento del modo de la *tupla* es discreto, cada *valor* especificado debe proporcionar un valor comprendido entre los límites del modo elemento (con inclusión de éstos) a menos que se trate de un valor indefinido.

c. *tupla de estructura*

1. El modo de la tupla debe ser un modo estructura.

2. Este modo no puede ser un modo estructura que tenga nombres de campo invisibles (véase el apartado 9.2.7).

En el caso de una *tupla de estructura no etiquetada:*

- Si el modo de la *tupla* no es un modo estructura variable, ni un modo estructura parametrizada:

3. Deben producirse tantas ocurrencias de *valor*, como nombres de campo haya en la lista de nombres de campo del modo de la *tupla*.

4. La clase de cada *valor* debe ser compatible con el modo del nombre de campo correspondiente (en posición) al modo de la *tupla*.

- Si el modo de la *tupla* es un modo estructura variable con marcadores o un modo estructura parametrizado con marcadores:

  5. Cada *valor* especificado por un campo marcador debe ser una *expresión literal*.

  6. Debe haber tantas ocurrencias de *valor* como nombres de campo haya indicados como existentes por el valor o valores proporcionados por las ocurrencias de *expresión literal* especificadas para los campos marcadores.

  7. La clase de cada *valor* debe ser compatible con el modo del nombre de campo correspondiente.

- Si el modo de la *tupla* es un modo estructura variable sin marcadores o un modo estructura parametrizada sin marcador(es):

  8. No se admite una *tupla de estructura no etiquetada*.

En el caso de una *tupla de estructura etiquetada:*

- Si el modo de la *tupla* no es ni un modo estructura variable, ni un modo estructura parametrizado:

  9. Cada nombre de campo de la lista de nombres de campo del modo de la *tupla*, debe mencionarse solamente una vez y solo una en una lista de *nombres de campo* y en el mismo orden que en el modo de la *tupla*.

  10. La clase de cada *valor*, debe ser compatible con el modo del nombre de campo especificado en la *lista de nombres de campo* que etiqueta dicho *valor*.

- Si el modo de la *tupla* es un modo estructura variable sin marcadores, o un modo estructura parametrizada con marcadores

  11. Cada *valor* especificado para un campo marcador, debe ser una *expresión literal*.

  12. Solamente pueden especificarse nombres de campo correspondientes a campos indicados como existentes, por el valor o valores proporcionados por las ocurrencias de la *expresión literal* especificada para campos marcadores y todos ellos deben especificarse en el mismo orden que en el modo de la *tupla*.

  13. La clase de cada *valor*, debe ser compatible con el modo del nombre de campo especificado en la *lista de nombres de campo* que etiqueta dicho *valor*.

- Si el modo de la *tupla* es un modo estructura variable sin marcadores o un modo estructura parametrizada sin marcador(es):

14. Los nombres de campo mencionados en la *lista de nombres de campo* y definidos en los mismos *campos alternativos*, deben definirse todos en la misma *alternativa variable* o después de *(ELSE)*. Todos los nombres de campo de una alternativa variable seleccionada o que se hayan definido después de *ELSE*, deben mencionarse una sola vez y en el mismo orden que en el modo de la *tupla*.

15. La clase de cada *valor*, debe ser compatible con el modo de cualquier nombre de campo especificado en la *lista de nombres de campo* enfrente de cada *valor*.

16. Si el modo de una *tupla* es un modo estructura parametrizada, con marcadores, la lista de valores proporcionados por las ocurrencias de la *expresión literal* especificadas para los campos marcadores, debe ser la misma que la lista de valores del modo de la *tupla*.

17. Para una tupla (estructura) constante, cada valor especificado para un campo con modo discreto, debe proporcionar un valor comprendido entre los límites del modo del campo (ambos incluidos) a menos que se trate de un *valor indefinido*.

18. Por lo menos una ocurrencia de *valor* deberá ser una *expresión*.

condiciones dinámicas: En caso de *tupla conjuntista*, *tupla matricial* o *tupla de estructura*, se aplican las condiciones de asignación de cualquier valor con respecto al modo primitivo, modo elemento o modo de campo asociado, respectivamente (véase el apartado 6.2.3) (véanse las condiciones a2, a3, b2, c4, c7, c10, c13 y c15).

Si la *tupla* tiene un modo matriz dinámico, se produce la excepción *RANGEFAIL* en caso de fallo de cualquiera de las condiciones b3 ó b5.

Si la *tupla* tiene un modo estructura parametrizada dinámica, se produce la excepción *TAGFAIL* si falla la verificación c16.

ejemplos:

9.5	*number_list[ ]*	*(1.1)*
9.6	*[2:max]*	*(2.1)*
8.24	*[('A'):3,('B', 'K', 'Z'):1,(ELSE):0]*	*(6.1)*
17.5	*[(*):' ']*	*(6.1)*
12.28	*(:NULL, NULL, 536:)*	*(7.1)*
11.16	*[.status:occupied,.p:[white, rook]]*	*(9.1)*

5.2.6 *VALORES DE ELEMENTOS DE CADENA*

sintaxis:

<valor elemento de cadena> ::= *(1)*
<expresión cadena> <posición> *(1.1)*

sintaxis derivada: La sintaxis derivada de un *valor subcadena* de longitud 1, es un *valor elemento de cadena* (véase el apartado 5.2.7) por ejemplo:
<expresión cadena>(<posición>)
se deriva de:
<expresión cadena>(<posición> UP 1)

5.2.7 *VALORES SUBCADENAS*

sintaxis:

<valor subcadena> ::= *(1)*
<expresión cadena>(<elemento izquierda> : <elemento derecha>) *(1.1)*
| *<expresión cadena>(<posición> UP <longitud de la cadena>)* *(1.2)*

Nota. Si la *expresión cadena* es una *localización cadena*, la construcción sintáctica es ambigua y se interpretará como una *subcadena* (véase el apartado 4.2.6).

semántica: Un *valor subcadena*, entrega un valor cadena que es un subvalor del valor cadena especificado.

propiedades estáticas: La clase de un *valor subcadena*, es la clase *CHAR (n)*-derivada o la clase *BIT(n)*-derivada, si la clase de la *expresión cadena* no es fuerte (dependiendo de que la *expresión cadena* sea de bits o de caracteres). En cualquier otro caso, es la clase *&nombre(n)*-valuada, donde *n* es *<longitud de cadena>* o *NUM (elemento derecha) - NUM (elemento izquierda) + 1* y *&nombre*, es un nombre de sínmodo virtual, sinónimo del modo de la *expresión cadena*.

condiciones estáticas: El *elemento izquierda*, *elemento derecha* y *longitud*, deben proporcionar valores enteros no negativos, tales que se cumplan las siguientes relaciones:

1. *NUM (elemento izquierda)* $\leqslant$ *NUM (elemento derecha)*

2. *NUM (elemento derecha)* $\leqslant$ *L-1*

3. *1* $\leqslant$ *NUM (longitud de cadena)* $\leqslant$ *L*

donde *L* es la longitud de la cadena del modo raíz de la clase de la *expresión cadena* (si la *expresión cadena*, tiene una clase dinámica, las verificaciones 2 y 3 solamente pueden realizarse durante la ejecución; véase más adelante).

condiciones dinámicas: El valor proporcionado por un *valor subcadena*, no debe ser indefinido.

Se produce la excepción *RANGEFAIL*, si no se cumple alguna de las relaciones 2 ó 3 anteriores, en caso de una *expresión cadena* con una clase dinámica o si se satisface alguna de las relaciones siguientes:

1. *NUM (posición) < 0*

2. *NUM (posición) + NUM (longitud de cadena)> L*

donde *L* es la longitud de la cadena del modo raíz de la clase de la *expresión cadena*.

5.2.8 *VALORES SEGMENTOS DE CADENA*

sintaxis:

<valor segmento de cadena> ::= *(1)*
<expresión cadena>(<principio> : <fin>) *(1.1)*

Nota: Si la *expresión cadena* es una *localización cadena* la construcción sintáctica es ambigua y será interpretada como un *segmento de cadena* (véase el apartado 4.2.13). Si tanto *principio* como *fin* son una *expresión literal entera*, la construcción sintáctica es ambigua y se interpretará como un *valor subcadena* (véase el apartado 5.2.7).

semántica: Un valor segmento de cadena, proporciona un valor cadena dinámico que es un subvalor del valor cadena especificado.

propiedades estáticas: La clase de un *valor segmento de cadena* se define de la misma forma que para el caso del valor subcadena (véase el apartado 5.2.7), pero con un parámetro dinámico *n* formado así: *NUM (fin) - NUM (principio) + 1*.

condiciones dinámicas: El valor proporcionado por un *valor segmento de cadena*, no debe ser indefinido.

Se produce la excepción *RANGEFAIL*, si se cumplen algunas de las relaciones siguientes:

1. *NUM (principio) > NUM (fin)*

2. *NUM (principio) < 0*

3. *NUM (fin) > L*

donde *L* es la longitud (posiblemente dinámica) del modo raíz de la clase de la *expresión cadena*.

5.2.9 *VALORES ELEMENTOS MATRICIALES*

sintaxis:

<valores elementos matriciales> ::= (1)
<expresión matricial> (<lista de expresiones>) (1.1)

Nota: Si la *expresión matricial* es una *localización matriz*, la contrucción sintáctica es ambigua y será interpretada como un *elemento matricial* (véase el apartado 4.2.7).

sintaxis derivada: Véase el apartado 4.2.7

semántica: Un valor elemento matricial, proporciona un valor que es un elemento del valor matricial especificado.

propiedades estáticas: La clase de un *valor elemento matricial*, es la clase M-valuada, donde M es el modo elemento del modo de la *expresión matricial*.

condiciones estáticas: La *expresión matricial*, debe ser fuerte. La clase de la *expresión*, debe ser compatible con el modo índice del modo de la *expresión matricial*.

condiciones dinámicas: El valor proporcionado por un *valor elemento matricial* no debe ser indefinido.

Se produce una excepción *RANGEFAIL*, si se cumple alguna de las relaciones siguientes:

1. *expresión <L*

2. *expresión >U*,

donde *L* y *U* son los límites inferior y superior (posiblemente dinámicos), respectivamente, del modo de la *expresión matricial*.

5.2.10 *VALORES SUBMATRICIALES*

sintaxis:

<valor submatricial> ::= (1)
<expresión matricial>
(<elemento inferior> : <elemento superior>) (1.1)
| *<expresión matricial>*
(<expresión entera> UP <longitud de matriz>) (1.2)

Nota: Si la *expresión matricial*, es una *localización matriz*, la construcción sintáctica es ambigua y se interpretará como una *submatriz* (véase el apartado 4.2.8).

semántica: Un valor submatricial, proporciona un valor (sub) matricial que es una parte del valor matricial especificado. El límite inferior del valor submatricial es igual al límite inferior del valor matricial especificado y el límite superior se determina mediante las expresiones (índice) especificadas.

propiedades estáticas: La clase de un *valor submatricial*, es la clase *M*-valuada, donde M es un modo matriz parametrizado definido como sigue:
&nombre (índice superior)
donde *&nombre*, es un nombre de sínmodo virtual, sinónimo del modo (posiblemente dinámico) de la *expresión matricial* e *índice superior* es o *L + longitud de la matriz - 1* donde *L* es el límite inferior del modo de la *expresión matricial*, o *lit*, donde *lit* es el literal cuya clase es compatible con las clases de *elemento inferior* y el *elemento superior*, tal que:
NUM(lit) = NUM(L) + NUM(elemento superior) - NUM(elemento inferior).

condiciones estáticas: La *expresión matricial*, debe ser fuerte. Las clases de *elemento inferior*, *elemento superior* o *expresión entera* y *longitud de la matriz* deben ser compatibles con el modo índice del modo de la *expresión matricial*. El *elemento inferior*, *elemento superior* y *longitud de la matriz*, deben proporcionar valores tales que se cumplan las siguientes relaciones:

1. $L \leq$ *elemento inferior* $\leq$ *elemento superior*

2. $1 \leq$ *longitud de la matriz*

3. *elemento superior* $< U$

4. *longitud de la matriz* $\leq U - L + 1$

donde *L* y *U* son los límites inferior y superior del modo matriz de la *expresión matricial*, respectivamente. (Si la *expresión matricial* contiene una clase dinámica, solamente pueden verificarse las relaciones 3 y 4 en el momento de la ejecución; véase más adelante.)

condiciones dinámicas: El valor proporcionado por un *valor submatricial* no debe ser indefinido.

Se produce una excepción *RANGEFAIL*, si no se cumple alguna de las relaciones 3 ó 4 anteriores, en caso de una clase dinámica o si se satisface alguna de las relaciones siguientes:

1. $L >$ *expresión entera*

2. *expresión entera* + *longitud matriz* $- 1 > U$,

donde *L* y *U*, son los límites superior e inferior del modo de la *expresión matricial*, respectivamente.

5.2.11 *VALORES SEGMENTOS DE MATRIZ*

sintaxis:

<valor segmento de matriz> ::= (1)
<expresión matricial> (<primero> : <último>) (1.1)

Nota: Si la *expresión matricial*, es una *localización matriz*, la construcción sintáctica es ambigua y se interpretará como un *segmento de matriz* (véase el apartado 4.2.14). Si *primero* y *último*, son una *expresión literal*, la construcción sintáctica es ambigua y se interpretará como un *valor submatricial* (véase el apartado 5.2.10).

semántica: Un valor segmento de matriz proporciona un valor matricial dinámico que es un subvalor del valor matricial especificado.

propiedades estáticas: La clase de un *valor segmento de matriz*, es la clase M-valuada, donde M es un modo matriz parametrizado dinámico definido de la misma forma que en el caso del *valor submatricial* (véase el apartado 5.2.10) pero con un parámetro dinámico de *índice superior* formado como *exp* donde *exp* es una expresión de clase compatible con las clases de *primero* y *último* y tal que:
NUM(exp) = NUM(L) + NUM(último) - NUM(primero)

condiciones estáticas: La *expresión matricial*, debe ser fuerte. Las clases de *primero* y *último*, deben ser compatibles con el modo índice del modo de la *expresión matricial*.

condiciones dinámicas: El valor proporcionado por un *valor segmento de matriz*, no debe ser indefinido.

Se produce la excepción *RANGEFAIL*, si se cumplen algunas de las relaciones siguientes:

1. *primero* > *último*

2. *primero* < *L*

3. *último* > *U*,

donde *L* y *U* designan el límite inferior y el límite superior (posiblemente dinámico) del modo de la *expresión matricial*, respectivamente.

5.2.12 *VALORES CAMPO DE ESTRUCTURA*

sintaxis:

<valor campo de estructura> ::= (1)
<expresión estructura> . <nombre de campo> (1.1)

Nota: Si la *expresión estructura* es una *localización estructura*, la construcción sintáctica es ambigua y se interpretará como un *campo de estructura* (véase el apartado 4.2.9).

semántica: Un valor campo de estructura proporciona un valor que es el campo del valor estructura especificado. Si la *expresión estructura* tiene un modo estructura variable sin marcadores y el *nombre de campo* es un nombre de campo variable, la semántica se define en la realización.

propiedades estáticas: La clase del *valor campo de estructura*, es la clase M-valuada, donde M es el modo del *nombre de campo*.

condiciones estáticas: La *expresión estructura*, debe ser fuerte. El *nombre de campo* debe ser un nombre perteneciente al conjunto de nombres de campo del modo de la *expresión estructura*.

condiciones dinámicas: El valor proporcionado por un *valor de campo de estructura*, no debe ser indefinido.

Se produce la excepción *TAGFAIL*, si la *expresión estructura* tiene:

- un modo estructura variable con marcadores, en el que el valor o valores de campo marcador indica que el campo designado no existe.
- un modo estructura dinámica parametrizada, cuya lista de valores asociada indica que el campo no existe.

ejemplos:

16.49 (RECEIVE USER_BUFFER). ALLOCATOR *(1.1)*

5.2.13 *LOCALIZACIONES REFERENCIADAS*

sintaxis:

<localización referenciada> ::= *(1)*
-> <localización> *(1.1)*

semántica: Una localización referenciada proporciona una referencia de la localización especificada si ésta es referenciable. Si no lo es, proporciona un valor de referencia que no puede desrreferenciarse (véase el apartado 4.2.4) y que puede referirse a una localización definida en la realización.

propiedades estáticas: Si la *localización* es referenciable, la clase de la *localización referenciada* es la clase M-referenciada, donde M es el modo de la *localización*. En cualquier otro caso la clase de la *localización referenciada* es la clase *PTR*-derivada. Una *localización referenciada* es constante única y exclusivamente si dicha *localización* es estática.

ejemplos:

8.23 ->c *(1.1)*

5.2.14 *CONVERSIONES DE EXPRESIONES*

sintaxis:

<conversión de expresión> ::= *(1)*
<nombre de modo> (<expresión>) *(1.1)*

semántica: Una conversión de expresión deroga y suplanta la verificación de modos y las reglas de compatibilidad de CHILL. Asocia explícitamente un modo a la expresión. La semántica dinámica precisa de una conversión de expresiones, se define en la realización y depende de la representación interna de los valores.

propiedades estáticas: La clase de *conversión de expresión*, es la clase M-valuada, donde M es el *nombre de modo*. Una *conversión de expresión* es constante, única y exclusivamente si lo es la *expresión*.

condiciones estáticas: La *expresión* no debe tener una clase dinámica. La clase de la *expresión* debe ser compatible al menos con un modo de igual tamaño que el del *nombre de modo*. El *nombre de modo*, no debe tener la propiedad de sincronización.

5.2.15 *LLAMADAS A PROCEDIMIENTO QUE PROPORCIONAN VALOR*

sintaxis:

<llamada a procedimiento que proporciona valor> ::= *(1)*
<llamada a procedimiento que proporciona valor> *(1.1)*

semántica: Una llamada a procedimiento que proporciona valor, proporciona el valor entregado por un procedimiento.

propiedades estáticas: La clase de la *llamada a procedimiento que proporciona valor*, es la clase M-valuada, donde M es el modo de la especificación resultante de la *llamada a procedimiento que proporciona valor*.

condiciones dinámicas: La *llamada a procedimiento que proporciona valor* no debe entregar un valor indefinido (véanse los apartados 5.3.1 y 6.8).

ejemplos:

6.51 julian_day_number([10,dec,1979]) *(1.1)*
11.68 ok_bishop(b,m) *(1.1)*

5.2.16 *LLAMADA A UNA RUTINA PREDEFINIDA QUE PROPORCIONA VALOR*

sintaxis:

<llamada a una rutina predefinida que proporciona valor> ::= (1)
<llamada a una rutina predefinida que proporciona valor de realización> (1.1)
| *<llamada a una rutina predefinida que proporciona valor CHILL>*

<llamada a una rutina predefinida que proporciona valor CHILL> ::= (2)
NUM(<expresión discreta>) (2.1)
| *PRED(<expresión discreta>)* (2.2)
| *PRED(<expresión de referencia ligada>)* (2.3)
| *SUCC(<expresión discreta>)* (2.4)
| *SUCC(<expresión de referencia ligada>)* (2.5)
| *ABS(<expresión entera>)* (2.6)
| *ADDR(<localización>)* (2.7)
| *CARD(<expresión conjuntista>)* (2.8)
| *MAX(<expresión conjuntista>)* (2.9)
| *MIN(<expresión conjuntista>)* (2.10)
| *SIZE({<nombre de modo> | <localización de modo estático>})* (2.11)
| *UPPER({<expresión matricial> | <expresión cadena>})* (2.12)
| *GETSTACK(<argumento de getstack>)* (2.13)

<argumento de getstack> ::= (3)
<nombre de modo> (3.1)
| *<nombre de modo matriz>(<expresión>)* (3.2)
| *<nombre de modo cadena>(<expresión entera>)* (3.3)
| *<nombre de modo estructura variable>(<lista de expresiones>)* (3.4)

sintaxis derivada: La sintaxis derivada de -> *<localización>* es *ADDR(<localización>)*

semántica: Una llamada a una rutina predefinida que proporciona valor, puede ser una llamada a una rutina predefinida en la realización que proporciona valor o una llamada a una rutina predefinida en CHILL que proporciona valor. Una llamada a una rutina predefinida en CHILL que proporciona valor es una invocación a una de las rutinas predefinidas en CHILL que proporciona valor.

NUM entrega un valor entero con la misma representación interna que el valor suministrado por el argumento discreto. Para valores de conjunto, *NUM* proporciona el valor entero especificado por el modo conjunto. Para valores de carácter, *NUM* proporciona el valor entero especificado en el alfabeto N.º 5 del CCITT (véase el apéndice A1). *NUM (TRUE)* proporciona 1, *NUM (FALSE)* proporciona 0. Para valores enteros, *NUM* suministra dicho valor entero.

Para valores discretos, *PRED* y *SUCC*, proporcionan, respectivamente, los valores discretos inferior y superior siguientes si existen. En cualquier otro caso, se produce una excepción. Si el valor discreto es un valor conjunto de un modo conjunto con huecos, se saltan los huecos (es decir en el ejemplo de propiedades estáticas del apartado 3.4.5, *SUCC (A)* proporciona B, *PRED (B)* proporciona A).

Para valores de referencia ligada, *PRED* y *SUCC* se definen solamente en relación con valores de referencia que se refieren a elementos matriciales. Entregan, respectivamente, el valor de referencia relativo al elemento matricial con los índices siguientes, superior e inferior, si existen.

ABS se define para valores enteros y proporciona el valor absoluto del entero.

ADDR, es una notación alternativa para referenciar una localización.

CARD, *MAX* y *MIN* se definen para valores conjuntistas. *CARD* proporciona el número de valores elemento del valor conjuntista. *MAX* y *MIN* proporcionan, respectivamente, los valores elemento máximo y mínimo del valor conjuntista. *SIZE* se define para los modos y localizaciones de modo estático referenciable. En el primer caso, proporcionan el número de unidades de memoria direccionables ocupadas por esa localización; en el segundo, el número de unidades de memoria direccionables que ocupará una localización referenciable del modo. En el primer caso no se evaluará en la ejecución la *localización de modo estático*. *UPPER*, se define para valores matriciales (posiblemente dinámicos) y valores cadena y proporciona el índice superior del valor matriz o el índice máximo de la cadena en el caso de valor cadena (esto es, la longitud de la cadena menos 1).

GETSTACK genera una localización del modo especificado en la pila (véase el apartado 7.4) y proporciona un valor de referencia para la localización originada. Si se especifica un nombre de modo, se crea una localización de modo estático de dicho modo y se proporciona un valor de referencia ligada. En cualquier otro caso, se genera una localización de modo dinámico cuyo modo es del tipo parametrizado con parámetros de ejecución especificados en el argumento de *GETSTACK*, entregándose un valor descriptor referido a la localización.

propiedades estáticas: La clase de la llamada a la rutina *NUM* predefinida es la clase *INT*-derivada. La llamada a la rutina predefinida, es constante (literal), única y exclusivamente si lo es el argumento.

La clase de la llamada a la rutina predefinida *PRED* o *SUCC*, es la de su argumento. La llamada a esta rutina predefinida es constante (literal), única y exclusivamente si lo es su argumento.

La clase de la llamada a la rutina predefinida *ABS*, es la de su argumento. La llamada a esta rutina predefinida, es constante (literal) única y exclusivamente si lo es su argumento.

La clase de la llamada a la rutina predefinida *CARD*, es la clase *INT*-derivada. La llamada a esta rutina predefinida es constante única y exclusivamente si lo es su argumento.

La clase de la llamada a la rutina predefinida *MAX* o *MIN*, es la clase M-valuada donde M es el modo miembro del modo de la *expresión conjuntista*. La llamada a esta rutina predefinida es constante única y exclusivamente si lo es su argumento.

La clase de la llamada a la rutina predefinida *SIZE*, es la clase *INT*-derivada. La llamada a esta rutina predefinida es constante.

Si el argumento de la llamada a una rutina predefinida *UPPER*, es una *expresión matricial*, la clase de dicha llamada a rutina es la clase M-valuada, donde M es el modo índice del modo matriz de la *expresión matricial* (fuerte). Si el argumento de la llamada a rutina predefinida *UPPER*, es una *expresión cadena*, la clase de dicha llamada es la *INT*-derivada. Una llamada a la rutina predefinida *UPPER* es constante y literal, única y exclusivamente si la clase de la *expresión matricial* o de la *expresión cadena* es una clase estática.

La clase de la llamada a la rutina predefinida *GETSTACK*, es la clase M-referenciada donde M es, o bien el *nombre de modo* o un modo parametrizado dinámico formado por:
&<nombre de modo matriz>(<expresión>) , o
&<nombre de modo cadena>(<expresión entera>) , o
&<nombre de modo estructura variable>(<lista de expresión>) ,
respectivamente, según sea el *argumento getstack*.

condiciones estáticas: Si el argumento de la llamada a la rutina predefinida *PRED* o *SUCC* es constante, no debe entregar los valores discretos mínimo o máximo definidos por el modo raíz de la clase del argumento, respectivamente.

Si el argumento de la llamada a la rutina predefinida *MAX* o *MIN* es constante, no debe entregar el valor conjuntista vacío.

La *expresión conjuntista* que actúe como argumento de la llamada a una rutina predefinida *CARD*, *MAX* o *MIN* debe ser fuerte.

La *expresión de referencia ligada* que sea argumento de una llamada a una rutina predefinida *PRED* o *SUCC*, debe ser fuerte.

La *expresión matricial* que sea argumento de una llamada a la rutina predefinida *UPPER* debe ser fuerte.

En el caso de un argumento *getstack* que no sea un *nombre de modo* simple, deben cumplirse los siguientes requerimientos de compatibilidad:

> La clase de *expresión*, debe ser compatible con el modo índice del modo del *nombre de modo matriz*.
>
> En la *lista de expresiones*, debe haber tantas expresiones como clases hay en la lista de clases del *nombre de modo estructura variable* y la clase de cada expresión deberá ser compatible con la clase correspondiente de la lista de clases del *nombre de modo estructura variable*.

condiciones dinámicas: Si *PRED* y *SUCC* se aplican a los valores discretos mínimo o máximo definidos por el modo raíz de la clase del argumento, se produce la excepción *OVERFLOW*. Si *PRED* y *SUCC* se aplican a un valor de referencia ligada relativo al elemento matricial de indice

mínimo o máximo, se produce la excepción *RANGEFAIL*. Si la *expresión de referencia ligada* entrega *NULL*, *PRED* y *SUCC* provocan la excepción *EMPTY*.

Si *MAX* y *MIN* se aplican a valores conjuntistas vacíos (es decir que no contienen valores miembros) producen la excepción *EMPTY*.

Si los requisitos de almacenamiento no pueden cumplirse, *GETSTACK* origina la excepción *SPACEFAIL*.

GETSTACK origina la excepción *RANGEFAIL* si en el *argumento getstack*:

- la *expresión* proporciona un valor que no pertenece al conjunto de valores definido por el modo índice del *nombre de modo matriz*;
- la *expresión entera* proporciona un valor negativo o un valor igual o mayor que la longitud del *nombre de modo cadena*;
- cualquier expresión de la *lista de expresiones* para la que la clase correspondiente de la lista de clases del *nombre de modo estructura variable* es una clase M-valuada (es decir, es fuerte), proporciona un valor que no pertenece al conjunto de valores definido por M.

ABS produce una excepción *OVERFLOW*, si el valor resultante rebasa los límites definidos por el modo raíz de la clase del argumento.

ejemplos:

9.11	*MIN(sieve)*	*(2.10)*
11.91	*PRED(col_1)*	*(2.2)*
11.91	*SUCC(col_1)*	*(2.4)*

5.2.17 *EXPRESIONES DE ARRANQUE*

sintaxis:

<expresión de arranque> ::= *(1)*
START <nombre de proceso> ([<lista de parámetros efectivos>]) *(1.1)*

semántica: La evaluación de la expresión de arranque, crea y activa un nuevo proceso cuya definición se indica mediante el nombre de proceso (véase el capítulo 8). La transferencia de parámetros es similar a la transferencia de parámetros de procedimiento; sin embargo, pueden facilitarse parámetros efectivos adicionales con un significado definido en la realización. La expresión de arranque proporciona un valor ejemplo único que identifica el proceso creado.

propiedades estáticas: La clase de una *expresión de arranque* es la clase *INSTANCE*-derivada.

condiciones estáticas: El número de ocurrencias de *parámetro efectivo* en la *lista de parámetros efectivos*, no debe ser inferior al número de ocurrencias de *parámetro formal* en la *lista de parámetros formales* de la definición del proceso del *nombre de proceso*. Si m es el número de parámetros efectivos y n el de parámetros formales ($m \geq n$), los requerimientos de compatibilidad para los primeros n parámetros efectivos son los mismos que para la transferencia de parámetros de procedimiento (véase el apartado 6.7).

condiciones dinámicas: La *expresión de arranque*, puede producir una excepción definida en la realización cuyo nombre se asocia al *nombre de proceso* (véase el apartado 7.5).

Para la transferencia de parámetros se aplican las condiciones de asignación de cualquier valor efectivo con relación al modo de su parámetro formal asociado (véase el apartado 6.7).

La *expresión de arranque* provoca la excepción *SPACEFAIL* si no pueden cumplirse los requerimientos de almacenamiento.

ejemplos:

15.25 START COUNTER( ) *(1.1)*

5.2.18 *EXPRESIONES RECIBIR*

sintaxis:

<expresión recibir> ::= *(1)*
RECEIVE <localización tampón> *(1.1)*

semántica: La expresión recibir proporciona un valor extraído del tampón especificado o de cualquier proceso demorado en una acción enviar. Si se ejecuta la expresión recibir, cuando el tampón no contiene ningún valor o no hay espera en ningún proceso ejecutante de una acción enviar, se demora el proceso ejecutante hasta que se envíe un valor al tampón (véase, para más detalle, el capítulo 8).

propiedades estáticas: La clase de la *expresión recibir*, es la clase M-valuada donde M es el modo elemento tampón del modo de la *localización tampón*.

condiciones dinámicas: El tiempo de vida de la localización tampón designada, no debe terminar en tanto el proceso ejecutante está en espera en dicha posición.

ejemplos:

16.49 RECEIVE USER_BUFFER *(1.1)*

5.2.19 *OPERADOR CERO-ADICO*

sintaxis:

<operador cero-ádico> ::= (1)
THIS (1.1)

semántica: El operador cero-ádico, proporciona el valor ejemplo específico que identifica al proceso que lo ejecuta.

propiedades estáticas: La clase del *operador cero-ádico*, es la clase *INSTANCE*-derivada.

5.3 *VALORES Y EXPRESIONES*

5.3.1 *CONSIDERACIONES GENERALES*

sintaxis:

<valor> ::= (1)
<expresión> (1.1)
| *<valor indefinido>* (1.2)

<valor indefinido> ::= (2)
* (2.1)
| *<nombre de sinónimo indefinido>* (2.2)

semántica: Un valor es, o bien un valor indefinido, o bien un valor (definido en CHILL) proporcionado como resultado de la evaluación de una expresión.

propiedades estáticas: La clase de un *valor* es la clase de la *expresión* o del *valor indefinido*, respectivamente.

La clase del *valor indefinido* es la clase general si el valor indefinido es un *; en cualquier otro caso, esta clase es la del *nombre de sinónimo indefinido*.

Un valor es constante, única y exclusivamente si se trata de un *valor indefinido* o una *expresión* constante.

propiedades dinámicas: Se dice que un valor es indefinido si se designa por el *valor indefinido* o si se indica explícitamente en este texto. Un valor compuesto es indefinido, única y exclusivamente si lo son sus subcomponentes (es decir valores subcadena, valores de elemento y valores de campo).

(Nota: Un *valor* puede designar un valor indefinido, solamente en los siguientes contextos:

- si es un *valor indefinido;*

- si se trata un *contenido de localización* que contenga un valor indefinido;

- si es una *llamada a procedimiento que produce un valor*, que proporcione un valor indefinido;

- si es un *valor subcadena*, un *valor segmento de candena*, un *valor elemento matricial*, un *valor submatriz*, un *valor segmento de matriz* o un *valor campo de estructura* que proporcionan un valor indefinido.

ejemplos:

6.40 `(146_097*c)/4+(1_461*y)/4 +(153+m+c)/5+day+1_721_119` *(1.1)*

5.3.2 *EXPRESIONES*

sintaxis:

<expresión> ::= *(1)*
<operando-1> *(1.1)*
| *<subexpresión>* { *OR* | *XOR* } *<operando-1>* *(1.2)*

<subexpresión> ::= *(2)*
<expresión> *(2.1)*

semántica: El orden de valuación de los términos de una expresión y de sus subtérminos, etc., no está definido por lo que puede considerarse que se evalúan en orden mixto. Necesitan evaluarse solamente hasta el punto en que se determina unívocamente el valor que ha de entregarse. Si la expresión es constante o literal, la evaluación no producirá nunca una excepción.

Si se especifican *OR* o *XOR*, la *subexpresión* y el *operando-1*, proporcionarán:

- valores booleanos, en cuyo caso *OR* y *XOR*, designan los operadores lógicos usuales que proporcionan un valor booleano;

- valores cadena de bits en cuyo caso *OR* y *XOR* designan las operaciones lógicas usuales con cadenas de bits que proporcionan un valor cadena de bits;

- valores conjuntistas, en cuyo caso *OR* designa la unión de ambos valores conjuntistas y *XOR* designa el valor conjuntista que consta de aquellos valores primitivos que pertenecen a uno solo de los valores conjuntistas especificados (por ejemplo *A XOR B = A-B OR B-A*).

propiedades estáticas: La clase de una *expresión* que sea un *operando-1*, es la de dicho *operando-1*. Si se especifica *OR* o *XOR*, la clase de la *expresión*, es la resultante de la clase de la *subexpresión* y del *operando-1*.

Una expresión es constante (literal), única y exclusivamente si, o bien lo es el *operando-1* que es constante (literal), o bien está construida a partir de una *expresión* y un *operando-1*, que son ambos constantes (literales).

condiciones estáticas: Si se especifica *OR* o *XOR*, la clase de la *subexpresión* y la del *operando-1* deben ser compatibles entre sí. Ambas clases deben tener un modo raíz booleano, conjuntista o cadena de bits.

condiciones dinámicas: En el caso de *OR* o *XOR*, se produce una excepción *RANGEFAIL* si uno o ambos operandos tienen una clase dinámica y falla la parte dinámica de la verificación de compatibilidad mencionada anteriormente.

ejemplos:

10.27 i<min (1.1)
10.27 i<min OR i>max (1.2)

5.3.3 *OPERANDO-1*

sintaxis:

<operando-1> ::= (1)
<operando-2> (1.1)
| *<suboperando-1> AND <operando-2>* (1.2)

<suboperando-1> ::= (2)
<operando-1> (2.1)

semántica: Si se especifican *AND*, el *sub operando-1* y el *operando-2* entregan:

- valores booleanos, en cuyo caso *AND* designa la operación lógica usual "y" que proporciona un valor booleano;
- valores cadena de bits, en cuyo caso *AND* designa la operación lógica usual "y" en cadenas de bits y proporciona un valor cadena de bits;
- valores conjuntistas, en cuyo caso *AND* designa la operación intersección de valores conjuntistas, que proporcionan como resultado un valor conjuntista.

propiedades estáticas: Si un *operando-1* es un *operando-2* la clase del *operando-1* es la clase del *operando-2*.

Si se especifica *AND*, la clase del *operando-1* es la clase resultante de las clases del *operando-2* y del *suboperando-1*.

Un *operando-1* es constante (literal) única y exclusivamente si es un *operando-2* constante (literal) o está formado por un *operando-1* y un *operando-2*, que son ambos constantes (literales).

condiciones estáticas: Si se especifica *AND*, la clase del *suboperando*-1 debe ser compatible con la clase del *operando-2*. Ambas clases deben tener un modo raíz booleano, conjuntista o de cadena de bits.

condiciones dinámicas: En el caso *AND*, se produce una excepción *RANGEFAIL* si un operando o los dos tienen una clase dinámica y falla la parte dinámica de la verificación de compatibilidad mencionada anteriormente.

ejemplos:

```
5.11        (a1 OR b1)                          (1.1)
5.11        NOT k2 AND (a1 OR b1)               (1.2)
```

5.3.4 *OPERANDO-2*

sintaxis:

```
<operando-2> ::=                                                    (1)
      <operando-3>                                                  (1.1)
    | <suboperando-2>  <operador-3> <operando-3>                    (1.2)

<suboperando-2>  ::=                                                (2)
      <operando-2>                                                  (2.1)

<operador-3> ::=                                                    (3)
      <operador de relación>                                        (3.1)
    | <operador de pertenencia>                                     (3.2)
    | <operador de inclusión conjuntista>                          (3.3)

<operador de relación> ::=                                          (4)
      = | / = | > | >= | < | <=                                     (4.1)

<operador de pertenencia> ::=                                       (5)
      IN                                                            (5.1)

<operador de inclusión conjuntista> ::=                             (6)
      <= | >= | < | >                                               (6.1)
```

semántica: Entre todos los valores de un modo dado se definen los operadores igualdad (=) y desigualdad (/=). Los restantes operadores de relación (menor que: <, menor o igual que: <= mayor que: > mayor o igual que: >=) se definen entre valores de un modo cadena o discreto dado. Todos los operadores de relación proporcionan como resultado un valor booleano.

El operador de pertenencia se define entre un valor primitivo y un valor conjuntista. El operador da como resultado *TRUE* si el valor miembro pertenece al valor conjuntista especificado; en cualquier otro caso produce *FALSE*.

Los operadores de inclusión conjuntista, se definen entre valores conjuntistas y comprueban si un valor conjuntista está o no contenido en: <=, está contenido propiamente en: <, contiene: >=, contiene propiamente: > a otro valor conjuntista. El operador de inclusión conjuntista proporciona como resultado un valor booleano.

propiedades estáticas: Si un *operando-2* es un *operando-3*, la clase del *operando-2* es la del *operando-3*. Si se especifica un *operador-3*, la clase del *operando-3* es la *BOOL*-derivada.

Un *operando-2* es constante (literal) única y exclusivamente si se trata de un *operando-3* a su vez constante (literal) o se obtiene por un *operando-2* y un *operando-3* que son constantes (literales).

condiciones estáticas: Si se especifica un *operador-3*, deben cumplirse los siguientes requerimientos de compatibilidad entre la clase del *sub operando-2* y la clase del *operando-3:*

- si el *operador-3* es = o /=, ambas clases deben ser compatibles;
- si el *operador-3* es un *operador de relación* diferente del = o /=, ambas clases deben ser compatibles y tener un modo raíz discreto o cadena;
- si el *operador-3* es un *operador de pertenencia*, la clase del *operando-3* debe tener un modo raíz conjuntista y la clase del *sub operando-2* debe ser compatible con el modo miembro del citado modo raíz;
- si el *operador-3*, es un operador de *inclusión conjuntista* ambas clases deben ser compatibles y poseer un modo raíz conjuntista.

condiciones dinámicas: En el caso de un *operador de relación*, se produce una excepción *RANGEFAIL* o *TAGFAIL* si uno o ambos operandos tienen una clase dinámica y falla la parte dinámica de la verificación de compatibilidad mencionada anteriormente. Se produce la excepción *TAGFAIL*, única y exclusivamente si una clase dinámica se basa en un modo estructura parametrizado dinámico.

ejemplos:

10.46 NULL *(1.1)*
10.46 last=NULL *(1.2)*

5.3.5 *OPERANDO-3*

sintaxis:

<operando-3> ::= *(1)*
<operando-4> *(1.1)*
| *<suboperando-3> <operador-4> <operando-4>* *(1.2)*

<suboperando-3> ::= *(2)*
<operando-3> *(2.1)*

<operador-4> ::= (3)
<operador aritmético aditivo> (3.1)
| *<operador de concatenación de cadena>* (3.2)
| *<operador de diferencia conjuntista>>* (3.3)

<operador aritmético aditivo> ::= (4)
\+ | - (4.1)

<operador de concatenación de cadena> ::= (5)
// (5.1)

<operador diferencia conjuntista> ::= (6)
\- (6.1)

semántica: Si el operador-4 es un operador aritmético aditivo, ambos operandos proporcionan valores enteros y el valor entero resultante es la suma (+) o la diferencia (-) de ambos valores.

Si el operador-4 es un operador de concatenación de cadena, ambos operandos proporcionan valores cadena de bits o de caracteres y el valor resultante es la concatenación de estos valores.

Si el operador-4 es un operador de diferencia conjuntista, ambos operandos proporcionan valores conjuntistas y el resultado es el valor conjuntista constituido por los valores miembros que pertenecen al valor proporcionado por el sub operando-3 y que no están en el valor proporcionado por el operando-4.

propiedades estáticas: Si un *operando-3* es un *operando-4*, la clase del primero es la del segundo. Si se especifica un *operador-4*, a partir de él se determina la clase del *operando-3*, como sigue:

- si el *operador-4* es un *operador de concatenación de cadena*, la clase del *operando-3*, en función de las clases del *operando-4* y *suboperando-3:*

  - si ninguna de ellas es fuerte, la clase es la *BIT(n)*-derivada o la *CHAR(n)*-derivada, según que ambos operandos sean cadena de bits o de caracteres, donde *n* es la suma de las longitudes de los modos raíz de ambas clases,

  - en los restantes casos, la clase es la *&nombre(n)*-valuada, donde *&nombre*, es un nombre de sínmodo virtual, sinónimo del modo de uno de los operandos fuertes y *n* representa la suma de la longitud de los modos raíz de ambas clases (esta clase es dinámica si uno o ambos operando tienen una clase dinámica).

- si el *operador-4* es un *operador aritmético aditivo* o un *operador diferencia conjuntista*, la clase del *operando-3* es la resultante de las clases del *operando-4* y del *sub operando-3*.

Un *operando-3* es constante (literal), única y exclusivamente si es o bien un *operando-4* constante (literal), o bien procede de un *operando-3* y un *operando-4* que son constantes (literales).

condiciones estáticas: Si se especifica un *operador-4*, deben cumplirse los siguientes requerimientos de compatibilidad:

- si el *operador-4* es un *operador aritmético aditivo*, las clases de ambos operandos deben ser compatibles entre sí y poseer ambas un modo raíz entero;
- si el *operador-4* es un *operador de concatenación de cadena*, los modos raíz de las clases de ambos operandos deben ser o modos cadena de bits o modos cadena de caracteres y, si ambas son clases valor, sus modos raíz deben tener la misma novedad;
- si el *operador-4* es un *operador de diferencia conjuntista*, las clases de ambos operandos deben ser compatibles entre sí y poseer un modo raíz conjuntista.

condiciones dinámicas: En el caso de un *operando-3* que no sea constante, se produce una excepción *OVERFLOW* si una adición (+) o una sustracción (-) originan un valor no comprendido entre los límites especificados por el modo raíz de la clase del *operando-3*.

ejemplos:

1.5 j *(1.2)*
1.5 i + j *(1.2)*

5.3.6 *OPERANDO-4*

sintaxis:

<operando-4> ::= *(1)*
<operando-5> *(1.1)*
| *<suboperando-4>*
<operador aritmético multiplicativo> <operando-5> *(1.2)*

<suboperando-4> ::= *(2)*
<operando-4> *(2.1)*

<operador aritmético multiplicativo> ::= *(3)*
** | / | MOD | REM* *(3.1)*

semántica: Si se especifica un operador aritmético multiplicativo, los operandos-4 y 5 proporcionan valores enteros que son el producto *(*)*, el cociente (/), módulo *(MOD)* o el resto de la división *(REM)* de ambos valores.

La operación módulo se define de manera que *I MOD J* proporciona el valor entero *K*, $0 \leq K < J$ de forma que existe un valor entero *N* tal que $I = N * J + K$. *J* debe ser mayor que *0*.

La operación resto se define de modo que $X \; REM \; Y = X - (X/Y) * Y$, da el valor *TRUE* para todos los valores enteros *X* e *Y*.

propiedades estáticas: Si el *operando-4* es un *operando-5*, la clase del *operando-4* es la clase del *operando-5*. En cualquier otro caso, la clase del *operando-4* es la clase resultante de las clases del *suboperando-4* y del *operando-5*.

Un *operando-4* es constante (literal), única y exclusivamente si es un *operando-5* constante (literal) o procede de un *operando-4* y un *operando-5* que son constantes (literales).

condiciones estáticas: Si se especifica un *operador aritmético multiplicativo*, las clases del *operando-5* y *suboperando-4*, deben ser compatibles entre sí y poseer un modo raíz entero.

condiciones dinámicas: En el caso de un *operando-4* que no sea constante, se produce una excepción *OVERFLOW*, si una operación de multiplicación (*), división (/), módulo (*MOD*) o resto (*REM*), da lugar a un valor que no pertenece al conjunto de valores definidos por el modo raíz de la clase del *operando-4*, o si dicha operación se efectúa con valores operando para los que el operador no está definido matemáticamente, por ejemplo una división o resto con un *operando-5* que proporcione 0 o una operación módulo con un *operando-5* que proporcione un valor entero no positivo.

ejemplos:

6.15 1_461 (1.1)
*6.15 (4 * d + 3) / 1_461* (1.2)

5.3.7 *OPERANDO-5*

sintaxis:

<operando-5> ::= (1)
[<operador monádico>] <operando-6> (1.1)

<operador monádico> ::= (2)
- | NOT (2.1)
| <operador repetición de cadena> (2.2)

<operador repetición de cadena> ::= (3)
(<expresión literal entera>) (3.1)

semántica: Si el operador monádico es el operador cambio de signo (-), el operando-6 proporciona un valor entero y el entero resultante es el valor entero previo con su signo cambiado.

Si el operador monádico es *NOT*, el operando-6 proporciona un valor booleano, un valor cadena de bits o un valor conjuntista. En los dos primeros casos, el resultado es la negación lógica del valor booleano o el valor cadena de bits; en el último es el valor complementario de conjunto, es decir el conjunto de valores miembro que no pertenecen al valor conjuntista del operando.

Si el operador monádico es un operador repetición de cadena, el operando es un *literal de cadena de caracteres* o un *literal de cadena de bits*. Si la *expresión literal entera* produce 0, el resultado es el valor cadena vacío; en cualquier otro caso, el resultado es el valor cadena formado mediante la concatenación de la cadena consigo misma tantas veces como especifique el valor proporcionado por la expresión literal menos 1.

propiedades estáticas: Si el *operando-5* es un *operando-6*, la clase del *operando-5* es la del *operando-6*.

Si se especifica un *operador monádico*, la clase del *operando-5* es:

- la clase resultante del *operador-6*, si el operador monádico es - o *NOT*;
- la clase *CHAR(n)*-derivada o *BIT(n)*-derivada (según que el literal sea un *literal de cadena de caracteres* o un *literal de cadena de bits*), si el operador monádico es el *operador repetición de cadena*, donde $n = r * L$, siendo r el valor proporcionado por la expresión literal entera y L, la longitud del literal de cadena.

Un *operando-5* es constante (literal), única y exclusivamente si lo es el *operando-6*.

condiciones estáticas: Si el *operador monádico* es -, la clase del *operando-6* debe tener un modo raíz entero.

Si el *operador monádico* es *NOT*, la clase del *operando-6* debe tener un modo raíz booleano, de cadena de bits o conjuntista.

Si el *operador monádico* es el *operador repetición de cadena*, el *operando-6* debe ser un *literal de cadena de caracteres* o un *literal de cadena de bits*. La *expresión literal entera*, debe proporcionar un valor entero no negativo.

condiciones dinámicas: Si el *operando-5* no es constante, se produce una expedición *OVERFLOW* si la operación inversión de signo (-) origina un valor que no pertenece al conjunto de valores definido por el modo raíz de la clase del *operando-5*.

ejemplos:

```
5.11      NOT k2                                   (1.1)
7.50      (6)' '                                   (1.1)
7.50      (6)                                      (2.2)
```

5.3.8 *OPERANDO-6*

sintaxis:

```
<operando-6> ::=                                   (1)
      <valor primitivo>                            (1.1)
    | <expresión parentizada>                      (1.2)
```

<expresión parentizada> ::= *(2)*
(<expresión>) *(2.1)*

semántica: Un operando-6 es un valor primitivo (véase el apartado 5.2), o una expresión parentizada.

propiedades estáticas: La clase del *operando-6* es la del *valor primitivo* o *expresión parentizada* respectivamente. La clase de la *expresión parentizada* es la de *expresión*.

Un *operando-6* es constante (literal), única y exclusivamente si el *valor primitivo* o la *expresión* es constante (literal), respectivamente.

ejemplos:

1.5 *i* *(1.1)*
5.11 *(a1 OR b1)* *(1.2)*

6.0 ACCIONES

6.1 CONSIDERACIONES GENERALES

sintaxis:

```
<sentencia de acción> ::=                                          (1)
     [<nombre> :] <acción> [<ejecutor>] [<nombre de etiqueta>];    (1.1)

<acción> ::=                                                       (2)
     <acción corchetada>                                           (2.1)
   | <acción de asignación>                                        (2.2)
   | <acción llamar>                                              (2.3)
   | <acción salir>                                               (2.4)
   | <acción retornar>                                            (2.5)
   | <acción resultado>                                           (2.6)
   | <acción dirigirse a>                                         (2.7)
   | <acción predicado>                                           (2.8)
   | <acción vacía>                                               (2.9)
   | <acción arrancar>                                            (2.10)
   | <acción parar>                                               (2.11)
   | <acción poner en espera>                                     (2.12)
   | <acción continuar>                                           (2.13)
   | <acción enviar>                                              (2.14)
   | <acción causa>                                               (2.15)

<acción corchetada> ::=                                            (3)
     <acción si>                                                   (3.1)
   | <acción caso>                                                 (3.2)
   | <acción hacer>                                               (3.3)
   | <módulo>                                                     (3.4)
   | <bloque principio-fin>                                       (3.5)
   | <acción poner en espera y elegir>                            (3.6)
   | <acción recibir y elegir>                                    (3.7)
```

semántica: Las sentencias de acción constituyen la parte algorítmica de un programa CHILL. Toda sentencia de acción puede etiquetarse y aquellas acciones que puedan provocar una excepción pueden tener añadido un ejecutor.

propiedades estáticas: Enfrente de cada *acción* se coloca un *nombre*, seguido por el signo:, y sólo este nombre se define como nombre de etiqueta.

condiciones estáticas: Se dará el *nombre de etiqueta* antes de punto y coma solamente cuando la *acción* sea una *acción corchetada* o, si se especifica un *ejecutor* y únicamente cuando se dé delante de la *acción* un nombre seguido por el signo:. El *nombre de etiqueta*, debe ser igual a este último *nombre*.

6.2 *ACCIÓN DE ASIGNACIÓN*

sintaxis:

```
<acción de asignación> ::=                                              (1)
     <acción de asignación simple>                                      (1.1)
   | <acción de asignación múltiple>                                    (1.2)

<acción de asignación simple> ::=                                       (2)
     <localización> {<símbolo de asignación> | <operador de
                                                asignación>}
     <valor>                                                            (2.1)

<acción de asignación múltiple>                                         (3)
     <localización> {,<localización>} + <símbolo de asignación>
     <valor>                                                            (3.1)

<operador de asignación> ::=                                            (4)
     <operador diádico cerrado> <símbolo de asignación>                 (4.1)

<operador diádico cerrado> ::=                                          (5)
     OR | XOR | AND                                                     (5.1)
   | <operador diferencia conjuntista>                                  (5.2)
   | <operador aritmético aditivo>                                      (5.3)
   | <operador aritmético multiplicativo>                               (5.4)

<símbolo de asignación> ::=                                             (6)
     := | =                                                            (6.1)
```

sintaxis derivada: La sintaxis derivada del símbolo :=, es el símbolo =.

semántica: La acción de asignación almacena un valor en una o más localizaciones.

Si se utiliza un símbolo de asignación, el valor proporcionado por el segundo miembro se almacena en la(s) localización(es) especificada(s) en el primer miembro.

Si se emplea un operador de asignación, el valor contenido en la localización se combina con el valor del segundo miembro (en ese orden) de acuerdo con la semántica del operador diádico cerrado especificado y el resultado se almacena nuevamente en la misma localización.

La evaluación de las localizaciones de los primeros miembros, de los valores de segundos miembros y de las asignaciones en sí mismas se verifican en un orden no especificado y posiblemente mixto. Puede efectuarse toda asignación tan pronto como se han evaluado el valor y la localización.

Si la localización (o alguna de las localizaciones) es el campo marcador de una estructura variable, los campos variables que dependen de él recibirán un valor indefinido.

condiciones estáticas: Los modos de todas las ocurrencias de *localización* deben ser equivalentes y no deben tener ni la propiedad de sólo lectura ni la de sincronización. Cada modo debe ser compatible con la clase del *valor*. Las verificaciones son dinámicas cuando afecten a localizaciones de modo dinámico y/o a valores con clase dinámica.

Si el *valor* es una *expresión regional* (véase el apartado 8.2.2), cada *localización* debe ser regional.

Si se especifica un *operador de asignación* en una *acción de asignación simple*, el *valor* especificado debe ser una *expresión*.

condiciones dinámicas: Se produce la excepción *TAGFAIL*, si falla la parte dinámica de la verificación de compatibilidad mencionada anteriormente, en caso de una localización y/o un valor de modo estructura dinámica parametrizada.

Se produce la excepción *RANGEFAIL*, si una *localización* tiene un modo intervalo y el *valor* proporcionado por la evaluación de *valor*, está fuera de los límites especificados por dicho modo intervalo.

Se produce la excepción *RANGEFAIL* si, en el caso de una localización y/o un valor de modo cadena dinámica parametrizada o de modo matriz, falla la parte dinámica de la verificación de compatibilidad mencionada anteriormente.

Las condiciones anteriores se denominan condiciones de asignación de un valor con relación a un modo (que es el modo de la localización).

En el caso de un *operador de asignación* se producen las mismas excepciones que en caso de evaluación de la expresión:
<localización> <operador diádico cerrado> (<expresión>)
y de almacenamiento del valor en la localización especificada (obsérvese que la localización se evalúa una vez solamente).

ejemplos:

```
4.11    a:=b+c                                  (1.1)
10.21   stackindex-:=1                          (2.1)
19.16   X.PREX, X.NEXT := NULL                  (3.1)
10.21   -:=                                     (4.1)
```

6.3 ACCIÓN SI

sintaxis:

```
<acción si> ::=                                 (1)
    IF <expresión booleana> <cláusula entonces>
    [<cláusula sino>] FI                        (1.1)
```

```
<cláusula entonces> ::=                                          (2)
     THEN <lista de sentencias de acción>                        (2.1)

<cláusula sino> ::=                                              (3)
     ELSE <lista de sentencias de acción>                        (3.1)
   | ELSIF <expresión booleana>
     <cláusula entonces> [<cláusula sino>]                       (3.2)
```

sintaxis derivada: La notación:

ELSIF <expresión booleana> <cláusula entonces> [<cláusula sino>]

es la sintaxis derivada de:

ELSE IF <expresión booleana> <cláusula entonces> [<cláusula sino>] FI;

semántica: La acción si constituye una ramificación condicional de dos salidas. Si la *expresión booleana* produce *TRUE*, se produce la entrada de la lista de sentencias de acción que sigue a *THEN*; en cualquier otro caso la lista de sentencias de acción que sigue a *ELSE*, si existe.

ejemplos:

```
7.24     IF n >= 10 THEN rn(r):='X';
                n-:=10;
                r+:=1;
         FI                                                      (1.1)

10.46    IF last = NULL
            THEN first, last:=p;
            ELSE last->.succ:=p;
            p->.pred:=last;
            last:=p;
         FI                                                      (1.1)
```

6.4 ACCIÓN CASO

sintaxis:

```
<acción caso> ::=                                                (1)
     CASE <lista de selectores de caso> OF [<lista de intervalo>;]
     {<alternativa de caso>}+
     [ELSE <lista de sentencias de acción>]
     ESAC                                                        (1.1)

<lista de selectores de caso> ::=                                (2)
     <expresión discreta> {,<expresión discreta>}*               (2.1)

<lista de intervalo> ::=                                         (3)
     <modo discreto> {,<modo discreto>}*                         (3.1)

<alternativa de caso> ::=                                        (4)
     <especificación de etiqueta de caso> : <lista de sentencias
                                              de acción>         (4.1)
```

semántica: La acción caso constituye una ramificación múltiple. Consta de la especificación de una o más expresiones discretas (lista de selectores de caso) y un cierto número de listas de sentencias de acción

etiquetadas (alternativas de caso). Cada lista de sentencias de acción, se etiqueta con una especificación de etiqueta de caso compuesta de una lista de especificaciones de etiqueta de caso (una para cada selector de caso). Cada etiqueta de caso define un conjunto de valores. La utilización de una lista de expresiones discretas en la lista de selectores del caso, permite seleccionar una alternativa basada en múltiples condiciones.

La acción de caso produce la entrada de la lista de sentencias de caso, cuyos valores dados en la especificación de etiqueta de caso corresponden a los valores de la lista de selectores de caso.

Las expresiones de la lista de selectores de caso, se evalúan en un orden indefinido y posiblemente mixto. Necesitan evaluarse solamente hasta el punto en que se determina unívocamente una alternativa de caso.

condiciones estáticas: Se aplican las condiciones de selección de caso (véase el apartado 9.1.3) para la lista de ocurrencias de *especificación de etiqueta de caso*.

En la *lista de selectores de caso*, el número de ocurrencias de *expresión discreta*, debe ser igual al número de clases de la lista resultante de clases de la lista de ocurrencias de *lista de etiquetas de caso* y, si existe, al número de ocurrencias de *modo discreto* de la *lista de intervalo*.

La clase de cualquier *expresión discreta* de la *lista de selectores de caso* debe ser compatible con la clase correspondiente (en posición) de la lista resultante de clases de las ocurrencias de *lista de etiquetas de caso* y, si está presente, compatible con el *modo discreto* correspondiente (en posición) de la *lista de intervalo*. Dicho modo debe ser asimismo compatible con la clase correspondiente de la lista de clases resultantes.

Todo valor proporcionado por una *expresión literal discreta*, o definido por un *intervalo literal* o un *modo discreto* en una *etiqueta de caso* (véase el apartado 9.1.3) debe estar en el intervalo del *modo discreto* correspondiente de la *lista de intervalo*, si existe, y asimismo en el intervalo definido por el modo de la *expresión discreta* correspondiente de la *lista de selectores de caso* si se trata de una *expresión discreta fuerte*. En este último caso, los valores definidos por el *modo discreto* correspondiente de la *lista de intervalo*, si existe, deben estar también incluidos en dicho intervalo.

La palabra clave facultativa *ELSE*, seguida de una *lista de sentencias de acción* sólo puede omitirse si la lista de ocurrencias de *lista de etiquetas de caso* es completa (véase el apartado 9.1.3).

condiciones dinámicas: Se produce la excepción *RANGEFAIL* si se especifica una *lista de intervalo* y el valor proporcionado por una *expresión discreta* de la lista de *selectores de caso* no está comprendido entre los límites especificados por el *modo discreto* correspondiente de la *lista de intervalo*.

ejemplos:

```
4.10    CASE order OF
          (1):a:=b+c;
             RETURN;
          (2):d:=0;
          (ELSE):d:=1;
        ESAC                                                    (1.1)
11.44   starting.p.kind, starting.p.color                       (2.1)
11.62   (rook),(*):
            IF NOT ok_rook(b,m)
               THEN
                  CAUSE illegal;
            FI;                                                 (4.1)
```

6.5 ACCIÓN HACER

6.5.1 CONSIDERACIONES GENERALES

sintaxis:

```
<acción hacer> ::=                                              (1)
    DO [<parte de control>;] <lista de sentencias de acción> OD (1.1)

<parte de control> ::=                                          (2)
    <control de iteración> [<control mientras>]                 (2.1)
  | <control mientras>                                          (2.2)
  | <componente de simultaneidad>                               (2.3)
```

semántica: La acción hacer tiene tres formas diferentes: las versiones hacer-iteración y hacer-mientras, ambas para formar bucles y versión hacer-simultaneidad que es una forma abreviada conveniente para acceder a campos de estructura de un modo eficaz. Si no se especifica una parte de control se produce una sola vez la entrada de la lista de sentencias de acción cada vez que se entra en la acción hacer.

Cuando se combinan las versiones hacer-iteración y hacer-mientras, el control mientras se evalúa después del control de iteración y solamente en el caso en que la acción hacer no se termine con el control de iteración.

condiciones dinámicas: Se produce la excepción ***SPACEFAIL***, si no pueden cumplirse los requisitos de almacenamiento.

ejemplos:

```
4.16    DO FOR i:=1 TO c;
           op(a,b,d,order-1);
           d:=a;
        OD                                                      (1.1)
15.48   DO WITH EACH;
           IF THIS_COUNTER = COUNTER
              THEN
```

```
            STATUS:=IDLE;
            EXIT FIND_COUNTER;
        FI;
    OD                                                          (1.1)
```

6.5.2 *CONTROL DE ITERACIÓN*

sintaxis:

```
<control de iteración> ::=                                      (1)
    FOR {<iteración> {,<iteración>}* | EVER}                    (1.1)

<iteración> ::=                                                 (2)
    <enumeración de valor>                                      (2.1)
  | <enumeración de localización>                                (2.2)

<enumeración de valor> ::=                                      (3)
    <enumeración por pasos>                                     (3.1)
  | <enumeración por intervalo>                                 (3.2)
  | <enumeración conjuntista>                                   (3.3)

<enumeración por pasos> ::=                                     (4)
    <contador de bucle> <símbolo de asignación>
    <valor de arranque> [<valor del paso>] [DOWN] <valor final> (4.1)

<contador de bucle> ::=                                         (5)
    <nombre>                                                    (5.1)

<valor inicial> ::=                                             (6)
    <expresión>                                                 (6.1)

<valor del paso> ::=                                            (7)
    BY <expresión entera>                                       (7.1)

<valor final> ::=                                               (8)
    TO <expresión>                                              (8.1)

<enumeración por intervalo> ::=                                 (9)
    <contador de bucle> [DOWN] IN <modo discreto>               (9.1)

<enumeración conjuntista> ::=                                   (10)
    <contador de bucle> [DOWN] IN <expresión conjuntista>       (10.1)

<enumeración de localización> ::=                               (11)
    <contador de bucle> [DOWN] IN <localización matriz>         (11.1)
```

semántica: De acuerdo con el control de iteración especificado, se produce la entrada repetida de la lista de sentencias de acción.

El control de iteración puede mencionar varios contadores de bucle. Los contadores de bucle, se evalúan cada vez en un orden no especificado, antes de ejecutar la lista de sentencias de acción; esta evaluación es necesaria solamente hasta el punto en que pueda

decidirse la terminación de la acción hacer. Dicha acción hacer se termina si al menos uno de los contadores de bucle indica la terminación.

Se distingue entre terminación normal o anormal. La terminación normal tiene lugar cuando la evaluación de uno de los contadores de bucle, por lo menos, indica terminación. Se produce la terminación anormal si la evaluación de una condición mientras proporciona *FALSE*, si se ejecuta una acción salir o dirigirse a con una etiqueta (destino) definida fuera de la lista de sentencias de acción, o si se produce una excepción cuyo ejecutor correspondiente es exterior y no está añadido a la acción hacer.

1. hacer permanente:

   Se repite indefinidamente la lista de acción; solamente es posible la terminación anormal.

2. enumeración de valores:

   Se produce repetidamente la entrada de la lista de sentencias de acción para el conjunto de valores especificados de los contadores de bucle. Dicho conjunto de valores se especifica ya sea mediante un modo discreto (enumeración por intervalo), o mediante un valor conjuntista (enumeración conjuntista), o mediante un valor inicial, un valor del paso y un valor final (enumeración por pasos).

   El contador de bucle siempre está definido implícitamente dentro de la lista de sentencias de acción. Sin embargo, si un nombre de acceso igual al nombre del contador de bucle es visible fuera de la acción hacer, el valor del contador de bucle se almacenará en la localización designada inmediatamente antes de la terminación anormal. En caso de terminación normal el valor almacenado en la localización designada por el nombre de acceso externo es indefinido.

   enumeración por intervalo:

   En caso de enumeración por intervalo, sin (con) la especificación *DOWN*, el valor inicial del contador de bucle es el mínimo (máximo) del conjunto de valores definido por el modo discreto. Para las ejecuciones subsiguientes de la lista de sentencias de acción, se evaluará el *"valor siguiente"* así:
   SUCC ("valor previo") (PRED ("valor previo")).
   Se termina la acción hacer (terminación normal) si se ha ejecutado la lista de sentencias de acción para el valor máximo (mínimo) definido por el modo discreto.

enumeración conjuntista:

En caso de enumeración conjuntista sin (con) la especificación *DOWN*, el valor inicial del contador de bucle es el valor miembro mínimo (máximo) del valor conjuntista designado. Si el valor conjuntista es vacío, no se ejecutará la lista de sentencias de acción. Para las ejecuciones subsiguientes de la lista de sentencias de acción, el valor siguiente será el siguiente mayor (menor) valor miembro del valor conjuntista. Se termina la acción hacer (terminación normal) cuando se ha ejecutado la lista de sentencias de acción para el valor máximo (mínimo). Cuando se ejecuta la acción hacer, se evalúa solamente una vez la expresión conjuntista.

enumeración por pasos:

En caso de una enumeración por pasos sin (con) la especificación *DOWN*, se determina el conjunto de valores del contador de bucle mediante un valor inicial, un valor final y posiblemente un valor del paso. Cuando se ejecuta la acción hacer, estas expresiones se evalúan una sola vez en un orden no especificado posiblemente mixto. El valor del paso es siempre positivo. Antes de ejecutar la lista de sentencias de acción, se comprueba la terminación. Inicialmente se verifica si el valor inicial del contador de bucle es superior (inferior) al valor final. Para las ejecuciones subsiguientes, el *"valor siguiente"* se evalúa así:
"valor previo" + valor del paso
("valor previo" - valor del paso)
en caso de una especificación del *valor del paso*. En otro caso, así:
SUCC ("valor previo") (PRED ("valor previo")).
Se termina la acción hacer (terminación normal) si la evaluación proporciona un valor mayor (menor) que el valor final o pudiera provocar una excepción *OVERFLOW*.

3. enumeración de localización:

En caso de una enumeración de localización sin (con) la especificación *DOWN*, se produce la entrada de la lista de sentencias de acción repetidamente, para el conjunto de localizaciones especificadas que son elementos de la localización matriz designada por la *localización matriz*. La semántica es equivalente a la que existiría si se encontrara inicialmente la declaración de identidad-loc:

DCL <contador de bucle> <modo> LOC := <primera localización>;
donde *<modo>*, es el modo elemento del modo de *localización matriz* y la *<primera localización>*, el elemento de índice mínimo (máximo). Para ejecuciones subsiguientes, es la misma que si antes de la ejecución de la lista de sentencias de acción, se encontrase la declaración de identidad-loc:
DCL <contador de bucle> <modo> LOC := <localización siguiente>;
donde *<localización siguiente>*, es el elemento matriz de índice:
"índice siguiente" = SUCC("índice previo")
(PRED (índice previo)).

Se termina la acción hacer (terminación normal), si el contador de bucle inmediatamente antes de la próxima evaluación indica el elemento matricial de índice máximo (mínimo). Cuando se ejecuta la acción hacer, la *localización matriz* se evalúa una sola vez.

propiedades estáticas:

enumeración de valores:

El *contador de bucle* es un nombre de enumeración de valores. Si un nombre está visible en el dominio donde está situada la *acción hacer*, que sea igual al *contador de bucle*, dicho contador es explícito; en cualquier otro caso es implícito.

enumeración por pasos:

La clase de un *contador de bucle* explícito, es la clase M-valuada, donde M es el modo del nombre de acceso externo (véase más adelante: condiciones estáticas).

La clase de un *contador de bucle* implícito, es la clase resultante de la clase del *valor inicial*, *valor del paso*, si existe, y *valor final*.

enumeración por intervalo:

La clase del *contador de bucle*, es la clase M-valuada, donde M es el *modo discreto*.

enumeración conjuntista:

La clase del *contador de bucle*, es la clase M-valuada, donde M es el modo miembro del modo de la *expresión conjuntista* (fuerte).

enumeración de localización:

El *contador de bucle*, es un nombre de enumeración de localización. Su modo, es el modo elemento del modo de la *localización matriz*.

Un nombre de enumeración de localización, es referenciable (por lenguaje), si la organización de elementos del modo de la *localización de matriz* es *NOPACK*.

condiciones estáticas:

enumeración por pasos:

Las clases del *valor inicial*, *valor final* y *valor del paso*, si existen, deben ser compatibles dos a dos. En caso de un *contador de bucle* explícito, el nombre visible externamente debe ser un nombre de acceso. El modo del nombre de acceso externo debe ser compatible con cada una de estas clases y no debe ser un modo de sólo lectura.

enumeración conjuntista, enumeración por intervalo:

En caso de un *contador de bucle* explícito, el nombre visible externamente debe ser un nombre de acceso. El modo del nombre de acceso externo, debe ser compatible con la clase del *contador de bucle*.

La *expresión conjuntista* debe ser fuerte.

condiciones dinámicas: Se produce una excepción *RANGEFAIL*, si el valor proporcionado por *valor de paso*, no es mayor que 0 ó si, en caso de un *contador de bucle* explícito, el valor que ha de almacenarse nuevamente en la localización externa antes de la terminación anormal no esté comprendido entre los límites especificados por el modo de la localización externa. Esta excepción se produce fuera del bloque de la acción hacer.

ejemplos:

```
4.16     FOR i:=1 TO c                                  (1.1)
15.27    FOR EVER                                       (1.1)
4.16     i:=1 TO c                                      (3.1)
9.11     j:=MIN(sieve) BY MIN(sieve) TO max             (3.1)
14.22    I IN INT(1:100)                                (3.2)
```

6.5.3 *CONTROL MIENTRAS*

sintaxis:

```
<control mientras> ::=                                  (1)
     WHILE <expresión booleana>                         (1.1)
```

semántica: Se evalúa la expresión booleana inmediatamente antes de efectuar la entrada de la lista de sentencias de acción (tras la evaluación del control de iteración, si existe). Si el resultado es *TRUE*, se produce la entrada de la lista de sentencias de acción; en cualquier otro caso se termina la acción hacer (terminación anormal).

ejemplos:

```
7.28     WHILE n >= 1                                   (1.1)
```

6.5.4 *COMPONENTE DE SIMULTANEIDAD*

sintaxis:

```
<componente de simultaneidad> ::=                       (1)
     WITH <control de simultaneidad> {,<control de simultaneidad>}*  (1.1)

<control de simultaneidad> ::=                          (2)
     <localización estructura>                          (2.1)
   | <expresión estructura>                             (2.2)
```

Nota: Si la *expresión estructura* es una *localización*, la construcción sintáctica es ambigua y se interpretará como una *localización estructura*.

semántica: Los nombre de campo (visibles) de las localizaciones o valores estructura especificados en cada *control de simultaneidad*, se hacen disponibles como accesos directos a los campos.

Si se especifica una *localización estructura*, se crean implícitamente nombres de acceso que son iguales a los nombres de campo del modo de la *localización estructura* y que designan las sublocalizaciones de ésta.

Si se especifica una *expresión estructura*, se crean implícitamente nombres de valor iguales a los nombres de campo del modo de la *expresión estructura* (fuerte), que designan los subvalores del valor estructura.

Cuando entra la acción hacer, las localizaciones estructura y/o valores estructura especificados, se evalúan una sola vez con la entrada de la acción en un orden no especificado y posiblemente mixto.

propiedades estáticas:

Expresión estructura: Todo nombre disponible en la acción hacer es un nombre de valor do-with. Su clase es la clase M-valuada, donde M es el modo de ese nombre de campo del modo estructura de la *expresión estructura* disponible como nombre de valor do-with.

Localización estructura: Cualquier nombre disponible en la acción hacer, es un nombre de localización do-with. Su modo es el modo del nombre de campo del modo de la *localización estructura* disponible como nombre de localización do-with. Un nombre de localización do-with, es referenciable (en lenguaje), si la organización del campo del nombre de campo asociado es ***NOPACK***.

condiciones estáticas: La *expresión estructura*, debe ser fuerte.

ejemplos:

15.48 WITH EACH *(1.1)*

6.6 ACCIÓN SALIR

sintaxis:

<acción salir> ::= *(1)*
EXIT <nombre de etiqueta> *(1.1)*

semántica: Se utiliza la acción salir, para abandonar una acción corchetada. La acción concluye inmediatamente después de la acción corchetada más interna etiquetada con el nombre de etiqueta.

condiciones estáticas: La *acción salir* debe estar dentro de la sentencia de acción corchetada etiquetada con el *nombre de etiqueta*. Si la *acción salir* está situada dentro de una definición de procedimiento o de proceso, la sentencia de salida de acción de salida de corchetes, debe estar comprendida también dentro de la misma definición de procedimiento o de proceso (es decir, la acción salir no puede utilizarse para abandonar procedimientos o procesos).

No puede añadirse ningún *ejecutor* a una *acción salir*.

ejemplos:

```
15.52    EXIT FIND_COUNTER                                    (1.1)
```

6.7 ACCIÓN LLAMAR

sintaxis:

```
<acción llamar> ::=                                           (1)
     [CALL] {<llamada a un procedimiento> | <llamada a una rutina
                                            predefinida>}     (1.1)

<llamada a un procedimiento> ::=                              (2)
     {<nombre de procedimiento> | <expresión procedimiento>}
     ([<lista de parámetros efectivos>])                      (2.1)

<lista de parámetros efectivos> ::=                           (3)
     <parámetro efectivo> {,<parámetro efectivo>}*            (3.1)

<parámetro efectivo> ::=                                      (4)
     <valor>                                                  (4.1)
   | <localización de modo estático>                          (4.2)
```

sintaxis derivada: La palabra clave *CALL* es optativa. Una *acción llamar* con *CALL*, se deriva de una *acción llamar* sin *CALL*.

semántica: Una acción llamar provoca la llamada al procedimiento general indicado por el valor proporcionado por la expresión procedimiento o al procedimiento indicado por el nombre de procedimiento. Se transfieren al procedimiento los valores y localizaciones efectivos especificados en la lista de parámetros efectivos.

propiedades estáticas: Una *llamada a un procedimiento*, tiene asociadas las siguientes propiedades: una lista de especificaciones de parámetro, posiblemente una especificación de resultado, un conjunto posiblemente vacío de nombres de excepción, una generalidad, una recurrencia, y posiblemente puede ser regional (esto último solamente es posible en caso de un *nombre de procedimiento* véase el apartado 8.2.2). Estas propiedades son consecuencia del *nombre de procedimiento* o de cualquier modo compatible con la clase de la *expresión procedimiento* (en este último caso la generalidad siempre es general).

Una *llamada a un procedimiento*, con una especificación de resultado, es una *llamada a un procedimiento que produce una localización*, única y exclusivamente si se especifica *LOC* en la especificación de resultado; en cualquier otro caso, es una *llamada a un procedimiento que proporciona valor*.

condiciones estáticas: El número de ocurrencias del *parámetro efectivo*, en la *llamada a un procedimiento*, debe ser el mismo que el de sus especificaciones de parámetro. Los requisitos de compatibilidad para el *parámetro efectivo* y la especificación de parámetro correspondiente (en posición) de la *llamada a un procedimiento*, son:

- Si la especificación del parámetro tiene el atributo *IN* (subsidiario), el *parámetro efectivo* debe ser un *valor* de clase compatible con el modo de la especificación del parámetro correspondiente. Este modo no debe tener la propiedad de sincronización. Si la *llamada a un procedimiento* no es regional, el *valor* (efectivo) no debe ser regional (véase el apartado 8.2.2).

- Si la especificación del parámetro tiene el atributo *INOUT* o el *OUT*, el *parámetro efectivo* debe ser una *localización de modo estático*, de modo compatible con la clase M-valuada, donde M es el modo de la especificación de parámetro correspondiente. El modo de la *localización de modo estático* (efectiva), no debe tener ni la propiedad de sólo lectura ni la de sincronización. Si la *llamada a un procedimiento* no es regional, la *localización* (efectiva), no debe ser regional (véase el apartado 8.2.2).

- Si la especificación de parámetro tiene el atributo *INOUT*, el modo de tal especificación debe ser compatible con la clase M-valuada, donde M es el modo de la *localización de modo estático*.

- Si la especificación de parámetro tiene el atributo *LOC*, el *parámetro efectivo* debe ser una *localización de modo estático* referenciable y tal que el modo de la especificación del parámetro sea de lectura compatible con el modo de esta *localización de modo estático* (efectiva), o un *valor* que no es una *localización*, pero cuya clase es compatible con el modo de la especificación del parámetro.

condiciones dinámicas: Una llamada a un procedimiento, puede originar cualquier excepción del conjunto de nombres de excepción asociado. Produce la excepción *EMPTY*, si el resultado de la *expresión procedimiento* es *NULL*; ocasiona la excepción *SPACEFAIL*, si no pueden cumplirse los requerimientos de almacenamiento y provoca la excepción *RECURSEFAIL*, si las llamadas de procedimiento por si mismas son recurrentes (es decir todavía está activa una invocación previa) y si su recursividad es no recursiva.

La transferencia de parámetro puede originar las siguientes excepciones:

- Si la especificación de parámetro tiene los atributos *IN*, *INOUT* o *LOC*, se aplican las condiciones de asignación del valor (efectivo) (contenido posiblemente en una localización efectiva) con relación a

modo de la especificación del parámetro, en el punto de la llamada (véase el punto 6.2), originándose las posibles excepciones antes de llamar al procedimiento.

- Si la especificación del parámetro tiene el atributo *INOUT* u *OUT*, se aplican, en el punto de retorno (véase el apartado 6.2), las condiciones de asignación del valor local del parámetro formal con respecto al modo de la localización (efectiva), produciéndose las posibles excepciones antes del retorno del procedimiento (véase el apartado 6.2).

- Si la especificación del parámetro tiene el atributo *LOC*, y el *parámetro efectivo* es un *valor* distinto de una *localización*, se aplican, en el punto de la llamada, las condiciones de asignación del *valor* (efectivo) con respecto al modo de la especificación del parámetro, produciéndose las posibles excepciones antes de llamar al procedimiento (véase el apartado 6.2).

La *expresión procedimiento*, no debe proporcionar un procedimiento definido dentro de una definición de proceso cuya activación no sea la misma que la activación del proceso que ejecuta la llamada de procedimiento (véase el apartado 8.1) y el tiempo de vida del procedimiento designado no debe haber concluido.

ejemplos:

4.17 op(a,b,d,order-1) *(1.1)*

6.8 ACCIÓN RESULTADO Y ACCIÓN RETORNAR

sintaxis:

<acción retornar> ::= *(1)*
RETURN [*<resultado>*] *(1.1)*

<acción resultado> ::= *(2)*
RESULT <resultado> *(2.1)*

<resultado> ::= *(3)*
<valor> *(3.1)*
| *<localización de modo estático>* *(3.2)*

sintaxis derivada: La *acción retornar* con *resultado*, se deriva de *RESULT <resultado>*; *RETURN*. Si a esta *acción retornar* se le añade un *ejecutor*, éste se considera asociado a la *acción resultado* de la que se deriva aquella.

semántica: La acción resultado permite establecer el resultado entregado por una llamada a un procedimiento. Este resultado puede ser una localización o un valor. La acción retornar produce el retorno de la invocación del procedimiento en cuya definición se ha colocado. Si el procedimiento entrega un resultado, éste viene determinado por la última acción resultado ejecutada. Si no se ha ejecutado ninguna acción resultado, la llamada a un procedimiento proporciona, respectivamente, una localización indefinida o un valor indefinido.

propiedades estáticas: La *acción resultado* y la *acción retornar*, tienen asociado un nombre de procedimiento, que es el nombre de la definición de procedimiento más próxima de su entorno.

condiciones estáticas: La *acción retornar* y la *acción resultado*, deben estar rodeadas textualmente por una definición de procedimiento. Una *acción resultado* sólo puede especificarse si su nombre de procedimiento tiene una especificación de resultado.

A una *acción retornar* (sin resultado) no puede agregársele un *ejecutor*.

Si se especifica *LOC*, en la especificación de resultado del nombre de procedimiento de la *acción resultado*, el *resultado* debe ser una *localización modo estático* tal que el modo de la especificación de resultado y el modo de la *localización de modo estático*, sean de lectura compatible entre sí. Si el nombre de procedimiento de una *acción resultado* no es regional, tampoco debe serlo la *localización de modo estático* del *resultado* (véase el apartado 8.2.2).

Si no se especifica *LOC*, en la especificación de resultado del nombre de procedimiento de la *acción resultado*, el *resultado* puede ser un *valor* de clase compatible con el modo de la especificación de resultado. Si el nombre de procedimiento de una *acción resultado* no es regional, tampoco debe serlo el *valor* del *resultado* (véase el apartado 8.2.2).

condiciones dinámicas: Si no se especifica *LOC* en la especificación de resultado del nombre de procedimiento, se aplican las condiciones de asignación del *valor* en la *acción resultado* con respecto al modo de la especificación de resultado de su nombre de procedimiento.

ejemplos:

```
4.20    RETURN                                  (1.1)
1.5     RESULT i+j                              (2.1)
5.20    c                                       (3.1)
```

6.9 ACCIÓN DIRIGIRSE A

sintaxis:

```
<acción dirigirse a> ::=                        (1)
    GOTO <nombre de etiqueta>                   (1.1)
```

semántica: La acción dirigirse a produce una transferencia de control. La acción finaliza con la sentencia de acción etiquetada con el nombre de etiqueta.

condiciones estáticas: Si la *acción dirigirse a* está situada dentro de una definición de procedimiento o de proceso, la etiqueta indicada por el *nombre de etiqueta*, debe estar también definida dentro de la definición (es decir no es posible salir fuera de una invocación de procedimiento o de proceso).

A una *acción dirigirse a*, no debe agregársele un *ejecutor*.

6.10 ACCIÓN PREDICADO

sintaxis:

<acción predicado> ::= (1)
ASSERT <expresión booleana> (1.1)

semántica: La acción predicado, proporciona un modo de verificar una condición.

condiciones dinámicas: Se produce la excepción *ASSERTFAIL*, si el resultado de la *expresión booleana* es *FALSE*.

ejemplos:

4.6 ASSERT b>0 AND c>0 AND order>0 (1.1)

6.11 ACCIÓN VACÍA

sintaxis:

<acción vacía> ::= (1)
<vacía> (1.1)

<vacía> ::= (2)

semántica: La acción vacía no origina ninguna acción.

condiciones estáticas: A una *acción vacía*, no debe agregársele ningún *ejecutor*.

6.12 ACCIÓN CAUSA

sintaxis:

<acción causa> ::= (1)
CAUSE <nombre de excepción> (1.1)

semántica: La acción causa produce una excepción.

condiciones estáticas: A una *acción causa* no debe agregársele ningún *ejecutor*.

condiciones dinámicas: La *acción causa* produce la excepción cuyo nombre está indicado por *nombre de excepción*.

ejemplos:

4.8 CAUSE wrong_input (1.1)

6.13 ACCIÓN ARRANCAR

sintaxis:

<acción arrancar> ::= (1)
<expresión de arranque> [*SET <localización ejemplo>*] (1.1)

sintaxis derivada: La *acción arrancar* con la opción *SET* es la sintaxis derivada para la acción de asignación simple:
<localización ejemplo> := <expresión de arranque>

semántica: La acción arrancar, evalúa la expresión de arranque (véase el apartado 5.2.17) sin utilizar el valor ejemplo resultante.

ejemplos:

14.37 START CALL_DISTRIBUTOR() (1.1)

6.14 ACCIÓN PARAR

sintaxis:

<acción parar> ::= (1)
STOP (1.1)

semántica: La acción parar termina el proceso ejecutando dicha acción (véase el apartado 8.1).

condiciones estáticas: A una *acción parar* no debe agregársele ningún *ejecutor*.

6.15 ACCIÓN CONTINUAR

sintaxis:

<acción continuar> ::= (1)
CONTINUE <localización suceso> (1.1)

semántica: La acción continuar permite activar el proceso de máxima prioridad demorado en la localización suceso especificada. Si hay más de un proceso de máxima prioridad, se seleccionará un proceso determinado de la máxima prioridad posible, de acuerdo con un algoritmo de ordenación definido en la realización. Si en la localización suceso especificada, no hay ningún proceso demorado, la acción continuar no tiene ningún efecto ulterior (véase el capítulo 8 para detalles adicionales).

ejemplos:

13.23 CONTINUE RESOURCE_FREED (1.1)

6.16 ACCIÓN PONER EN ESPERA

sintaxis:

<*acción poner en espera*> ::= (1)
DELAY <*localización suceso*> [<*prioridad*>] (1.1)

<*prioridad*> ::= (2)
PRIORITY <*expresión literal entera*> (2.1)

semántica: La acción poner en espera origina la demora del proceso que la ejecuta. Puede activarse mediante una acción continuar en la localización suceso especificada. La prioridad expresa la prioridad del proceso puesto en espera dentro del conjunto de procesos demorados en la localización suceso indicada. La prioridad mínima y la subsidiaria es 0 (véase el capítulo 8 para detalles adicionales).

condiciones estáticas: La *expresión literal entera* no debe proporcionar un valor negativo.

condiciones dinámicas: Se produce la excepción *DELAYFAIL*, si el modo de la *localización suceso*, tiene asociada una longitud y el número de procesos demorados en la localización suceso especificada es igual a dicha longitud inmediatamente antes de la evaluación de la localización suceso. Esta excepción se produce antes de la demora del proceso.

El tiempo de vida de la localización de proceso proporcionada no debe haber concluido mientras que el proceso que ejecuta la acción de puesta en espera está en espera en ella.

ejemplos:

13.17 DELAY RESOURCE_FREED (1.1)

6.17 ACCIÓN PONER EN ESPERA Y ELEGIR

sintaxis:

<*acción poner en espera y elegir*> ::= (1)
DELAY CASE [*SET* <*localización ejemplo*>;] [<*prioridad*>;]
{<*alternativa de puesta en espera*>}+
ESAC (1.1)

<*alternativa de puesta en espera*> ::= (2)
(<*lista de sucesos*>) : <*lista de sentencias de acción*> (2.1)

<*lista de sucesos*> ::= (3)
<*localización suceso*> {,<*localización suceso*>}* (3.1)

semántica: La acción esperar, produce la puesta en espera del proceso que la ejecuta. Se activa mediante una acción continuar aplicada a una de las localizaciones de suceso especificadas. En este caso se ejecutará una lista de sentencias de acción etiquetada por la localización suceso sobre la cual se ha desarrollado la acción continuar que ha reactivado el

proceso (véase el capítulo 8 para más detalle). Con anterioridad a la puesta en espera del proceso, se evalúa cada localización suceso y la localización ejemplo, si se ha especificado. Ambas se evalúan en un orden no especificado y posiblemente mixto. Si dos o más evaluaciones conducen a la misma localización suceso, la elección de una lista de sentencias de acción es no-determinística.

Si se especifica una localización ejemplo, se almacenará en dicha localización el valor ejemplo que identifica el proceso que ejecutó la acción continuar activadora.

condiciones estáticas: El modo de la *localización ejemplo*, no debe tener la propiedad de sólo lectura. En *prioridad*, la *expresión literal entera* no debe proporcionar un valor negativo.

condiciones dinámicas: Se produce la excepción *DELAYFAIL*, si el modo de una *localización suceso*, al menos, tiene asociada una longitud tal que el número de procesos en espera en la localización suceso especificada, es igual al valor de la longitud después de la evaluación de la *localización suceso*. Se produce esta excepción antes de la puesta en espera del proceso.

El tiempo de vida de las localizaciones suceso proporcionadas no debe acabar en ningún caso mientras el proceso que ejecuta la acción poner en espera y elegir está en espera en ellas.

ejemplos:

```
14.20   DELAY CASE
          (OPERATOR_IS_READY):/* some actions */
          (SWITCH_IS_CLOSED):DO FOR I IN INT(1:100);
                               CONTINUE OPERATOR_IS_READY;
                               /* empty the queue */
                               OD;
        ESAC                                                    (1.1)
```

6.18 ACCIÓN ENVIAR

6.18.1 CONSIDERACIONES GENERALES

sintaxis:

```
<acción enviar> ::=                                             (1)
    <acción enviar señal>                                       (1.1)
  | <acción enviar tampón>                                      (1.2)
```

semántica: La acción de enviar inicia la transferencia de información de sincronización a partir de un proceso que contenga una acción enviar. La semántica detallada depende de que el objeto de sincronización sea una señal o un tampón.

6.18.2 *ACCIÓN ENVIAR SEÑAL*

sintaxis:

<acción enviar señal> ::= *(1)*
SEND <nombre de señal> [(<valor> {,<valor>})]*
[TO <expresión ejemplo>] [<prioridad>] *(1.1)*

semántica: La señal especificada se envía conjuntamente con la lista de valores y la prioridad (si está presente). La prioridad mínima y la subsidiaria es 0. Si el nombre de señal tiene asociado un nombre de proceso, esto implica que sólo pueden recibir la señal los procesos de ese nombre. Si se especifica la opción *TO*, ésta identifica los únicos procesos que pueden recibir la lista de valores enviados en la acción enviar señal. Esta identificación de procesos no debe estar en contradicción con un posible nombre de proceso asignado al nombre de señal. Tanto el posible nombre de proceso de la señal como el posible valor ejemplo, se asignan dinámicamente a la lista de valores enviada (véase el capítulo 8 para más detalle).

condiciones estáticas: El número de ocurrencias de *valor* debe ser igual al número de modos del *nombre de señal*. La clase de cada *valor* debe ser compatible con el modo correspondiente del *nombre de señal*. Ninguna ocurrencia de *valor* puede ser regional (véase el apartado 8.2.2). La *expresión literal entera* de *prioridad*, no debe proporcionar un valor negativo.

condiciones dinámicas: Se aplican las condiciones de asignación de cada *valor* con respecto a su modo correspondiente del *nombre de señal*.

Se produce la excepción *EMPTY*, si la *expresión ejemplo* da como resultado *NULL*.

Se produce la excepción *EXTINCT*, única y exclusivamente si el tiempo de vida del proceso indicado por el valor proporcionado por la *expresión ejemplo*, ha concluido en el momento de la ejecución de una acción enviar señal.

Se produce la excepción *MODEFAIL*, si el *nombre de señal* tiene asociado un nombre de proceso que no es el indicado por el valor proporcionado por la *expresión ejemplo*.

ejemplos:

15.68 SEND READY TO RECEIVED_USER *(1.1)*
15.76 SEND READOUT(COUNT) TO USER

6.18.3 *ACCIÓN ENVIAR TAMPÓN*

sintaxis:

<acción enviar tampón> ::= *(1)*
SEND <localización tampón> (<valor>) [<prioridad>] *(1.1)*

semántica: El valor especificado, junto con la prioridad, se almacena en la localización tampón si lo permite su capacidad. Esto no ocurre si el modo de la *localización tampón* tiene una longitud asociada tal que el número de valores almacenados en el tampón es igual a dicha longitud inmediatamente antes de ejecutar la acción enviar tampón. En consecuencia, el proceso de envío se pondrá en espera hasta que haya capacidad en la localización tampón o se consuma el valor enviado. La prioridad mínima y la subsidiaria es 0 (véase el capítulo 8 para más detalle).

condiciones estáticas: La clase del *valor*, debe ser compatible con el modo elemento tampón del modo de la *localización tampón*. El *valor* no debe ser regional (véase el punto 8,2.2). La *expresión literal entera* de *prioridad*, no debe proporcionar un valor negativo.

condiciones dinámicas: Para la *acción enviar tampón*, se aplican las condiciones de asignación del *valor* con respecto al modo elemento tampón del modo de la *localización tampón*. Las posibles excepciones se producen antes de puesta en espera del proceso.

El tiempo de vida de la localización tampón entregada no debe finalizar mientras el proceso que ejecuta la acción enviar tampón está en espera en ella.

ejemplos:

```
16.115  SEND USER->([READY, ->COUNTER_BUFFER])                    (1.1)
```

6.19 ACCIÓN RECIBIR Y ELEGIR

6.19.1 CONSIDERACIONES GENERALES

sintaxis:

```
<acción recibir y elegir> ::=                                     (1)
    <acción recibir señal y elegir>                               (1.1)
  | <acción recibir tampón y elegir>                              (1.2)
```

semántica: La acción recibir y elegir recibe información de sincronización transmitida por la acción enviar. La semántica detallada depende del objeto de sincronización utilizado que es una señal o un tampón. La entrada de una acción recibir y elegir no produce necesariamente una puesta en espera del proceso que la ejecuta (véase el capítulo 8 para más detalle).

6.19.2 *ACCIÓN RECIBIR SEÑAL Y ELEGIR*

sintaxis:

<acción recibir señal y elegir> ::= (1)
RECEIVE CASE [*SET <localización ejemplo>*;]
{*<alternativa de recepción de señal>*}+
[*ELSE <lista de sentencias de acción>*] *ESAC* (1.1)

<alternativa de recepción de señal> ::= (2)
(*<nombre de señal>* [*IN <lista de nombres>*])
: *<lista de sentencias de acción>* (2.1)

semántica: La acción recibir señal y elegir recibe una señal, posiblemente con una lista de valores, cuyo nombre de señal se especifica en una alternativa de recepción de señal.

Cuando se produce la entrada de la acción recibir señal y elegir, estando presente una señal perteneciente a uno de los nombres especificados que pueden recibirse por el proceso que ejecuta la acción, se recibe dicha señal. Si tal señal no está presente y no se especifica *ELSE*, se pone en espera el proceso que ejecuta la acción recibir señal y elegir. Si se especifica *ELSE*, entrará la lista de sentencias de acción que la siga.

Mediante un proceso, sólo puede recibirse una señal si se cumplen las siguientes condiciones:

- Si la señal tiene asociado un nombre de proceso, el nombre de proceso receptor es dicho nombre de proceso.

- Si la señal tiene asociado un valor ejemplo, éste identifica el proceso receptor.

Si puede recibirse una señal se producirá la entrada de la lista de sentencias de acción etiquetada con el nombre de señal de la señal recibida. Si puede recibirse más de una señal, se seleccionará la señal de prioridad más alta de acuerdo con un algoritmo de asignación definido en la realización. Si el nombre de señal tiene agregada una lista de modos, es decir se envía con la señal una lista de valores, debe especificarse después de *IN* esta última. Son nombres de valor introducidos que designan los valores recibidos. Si en el dominio en el que se ha colocado la acción de recibir señal y elegir es visible un nombre de acceso igual a un nombre introducido, se almacenará el valor recibido en la localización designada, inmediatamente después de la recepción de señal y antes de la ejecución de la lista de sentencias de acción.

Si se especifica la opción *SET*, se almacenará el valor ejemplo, que indica el proceso que ha enviado la señal recibida, en la localización ejemplo especificada inmediatamente después de la recepción de señal.

propiedades estáticas: Todo nombre definido en la *lista de nombres* de la *alternativa de recepción de señal* es un nombre de valor a recibir. Su clase es la clase M-valuada donde M es el modo correspondiente del *nombre de señal* enfrente de él. Si un nombre es visible en el dominio donde se sitúa una *acción recibir señal y elegir*, y es igual a uno de los nombres introducidos después de *IN*, el nombre de valor a recibir es explícito, de lo contrario es implícito.

condiciones estáticas: El modo de la *localización ejemplo* no debe tener la propiedad de sólo lectura.

Todas las ocurrencias de *nombre de señal* deben ser diferentes.

En la *alternativa de recepción de señal* deben especificarse la *IN* facultativa y la *lista de nombres*, única y exclusivamente si el *nombre de señal* tiene un conjunto de modos no vacío. El número de nombres de la *lista de nombres* debe ser igual al número de modos del *nombre de señal*.

Si el nombre de valor a recibir es explícito, el nombre visible externamente debe ser un *nombre de acceso* de modo compatible con la clase del nombre de valor a recibir. El modo del *nombre de acceso*, no debe tener la propiedad de sólo lectura.

condiciones dinámicas: Si el nombre de valor a recibir es explícito, se aplican las condiciones de asignación del valor recibido con respecto al modo del *nombre de acceso* externo. Las posibles excepciones se producen después de recibir la señal y antes de la entrada de la lista de sentencias de acción.

Se produce la excepción *SPACEFAIL*, si al entrar una lista de sentencias de acción no pueden cumplirse los requisitos de almacenamiento.

ejemplos:

```
15.73   RECEIVE CASE
          (STEP): COUNT +:= 1;
          (TERMINATE):
            SEND READOUT(COUNT) TO USER;
            EXIT WORK_LOOP;
        ESAC                                                    (1.1)
```

6.19.3 ACCIÓN RECIBIR TAMPÓN Y ELEGIR

sintaxis:

```
<acción recibir tampón caso> ::=                                (1)
    RECEIVE CASE [SET <localización ejemplo>;]
    {<alternativa de recepción de tampón>}+
    [ELSE <lista de sentencias de acción>]
    ESAC                                                        (1.1)

<alternativa de recepción de tampón> ::=                        (2)
    (<localización tampón> IN <nombre>)
    <lista de sentencias de acción>                             (2.1)
```

semántica: La acción recibir tampón y elegir recibe un valor de una localización tampón o de un proceso remitente, en espera en una localización tampón, cuya localización se indica en una alternativa de recepción de tampón.

Cuando se produce la entrada de una acción recibir tampón y elegir, estando presente un valor en ella o en espera un proceso remitente, en una de las localizaciones tampón especificadas, se recibirá el valor y se ejecutará una lista de sentenciación etiquetada con la *localización tampón* que proporciona la localización tampón desde la que se ha recibido el valor.

Cuando se produce la entrada de una acción recibir tampón caso se evalúan las localizaciones tampón en un orden no especificado y quizá heterogéneo hasta un punto suficiente para seleccionar una alternativa. Si ninguna de las localizaciones tampón especificadas contienen un valor ni se ha puesto en espera proceso alguno en la localización tampón especificada y tampoco se ha especificado *ELSE*, se pone en espera el proceso ejecutante. Si se especifica *ELSE*, se ejecutará la lista de sentencias de acción que siguen al mismo. Si se recibe más de un valor, se seleccionará el de mayor prioridad de acuerdo con un algoritmo de ordenación definido en la realización. Si dos o más ocurrencias de *localización tampón* proporcionan la misma localización tampón desde la que se ha recibido el valor, la selección de la lista de sentencias de acción es no-determinística.

El valor se recibe inmediatamente antes de la entrada de la lista de sentencias de acción que sigue a la coma. El nombre siguiente a *IN* es un nombre de valor a recibir introducido que designa el valor recibido. Si en el intervalo en que se ha colocado la *acción recibir tampón y elegir* es visible un nombre de acceso igual a un nombre de valor a recibir creado, el valor recibido se almacena en la localización designada, inmediatamente antes de la entrada de la lista de sentencias de acción.

Si se especifica la opción *SET*, la localización ejemplo especificada ha almacenado en ella, inmediatamente después de la recepción, el valor ejemplo que designa el proceso que ha enviado el valor recibido.

propiedades estáticas: El *nombre* que sigue a *IN* en la *alternativa recibir tampón* es un nombre de valor a recibir. Su clase es la clase M-valuada, donde M es el modo elemento tampón del modo de la *localización tampón* que etiqueta la *alternativa de recepción de tampón*.

Si en el dominio en el que se ha situado la *acción recibir tampón y elegir* es visible un nombre igual al *nombre* introducido después de *IN*, el nombre de valor a recibir se llama explícito; de lo contrario es implícito.

condiciones estáticas: El modo de la *localización ejemplo* no debe tener la propiedad de sólo lectura. Si el nombre de valor a recibir es explícito, el modo del nombre visible externamente debe ser un *nombre de acceso* y su modo debe ser compatible con la clase del nombre de valor a recibir del mismo nombre. Este modo no debe tener la propiedad de sólo lectura.

condiciones dinámicas: Si el nombre de valor a recibir es explícito, se aplican las condiciones de asignación del valor recibido con respecto al modo del *nombre de acceso* externo. Las posibles excepciones se producen después de recibir el valor y antes de la entrada de la lista de sentencias de acción.

Se produce una excepción ***SPACEFAIL***, si al producirse la entrada de una lista de sentencias de acción, no pueden cumplirse los requisitos de almacenamiento.

No debe haber finalizado el tiempo de vida de ninguna de las localizaciones tampón proporcionadas mientras esté en espera en ellas el proceso que ejecuta la acción recibir tampón y elegir.

7.0 ESTRUCTURA DEL PROGRAMA

7.1 CONSIDERACIONES GENERALES

Las acciones parentizadas *hacer, bloque principio-fin, módulo, región, poner en espera y elegir, recibir y elegir, definición de procedimiento* y *definición de proceso,* determinan la estructura del programa, es decir, el alcance de los nombres y el tiempo de vida de las localizaciones creadas en ellas.

- Se utilizará la palabra bloque para designar:
  - la *lista de sentencias de acción,* en la *acción hacer* con inclusión del *contador de bucle* y el *control mientras*;
  - el *bloque principio-fin*;
  - la *definición de procedimiento,* excluyendo la *especificación de resultado*;
  - la *definición de proceso*;
  - la *lista de sentencias de acción,* en una *alternativa de recepción de tampón* o en una *alternativa de recepción de señal* con inclusión del *nombre* o *lista de nombres* después de *IN*;
  - la *lista de sentencias de acción,* después de *ELSE* en una *acción recibir caso* o en un *ejecutor*;
  - la *alternativa-on,* en un *ejecutor.*
- Se utilizará la palabra modulión para designar un *módulo* o una *región.*
- La palabra grupo, designará un bloque o un modulión.
- La palabra dominio o dominio de un grupo, designará aquella parte del grupo no rodeada por un grupo más interno del grupo (es decir, la parte consistente en el nivel jerárquico más externo del grupo).

Un grupo define un alcance para los nombres creados en su dominio. Los nombres pueden crearse de las siguientes formas:

- Un nombre que aparezca en la *lista de nombres* de una *declaración,* en una *definición de modo* o en una *definición de sinónimo* o en una *definición de señal* se crea en el dominio en el que están situados, respectivamente, la *declaración, definición de modo, definición de sinónimo* o *definición de señal.*
- Un nombre que aparezca en la *lista de nombres* de una *lista de parámetro formal* se crea en el dominio de la *definición de procedimiento* o *definición de proceso* asociadas.
- Un nombre situado enfrente del signo: seguido por una *acción, región, definición de procedimiento, definición de entrada* o *definición de proceso,* se crea en el dominio donde están colocados, respectivamente, la *acción, región, definición de procedimiento, definición de procedimiento* conteniendo la *definición de entrada,* y *definición de proceso.*

- Cada nombre de enumeración de valor, nombre de enumeración de localización, nombre de valor do-with y de localización do-with se crea en el dominio del bloque de la *acción hacer* asociada.

- Cada nombre de valor a recibir, se crea en el dominio del bloque de la *alternativa de recepción de señal* o *alternativa de recepción de tampón*.

- Un nombre de campo o nombre de elemento de conjunto, se crean en el dominio en el que están colocadas la ocurrencia de definición de su *modo estructura* o *modo conjunto* asociados.

- Un nombre de excepción se crea mediante una *acción causar* o una *alternativa-on* (nota: para un nombre de excepción no se indica ningún punto de creación específico, véase el capítulo 10).

- En el dominio de un módulo preludio normalizado (véase el apartado 7.8), se considera creado un nombre lenguaje predefinido.

Los nombres introducidos (creados) por el programador, salvo los nombres de excepción deben crearse (declararse o definirse) en un lugar determinado. Este lugar se denomina ocurrencia de definición del nombre. Los lugares donde se utiliza el nombre se denominan ocurrencias de aplicación del nombre. Las reglas de identificación asocian unívocamente una ocurrencia de definición con cada ocurrencia de aplicación del nombre (véase el apartado 9.2.8). Para los nombres de excepción (véase el capítulo 10) no se hace ninguna distinción entre las ocurrencias de definición y de aplicación.

Un nombre tiene un cierto alcance, es decir aquella parte del programa donde puede verse su definición o declaración y, en consecuencia, donde puede usarse libremente. Se dice que el nombre es visible en dicha parte. Las localizaciones tienen un cierto tiempo de vida que es la parte del programa donde existen. Los bloques determinan la visibilidad de los nombres y el tiempo de vida de las localizaciones creadas en ellos. Los moduliones determinan la visibilidad solamente; el tiempo de vida de las localizaciones creadas en el dominio de un modulión será el mismo que si hubieran sido creadas en el dominio del primer bloque que le rodea. Los moduliones permiten restringir la visibilidad de los nombres. Por ejemplo, un nombre creado en el dominio de un módulo no será automáticamente visible en módulos externos o internos aunque su tiempo de vida lo permita.

7.2 DOMINIOS Y FASCICULIZACIÓN

sintaxis:

```
<cuerpo principio-fin> ::=                                          (1)
     <lista de sentencias de datos> <lista de sentencias
                                        de acción>                  (1.1)

<cuerpo proc> ::=                                                   (2)
     <lista de sentencias de datos>
     {<sentencia de acción>|<sentencia de entrada>}*                (2.1)

<cuerpo de proceso> ::=                                             (3)
     <lista de sentencias de datos> <lista de sentencias
                                        de acción>                  (3.1)

<cuerpo de módulo> ::=                                              (4)
     {<sentencia de datos>|<sentencia de visibilidad>|
     <región>}* <lista de sentencias de acción>                     (4.1)
```

```
<cuerpo de región> ::=                                          (5)
     {<sentencia de datos>|<sentencia de visibilidad>}*         (5.1)

<lista de sentencias de acción> ::=                             (6)
     {<sentencia de acción>}*                                   (6.1)

<lista de sentencias de datos> ::=                              (7)
     {<sentencia de datos>}*                                    (7.1)

<sentencia de datos> ::=                                        (8)
     <sentencia de declaración>                                 (8.1)
     |<sentencia de definición>                                (8.2)

<sentencia de definición> ::=                                   (9)
     <sentencia de definición de sinmodo>                       (9.1)
     |<sentencia de definición de neomodo>                      (9.2)
     |<sentencia de definición de sinónimo>                     (9.3)
     |<sentencia de definición de procedimiento>                (9.4)
     |<sentencia de definición de proceso>                      (9.5)
     |<sentencia de definición de señal>                        (9.6)
     |<vacía>;                                                  (9.7)
```

semántica: Cuando se produce la entrada del dominio de un bloque se efectúan las inicializaciones ligadas al tiempo de vida de las localizaciones creadas por la entrada del bloque. Subsiguientemente, se realizan las inicializaciones ligadas al dominio en el dominio del bloque y las evaluaciones, posiblemente dinámicas, de las declaraciones de identidad-loc siguiendo orden textual especificado.

Cuando se produce la entrada del dominio de un modulión, se efectúan, en el orden textualmente especificado, las inicializaciones ligadas al dominio y las evaluaciones posiblemente dinámicas de las declaraciones de identidad-loc en el dominio del modulión.

propiedades estáticas: Todo dominio tiene un grupo de cierre directo, definido como sigue:

- Si el dominio es el de una *acción hacer*, un *bloque principio-fin*, una *definición de procedimiento*, una *definición de proceso*, un *módulo* o *región*, su grupo de cierre directo es el grupo en cuyo dominio están situadas la *acción hacer*, el *bloque principio-fin*, la *definición de procedimiento*, la *definición de proceso*, el *módulo* o *región*, respectivamente.

- Si el dominio es la *lista de sentencias de acción* con la posible inclusión de nombres introducidos, de una *alternativa de recepción de tampón*, de una *alternativa de recepción de señal* o la *lista de sentencias de acción* que sigue a *ELSE* en una *acción recibir tampón y elegir*, o una *acción recibir señal y elegir*, su grupo de cierre directo es el grupo en cuyo dominio están situadas la *acción recibir tampón y elegir* o la *acción recibir señal y elegir*.

- Si el dominio es la *lista de sentencias de acción* en una *alternativa-on* o la *lista de sentencias acción* que sigue a *ELSE*, en un *ejecutor* no anexado a un grupo, el grupo de cierre directo es el grupo en cuyo dominio está colocada la sentencia a la cual está añadido el *ejecutor*.

- Si el dominio es una *alternativa-on,* o una *lista de sentencias de acción* situadas después de *ELSE,* en un *programa ejecutor* añadido a un grupo, su grupo de cierre directo es el grupo al cual está añadido dicho *ejecutor.*

Un dominio tiene un dominio de cierre directo único que es el dominio del grupo de cierre directo. Una sentencia tiene un grupo de cierre directo único, que es el grupo donde está situado el dominio de la sentencia. Se dice que un dominio cierra directamente un grupo (dominio), si y sólo si el dominio es el dominio de cierre directo del grupo (dominio).

Se dice que una sentencia (dominio) está rodeada por un grupo, única y exclusivamente si el grupo es el grupo de cierre directo de la sentencia (dominio) o el dominio de cierre directo está rodeado por el grupo.

Se produce la entrada de un dominio cuando:

- Dominio de módulo: se ejecuta el módulo como una acción (por ejemplo no se dice que se entra el módulo cuando una acción dirigirse a transfiere el control a un nombre de etiqueta definido dentro del módulo).

- Dominio principio-fin: se ejecuta el bloque principio-fin como una acción.

- Dominio de región: se encuentra la región (es decir, no se dice que se entra la región cuando se llama a uno de sus procedimientos críticos).

- Dominio de procedimiento: se produce la entrada del procedimiento mediante su entrada principal (es decir, no se utiliza un punto de entrada definido adicionalmente).

- Dominio de proceso: se activa el proceso mediante una sentencia de arranque.

- Dominio hacer: se ejecuta la acción hacer como una acción, posterior a la evaluación de las expresiones o posiciones en la parte de control.

- Dominio alternativa de recepción de tampón, dominio alternativa de recepción de señal: se ejecuta la alternativa al recibir el valor tampón o la señal.

- Dominio alternativa-on: se ejecuta la alternativa-on en la causa de una excepción.

Se produce la entrada de una sentencia de acción, solamente en el caso en que la primera acción, si está presente, reciba el control desde fuera de la lista de sentencias de acción.

7.3 BLOQUES PRINCIPIO-FIN

sintaxis:

```
<bloque principio-fin> ::=                                    (1)
     BEGIN <cuerpo principio-fin> END                         (1.1)
```

semántica: Un bloque principio-fin es una acción (acción compuesta) que contiene, posiblemente, declaraciones y definiciones locales. Determina la visibilidad de los nombres creados localmente y el tiempo de vida de las localizaciones creadas localmente (véanse los apartados 7.9 y 9.2.5).

condiciones dinámicas: Se produce una excepción *SPACEFAIL*, si el *bloque principio-fin*, requiere un almacenamiento local para el cual no pueden cumplirse los requisitos de almacenamiento.

ejemplos:

see 15.63 - 15.80

7.4 DEFINICIONES DE PROCEDIMIENTO

sintaxis:

```
<sentencia de definición de procedimiento> ::=                (1)
     <nombre> : <definición de procedimiento>
     /<ejecutor>/ /nombre de procedimiento>/;                 (1.1)

<definición de procedimiento> ::=                             (2)
      PROC (/<lista de parámetros formales>/) /<especifi-
                                      cación de resultado>/
     /EXCEPTIONS (<lista de excepciones>)/ <atributos de
                                          procedimiento>;
     <cuerpo proc> END                                        (2.1)

<lista de parámetros formales> ::=                            (3)
     <parámetro formal>{<parámetro formal>}*                  (3.1)

<parámetro formal> ::=                                        (4)
     <lista de nombres> <especificación de parámetro>         (4.1)

<atributos de procedimiento> ::=                              (5)
     /<generalidad>/ /RECURSIVE/                              (5.1)

<generalidad> ::=                                             (6)
      GENERAL                                                 (6.1)
     |SIMPLE                                                  (6.2)
     |INLINE                                                  (6.3)

<sentencia de entrada> ::=                                    (7)
     <nombre> : <definición de entrada>                       (7.1)

<definición de entrada> ::=                                   (8)
      ENTRY                                                   (8.1)
```

sintaxis derivada: Un *parámetro formal*, cuya *lista de nombres* contiene más de un nombre se deriva de varias ocurrencias de *parámetros formales*

separadas por comas, una para cada nombre, cada uno de los cuales tiene la misma *especificación de parámetro*. Por ejemplo: *I, J INT LOC* se deriva de *I INT LOC, J INT LOC*.

semántica: Una definición de procedimiento define una secuencia (posiblemente) parametrizada de acciones que pueden llamarse desde diferentes lugares de un programa. Se devuelve el control al punto de llamada ejecutando una acción retorno o al alcanzar el final del cuerpo del procedimiento o de una alternativa-on de un ejecutor añadido a la definición de procedimiento (caída directa). Pueden especificarse distintos grados de complejidad del procedimiento; como sigue:

Procedimientos simples *(SIMPLE)*, son procedimientos que no pueden manipularse dinámicamente. No se tratan como valores, es decir no pueden almacenarse en una localización procedimiento ni transferirse como parámetros a una llamada a un procedimiento u obtenerse como resultados de ésta.

Procedimientos generales *(GENERAL)*, no tienen las restricciones de los procedimientos simples, pudiendo tratarse como valores procedimiento.

Procedimientos en línea *(INLINE)*. Tienen las mismas restricciones que los procedimientos simples y no pueden ser recurrentes. Poseen la misma semántica que los procedimientos normales, pero el compilador insertará el código objeto generado en el punto de invocación en lugar de generar el código para llamar realmente al procedimiento.

Solamente pueden especificarse como (mutuamente) recurrentes los procedimientos simples y generales. Cuando no se especifican atributos de procedimiento se aplicará un valor subsidiario definido en la realización.

Un procedimiento puede suministrar un valor o una localización (indicada pero el atributo *LOC* en la especificación de resultado).

El nombre situado enfrente de la definición del procedimiento establece el nombre de dicho procedimiento. Si el nombre de procedimiento es general, es un procedimiento literal para el valor procedimiento definido. Su clase se determina por los modos y atributos de la lista de parámetros formales y la especificación de resultado.

Un procedimiento puede tener múltiples puntos de entrada establecidos mediante sentencias de entrada. Dichas sentencias se consideran definiciones de procedimiento adicionales. El nombre en la sentencia de entrada define el nombre del punto de entrada en el procedimiento en cuyo dominio está situado. El punto de entrada se determina por la posición textual de la sentencia de entrada.

transferencia de parámetros:

Hay básicamente dos procedimientos de transferencia de parámetros: el paso por valor y el paso por localización (atributo *LOC)*. Los atributos *OUT* e *INOUT* constituyen variaciones del sistema de paso por valor.

Paso por valor

En la transferencia de parámetros mediante paso por valor se pasa un valor en forma de parámetro a un procedimiento y se almacena en la localización local del modo parámetro especificado. El efecto es el mismo que si se encontrase al principio de la llamada a un procedimiento la declaración de localización:

DCL <nombre de parámetro formal> <modo> := <parámetro efectivo>.

Sin embargo, la inicialización no puede provocar una excepción dentro del cuerpo procedimiento. Puede especificarse, facultativamente, la palabra clave *IN* para indicar explícitamente el paso por valor.

Si se especifica un atributo *INOUT,* el valor del parámetro efectivo se obtiene de una localización, restituyéndose en la localización efectiva el valor actual del parámetro formal inmediatamente antes del retorno.

El efecto de *OUT* es el mismo de el *INOUT,* salvo que el valor inicial de la localización efectiva no se copia en la localización del parámetro formal tras la entrada del procedimiento. En consecuencia, el parámetro formal tiene un valor inicial indefinido. No es necesario efectuar la operación de nuevo almacenamiento si el procedimiento produce una excepción en el punto de llamada.

Paso por localización

En la transferencia de parámetro mediante paso por localización, se transfiere una localización en forma de parámetro al cuerpo procedimiento. No pueden transferirse de esta forma ni las localizaciones no referenciables ni las localizaciones modo dinámico. El efecto es el mismo que si se encontrase en el punto de entrada del procedimiento la sentencia de declaración de identidad-loc:

DCL <nombre de parámetro formal> <modo> LOC:= <parámetro efectivo>;

sin embargo, tal declaración no puede producir una excepción dentro del cuerpo de un procedimiento.

Si se especifica un *valor* que no es una *localización de modo estático,* se creará implícitamente y se pasará en el momento de la llamada una localización que contenga el valor especificado. El tiempo de la localización creada es la llamada de procedimiento.

transmisión del resultado:

Un procedimiento puede proporcionar un valor o una localización. En el primer caso se especifica un *valor* en cualquier *acción resultado* y en el segundo una *localización de modo estático* (véase el apartado 6.8). El valor o localización proporcionado están determinados por la acción resultado ejecutada más recientemente antes del retorno. Si se produce el retorno de un procedimiento con una especificación resultado sin haberse ejecutado una acción resultado, el procedimiento entrega un valor o localización indefinido. En este caso la llamada al procedimiento no puede utilizarse como llamada a un procedimiento que produce localización (véase el

apartado 4.2.10) ni una llamada a un procedimiento que proporciona valor (véase el apartado 5.2.15) sino simplemente como una acción de llamada (apartado 6.7).

especificación de registro:

La especificación de registro puede facilitarse en el parámetro formal del procedimiento y en la especificación de resultado. En el caso del paso por valor, significa que el valor efectivo está contenido en el registro especificado; en el caso del paso por localización, indica que el puntero (oculto) de la localización efectiva está contenido en el registro especificado. Si se especifica en la especificación de resultado, indica que el valor proporcionado o el puntero (oculto) de la localización suministrada está contenido en el registro especificado.

propiedades estáticas: Un nombre es un nombre de procedimiento, única y exclusivamente si se ha definido en una *sentencia de definición de procedimiento* o en una *sentencia de entrada* (es decir está situado frente al signo: y una *definición de procedimiento* o *definición de entrada).*

Un nombre de procedimiento, tiene asociada una *definición de procedimiento* definida como sigue:

- Si el nombre de procedimiento está definido en una *sentencia de definición de procedimiento,* será la *definición de procedimiento* de dicha sentencia.

- Si el nombre de procedimiento está definido en una *sentencia de entrada,* será la *definición de procedimiento* en cuyo dominio esté colocada la *sentencia de entrada.*

Un nombre de procedimiento, tiene asociadas las propiedades que siguen, definidas por su *definición de procedimiento:*

- Tiene una lista de especificaciones de parámetro, definidas por las ocurrencias de *especificación de parámetro* en la *lista de parámetros formales.* Cada parámetro consta de un modo, posiblemente un atributo de parámetro y/o un nombre de registro.

- Tiene, posiblemente, una especificación de resultado que consta de un modo, posiblemente un atributo *LOC* y/o un nombre de registro.

- Tiene un conjunto, posiblemente, vacío de nombres de excepción que son los mencionados en la *lista de excepciones.*

- Tiene una generalidad que es, si se especifica *generalidad,* general, simple o inline dependiendo que se especifique *GENERAL, SIMPLE* o *INLINE*; en cualquier otro caso, se especifica subsidiariamente en la realización general o simple. Si el nombre de procedimiento se define dentro de una región, su generalidad es simple.

- Tiene una recursividad que es recursiva si se especifica *RECURSIVE.* En cualquier otro caso, se especifica subsidiariamente en la realización *recursiva* o *no recursiva.* Sin embargo,

si la generalidad es inline, o si el nombre de procedimiento es crítico (véase el apartado 8.2) la recursividad es no recursiva.

Un nombre de procedimiento general, es un procedimiento literal. Un nombre de procedimiento general, tiene asociado un modo procedimiento formado como sigue:

PROC (/<lista de parámetros>/) /<especificación de resultado >/ /EXCEPCIONES (<lista de excepciones>)/ /RECURSIVE/

donde *<especificación de resultado >*, si existe, y *<lista de excepciones>* son los mismos que en su *definición de procedimiento* y *<lista de parámetros>* es la secuencia de ocurrencias de *<especificación de parámetro>* en la *lista de parámetros formales*, separados por comas.

Un nombre definido en la *lista de nombres* de *parámetro formal*, es un nombre de localización, única y exclusivamente si la *especificación de parámetros* del *parámetro formal* no contiene el atributo *LOC*. En caso contrario, se trata de un nombre de identidad-loc.

Cualquiera de tales nombres de localización o de identidad-loc, son referenciables (por lenguaje).

condiciones estáticas: Si un nombre de procedimiento es regional, (véase el apartado 8.2.2), su definición de procedimiento no puede especificar *GENERAL*.

Si un nombre de procedimiento es crítico (véase el apartado 8.2), su definición no puede especificar *GENERAL* ni *RECURSIVE*.

Ninguna definición de procedimiento puede especificar *INLINE* y *RECURSIVE*.

Cuando se especifique el *nombre de procedimiento* facultativo situado antes del punto y coma, debe ser igual al *nombre* colocado enfrente de la *definición de procedimiento*.

El modo de la *especificación de parámetro* o de la *especificación de resultado*, sólo podrá tener la propiedad de sincronización si se especifica *LOC*.

ejemplos

```
1.3  add:
     PROC (i, j INT) (INT) EXCEPTIONS (OVERFLOW);
        RESULT i + j;
     END add;                                          (1.1)
```

7.5 DEFINICIONES DE PROCESO

sintaxis:

<sentencia de definición de proceso> ::= (1)

<nombre> : <definición de proceso>

/<ejecutor>/ /<nombre de proceso> / (1.1)

<definición de proceso> ::= (2)
PROCESS (/<lista de parámetros formales>/);
<cuerpo de proceso> END (2.1)

semántica: Una definición de proceso establece una secuencia de acciones, posiblemente parametrizada, que puede ponerse en marcha para su ejecución concurrente desde distintos puntos del programa (véase el capítulo 8).

propiedades estáticas: Un nombre es un nombre de proceso, única y exclusivamente si está definido en una *sentencia de definición de proceso* (es decir colocada enfrente del signo : y de una *definición de proceso).*

Un nombre de proceso, debe tener asociados un conjunto de nombres de excepción definidos en la realización.

condiciones estáticas: Cuando se especifique, el *nombre de proceso* facultativo situado delante del punto y coma, debe ser igual al nombre colocado enfrente de la *definición de proceso.*

Una *sentencia de definición de proceso,* no puede estar rodeada por una región m por un bloque distinto de la definición del proceso imaginario más externo (véase el apartado 7.8).

Los atributos de parámetro de la *lista de parámetros formales,* no pueden ser *INOUT* ni *OUT.* Sólo si se especifica *LOC* en la *especificación de parámetro* en la *lista de parámetros formales* puede el modo en ella tener la propiedad de sincronización.

ejemplos:

```
14.12  PROCESS ( );
         DO FOR EVER;
           WAIT (10 /* seconds */);
           CONTINUE OPERATOR-IS-READY;
         OD;
       END                                                    (2.1)
```

7.6 MÓDULOS

sintaxis:

<módulo> ::= (1)
MODULE <cuerpo de módulo> END (1.1)

semántica: Un módulo es una acción que contiene posiblemente declaraciones y definiciones locales. Un módulo es una forma de restringir la visibilidad de los nombres; no influye en el tiempo de vida de las posiciones creadas localmente.

En el apartado 9.2 se facilitan las reglas de visibilidad detalladas para los módulos.

propiedades estáticas: Un nombre es un nombre de módulo única y exclusivamente si se define colocándolo enfrente del signo : delante de *MODULE.*

ejemplos:

```
7.42  MODULE
        SEIZE convert;
        DCL n INT INIT:= 1979;
        DCL rn CHAR (20) INIT: = (20)' ';
        GRANT n, rn;
        convert( );
        ASSERT rn = 'MDCCCCLXXVIIII'//(6)' ';
      END                                                    (1.1)
```

7.7 REGIONES

sintaxis:

```
<región> ::=                                                 (1)
    /<nombre>:/ REGIÓN <cuerpo de región> END
    /<ejecutor>/ /<nombre de región>/;                       (1.1)
```

semántica: Una región es un modo de proporcionar un acceso mutuamente exclusivo a su objeto de datos declarado localmente para la ejecución concurrente de procesos (véase el capítulo 8). Determina la visibilidad de los nombres creados localmente de la misma forma que el módulo.

propiedades estáticas: Un nombre es un nombre de región, única y exclusivamente si se ha definido colocándolo enfrente del signo : delante de *REGION*.

condiciones estáticas: El *nombre de región* facultativo, situado antes del punto y coma, debe ser igual al *nombre* de la región.

Una *región* no debe estar rodeada por un bloque distinto de la definición del proceso imaginario más externa.

ejemplos:

see 13.1 - 13.25

7.8 PROGRAMA

sintaxis:

```
<programa> ::=                                               (1)
    {<sentencia de acción módulo>|<región>}*                 (1.1)
```

semántica: Los programas constan de una lista de módulos o regiones rodeados por una definición del proceso imaginario más externo. Se considera que esta definición de proceso contiene en su dominio un módulo preludio CHILL normalizado. Este módulo contiene las definiciones de los nombres predefinidos y las rutinas predefinidas incorporadas de realización, los modos predefinidos y los nombres de registro predefinidos CHILL.

propiedades estáticas: El lenguaje y los nombres definidos en la realización (véase el apéndice C2) se consideran creados en un módulo en el dominio de la definición del proceso imaginario más externo, y cedidos mediante *PERVASIVE* por dicho módulo (véase el apartado 9.2.6.2).

7.9 *ASIGNACIÓN DE MEMORIA Y TIEMPO DE VIDA*

Se denomina tiempo de vida de una localización o procedimiento, al tiempo durante el cual éstas existen dentro de su programa.

Se crea una localización mediante una declaración o ejecutando una llamada a la rutina predefinida *GETSTACK*.

El tiempo de vida de una localización declarada en el dominio de un bloque, es el tiempo durante el cual el control reside en dicho bloque, a menos que se declare con el atributo *STATIC*. El tiempo de vida de una localización declarada en el dominio de un modulión, es el mismo que si se hubiera declarado en el dominio del bloque más próximo que rodea al modulión. El tiempo de vida de una localización declarada con el atributo *STATIC*, es el mismo que si se hubiese declarado en el dominio de la definición del proceso imaginario más externo. Esto significa que para una declaración de localización con el atributo *STATIC* la asignación de memoria se efectúa una sola vez, concretamente al principio del proceso imaginario más externo. Si tal declaración aparece dentro de una definición de procedimiento o de proceso solamente existirá una localización para todas las invocaciones o activaciones.

El tiempo de vida de una localización creada por la ejecución de la llamada a la rutina predefinida *GETSTACK*, es el tiempo comprendido entre la ejecución y la salida del bloque rodeante más próximo. Si se ejecuta la rutina predefinida *GETSTACK* mientras se evalúa un parámetro efectivo de una llamada a un procedimiento o una expresión de arranque, el tiempo de vida de la localización creada será la llamada al procedimiento o el tiempo de vida del proceso creado.

El tiempo de vida de un acceso creado en una declaración identidad-loc, es el bloque más próximo que rodea a la declaración identidad-loc.

El tiempo de vida de un procedimiento, es el bloque más próximo que rodea a la definición del procedimiento.

propiedades estáticas: Se dice que una *localización* es estática, única y exclusivamente si es una *localización de modo estático* de uno de los tipos siguientes:

- Un *nombre de localización*, declarado con el atributo *STATIC* o cuya definición no está rodeada por un bloque diferente de la definición del proceso imaginario.
- Un *nombre de identidad-loc*, tal que la *localización de modo estático* que aparece en su definición es estática.
- Un *elemento cadena* o *subcadena* en el que la *localización cadena* es estática y el *elemento izquierda* y el *elemento derecha*, o la *posición* son constantes.
- Un *elemento matricial* o *submatricial*, en el que la *localización matriz* es estática y la *expresión*, o el *elemento inferior* y el *elemento superior* o la *expresión entera* que aparecen en él son constantes.
- Un *campo de estructura* en el que la *localización estructura* es estática. Si la *localización estructura* no es una *localización estructura parametrizada*, el *nombre de campo* no debe ser un *nombre de campo variable*.
- Una *conversión de localización* tal que la *localización* que aparece en ella es estática.

8.0 EJECUCIÓN SIMULTÁNEA

8.1 PROCESOS Y SUS DEFINICIONES

Un proceso, es la ejecución secuencial de una serie de sentencias. Dicha ejecución secuencial puede ser concurrente con otros procesos. Se describe el comportamiento de un proceso, mediante una definición de proceso (véase el punto 7.5), que describe los objetos locales de un proceso y la serie de sentencias de acción que deben ejecutarse secuencialmente.

Se crea un proceso, mediante la evaluación de una expresión de arranque (véase el punto 5.2.17). Se vuelve activo (es decir en ejecución) y se considera que se ejecuta concurrentemente con otros procesos. El proceso creado es una activación de la definición indicada por el nombre de proceso en la definición de proceso. Pueden crearse y ejecutarse concurrentemente un número no especificado de procesos con la misma definición. Cada proceso se identifica unívocamente por un valor ejemplo, obtenido como resultado de la expresión de arranque o la evaluación del operador *THIS*. La creación de un proceso implica la creación de sus localizaciones declaradas localmente con excepción de las declaradas con el atributo *STATIC* (véase el punto 7.9) y de los valores y procedimientos definidos localmente. Se dice que las localizaciones, valores y procedimientos declarados localmente, tienen la misma activación que el proceso creado al que pertenecen. El proceso imaginario más externo (véase el punto 7.8) que constituye la totalidad del programa CHILL en ejecución, se considera creado por una expresión de arranque ejecutada por el sistema bajo cuyo control se está ejecutando el programa. En la creación de un proceso, sus parámetros formales, si están presentes, designan los valores y localizaciones según son entregados por los parámetros efectivos correspondientes de la expresión de arranque.

Se termina un proceso, mediante la ejecución de una acción parar o cuando se alcanza el final del cuerpo del proceso, o el de una alternativa-on de un programa ejecutor especificado al final de la definición del proceso (caída directa). Si el proceso imaginario más externo ejecuta una acción de parar o experimenta una caída directa se completará la terminación cuando y sólo cuando finalicen todos sus procesos subsidiarios (es decir, los procesos creados en él por las expresiones de arranque).

En el nivel de programación CHILL, un proceso siempre está en uno de entre dos estados: o es activo (es decir, en ejecución) o está en espera (es decir, esperando a que se cumpla una condición). La transición de activo a espera se denomina puesta en espera del proceso, la transición de en espera a activo, se denomina reactivación del proceso.

8.2 EXCLUSIÓN MUTUA Y REGIONES

8.2.1 CONSIDERACIONES GENERALES

Las regiones (véase 7.7) constituyen un modo de proporcionar procesos con accesos mutuamente exclusivos a las localizaciones declaradas en ellas. Las condiciones estáticas de contexto (véase el punto 8.2.2) son tales que los accesos mediante un proceso (que no es el proceso imaginario más externo) a las localizaciones declaradas en una región solamente pueden realizarse mediante llamada de procedimientos definidos dentro de la región y cedidos por ésta.

Se dice que un nombre de procedimiento designa un procedimiento crítico (y es un nombre de procedimiento crítico), si y sólo si se ha definido dentro de una región y se ha cedido por ésta o si un nombre de procedimiento con la misma definición de procedimiento

(vease el punto 7.4) es crítico (esto último tiene importancia solamente cuando afecte a definiciones de entrada).

Se dice que una región es libre, si y sólo si, el control no reside en ninguno de sus procedimientos críticos ni en la región misma efectuando las inicializaciones ligadas al dominio.

La región estará bloqueada (para evitar la ejecución concurrente) si:

- Se produce la entrada de la región (obsérvese que, debido a que las regiones no están rodeadas por un bloque, no pueden hacerse tentativas concurrentes para producir la entrada de la región).
- Se llama a un procedimiento crítico de la región.
- Se reactiva un proceso puesto en espera en la región.

La región será liberada, volviendo nuevamente a estar libre si:

- Se ha dejado a la región.
- Retorna el procedimiento crítico.
- El procedimiento crítico ejecuta una acción que causa la puesta en espera del proceso ejecutante (véase el punto 8.3). En el caso de llamadas de procedimiento crítico fasciculadas dinámicamente sólo se liberará la última región bloqueada.

Si, estando una región bloqueada, un proceso intenta llamar a uno de sus procedimientos críticos o entrar en la región, se suspende dicho proceso hasta que se libere la región (obsérvese que tal proceso permanece activo en sentido CHILL).

Cuando se libera una región y hay más de un proceso detenido mientras intentaba entrar en la región o llamar a uno de sus procedimientos críticos, o ser reactivado por uno de sus procedimientos críticos, solamente se seleccionará un proceso para entrar en la región de acuerdo con un algoritmo de ordenación definido en la realización.

8.2.2 REGIONALIDAD

Para permitir la verificación estática de que solamente puede accederse a una posición declarada en una región llamando a procedimientos críticos o haciendo entrar la región para efectuar inicializaciones ligadas al dominio, deben forzarse las siguientes condiciones de contexto estático:

- Los requerimientos de regionalidad mencionados en los puntos apropiados (acción asignar, llamada a un procedimiento, acción enviar, acción resultado).
- Los procedimientos regionales no son generales (véase el punto 7.4).
- Los procedimientos críticos no son generales ni recurrentes (véase el punto 7.3).

Una ***localización, valor*** o ***nombre de procedimiento,*** puede ser regional. Se define esta propiedad como sigue:

1. Localización

   Una ***localización*** es regional, si y sólo si se cumplen algunas de las condiciones siguientes:

- Es un *nombre de acceso* que, a su vez, es:

  - un *nombre de localización* declarado textualmente dentro de una *región* y que no está definido en un *parámetro formal* de un procedimiento crítico,

  - un *nombre de identidad-loc*, tal que la *localización de modo estático* de su declaración es regional o se ha definido en un *parámetro formal* de un procedimiento regional,

  - un *nombre referido a una base*, en el que el nombre *de localización de referencia libre o ligada* de su declaración es regional,

  - un *nombre de enumeración de localización*, en el que la *localización matriz* en la *acción hacer* asociada, es regional,

  - un *nombre de localización do-with*, en el que la *localizacion estructura* de la *acción do* asociada es regional.

- Es una *referencia ligada desreferenciada*, en la que *expresión de referencia ligada*, es regional.

- Es una *referencia libre desreferenciada*, en la que *expresión de referencia libre*, es regional.

- Es un *descriptor desreferenciado*, en el que la *expresión descriptor* es regional.

- Es un *elemento matricial, submatricial* o *segmento de matriz*, en el que la *localizacion matriz*, es regional.

- Es un *elemento cadena, subcadena* o *segmento de cadena*, en el que la *localización cadena* es regional.

- Es un *campo de estructura*, en el que la *localización estructura* es regional.

- Es una *llamada a un procedimiento que produce una localización*, tal que en la misma se ha especificado un *nombre de procedimiento* que es regional.

- Es una *llamada a una rutina predefinida que produce una localización*, para la cual, la realización especifica que es regional.

- Es una *conversión de localización*, cuya *localización de modo estático* es regional.

2. Valor

Un *valor* o *expresión* es regional, si y sólo si se trata de un *valor primitivo* regional o una *expresión parentizada* que contiene una expresión regional.

Un *valor primitivo* es regional, si y sólo si se cumple una cualquiera de las siguientes condiciones:

- Es un *contenido de localización* regional, y cuyo modo tiene la propiedad referenciante.

- Es un *nombre de valor* que, a su vez es:

  - un *nombre de sinónimo*, en cuya definición el *valor constante* es regional,

  - un *nombre de valor do-with*, tal que la *expresión estructura* en su acción *hacer* asociada es regional y cuyo modo tiene la propiedad referenciante.

- Es una *tupla,* que contiene una *tupla matricial* o *tupla de estructura,* en la cual al menos una de las ocurrencias del *valor* especificado, es regional.

- Es un *valor elemento matricial*, un *valor submatricial* o un *valor segmento de matriz* tal que su *expresión matricial* es regional y el modo de la *expresión matricial* tiene la propiedad referenciante.

- Es un *valor campo de estructura,* tal que la *expresión estructura* del mismo es regional, y el modo del campo tiene la propiedad referenciante.

- Es una *localización referenciada* en la que la *localización* es regional.

- Es una *conversión de expresión,* tal que su *expresión* es regional.

- Es una *llamada a un procedimiento que produce valor,* tal que en la *llamada a un procedimiento que produce valor,* se especifica un *nombre de procedimiento* que es regional y cuyo modo resultado tiene la propiedad referenciante.

- Es una *llamada a una rutina predefinida que produce valor,* que a su vez, es bien una *llamada a una rutina predefinida que produce valor definido en la realización* que entrega valor de clase compatible con un modo referenciante para el cual la realización especifica que es regional, o *ADDR (<localización>)*, donde *localización* es regional.

3. Nombre de procedimiento

Un *nombre de procedimiento* es regional, si y sólo si se ha definido dentro de una *región* y no es crítico (es decir no cedido por la región).

8.3 PUESTA EN ESPERA DE UN PROCESO

Cuando un proceso es activo, puede ponerse en espera ejecutando o evaluando una de las siguientes acciones o expresiones:

Acción poner en espera (véase el apartado 6.16). Cuando un proceso ejecuta una acción poner en espera, queda en espera. Se transforma en un miembro, con cierta prioridad, de un conjunto de procesos en espera asociados a la localización suceso especificada.

Acción poner en espera y elegir (véase el punto 6.17). Cuando un proceso ejecuta una acción poner en espera y elegir, queda en espera. Se transforma en miembro, con la prioridad especificada, de cada conjunto de procesos en espera asociados a la localización suceso especificada en una alternativa de puesta en espera de la acción poner en espera y elegir.

Expresión recibir (véase el apartado 5.2.18). Cuando un proceso evalúa una expresión recibir, queda en espera, si y sólo si la localización tampón especificada no contiene valores ni procesos remitentes en espera. Se transforma en un miembro de un conjunto de procesos receptores en espera, asociados a la localización tampón (vacía) especificada.

Acción recibir tampón y elegir (véase el apartado 6.19.3). Cuando un proceso ejecuta una acción recibir tampón y elegir queda en espera, si y sólo si no existe ningún valor ni hay en espera ningún proceso remitente en ninguna de las localizaciones tampón especificadas y si tampoco se ha especificado *ELSE*. Se convierte en un miembro de cada conjunto de procesos receptores en espera, asociados a una localización tampón especificada en una alternativa de recepción de tampón o en la acción recibir tampón y elegir.

Acción recibir señal y elegir (véase el apartado 6.19.2). Cuando un proceso ejecuta una acción recibir señal y elegir, queda en espera si no hay pendiente ninguna señal que pueda recibirse por el proceso que ejecuta aquella acción y si no se ha especificado *ELSE*. El proceso se convierte en un miembro de cada conjunto de procesos en espera, asociado a un nombre de señal especificado en la alternativa de recepción de señal.

Acción enviar tampón (véase el apartado 6.18.3). Cuando un proceso ejecuta una acción enviar tampón, queda en espera, si y sólo si el modo de la localización tampón tiene asociada una longitud y el número de valores del tampón es igual a esta longitud inmediatamente antes de la operación de envío. El proceso se convierte en miembro, con la prioridad especificada, del conjunto de procesos remitentes en espera asociado a la posición tampón.

Cuando un proceso ejecuta una acción que produce su puesta en espera mientras su control reside en un procedimiento crítico, se liberará la región asociada. Se retendrá el contexto dinámico del procedimiento hasta la reactivación del proceso en el punto de la región donde esté en espera. Entonces dicha región quedará nuevamente bloqueada.

8.4 REACTIVACIÓN DE UN PROCESO

Un proceso en espera, puede reactivarse si y sólo si otro proceso ejecuta una de las acciones siguientes:

Acción continuar (véase el apartado 6.15). Cuando un proceso ejecuta una acción continuar, reactiva otro proceso si y sólo si el conjunto de procesos en espera de la localización suceso especificada no es vacío. De acuerdo con un algoritmo de ordenación definido en la realización, se transforma en activo el proceso de máxima prioridad. En consecuencia este proceso reactivado se retira de todos los conjuntos de procesos en espera.

Acción enviar tampón (véase el apartado 6.18.3). Si un proceso ejecuta una acción enviar tampón, reactivará otro proceso si y sólo si no es vacío el conjunto de procesos receptores en espera de la localización tampón especificada. Se elige un proceso para transformarlo en activo de acuerdo con un algoritmo de ordenación definido en la realización. De esta forma un proceso reactivado se retira de todos los conjuntos de procesos en espera. Si el conjunto de procesos de recepción en espera, de una localización tampón especificada, está vacío, el valor enviado se almacenará en el tampón con su prioridad especificada si lo admite la capacidad del tampón (véase el apartado 8.3).

Acción enviar señal (véase el apartado 6.18.2). Cuando un proceso ejecuta una acción enviar señal, reactiva otro proceso si y sólo si el conjunto de procesos en espera, con el nombre de señal especificado, contiene un proceso que puede recibir la señal. La selección de un proceso para transformarlo en activo se realiza mediante un algoritmo de ordenación definido en la realización. De este modo, el proceso reactivado, se retira de todos los conjuntos de procesos en espera. Si no hay presente ningún proceso en espera para recibir la señal, ésta queda pendiente con su prioridad específica, lista posible de valores, nombre de proceso y/o valor ejemplo.

Acción recibir tampón y elegir (véase el apartado 6.19.3). Cuando un proceso ejecuta una acción recibir tampón y elegir, reactiva otro proceso si y sólo si no es vacío el conjunto de procesos remitentes en espera o alguna de las localizaciones tampón especificadas. En este caso recibe un valor de máxima prioridad de entre los valores de la localización tampón o de los procesos remitentes en espera. Al recibir un valor de un tampón, el proceso elimina el valor del tampón seleccionándose, para su conversión en activo, el proceso enviar en espera cuyo valor sea de máxima prioridad, de acuerdo con un algoritmo de ordenación definido en la realización. De este modo, el proceso reactivado se elimina de todos los conjuntos de procesos remitentes en espera, almacenándose su valor en el tampón con la prioridad especificada. Al recibirse directamente un valor procedente de un proceso remitente en espera, se selecciona para su transformación en activo el proceso en espera que tenga asociado el valor de máxima prioridad, de acuerdo con un algoritmo definido en la realización. Este proceso reactivado se suprime en todos los conjuntos de procesos remitentes en espera, recibiéndose su valor.

Cuando un proceso ejecuta una acción que provoca la reactivación de otro proceso, mientras el proceso en fase de reactivación está activo, dentro de un procedimiento crítico, dicho proceso permanecerá activo, es decir no liberará la región en aquel punto.

8.5 SENTENCIAS DE DEFINICIÓN DE SEÑAL

sintaxis:

<sentencia de definición de señal> ::= (1)
*SIGNAL <definición de señal> {,<definición de señal>}** (1.1)

<definición de señal> ::= (2)
<nombre> /=(<modo> {,<modo>})/ /TO <nombre de proceso>/* (2.1)

semántica: Una definición de señal, define una función de composición y descomposición para valores que han de transmitirse entre procesos. Si se envía una señal, se transmite la lista de valores especificada. Si en una acción recibir y elegir, no hay ningún proceso en espera de la señal, se guardan los valores hasta que un proceso los reciba.

propiedades estáticas: Un nombre es un nombre de señal, si y sólo si se define en una *definición de señal*. Un nombre de señal, tiene las siguientes propiedades:

- Tiene asociada una lista facultativa de modos que son los mencionados en la *definición de señal*.
- Tiene asociado, facultativamente, un nombre de proceso, que es el *nombre de proceso* especificado después de *TO*.

condiciones estáticas: En una *definición de señal* ningún *modo* puede tener la propiedad de sincronización.

ejemplos:

```
15.16  SIGNAL INITIATE = (INSTANCE),
15.17         TERMINATE;                                   (1.1)
```

9.0 PROPIEDADES SEMÁNTICAS GENERALES

9.1 VERIFICACIÓN DE MODOS

9.1.1 PROPIEDADES DE MODOS Y CLASES

9.1.1.1 Novedad

Informal

La novedad de un modo indica si se ha definido o no mediante una sentencia de definición de neomodo. La novedad de un modo es o nula es decir se trata de un modo (base) no definido mediante un neomodo o es el nombre de neomodo mediante el cual se define.

Definición

Se define la novedad de un modo como sigue:

- Si el modo se designa por un *nombre de neomodo*, su novedad es la de nombre de neomodo,
- en cambio, si el modo se designa por un *nombre de sinmodo*, su novedad es la del *modo definidor* en su definición.
- en cambio si el modo se designa por un *modo matriz parametrizada*, *un modo cadena parametrizada* o un *modo estructura parametrizada*, su novedad es la novedad del *nombre de modo matriz origen*, *nombre de modo cadena origen* o *nombre de modo estructura variable origen*, en él, respectivamente,
- en cambio si el modo se designa por un *modo intervalo*, su novedad es la de su modo progenitor,
- en cambio si el modo se designa por un modo progenitor introducido virtualmente, su novedad es la del nombre de neomodo que ha causado su introducción (véase el apartado 3.2.3.3),
- en cambio si el modo se designa por *READ* <*modo*>, su novedad es la de <*modo*>,
- en cualquier otro caso la novedad es nula.

9.1.1.2 Modos de sólo lectura

Informal

Se dice que un modo es de sólo lectura si lo es una localización del mismo, considerada en su totalidad, es decir no puede escribirse sobre él ni sobre ninguna parte del mismo.

Definición

Un modo tiene la siguiente propiedad hereditaria: es un modo de sólo lectura, si y solo si se cumple alguna de las siguientes condiciones:

- Se designa mediante un *modo* que tiene la forma *READ <modo>*.

- Se designa mediante un *modo matriz parametrizada, un modo cadena parametrizada* o un *modo estructura parametrizada,* donde el *nombre de modo matriz origen,* el *nombre de modo cadena origen* o el *nombre de modo estructura variable origen,* respectivamente, designa un modo de sólo lectura.

9.1.1.3 Propiedad de sólo lectura

Informal

Un modo tiene la propiedad de sólo lectura, si una localización del mismo, es de sólo lectura o contiene una componente, subcomponente, etc., que es de sólo lectura.

Definición

Un modo tiene la propiedad de sólo lectura, si y sólo si se cumple una de las condiciones siguientes:

- El modo es un nombre de modo definido mediante un modo que posee la propiedad de sólo lectura.

- El modo es un modo matriz con un modo elemento que posee la propiedad de sólo lectura, o un modo estructura en el que por lo menos uno de sus modos de campo tiene la propiedad de sólo lectura.

- El modo es un modo de sólo lectura.

9.1.1.4 Propiedad de referenciar

Informal

Un modo tiene la propiedad de referenciar, si una localización del mismo posee un modo referencia o contiene una componente, subcomponente, etc., que posee un modo referencia.

Definición

Un modo tiene la propiedad de referenciar, si y solo si se cumple una de las condiciones siguientes:

- El modo es un nombre de modo definido por un modo que posee la propiedad de referenciar.

- El modo es un modo matriz con un modo elemento que tiene la propiedad de referenciar o un modo estructura tal que al menos uno de sus modos de campo posee la propiedad de referenciar.

- El modo es un modo de referencia.

9.1.1.5 Propiedad parametrizada marcada

Informal

Un modo tiene la propiedad parametrizada marcada, si una localización del mismo tiene un modo estructura parametrizada con marcadores o contiene una componente, subcomponente, etc., que posee un modo estructura parametrizada con marcadores.

Definición

Un modo tiene la propiedad parametrizada marcada si y sólo si se cumple una de las siguientes condiciones:

- El modo es un nombre de modo definido por un modo que posee la propiedad parametrizada marcada.
- El modo es un modo matriz con un modo elemento que posee la propiedad parametrizada marcada o un modo estructura en el que al menos uno de sus modos de campo, posee la propiedad parametrizada marcada.
- El modo es un modo estructura parametrizada con marcadores.

9.1.1.6 Propiedad de sincronización

Informal

Un modo tiene la propiedad de sincronización si una localización del mismo posee un modo sincronización o contiene una componente, subcomponente, etc., que posee un modo sincronización.

Definición

Un modo tiene la propiedad de sincronización, si y sólo si se cumple una de las siguientes condiciones:

- El modo es un nombre de modo definido por un modo que posee la propiedad de sincronización.
- El modo es un modo matriz con un modo elemento que posee la propiedad de sincronización o un modo estructura en el que al menos uno de sus modos de campo posee la propiedad de sincronización.
- El modo es un modo suceso o un modo tampón.

9.1.1.7 Modo raíz

Toda clase M-valuada o M-derivada en la que M no es un modo compuesto, tiene un modo raíz definido así:

- si M no es un modo intervalo, el modo raíz es M,
- si M es un modo intervalo, el modo raíz es el modo progenitor de M.

9.1.1.8 Clase resultante

Dadas dos clases compatibles (véase el apartado 9.1.2.6) que son o la clase total, una clase M-valuada o una clase M-derivada donde M es un modo discreto, un modo conjuntista o un modo cadena, se define la clase resultante así:

- la clase resultante de la clase M-derivada y de la N-derivada es la clase M-derivada;
- la clase resultante de las clases M-valuada y N-derivada, es la clase M-valuada, si M no es un modo intervalo; en cualquier otro caso es la clase P-valuada donde P es el modo progenitor de M;
- la clase resultante de las clases M-valuada y N-valuada, es la clase M-valuada si M no es un modo intervalo; en cualquier otro caso es la clase P-valuada donde P es el modo progenitor de M;
- la clase resultante de la clase general y cualquier otra clase es esta última.

Dada una lista C_i de clases compatibles dos a dos (i=1, ..., n), se define recurrentemente la clase resultante de esta lista de clases del siguiente modo: si n>1, es la clase resultante de la clase resultante de la lista C_i (i=1, ..., n-1) y la clase C_n, en cualquier otro caso es la clase resultante de C_1 y C_1.

(Obsérvese que se ha definido el CHILL de tal modo que el orden en que se toman las clases C_i es indiferente, es decir todas estas clases resultantes son compatibles).

9.1.2 RELACIONES SOBRE LOS MODOS Y CLASES

En los siguientes puntos, se definen las relaciones de compatibilidad entre modos, entre clases y entre modos y clases. Estas relaciones se utilizan a lo largo de este documento para definir condiciones estáticas.

Las propias relaciones de compatibilidad se definen en términos de algunas otras relaciones que se utilizan principalmente en el capítulo 9 para la finalidad antes mencionada.

9.1.2.1 Relación "definido por"

Informal

Se dice que un nombre de modo está definido por su modo de definición y, de forma transitiva, si este último es también un nombre de modo, el primero está también definido por el modo definidor de su modo definidor etc.

Definición

Se dice que un nombre de modo N está definido por un modo M, si y sólo si:

- M es el modo definidor de N
- el modo de definición de N es un nombre de modo definido por M.

9.1.2.2 Relaciones de equivalencia sobre modos

CONSIDERACIONES GENERALES

Informal

Las siguientes relaciones de equivalencia desempeñan un papel en la formulación de las relaciones de compatibilidad:

- Se dice que dos modos son similares, si son de misma naturaleza, es decir, tienen las mismas propiedades hereditarias.
- Se dice que dos modos son v-equivalentes,(equivalentes en valor) si son similares y tienen también la misma novedad.
- Se dice que dos modos son equivalentes, si son v-equivalentes y se tienen también en cuenta las posibles diferencias en la representación del valor en el almacenamiento en memoria o de la dimensión mínima de almacenamiento.
- Se dice que dos modos son l-equivalentes (equivalentes en localización) si son equivalentes y tienen también la misma especificación de solo-lectura.

Definición

En los puntos siguientes, se dan las relaciones de equivalencia sobre los modos en forma de un conjunto (parcial) de relaciones. Los algoritmos de equivalencia total se obtienen tomando el cierre simétrico, reflexivo y transitivo de este conjunto de relaciones. Los modos mencionados en las relaciones pueden ser dinámicos o introducidos virtualmente. En el primer caso, sólo se puede realizar la verificación de la equivalencia completa en el momento de la ejecución. El fallo de la verificación de la parte dinámica dará como resultado la excepción *RANGEFAIL* o *TAGFAIL* (véanse los puntos convenientes).

La verificación de dos modos recurrentes para cualquier equivalencia requiere la comprobación de los modos asociados en los caminos correspondientes del conjunto de modos recurrentes mediante los cuales se definen. Los modos son equivalentes si no se encuentra ninguna contradicción. (En consecuencia, un camino del algoritmo de verificación se termina satisfactoriamente si se comparan dos modos que ya han sido comparados previamente.)

La relación "similar"

Dos modos son similares, si y solo si se cumple alguna de las condiciones siguientes:

- son modos enteros;
- son modos booleanos;
- son modos carácter;
- son modos conjunto tales que definen el mismo número de valores, los mismos nombres de elementos de conjunto y para los mismos nombres, la llamada a la rutina predefinida *NUM*, entrega el mismo valor;
- son modos intervalo con modos progenitores similares;
- uno es un modo intervalo cuyo modo progenitor es similar al otro modo;
- uno es un modo booleano y el otro es un modo cadena de bits de longitud 1;
- uno es un modo carácter y el otro es un modo cadena de caracteres de longitud 1;
- son modos conjuntistas tales que sus modos miembro son equivalentes;
- son modos de referencia ligada tales que sus modos referenciados son equivalentes;
- son modos de referencia libre;
- son modos descriptor tales que sus modos referenciado origen son equivalentes;
- son modos procedimiento tales que:

  1. tienen el mismo número de especificaciones de parámetro y las especificaciones de parámetro correspondientes (en posición) tienen modos 1-equivalentes, los mismos atributos de parámetro y, si existe, la misma especificación de registro.

  2. ambos tienen o ambos no tienen una especificación de resultado. Si existen, ambas especificaciones de resultado deben tener modos 1-equivalentes, los mismos atributos y la misma especificación de registro, si existe:

  3. tienen el mismo conjunto de nombres de excepción;

  4. tienen la misma recursividad;

- son modos ejemplo;
- son modos suceso tales que ambos tienen la misma longitud o no tienen longitud;
- son modos tampón tales que:

  1. ambos no tienen longitud o tienen la misma longitud;
  2. tienen modos elemento tampón l-equivalentes;

- son modos cadena tales que:

  1. ambos son modos cadena de bits o modos cadena de caracteres;
  2. tienen la misma longitud. Esta verificación es dinámica cuando uno o ambos modos es (son) dinámico(s). El fallo de la verificación, producirá la excepción *RANGEFAIL*;

- son modos matriz tales que:

  1. sus modos índice son v-equivalentes;
  2. sus modos elemento son equivalentes;
  3. sus organizaciones de elemento son equivalentes (véase apartado 9.1.2.2);
  4. tienen el mismo número de elementos. Esta verificación es dinámica si uno o ambos modos es (son) dinámico(s). El fallo de la verificación, producirá la excepción *RANGEFAIL*;

- son modos estructura que no son modos estructura parametrizada tales que:

  1. tienen el mismo número de campos y los campos correspondientes (en posición) son equivalentes (véase el punto 9.1.2.2);
  2. si son modos estructura variable parametrizable, sus listas de clases deben ser compatibles;

- son modos estructura parametrizada, tales que:

  1. sus modos estructura variable origen son similares;
  2. sus valores correspondientes (en posición) deben ser los mismos. Esta verificación es dinámica si uno o ambos modos es (son) dinámico(s). El fallo de la verificación producirá la excepción *TAGFAIL*.

La relación "v-equivalente"

Dos modos son v-equivalentes, si y solo si son similares y tienen la misma novedad.

La relación "equivalente"

Dos modos son equivalentes, si y solo si son v-equivalentes y:

- si uno de los modos es un modo booleano, el otro modo también debe ser un modo booleano;
- si uno de los modos es un modo carácter, el otro modo debe ser también un modo carácter;
- si uno de los modos es un modo intervalo, el otro modo debe ser también un modo intervalo y ambos límites superiores deben ser iguales y ambos límites inferiores deben ser iguales.

La relación "l-equivalente"

Dos modos son l-equivalentes, si y solo si son equivalentes y si uno de los modos tiene la propiedad de solo lectura, el otro modo también debe tener la propiedad de solo lectura, y:

- si ambos modos son de referencia ligada, sus modos referenciados deben ser l-equivalentes;
- si ambos son modos descriptor, sus modos referenciados origen, deben ser l-equivalentes;
- si ambos son modos matriz, sus modos elemento deben ser l-equivalentes;
- si ambos son modos estructura los campos correspondientes (en posición) deben ser l-equivalentes.

Las relaciones "equivalente" y "l-equivalente" para campos

Dos campos (ambos campos en el contexto de dos modos estructura dados) son 1. equivalente, 2. l-equivalente, si y solo si ambos campos son campos fijos que son 1. equivalentes, 2. l-equivalentes o ambos son campos alternativos que son 1. equivalentes, 2. l-equivalentes.

Las relaciones "equivalente" y "l-equivalente" se definen de forma recurrente para los campos fijos (correspondientes), los campos variables, los campos alternativos y las alternativas variables respectivamente, de la siguiente forma:

1. Campos fijos y campos variables

   a. Ambos campos deben tener una organización equivalente.

   b. Ambos campos deben ser 1. equivalentes, 2. l-equivalentes.

2. Campos alternativos

a. Ambos campos alternativos, tienen marcadores o ninguno de ellos tiene marcadores. En el primer caso, los marcadores deben tener el mismo número de nombres de campos marcador y los correspondientes nombres (en posición) de campos marcador deben designar los campos fijos correspondientes.

b. Ambos deben tener el mismo número de alternativas variables y las alternativas variables correspondientes (en posición) deben ser 1. equivalentes, 2. 1-equivalentes.

c. Ambos deben tener especificado *ELSE* o ninguno debe tener especificado *ELSE*. En el primer caso, debe seguir el mismo número de campos variables y los campos variables correspondientes (en posición) deben ser 1. equivalentes, 2. 1-equivalentes.

3. Alternativas variables

a. Ambas alternativas variables deben tener el mismo número de listas de etiquetas de caso y las listas de etiquetas de caso correspondientes (en posición) deben ser o bien ambas *indiferentes*, o bien *(ELSE)*, o bien definir el mismo conjunto de valores.

b. Ambas alternativas variables deben tener el mismo número de campos variables y los campos variables correspondientes (en posición), deben ser 1. equivalentes, 2. 1-equivalentes.

La relación "equivalente" para organización

En lo que sigue, se supondrá que cada *pos* es de la forma:
POS(<número de palabra>,<bit de arranque>,<longitud>)

y que cada *paso* es de la forma:
STEP(<pos>,<longitud del paso>,<tamaño de la configuración>)

El punto 3.10.6 facilita las reglas convenientes para poner *pos* o *paso* en la forma requerida.

1. Organización de campo

Dos organizaciones de campo son equivalentes si ambas son *NOPACK*, o ambas son *PACK*, o ambas son *pos*. En el último caso una *pos*, debe ser equivalente a la otra (véase más adelante).

2. Organización de elementos

Dos organizaciones de elementos son equivalentes, si ambas son *NOPACK*, ambas son *PACK*, o ambas son *paso*. En el último caso la *pos* de un *paso* debe ser equivalente a la *pos* del otro (véase más adelante) y *NUM(longitud del paso)* debe proporcionar los mismos valores para las dos organizaciones de elementos y *NUM(tamaño de la configuración)*, debe entregar los mismos valores para las dos organizaciones de elementos.

3. Pos

Una *pos* es equivalente a otra *pos*, si y solo si ambas apariciones de *NUM (número de palabra)* entregan el mismo valor, ambas apariciones de *NUM (bit de arranque)* entregan el mismo valor y ambas apariciones de *NUM (longitud)* entregan el mismo valor.

9.1.2.3 Relación "de lectura compatible"

Informal

Se dice que un modo M es de lectura compatible con un modo N, si y solo si M y N son equivalentes y M y sus posibles (sub-) componentes tienen especificaciones de sólo lectura más restrictivas. Esta relación es por tanto no simétrica.

Ejemplo:
READ REF READ CHAR es de lectura compatible con *REF CHAR*

Definición

Se dice que un modo M es de lectura compatible con un modo N (relación no simétrica) si y solo si M y N son equivalentes y, si N es un modo de sólo lectura, entonces M debe ser también un modo de solo lectura y además:

- si M y N son modos de referencia ligada, el modo referenciado de M debe ser de lectura compatible con el modo referenciado de N;
- si M y N son dos modos descriptor, el modo referenciado origen de M debe ser de lectura compatible con el modo referenciado origen de N;
- si M y N son dos modos matriz, el modo elemento de M debe ser de lectura compatible con el modo elemento de N;
- si M y N son dos modos estructura, cualquier modo de campo de M debe ser de lectura compatible con el modo de campo de N correspondiente.

9.1.2.4 Relación "restringible a"

Informal

La relación "restringible a" se refiere a modos equivalentes con la propiedad de referenciar. Se dice que un modo M es restringible a un modo N, si él o sus posibles subcomponentes se refieren a localizaciones con una especificación de sólo lectura menor o igualmente restrictiva que los referenciados por N. Esta relación, es por tanto, no simétrica. La relación se utiliza en asignaciones (véase el punto 9.1.2.5).

Ejemplo:
REF INT es restringible a *REF READ INT*
STRUCT(P REF BOOL) es restringible a *STRUCT(Q REF READ BOOL)*

Definición

Un modo M es restringible a un modo N (relación no simétrica), si y solo si se cumple una de las siguientes condiciones:

- M no tiene la propiedad de referenciar y M es equivalente a N.
- M y N son modos de referencia ligada y el modo referenciado de N es de lectura compatible con el modo referenciado de M.
- M y N son modos de referencia libre y M y N son equivalentes.
- M y N son modos descriptor y el modo de origen referenciado de N es de lectura compatible con el modo de origen referenciado de M.
- M y N son modos matriz y el modo elemento de M es restringible al modo elemento de N.
- M y N son modos estructura y cada modo campo de M es restringible al modo de campo correspondiente de N.

9.1.2.5 Compatibilidad entre un modo y una clase

- cualquier modo M es compatible con la clase general;
- un modo M es compatible con la clase vacía, si y solo si M es un modo de referencia o un modo procedimiento o un modo ejemplo;
- un modo M es compatible con la clase de referencia N, si y solo si es un modo de referencia y se cumple una de las siguientes condiciones:

  1. N es un modo estático y M es un modo de referencia ligada cuyo modo referenciado es de lectura compatible con N;
  2. N es un modo estático y M es un modo de referencia libre;
  3. M es un modo descriptor con un modo referenciado origen V y:
     - si V es un modo cadena, N debe ser un modo cadena tal que V(p) sea de lectura compatible con N, donde p es la longitud (posiblemente dinámica) de N;
     - si V es un modo matriz, N debe ser un modo matriz tal que V(p) sea de lectura compatible con N, donde p es el límite superior (posiblemente dinámico) de N;
     - si V es un modo estructura variable, N debe ser un modo estructura parametrizada tal que V (p_1, ... p_n) sea de lectura compatible con N, donde $p_1 \ldots p_n$ designan la lista de valores de N;

- un modo M es compatible con una clase N-derivada, si y solo si M y N son similares;

- un modo es compatible con la clase de N-valuada, si y solo si se cumple una de las siguientes condiciones:

  1. si M no tiene la propiedad de referenciar, M y N deben ser v-equivalentes;

  2. si M tiene la propiedad de referenciar, N debe ser restringible a M.

9.1.2.6 Compatibilidad entre clases

- Cualquier clase es compatible consigo misma.

- La clase general es compatible con cualquier otra clase.

- La clase vacía es compatible con cualquier clase de referencia M.

- La clase vacía es compatible con la clase M-derivada o con la clase M-valuada, si y solo si M es un modo de referencia, es un modo procedimiento o un modo ejemplo.

- La clase M-referencia es compatible con la clase N-referencia si, y sólo si M y N son equivalentes. Si M y/o N es (son) un modo dinámico, se ignora la parte dinámica de la verificación de equivalencia es decir, no pueden ocurrir excepciones.

- La clase M-referencia es compatible con la clase N-derivada o la clase N-valuada, si y solo si N es un modo de referencia y se cumple una de las condiciones siguientes:

  1. M es un modo estático y N es un modo de referencia ligada cuyo modo referenciado es equivalente a M.

  2. M es una localización de modo estático y N es un modo de referencia libre.

  3. N es un modo descriptor con modo referenciado origen V y:

     - si V es un modo cadena, M debe ser un modo cadena tal que V(p) sea equivalente a M, donde p es la longitud (posiblemente dinámica) de M;

     - si V es un modo matriz, M debe ser un modo matriz tal que V(p) sea equivalente a M, donde p es el límite superior (posiblemente dinámico) de M;

     - si V es un modo estructura variable, M debe ser un modo estructura parametrizada tal que V (p_1,...p_n) sea equivalente a M, donde p_1,...p_n designa la lista de valores de N.

- La clase M-derivada es compatible con la clase N-derivada o con la clase de N-valuada, si y solo si M y N son similares.

- La clase de M-valuada, es compatible con la clase N-valuada, si y solo si M y N son v-equivalentes.

Dos listas de clases son compatibles si y solo si ambas listas tienen el mismo número de clases y las clases correspondientes (en posición) son compatibles.

9.1.3 SELECCIÓN DE CASO

sintaxis:

```
<especificación de etiqueta de caso> ::=                          (1)
    <lista de etiquetas de caso> {, <lista de etiquetas de caso>}* (1.1)

<lista de etiquetas de caso> ::=                                  (2)
    (<etiqueta de caso> {,<etiqueta de caso>}*)                    (2.1)
  | (ELSE) | <indiferente>                                         (2.2)

<etiqueta de caso> ::=                                            (3)
    <expresión literal discreta>                                  (3.1)
  | <intervalo literal>                                           (3.2)
  | <nombre de modo discreto>                                     (3.3)

<indiferente> ::=                                                 (4)
    (*)                                                           (4.1)
```

semántica: La selección de caso es un modo de seleccionar una alternativa de entre una lista de alternativas. La selección se basa en una lista especificada de valores de un selector.

La selección de caso puede aplicarse a:

- campos alternativos (véase el punto 3.10.4), en cuyo caso se selecciona una lista de campos variables,
- tuplas matriciales etiquetadas (véase el punto 5.2.5), en cuyo caso se selecciona un valor elemento de matriz,
- acción caso (véase el punto 6.4), en cuyo caso se selecciona una lista de sentencias de acción.

En las situaciones primera y última, se etiqueta cada alternativa con una especificación de etiqueta de caso; en la tupla matricial etiquetada, se etiqueta cada valor con una lista de etiquetas de caso. Para facilidad de descripción, la lista de etiquetas de caso de la tupla matricial etiquetada se considerará aquí como una especificación de etiqueta de caso con una aparición de lista de etiquetas de caso solamente.

La selección de caso, selecciona aquella alternativa que está etiquetada por la especificación de la etiqueta de caso que empareja con la lista de valores de selector. (El número de valores de selector será siempre el mismo que el número de apariciones de las listas de etiquetas de caso en la especificación de la etiqueta de caso.) Se dice

que una lista de valores empareja con una especificación de etiqueta de caso, si y solo si cada valor se adapta a la lista de etiquetas de caso correspondiente (en posición) en la especificación de etiqueta de caso.

Se dice que un valor empareja con una lista de etiquetas de caso, si y sólo si:

- la lista de etiquetas de caso consta de etiquetas de caso y el valor es uno de los valores indicados explícitamente por una de las etiquetas de caso,

- la lista de etiquetas de caso consiste en *(ELSE)* y el valor es uno de los valores implícitamente indicados por *(ELSE)*

- la lista de etiquetas de caso consiste en *indiferente*.

Los valores indicados explícitamente por una etiqueta de caso, son valores entregados por cualquier *expresión discreta* o definidos por un *intervalo literal* o *nombre de modo discreto*. Los valores indicados implícitamente por *(ELSE)*, son todos los valores posibles del selector que no están indicados explícitamente en alguna lista de etiquetas de caso asociada (es decir perteneciente al mismo valor del selector) en alguna especificación de etiqueta de caso.

propiedades estáticas:

- Un *campo alternativo* con *especificación de etiqueta de caso*, una *tupla matricial etiquetada*, o una *acción caso* tienen asociada una lista de especificaciones de etiqueta de caso formada tomando la *especificación de etiqueta de caso* enfrente de cada *alternativa variable*, *valor* o *alternativa de caso*, respectivamente.

- Una *etiqueta de caso*, tiene asociada una clase que es, si se trata de una *expresión literal discreta*, la clase de la *expresión literal discreta*; si es un *intervalo literal*, la clase resultante de las clases de cada *expresión literal discreta* en *intervalo literal*; si es un *nombre de modo discreto*, la clase resultante de la clase M-valuada, donde M es un *nombre de modo discreto*.

- Una *lista de etiquetas de caso*, tiene asociada una clase que es, si se trata de *(ELSE)* o <*indiferente*>, la clase general; en cualquier otro caso, la clase resultante de las clases de cada *etiqueta de caso*.

- Una *especificación de etiqueta de caso*, tiene asociada una lista de clases que son las clases de cada etiqueta de caso.

- Una lista de especificaciones de etiquetas de caso, tiene asociada una lista resultante de clases (siempre que las especificaciones de etiqueta de caso tengan el mismo número de clases; éste será siempre el caso). Esta lista de clases resultante se forma determinando, para cada posición de la lista, la clase resultante de todas las clases que tienen esta posición.

Una lista de especificaciones de etiquetas de caso está completa, si y solo si para todas las listas de posibles valores del selector está presente una especificación de etiqueta de caso, que empareja con la lista de valores del selector. El conjunto de todos los posibles valores del selector, se determina según el contexto como sigue:

- Para un modo estructura variable con marcadores, es el conjunto de valores definidos por el modo del campo marcador correspondiente.
- Para un modo estructura variable sin marcadores, es el conjunto de valores definidos por el modo raíz de la clase resultante correspondiente (esta clase no es nunca la clase general, véase el punto 3.10.4).
- Para una tupla matricial, es el conjunto de valores definidos por el modo índice de cada modo de la tupla matricial.
- Para una acción caso con lista de intervalos, es el conjunto de valores definidos por el modo discreto correspondiente en la lista de intervalos.
- Para una acción caso sin lista de intervalos, es el conjunto de valores definidos por M tal que la clase del selector correspondiente, es la clase de M-valuada, o la clase M-derivada.

condiciones estáticas: Para cada *especificación de etiqueta de caso*, el número de ocurrencias de *lista de etiquetas de caso*, debe ser igual.

Para dos ocurrencias cualesquiera de *especificación de etiqueta de caso*, sus listas de clases deben ser compatibles.

La lista de ocurrencias de *especificación de etiqueta de caso* debe ser coherente, es decir, que cada lista de posibles valores del selector se ajuste, como máximo, a una especificación de etiqueta de caso.

ejemplos:

11.7	*(occupied)*	*(3.1)*
11.62	*(rook),(*)*	*(1.1)*
8.24	*(ELSE)*	*(2.2)*

9.1.4 DEFINICIÓN Y RESUMEN DE LAS CATEGORÍAS SEMÁNTICAS

Este punto resume las categorías semánticas que se indican en la descripción de la sintaxis mediante una parte subrayada. Si estas categorías no se han definido en los puntos apropiados, se facilita la definición aquí, de lo contrario se dará la referencia del punto adecuado.

9.1.4.1 Nombres

Nombres de modo

nombre de modo matriz:	*nombre* definido por un modo matriz.
nombre de modo booleano:	*nombre* definido por un modo booleano.
nombre de modo de referencia ligada:	*nombre* definido por un modo de referencia ligada.
nombre de modo tampón:	*nombre* definido por un modo tampón.
nombre de modo carácter:	*nombre* definido por un modo carácter.
nombre de modo discreto:	*nombre* definido por un modo discreto.
nombre de modo suceso:	*nombre* definido por un modo suceso.
nombre de modo de referencia libre:	*nombre* definido por un modo de referencia libre.
nombre de modo ejemplo:	*nombre* definido por un modo ejemplo.
nombre de modo entero:	*nombre* definido por un modo entero.
nombre de modo:	véase apartado 3.2.1
nombre de neomodo:	véase apartado 3.2.3
nombre de modo matriz parametrizada:	*nombre* definido por un modo matriz parametrizado.
nombre de modo cadena parametrizada:	*nombre* definido por un modo cadena parametrizado.
nombre de modo estructura parametrizada:	*nombre* definido por un modo estructura parametrizada.
nombre de modo conjuntista:	*nombre* definido por un modo conjuntista.
nombre de modo procedimiento:	*nombre* definido por un modo procedimiento.
nombre de modo intervalo:	*nombre* definido por un modo intervalo.
nombre de modo descriptor:	*nombre* definido por un modo descriptor.
nombre de modo conjunto:	*nombre* definido por un modo conjunto.
nombre de modo cadena:	*nombre* definido por un modo cadena.
nombre de modo estructura:	*nombre* definido por un modo estructura.
nombre de sinmodo:	véase apartado 3.2.2
nombre de modo estructura variable:	*nombre* definido por un modo estructura variable.

Nombres de acceso

nombre referido a una base:	véase apartado 4.1.4
nombre de localización:	véanse apartados 4.1.2, 7.4
nombre de localización do-with:	véase apartado 6.5.4
nombre de enumeración de localización:	véase apartado 6.5.2
nombre de identidad-loc:	véanse apartados 4.1.3, 7.4

Nombres de valor

nombre de sinónimo:	véase apartado 5.1
nombre de valor do-with:	véase apartado 6.5.2
nombre de enumeración de valor:	véase apartado 6.5.4
nombre de valor a recibir:	véanse apartados 6.19.2, 6.19.3

Nombres varios

nombre de localización de referencia libre o ligada:	*nombre de localización* con un modo de referencia ligada o un modo de referencia libre.
nombre de rutina predefinida:	cualquier nombre definido de realización que designe una rutina predefinida de realización.
nombre de campo:	véase apartado 3.10.4
nombre de procedimiento general:	*nombre de procedimiento* cuya generalidad es general
nombre de etiqueta:	véase apartado 6.1
nombre de módulo:	véase apartado 7.6
nombre no reservado:	*nombre* que no es ninguno de los nombres reservados mencionados en el apéndice C1
nombre de procedimiento:	véase apartado 7.4
nombre de proceso:	véase apartado 7.5
nombre de región:	véase apartado 7.7
nombre de registro:	nombre definido en la realización que designa un registro de máquina.
lista de nombres reservados:	*lista de nombres* que consta únicamente de nombres reservados (véase apéndice C1)
nombre de elemento de conjunto:	véase apartado 3.4.5
nombre de señal:	véase apartado 8.5.2
nombre de campo marcador:	véase apartado 3.10.4
nombre de sinónimo indefinido:	véase apartado 5.1

9.1.4.2 Localizaciones

localización matriz:	*localización* con un modo matriz.
localización tampón:	*localización* con un modo tampón.
localización suceso:	*localización* con un modo suceso.
localización ejemplo:	*localización* con un modo ejemplo.
localización cadena:	*localización* con un modo cadena.
localización estructura:	*localización* con un modo estructura.

9.1.4.3 Expresiones

expresión matricial:	*expresión* de clase compatible con un modo matriz.
expresión booleana:	*expresión* de clase compatible con un modo booleano.
expresión de referencia ligada:	*expresión* de clase compatible con un modo de referencia ligada.
expresión discreta:	*expresión* de clase compatible con un modo discreto.
expresión literal discreta:	*expresión discreta* que es literal.
expresión de referencia libre:	*expresión* de clase compatible con un modo de referencia libre.
expresión ejemplo:	*expresión* de clase compatible con un modo ejemplo.
expresión entera:	*expresión* de clase compatible con un modo entero.
expresión literal entera:	*expresión entera* que es literal.

expresión conjuntista:	*expresión* de clase compatible con un modo conjuntista.
expresión procedimiento:	*expresión* de clase compatible con un modo procedimiento.
expresión descriptor:	*expresión* de clase compatible con un modo descriptor.
expresión cadena:	*expresión* de clase compatible con un modo cadena.
expresión estructura:	*expresión* de clase compatible con un modo estructura.

condiciones estáticas: Ni una *expresión booleana* ni una *expresión discreta* (cuando se indica en la sintaxis), pueden tener una clase dinámica. Es decir la verificación de si la *expresión* es compatible con un modo booleano o un modo discreto puede realizarse estáticamente.

9.1.4.4 Categorías semánticas varias

llamada a rutina predefinida que produce valor de realización:	véase apartado 11.1.3
llamada a procedimiento que proporciona una localización:	véase apartado 6.7
sentencia de acción módulo:	*sentencia de acción* en la que la *acción* directamente contenida es un *módulo*.
carácter distinto del apóstrofo	*carácter* que no es un apóstrofo.
llamada a un procedimiento que proporciona un valor:	véase apartado 6.7

9.2 VISIBILIDAD E IDENTIFICACIÓN

9.2.1 CONSIDERACIONES GENERALES

Los elementos constructivos específicos CHILL mencionados en el punto 7.1, crean nombres nuevos dentro de un programa. Las sentencias de estructuración del programa y las sentencias de visibilidad determinan la visibilidad de los nombres a lo largo del programa. Este punto trata de la visibilidad de los nombres con exclusión de los nombres de excepción, es decir, cada nombre se considera que no está en el contexto de un *nombre de excepción*. Para los nombres de excepción, véase el capítulo 10.

Para que sea posible una descripción precisa de la estructura de visibibilidad de un programa, se introducen justamente para este punto 9.2 los siguientes perfeccionamientos de terminología:

- Una cadena de nombre (de un nombre) es una cadena de caracteres (utilizada como designación de un nombre) considerada como un elemento de léxico aislado de cualquier contexto. Un nombre es una cadena de nombre asociada a una definición (ocurrencia de definición, véase el punto 9.2.2) de dicha cadena de nombre.

Ejemplo:

```
B : BEGIN
    MODULE DCL I INT; END;
    MODULE DCL I PTR; END;
    END B;
```

En el cuerpo principio-final del bloque etiquetado *B* se introducen dos nombres, ambos con la cadena de nombres *I*.

Dentro de un dominio, cada nombre tiene uno de los cuatro grados siguientes de visibilidad:

Cuadro 1. Grados de visibilidad

Visibilidad	Propiedades (informales)
fuertemente visible directamente	El nombre es visible por creación, cesión o toma
fuertemente visible indirectamente	El nombre es heredado a través de una fasciculación de bloques o mediante su atributo de penetración
débilmente visible	El nombre viene implicado por un nombre fuertemente visible
invisible	El nombre no puede aplicarse

Se dice que un nombre es fuertemente visible si es fuertemente visible directamente o si es fuertemente visible indirectamente. Se dice que un nombre es visible si es débilmente o fuertemente visible, de lo contrario se dice que el nombre es invisible. Las sentencias de estructuración del programa y las sentencias de visibilidad, determinan unívocamente a qué clase de visibilidad pertenece cada nombre. Las propiedades precisas de las clases de visibilidad se explican en los puntos siguientes.

La identificación, es el mecanismo de asociación de un nombre único a cualquier cadena de nombre, es decir se asocia un significado único a la cadena de nombre.

9.2.2 VISIBILIDAD Y CREACIÓN DE NOMBRES

Los nombres se crean mediante los elementos constructivos mencionados en el punto 7.1. Excepto para los nombres de campo y los nombres de elemento de conjunto, los nombres tienen una sola ocurrencia de definición, que es el elemento constructivo que introduce el nombre. A fin de dar un tratamiento uniforme a cualquier nombre al establecer la visibilidad y la unión de nombres, se considera que se aplica el siguiente mecanismo para proporcionar una ocurrencia de definición única a cualquier nombre creado:

- Dentro del dominio de un grupo, cada ocurrencia de *modo* se considera que es una ocurrencia aplicada de un nombre de sínmodo virtual, definido dentro de este dominio. Para las definiciones de procedimiento, la definición de sínmodo virtual del modo de resultado se sitúa en el dominio del grupo que rodea al procedimiento. Las definiciones de sínmodo virtual de los modos de parámetro formal, se sitúan en el dominio del procedimiento.

Se aplican las reglas de visibilidad y de identificación, teniendo en cuenta estas definiciones virtuales

Ejemplo:

```
DCL I SET(A,B),
    K INT,
    J ARRAY (SET(A,B)) INT;
```

se considera que se reemplaza por:

```
SYNMODE &1 = SET(A,B), &2 = INT;
        &3 = ARRAY (&1)&2;
DCL I &1, K &2, J &3;
```

&1, *&2* y *&3* son nombres de sínmodos virtuales. Las reglas de visibilidad se aplican a estas sustituciones virtuales. Las sustituciones virtuales, tienen como consecuencia que los modos de creación de nombres (*SET*, *STRUCT*) aparezcan una sola vez en un dominio, a la derecha de las definiciones de sínmodo virtual. Esta definición de sínmodo se considera como una ocurrencia de definición única de los nombres de elemento de conjunto o de los nombres de campo.

Las propiedades de visibilidad y de identificación de los nombres de campo son diferentes (más sencillas) que las de otros nombres. Por tanto, en lo que resta del punto 9.2, la palabra "nombre" no incluye los nombres de campo, a menos que se indique lo contrario.

9.2.3 NOMBRES IMPLICADOS

Cada nombre fuertemente visible en un dominio, tiene un conjunto (posiblemente vacío) de nombres implicados, definidos como sigue:

- Cada modo tiene un conjunto (posiblemente vacío) de nombres implicados enumerados en el cuadro 2.

Los nombres implicados de un nombre (fuertemente visible) son:

- Si el nombre es un nombre de acceso, los nombres implicados son los nombres implicados por el modo del nombre de acceso.

- Si el nombre es un nombre de modo, los nombres implicados son los nombres implicados por el modo definidor.

- Si el nombre es un nombre de procedimiento, los nombres implicados son los nombres implicados por el modo de la especificación de resultado.

- Si el nombre es un nombre de señal, los nombres implicados son todos nombres implicados por sus modos adjuntos.

- En cualquier otro caso no hay nombres implicados.

Cuadro 2. Nombres implicados de modos

Modos	conjunto de nombres implicados
INT, BIN, CHAR **INSTANCE, PTR** **BOOL, EVENT** **CHAR(n), BIN(n)** **BIT(n), RANGE(....)**	Vacío
nombre de modo	El conjunto de nombres implicados de su modo de definición
M(m:n)	El conjunto de nombres implicados de *M*
REF M, ROW M **READ M, POWERSET M** **PROC(---) (M)** **BUFFER M**	El conjunto de nombres implicados de *M*
ARRAY (M) N	La unión de los conjuntos de los nombres implicados por *M* y *N*
STRUCT(N_1 M_1,---,N_n M_n)	La unión de los conjuntos de nombres implicados por M_1 hasta M_n. Para las estructuras variables, la unión de los nombres implicados de todos los campos de la estructura variable
Parametrizado *VM (---)*	El conjunto de nombres implicados de *VM*
SET(.....)	El conjunto de los nombres de elemento de conjunto

(obsérvese que los nombres implicados, siempre que sean nombres de elemento de conjunto, nunca tienen ellos mismos nombres implicados.)

9.2.4 VISIBILIDAD EN LOS DOMINIOS

Un nombre, que es fuertemente visible en un dominio, es o fuertemente visible directamente o fuertemente visible indirectamente en él.

Un nombre es fuertemente visible directamente en un dominio, únicamente en los siguientes casos:

- El nombre tiene su ocurrencia de definición en el dominio.
- El nombre está tomado dentro de un dominio (véase el punto 9.2.6.3)
- El nombre está cedido dentro del dominio (véase el punto 9.2.6.2)

Un nombre es fuertemente visible indirectamente en un dominio, únicamente en los siguientes casos:

- El dominio es un dominio de bloque y el nombre es heredado (véase el punto 9.2.5).
- El nombre es fuertemente visible en dominio que lo engloba y tiene la propiedad de penetración en este dominio (véase apartado 9.2.6.2) y el dominio no tiene ningún nombre fuertemente visible directamente en él con la misma cadena de nombre. En este caso, el nombre tiene también la propiedad de penetración en el dominio.

Un nombre es débilmente visible en un dominio, únicamente en el siguiente caso:

- El nombre está implicado por un nombre fuertemente visible en este dominio.

Las reglas de visibilidad se definen de tal forma que, en cualquier dominio, todos los nombres fuertemente visibles tienen diferentes cadenas de nombres. Sin embargo, dos o más nombres débilmente visibles pueden tener la misma cadena de nombre. En algunos casos no se puede aplicar entonces tal nombre (véase apartado 9.2.8).

9.2.5 VISIBILIDAD Y BLOQUES

A los bloques se les aplica la siguiente regla de visibilidad:

Un nombre, fuertemente visible en el dominio de un grupo, es fuertemente visible indirectamente en el dominio de cada bloque encerrado directamente que no tiene un nombre fuertemente visible directamente con la misma cadena de nombre (herencia de nombres por bloques).

9.2.6 VISIBILIDAD Y MODULIONES

9.2.6.1 Consideraciones generales

sintaxis:

<sentencia de visibilidad >::= *(1)*
<sentencia de cesión> *(1.1)*
| *<sentencia de toma>* *(1.2)*

semántica: Las sentencias de visibilidad, que se permiten únicamente en dominios de modulión, controlan la visibilidad de los nombres explícitamente mencionados en ellos (e implícitamente sus nombres implicados).

9.2.6.2 Sentencias de cesión

sintaxis:

<sentencia de cesión> ::= (1)
GRANT <ventana de cesión> [PERVASIVE]; (1.1)

<ventana de cesión> ::= (2)
*<elemento cedido> {,<elemento cedido>}** (2.1)
| *ALL* (2.2)

<elemento cedido> ::= (3)
<nombre no reservado> (3.1)
| *<nombre de neomodo> <cláusula prohibir>* (3.2)

<cláusula prohibir> ::= (4)
FORBID {<lista de nombres prohibidos> | ALL} (4.1)

<lista de nombres prohibidos> ::= (5)
(<nombre de campo> {, nombre de campo>})* (5.1)

semántica: Las sentencias de cesión, son procedimientos de extensión de la visibilidad de los nombres en un dominio de modulión dentro del dominio que lo engloba directamente. *FORBID* puede especificarse únicamente para nombres de neomodos que sean modos estructura. Esto significa que todas las localizaciones y valores de este modo tienen campos que pueden seleccionarse únicamente dentro del módulo de cesión y no fuera.

Se aplican las siguientes reglas de visibilidad:

- Un nombre, visible en el dominio de un modulión, es fuertemente visible directamente, en el dominio del grupo directamente circundante, si se menciona en una sentencia de cesión en el dominio del modulión. Se dice que el nombre está cedido en el dominio circundante.

- La notación *FORBID ALL*, es una abreviatura sintáctica que prohibe todos los nombres de campo de un *nombre de neomodo* (véase apartado 9.2.7).

- La notación *GRANT ALL [PERVASIVE]*, es una abreviatura sintáctica para la cesión de todos los nombres (con la propiedad de penetración, si se especifica) que son fuertemente visibles en el dominio del modulión de cesión y cuyas ocurrencias de definición están dentro del modulión de cesión.

propiedades estáticas: Un nombre cedido con el atributo *PERVASIVE*, tiene la propiedad de penetración en el dominio que lo engloba.

condiciones estáticas: La ocurrencia de definición de cualquier *nombre no-reservado* debe estar dentro del modulión de cesión.

El *nombre de neomodo* con la especificación *FORBID*, debe tener su ocurrencia de definición en el dominio del modulión de cesión y debe ser un modo estructura y cada *nombre de campo* en una *lista de nombres prohibidos*, debe ser un nombre de campo del *nombre de neomodo*.

Si una sentencia de cesión se coloca en el dominio de una región no debe ceder un nombre que sea un nombre de valor regional o un nombre de acceso regional.

ejemplos:

1.11 GRANT add,mult; *(1.1)*

9.2.6.3 Sentencias de toma

sintaxis:

<sentencia de toma> ::= *(1)*
SEIZE <ventana de toma>; *(1.1)*

<ventana de toma> ::= *(2)*
*< elemento tomado> {,<elemento tomado>}** *(2.1)*
| ALL *(2.2)*

<elemento tomado> ::= *(3)*
< nombre de modulión> ALL *(3.1)*
|< nombre no reservado> *(3.2)*

<nombre de modulión> ::= *(4)*
< nombre de módulo> *(4.1)*
< nombre de región> *(4.2)*

semántica: Las sentencias de toma constituyen un procedimiento de extender la visibilidad de los nombres en los dominios de grupo a los dominios de los moduliones directamente encerrados.

Se aplican las siguientes reglas de visibilidad:

- Un nombre, visible el dominio de un grupo, es fuertemente visible directamente, en el dominio de un modulión directamente encerrado, si se menciona en una sentencia de toma en el dominio del modulión. Se dice que el nombre está tomado en el dominio del modulión.

- Si un nombre que tiene la propiedad de penetración en el dominio circundante es tomado, éste será fuertemente visible directamente en el dominio del módulo de toma y conserva la propiedad de penetración.

- La notación *SEIZE ALL*, es una abreviatura sintáctica para tomar todos los nombres que son fuertemente visibles en el dominio del grupo que los engloba y cuyas ocurrencias de definición están fuera del modulión de toma.

- La notación *SEIZE <nombre de modulión> ALL*, es una abreviatura sintáctica para tomar todos los nombres que son fuertemente visibles en el dominio del grupo que los engloba y que están cedidos por el módulo o región designada por el *nombre del modulión*.

condiciones estáticas: La ocurrencia de definición de cualquier *nombre no-reservado* o *nombre de modulión*, debe estar fuera del modulión de toma.

Un nombre mencionado en un *elemento tomado*, no debe ser un *nombre de valor do-with* ni un *nombre de localización do-with*.

ejemplos:

*15.14 SEIZE/*external signals*/*
ACQUIRE, RELEASE, CONGESTED, STEP, READOUT *(1.1)*

9.2.7 VISIBILIDAD DE NOMBRES DE CAMPO

Los nombres de campo, pueden aparecer fuera de su ocurrencia de definición, solamente en el siguiente contexto:

- Selección de campo de una localización estructura o un valor estructura.
- Tuplas de estructuras etiquetadas.
- Cláusulas prohibir en la sentencia de cesión.

En los dos primeros contextos, son visibles aquellos nombres de campo que estén unidos al modo de la localización estructura, al valor estructura (fuerte) o a la tupla, excepto si la novedad de este modo es un nombre de neomodo que ha sido cedido por un modulión con una cláusula prohibir. En este último caso, fuera del modulión de cesión, únicamente son visibles aquellos nombres de campo que no se mencionan en la *cláusula prohibir*.

En el último contexto todos los nombres de campo del nombre de neomodo cedido son visibles, y solamente ellos.

9.2.8 IDENTIFICACIÓN

La unión de nombres, es el mecanismo de asociar un nombre único con cualquier aparición de una cadena de nombre.

Las reglas de unión, dependen de si la cadena de nombres que aparece en el contexto de:

1. un *nombre de directivo*,
2. un *nombre de excepción*,
3. un *nombre reservado*,
4. un *nombre de campo*,
5. un *nombre no-reservado*, un *nombre de módulo*, o un *nombre de región* en un *elemento tomado*,
6. cualquier otro *nombre*.

Reglas de unión

1. Una cadena de nombre de directivo sigue una pauta de unión definida en la realización que no debe influir sobre las reglas de unión del CHILL (véase el punto 2.6)

2. Una cadena de nombre de excepción se trata de acuerdo con las reglas de identificación del programa ejecutor dadas en el punto 10.3

3. Una cadena de nombre reservado que no esté liberada por un directivo libre en una unidad de compilación en la que aparece, tiene su significado reservado. Si está liberada, sigue las reglas del punto 6. Aunque esté liberada en una unidad de compilación, puede no estar cedida fuera de esta unidad.

4. Una cadena de nombre de campo está ligada como sigue, dependiendo de los contextos mencionados en el punto 9.2.7:

   - al nombre de campo visible del modo estructura de la localización estructura o de la expresión estructura (fuerte);
   - al nombre de campo visible del modo estructura de la tupla (fuerte);
   - al nombre de campo visible del nombre de neomodo.

   Si la cadena de nombre no puede adjudicarse a uno de tales nombres de campo, el programa es erróneo.

5. Una cadena de nombre que aparece en el contexto de un *elemento tomado* está ligada según las reglas mencionadas en el punto 6, pero en el dominio que engloba directamente al dominio en el que está situada la sentencia de toma.

6. Para cualquier otra ocurrencia de una cadena de nombre en el dominio de un grupo:

   a. si existe más de un nombre fuertemente visible en el dominio con esta cadena de nombre, entonces el programa es erróneo;

   b. por el contrario si existe un nombre fuertemente visible en el dominio con esta cadena de nombre, la cadena de nombre está ligada a este nombre;

   c. por el contrario si hay exactamente un nombre débilmente visible con esta cadena de nombre en el dominio, la cadena de nombre está ligada a este nombre;

   d. por el contrario si hay más de un nombre débilmente visible en el dominio con esta cadena de nombre y si todos estos nombres (de elemento de conjunto) tienen clases compatibles, entonces la cadena de nombre está ligada a uno de estos nombres (arbitrario);

   e. en cualquier otro caso, el programa es erróneo.

Además de las reglas mencionadas anteriormente, una cadena de nombres que aparezca en una sentencia de cesión o una sentencia de toma debe estar ligada a un nombre cuya aparición de definición esté dentro o fuera del modulión de cesión o de toma, respectivamente, (si existe una elección según la regla d.).

10.1 TRATAMIENTO DE EXCEPCIONES

10.1 CONSIDERACIONES GENERALES

Una excepción puede ser una excepción definida por lenguaje, en cuyo caso puede tener un nombre definido por lenguaje, una excepción definida por el usuario o bien una excepción definida por la realización. Una excepción definida por lenguaje se deberá a una violación dinámica de una condición dinámica. Puede producirse cualquier excepción nominada por la ejecución de una acción causar.

Cuando se produce una excepción, ésta puede tramitarse, es decir se ejecutará una lista de sentencias de acción de un programa ejecutor adecuado.

El tratamiento de las excepciones se define de tal forma que en cualquier sentencia se conoce estáticamente qué excepciones podrían ocurrir (es decir se conoce estáticamente qué excepciones no pueden ocurrir) y para qué excepciones se puede encontrar un programa ejecutor adecuado o qué excepciones pueden pasarse al punto de llamada de un procedimiento. Si se produce una excepción y no se puede encontrar un programa ejecutor para ella, el programa es erróneo.

10.2 EJECUTORES

sintaxis:

<programa ejecutor> ::= *(1)*
*ON {<alternativa-on>}**
[ELSE <lista de sentencias de acción>] END *(1.1)*

<alternativa-on> ::= *(2)*
(<lista de excepciones>) : <lista de sentencias de acción> *(2.1)*

semántica: Se produce la entrada de una lista de sentencias de acción en una alternativa-on, si ocurre una excepción en la sentencia a la cual va asociado el programa ejecutor y cuyo nombre se menciona en la *lista de excepciones* de la alternativa-on. Si se especifica *ELSE*, se efectuará la entrada de la lista de sentencias de acción que la sigue, si ocurre una excepción en la sentencia a la cual está asociado el programa ejecutor y cuyo nombre no está especificado en ninguna *lista de excepciones* directamente contenida en el programa ejecutor.

Si el programa ejecutor está anexado a una acción, cuando se alcanza el final de una lista de sentencias de acción en una alternativa-on, se pasará el control a la sentencia de acción que sigue a la sentencia de acción en la que está situado el programa ejecutor.

Si el programa ejecutor está anexado a una definición de procedimiento, se devolverá el control al punto de llamada cuando se alcance el final de una lista de sentencias de acción. Si el programa ejecutor está anexado a una definición de proceso, el proceso en ejecución terminará cuando se alcance el final de una lista de sentencias de acción en la alternativa-on.

condiciones estáticas: Todos los nombres en todas las ocurrencias de *lista de excepciones* deben ser diferentes.

condiciones dinámicas: Se produce la excepción *SPACEFAIL*, si se entra una lista de sentencias de acción y no se pueden satisfacer los requisitos de almacenamiento.

ejemplos:

```
10.43  ON
         (no_space): CAUSE overflow;
       END                                                    (1.1)
```

10.3 IDENTIFICACIÓN DEL EJECUTOR

Cuando se produce una excepción E en una acción A o en una sentencia de datos o en una región D, se puede tramitar la excepción mediante el programa ejecutor adecuado, es decir en el programa ejecutor se ejecutará una lista de sentencias de acción o puede pasarse la excepción al punto de llamada de un procedimiento o, si nada de lo anterior es posible, el programa es erróneo.

Para cualquier acción A, o sentencia de datos o región D, puede determinarse estáticamente si, para una excepción dada E, en A o en D, puede encontrarse un programa ejecutor adecuado o si la excepción puede pasarse al punto de llamada.

Un programa ejecutor adecuado para A o D, con respecto a E se determina como sigue:

1. si se anexa un programa ejecutor a A o a D, que mencione E en una lista de excepciones o especifique ***ELSE***, entonces este programa ejecutor es el adecuado con respecto a E;

2. si A o D están encerrados directamente por una acción parentizada, el programa ejecutor adecuado (si está presente) es el programa ejecutor adecuado para la acción parentizada con respecto a E;

3. si A o D se sitúan en el dominio de una definición de procedimiento entonces:

   - si se especifica un programa ejecutor después de la definición de procedimiento, cuyo programa ejecutor especifica E en una lista de excepciones, o especifica ***ELSE***, entonces este programa ejecutor es el adecuado,

   - si se menciona E en la lista de excepciones de la definición de procedimiento, entonces se provoca E en el punto de llamada,

   - en cualquier otro caso, no existe programa ejecutor;

4. si A o D se colocan en el dominio de una definición de proceso (posiblemente el imaginario) entonces:

   - si se especifica un programa ejecutor después de la definición de proceso, cuyo programa ejecutor especifica E, en una lista de excepciones, o especifica ***ELSE***, este programa ejecutor es adecuado,

- en cualquier otro caso, no existe programa ejecutor;

5. si A es una acción de una lista de sentencias de acción, el programa ejecutor adecuado es el programa ejecutor adecuado para la acción A' o la definición D' con respecto a E a la que está anexado el programa ejecutor, pero considerado como si este programa ejecutor no estuviera especificado.

Si se produce una excepción y la transferencia del control al programa ejecutor adecuado implica la salida de los bloques, se liberará el almacenamiento local al salir del bloque.

11.0 OPCIONES DE REALIZACIÓN

11.1 RUTINAS PREDEFINIDAS DEFINIDAS EN LA REALIZACIÓN

sintaxis:

```
<llamada a una rutina predefinida> ::=                                  (1)
    <nombre de rutina predefinida>
    ([<lista de parámetros de rutina predefinida>])                     (1.1)

<lista de parámetros de rutina predefinida> ::=                         (2)
    <parámetro de rutina predefinida>
    {, <parámetro de rutina predefinida>}*                              (2.1)

<parámetro de rutina predefinida> ::=                                   (3)
    <valor>                                                             (3.1)
  | <localización>                                                      (3.2)
  | <nombre no reservado>                                               (3.3)
```

semántica: Una realización puede proporcionar un conjunto de rutinas predefinidas en la realización, además del conjunto de rutinas predefinidas en el lenguaje.

Un valor, una localización o cualquier nombre de programa definido que no sea un nombre reservado puede pasarse como parámetro. La llamada a una rutina predefinida, puede proporcionar un valor o una localización. El mecanismo de pasar parámetros se define en la realización.

Una rutina predefinida puede ser genérica, es decir, su clase (si es una llamada a rutina predefinida que produce un valor) o su modo (si es una llamada a rutina predefinida que produce una localización), pueden depender no sólo del nombre de la rutina predefinida, sino también de las propiedades estáticas de los parámetros efectivos pasados y del contexto estático de la llamada.

propiedades estáticas: Un *nombre de rutina predefinida* es un nombre definido en la realización que se considera definido en el módulo de preludio normalizado (véase el punto 7.8). Puede tener asociado un conjunto de nombres de excepción definidos en la realización. Una *llamada a una rutina predefinida*, es una *llamada a una rutina predefinida que produce un valor (localización)*, si y sólo si en la realización se especifica que para una elección dada de las propiedades estáticas de los parámetros y para el contexto estático de la llamada, la llamada a rutina predefinida entrega un valor (localización).

11.2 MODOS ENTEROS DEFINIDOS EN LA REALIZACIÓN

Una realización puede definir modos enteros diferentes de los definidos por *INT*, p.ej. enteros cortos, enteros largos, enteros sin signo. Estos modos enteros, deben designarse mediante nombres de modo entero definidos en la realización.

Estos nombres se consideran como nombres de neomodo, similares a *INT*. Sus gamas de valores se definen en la realización. Estos modos enteros, pueden definirse como modos raíz de las clases adecuadas.

11.3 NOMBRES DE REGISTRO DEFINIDOS EN LA REALIZACIÓN

En una realización se puede definir un conjunto de nombres de registro predefinidos (véanse los puntos 3.7 y 7.8).

11.4 NOMBRES DE PROCESO Y DE EXCEPCIÓN DEFINIDOS EN LA REALIZACIÓN

En una realización se puede definir un conjunto de nombres de procesos definidos en la realización, es decir nombres de procesos cuya definición no está especificada en CHILL. Se considera que la definición está colocada en el dominio del módulo de preludio normalizado. Los procesos de este nombre pueden iniciarse y los valores de ejemplos que designan tales procesos pueden manipularse.

En una realización se puede definir un conjunto de nombres de excepción para cualquier nombre de proceso o grupo de nombres de procesos. Estas excepciones pueden causarse al arrancar el proceso (véase apartado 6.14).

11.5 EJECUTORES DEFINIDOS EN LA REALIZACIÓN

Una realización puede especificar que un programa ejecutor definido en la realización está anexado a la definición del proceso imaginario más externo (véase el punto 7.8). Los nombres de excepción y las acciones del programa ejecutor definido en la realización, pueden especificar cualquier nombre de excepción o acción CHILL que sean legales. Obsérvese que solamente puede entrar en tal programa de manejo una alternativa-on, mediante una excepción provocada por el proceso más externo y no por un proceso interior.

11.6 OPCIONES DE SINTAXIS

En algunos lugares, CHILL permite más de una descripción sintáctica para la misma semántica. La elección de una de las siguientes opciones debiera fijarse dentro del programa total.

Símbolo de asignación

El signo de asignación es := ó =

ARRAY

El nombre reservado *ARRAY* debiera ser preceptivo o no permitido.

RETURNS

En las definiciones de procedimiento con una especificación de resultado, el nombre reservado *RETURNS*, debiera ser preceptivo o no permitido.

Modos estructura

Los modos estructura deben estar o bien en la notación de estructura fascicular o en la notación numerada en niveles.

Paréntesis de literal y de tupla

En el caso de que en la representación del alfabeto se disponga de corchetes [y], pueden utilizarse en vez de *(:* y *:)*, respectivamente.

APÉNDICE A: JUEGOS DE CARACTERES PARA LOS PROGRAMAS CHILL

A.1 VERSIÓN DE REFERENCIA DEL ALFABETO INTERNACIONAL N.º 5 DEL CCITT

Recomendación V3 (la representación interna es el número binario formado por los bits b7 a b1, donde b1 es el bit menos significativo).

				b7	0	0	0	0	1	1	1	1
				b6	0	0	1	1	0	0	1	1
				b5	0	1	0	1	0	1	0	1
					0	1	2	3	4	5	6	7
b_4	b_3	b_2	b_1									
0	0	0	0	0	NUL	TC_7	SP	0	@	P	`	p
0	0	0	1	1	TC_1	DC_1	!	1	A	Q	a	q
0	0	1	0	2	TC_2	DC_2	"	2	B	R	b	r
0	0	1	1	3	TC_3	DC_3	#	3	C	S	c	s
0	1	0	0	4	TC_4	DC_4	¤	4	D	T	d	t
0	1	0	1	5	TC_5	TC_8	%	5	E	U	e	u
0	1	1	0	6	TC_6	TC_9	&	6	F	V	f	v
0	1	1	1	7	BEL	TC_{10}	'	7	G	W	g	w
1	0	0	0	8	FE_0	CAN	(	8	H	X	h	x
1	0	0	1	9	FE_1	EM	)	9	I	Y	i	y
1	0	1	0	10	FE_2	SUB	*	:	J	Z	j	z
1	0	1	1	11	FE_3	ESC	+	;	K	[	k	{
1	1	0	0	12	FE_4	IS_4	,	<	L	\	l	|
1	1	0	1	13	FE_5	IS_3	-	=	M	]	m	}
1	1	1	0	14	SO	IS_2	.	>	N	^	n	~
1	1	1	1	15	SI	IS_1	/	?	O	_	o	DEL

A.2 JUEGO MÍNIMO DE CARACTERES PARA REPRESENTAR LOS PROGRAMAS CHILL

En este documento se utiliza para representar programas CHILL el subconjunto siguiente del código Básico de alfabeto N.º 5 del CCITT.

b_7	0	0	0	0	1	1	1	1
b_6	0	0	1	1	0	0	1	1
b_5	0	1	0	1	0	1	0	1
	0	1	2	3	4	5	6	7
b_4 b_3 b_2 b_1								
0 0 0 0	0		SP	0		P		
0 0 0 1	1			1	A	Q		
0 0 1 0	2			2	B	R		
0 0 1 1	3			3	C	S		
0 1 0 0	4			4	D	T		
0 1 0 1	5			5	E	U		
0 1 1 0	6			6	F	V		
0 1 1 1	7		'	7	G	W		
1 0 0 0	8		(	8	H	X		
1 0 0 1	9		)	9	I	Y		
1 0 1 0	10		¤	:	J	Z		
1 0 1 1	11		+	;	K			
1 1 0 0	12		,	<	L			
1 1 0 1	13		-	=	M			
1 1 1 0	14		.	>	N			
1 1 1 1	15		/		O	_		

APÉNDICE B: SÍMBOLOS ESPECIALES

;	punto y coma, fin de sentencia
,	coma
(	abrir paréntesis
)	cerrar paréntesis
[	abrir corchetes
]	cerrar corchetes
(:	abrir corchetes de tupla
:)	cerrar corchetes de tupla
:	dos puntos
.	punto
:=	símbolo de asignación
<	menor que
<=	menor o igual
=	igual
/=	distinto
>=	mayor o igual
>	mayor
+	más
-	menos
*	asterisco
/	dividir
//	concatenación
->	símbolo de referenciar y de desrreferenciar
<>	paréntesis de abrir y cerrar directivo de compilador
/*	paréntesis de abrir comentario
*/	paréntesis de cerrar comentario
'	apóstrofo
"	doble apóstrofo
-	guión

APÉNDICE C: NOMBRES ESPECIALES CHILL

C.1 NOMBRES RESERVADOS

ALL
ARRAY
ASSERT

BASED
BEGIN
BUFFER
BY

CALL
CASE
CAUSE
CONTINUE

DCL
DELAY
DO
DOWN

ELSE
ELSIF
END
ENTRY
ESAC
EVENT
EVER
EXCEPTIONS
EXIT

FI
FOR
FORBID

GENERAL
GOTO
GRANT

IF
IN
INIT
INLINE
INOUT

LOC

MODULE

NEWMODE
NOPACK

OD
OF
ON
OUT

PACK
PERVASIVE
POS
POWERSET
PRIORITY
PROC
PROCESS

RANGE
READ
RECEIVE
RECURSIVE
REF
REGION
RESULT
RETURN
RETURNS
ROW

SEIZE
SEND
SET
SIGNAL
SIMPLE
START
STATIC
STEP
STOP
STRUCT
SYN
SYNMODE

THEN
TO

UP

WHILE
WITH

C.2 NOMBRES PREDEFINIDOS

ABS	FALSE	NOT	SIZE
ADDR		NULL	SUCC
AND		NUM	
	GETSTACK		
			THIS
BIN		OR	TRUE
BIT	INSTANCE		
BOOL	INT		
		PRED	UPPER
		PTR	
CARD	MAX		
CHAR	MIN		XOR
	MOD	REM	

C.3 NOMBRES DE EXCEPCIÓN CHILL

ASSERTFAIL
DELAYFAIL
EMPTY
EXTINCT
MODEFAIL
OVERFLOW
RANGEFAIL
RECURSEFAIL
SPACEFAIL
TAGFAIL

C.4 DIRECTIVOS CHILL

FREE

1. Operaciones con enteros

```
integer_operations:
MODULE
   add:
   PROC (i,j INT)(INT) EXCEPTIONS (OVERFLOW);
      RESULT i+j;
   END add;
   mult:
   PROC (i,j INT)(INT) EXCEPTIONS (OVERFLOW);
      RESULT i*j;
   END mult;
   GRANT add, mult;
   SYNMODE operand_mode=INT;
   GRANT operand_mode;
   SYN neutral_for_add=0,
      neutral_for_mult=1;
   GRANT neutral_for_add,
         neutral_for_mult;
END integer_operations;
```

2. Las mismas operaciones con fracciones

```
fraction_operations:
MODULE
   NEWMODE fraction=STRUCT (num,denum INT);
   add:
   PROC (f1,f2 fraction)(fraction) EXCEPTIONS (OVERFLOW);
      RETURN [f1.num*f2.denum+f2.num*f1.denum,
          f1.denum*f2.denum];
   END add;
   mult:
   PROC (f1,f2 fraction)(fraction) EXCEPTIONS (OVERFLOW);
      RETURN [f1.num*f2.num,f2.denum*f1.denum];
   END mult;

   GRANT add, mult;
   SYNMODE operand_mode=fraction;
   GRANT operand_mode;
   SYN neutral_for_add fraction=[0,1],
      neutral_for_mult fraction=[1,1];
   GRANT neutral_for_add,
        neutral_for_mult;

END fraction_operations;
```

3. Las mismas operaciones con números complejos

```
1  complex_operations
2  MODULE
3     NEWMODE complex=STRUCT (re,im INT);
4     add:
5     PROC (c1,c2 complex)(complex) EXCEPTIONS (OVERFLOW);
6        RETURN [c1.re+c2.re,c1.im+c2.im];
7     END add;
8     mult:
9     PROC (c1,c2 complex)(complex) EXCEPTIONS (OVERFLOW);
10       RETURN [c1.re*c2.re-c1.im*c2.im ,
11               c1.re*c2.im+c1.im*c2.re];
12    END mult;
13
14    GRANT add, mult
15    SYNMODE operand_mode=complex;
16    GRANT operand_mode;
17    SYN neutral_for_add=complex [0,0],
18       neutral_for_mult=complex [1,0];
19    GRANT neutral_for_add,
20          neutral_for_mult;
21
22 END complex_operations;
```

4. Aritmética de orden general

```
1  general_order_arithmetic: /*from collected algorithms from CACM no.93*/
2  MODULE
3     op:
4     PROC (a INOUT, b,c,order INT) EXCEPTIONS (wrong_input) RECURSIVE;
5        DCL d INT;
6        ASSERT b>0 AND c>0 AND order>0
7           ON (ASSERTFAIL):
8              CAUSE wrong_input;
9           END;
10       CASE order OF
11          (1): a :=b+c;
12             RETURN;
13          (2): d :=0;
14          (ELSE): d :=1;
15       ESAC;
16       DO FOR i :=1 TO c;
17          op (a,b,d,order-1);
18          d :=a;
19       OD;
20       RETURN;
21    END op;
22
23    GRANT op;
24
25 END general_order_arithmetic;
```

5. Sumar bit a bit y verificar el resultado

```
1  add_bit_by_bit:
2  MODULE
3        adder:
4        PROC (a STRUCT(a2,a1 BOOL) IN, b STRUCT(b2,b1 BOOL) IN)
5            RETURNS(STRUCT(c4,c2,c1 BOOL));
6
7        DCL c STRUCT (c4,c2,c1 BOOL);
8        DCL k2,x,w,t,s,r BOOL;
9        DO WITH a,b,c;
10           k2 :=a1 AND b1;
11           c1 :=NOT k2 AND (a1 OR b1);
12           x :=e2 AND b2 AND k2;
13           w :=a2 OR b2 OR k2;
14           t :=b2 AND k2;
15           s :=a2 AND k2;
16           r :=a2 AND b2;
17           c4 :=r OR s OR t;
18           c2 :=x OR (w AND NOT c4);
19       OD;
20       RETURN c;
21    END adder;
22    GRANT adder;
23 END add_bit_by_bit;
24
25 exhaustive_checker:
26 MODULE
27    SEIZE adder;
28    DCL a STRUCT (a2,a1 BOOL),
29       b STRUCT (b2,b1 BOOL),
30    SYNMODE res=ARRAY (1:16) STRUCT (c4,c2,c1 BOOL);
31    DCL r INT, results res;
32    DO WITH a,b;
33       r :=0;
34       DO FOR a2 IN BOOL;
35          DO FOR a1 IN BOOL;
36             DO FOR b2 IN BOOL;
37                DO FOR b1 IN BOOL;
38                   r+ :=1;
39                   results (r) :=adder (a,b);
40                OD;
41             OD;
42          OD;
43       OD;
44       ASSERT result=res [[FALSE,FALSE,FALSE],[FALSE,FALSE,TRUE],
45                          [FALSE,FALSE,TRUE],[FALSE,TRUE,TRUE],
46                          [FALSE,FALSE,TPUE],[FALSE,TRUE,FALSE],
47                          [FALSE,TRUE,TRUE],[TRUE,FALSE,FALSE],
48                          [FALSE,TRUE,FALSE],[FALSE,TRUE,FALSE],
49                          [TRUE,FALSE,FALSE],[TRUE,FALSE,TRUE.],
50                          [FALSE,TRUE,TRUE],[TRUE,FALSE,FALSE],
51                          [TRUE,FALSE,TRUE],[TRUE,TRUE,FALSE]];
52 END exhaustive_checker;
```

6. Operaciones con fechas

```
playing_with_dates:
MODULE /* from collected algorithms from CACM no. 199 */
   SYNMODE month=SET(jan,feb,mar,apr,may,jun,
                     jul,aug,sep,oct,nov,dec);
   NEWMODE date=STRUCT (day INT (1:31), mo month, year INT);

   gregorian-date:
   PROC (julian_day_number INT)(date);
      DCL j INT :=julian_day_number,
          d,m,y INT;
      j-:=1_721_119;
      y :=(4 * j - 1) / 146_097;
      j :=4 * j - 1 - 146_097 * y;
      d :=j / 4;
      j :=(4 * d + 3) / 1_461;
      d :=4 * d + 3 - 1_461 * j;
      d :=(d + 4) / 4;
      m :=(5 * d - 3) / 153;
      d :=5 * d - 3 - 153 * m;
      d :=(d + 5) / 5;
      y :=100 * y + j;
      IF m<100 THEN m+:=3;
              ELSE m-:=9;
                   y+:=1;
      FI;
      RETURN [d,month (m+1), y];
   END gregorian_date;

   julian_day_number
   PROC (d date)(INT);
      DCL c,y,m INT;
      DO WITH d;
         m :=NUM (mo)+1;
         IF m>2 THEN m-:=3;
               ELSE m+:=9;
                    year-:=1;
         FI;
         c :=year/100;
         y :=year-100*c;
         RETURN (146_097*c)/4+(1_461*y)/4
            +(153*m+c)/5+day+1_721_119;
      OD;
   END julian_day_number;
   GRANT gregorian_date, julian_day_number;
END playing_with_dates

test:
MODULE
  SEIZE gregorian_date, julian_day_number;
  ASSERT julian_day_number ([10,dec,1979])=julian_day_number(
        gregorian_date(julian_day_number([10,dec,1979])));
END test;
```

7. Números romanos

```
1  Roman:
2  MODULE
3     SEIZE n,rn;
4     convert:
5     PROC () EXCEPTIONS (string_too_small);
6        DCL r INT :=0;
7        DO WHILE n>=1_000;
8           rn(r):='M';
9           r+:=1;
10          n-:=1_000;
11       OD;
12       IF n>500 THEN rn(r):='D';
13                    n-:=500;
14                    r+:=1;
15       FI;
16       IF n>=100 THEN rn(r):='C';
17                     n-:=100;
18                     r+:=1;
19       FI;
20       IF n>=50 THEN rn(r):='L';
21                    n-:=50;
22                    r+:=1;
23       FI;
24       IF n>=10 THEN rn(r):='X';
25                    n-:=10;
26                    r+:=1;
27       FI;
28       DO WHILE n>=1;
29          rn(r):='I';
30          r+:=1;
31          n-:=1;
32       OD;
33       RETURN;
34    END ON (RANGEFAIL): DO FOR i :=0 TO UPPER (rn);
35                           rn(i) :='.';
36                        OD;
37                        CAUSE string_too_small;
39    END convert;
40 END Roman;
41 test:
42 MODULE
43    SEIZE convert;
44    DCL n INT INIT :=1979;
45    DCL rn CHAR (20) INIT :=(20)' ';
46    GRANT n,rn;
47
48    convert ();
49
50    ASSERT rn='MDCCCCLXXVIIII'//(6)' ';
51
52 END test;
```

8. Cómputo de letras en una cadena de caracteres de longitud arbitraria

```
1  letter_count:
2  MODULE
3     SEIZE max;
4     DCL letter POWERSET CHAR INIT :=['A' : 'Z'];
5     count:
6     PROC (input ROW CHAR (max) IN, output ARRAY('A':'Z') INT OUT);
7        DO FOR i :=0 TO UPPER (input ->);
8           IF input -> (i) IN letter
9              THEN
10                output (input -> (i))+:=1;
11          FI;
12       OD;
13    END count;
14    GRANT count;
15 END letter-count;
16 test:
17 MODULE
18    DCL c CHAR (10) INIT :='A-B<ZAA9K''';
19    DCL output ARRAY ('A' : 'Z') INT;
20    SYN max=10_000;
21    GRANT max;
22    SEIZE count;
23    count (-> c,output);
24    ASSERT output=[('A') : 3,('B','K','Z') : 1, (ELSE) : 0];
25
26 END test;
```

9. Números primos

```
1  prime:
2  MODULE
3     SEIZE max;
4     NEWMODE number_list =POWERSET INT(2:max);
5     SYN  empty = number_list [];
6     DCL sieve number_list INIT := [2:max];
7        primes number_list INIT :=empty;
8     GRANT primes;
9     DO WHILE sieve/=empty;
10       primes OR :=[MIN (sieve)];
11       DO FOR j :=MIN (sieve) BY MIN (sieve) TO max;
12          sieve-:=[j];
13       OD;
14    OD;
15 END prime;
```

10. Realización de pilas de dos formas distintas transparentes al usuario

```
 1  stacks_1:
 2  MODULE
 3     SEIZE element
 4     SYN max=10_000,min=1;
 5     DCL stack ARRAY (min : max) element,
 6        stackindex INT INIT :=min;
 7     push:
 8     PROC (e element) EXCEPTIONS (overflow);
 9        IF stackindex=max
10           THEN CAUSE overflow;
11        FI;
12        stackindex+:=1;
13        stack (stackindex) :=e;
14        RETURN;
15     END push;
16     pop:
17     PROC () EXCEPTIONS (underflow);
18        IF stackindex=min
19           THEN CAUSE underflow;
20        FI;
21        stackindex-:=1;
22        RETURN;
23     END pop;
24
25     elem:
26     PROC (i INT)(element LOC) EXCEPTIONS (bounds);
27        IF i<min OR i>max
28           THEN CAUSE bounds;
29        FI;
30        RETURN stack (i);
31     END elem;
32
33     GRANT push,pop,elem;
34  END stacks-1;
```

```
35  stacks_2:
36  MODULE
37     SEIZE element;
38     NEWMODE cell=STRUCT (pred,succ REF cell,
39                          info element);
40     DCL p,last,first REF cell INIT :=NULL;
41     push:
42     PROC (e element) EXCEPTIONS (overflow);
43        p :=allocate (cell) ON
44                            (nospace) : CAUSE overflow;
45                            END;
46        IF last=NULL
47           THEN first,last :=p;
48           ELSE last ->. succ :=p;
49              p ->. pred :=last;
50              last :=p;
51        FI;
52        last ->. info :=e;
53        RETURN;
54     END push;
55     pop:
56     PROC () EXCEPTIONS (underflow);
57        IF last=NULL;
58           THEN CAUSE underflow;
59        FI;
60        last :=last ->. pred; IF last = NULL THEN first := NULL FI;
61        last ->. succ :=NULL;
62        RETURN;
63     END pop;
64     elem:
65     PROC (i INT) (element LOC) EXCEPTIONS (bounds);
66        IF first=NULL
67           THEN CAUSE bounds;
68        FI;
69        p :=first;
70        DO FOR j=2 TO i;
71           IF p ->. succ=NULL
72              THEN CAUSE bounds;
73           FI;
74           p :=p ->. succ;
75        OD;
76        RETURN p ->. info;
77     END elem;
78
79     GRANT push,pop,elem;
80
81  END stacks_2;
```

11. Fragmento para jugar al ajedrez

```
1  NEWMODE piece=STRUCT(color SET(white,black),
2                       kind SET(pawn,rook,knight,bishop,queen,king));
3  NEWMODE column=SET (a,b,c,d,e,f,g,h);
4  NEWMODE line=INT (1 : 8);
5  NEWMODE square=STRUCT (status SET (occupied,free),
6                         CASE status OF
7                         (occupied) : p piece,
8                         (free) :
9                         ESAC);
10 NEWMODE board=ARRAY (line) ARRAY (column) square;
11 NEWMODE move=STRUCT (lin_1,lin_2 line,
12                      col_1,col_2 column);
13
14 initialise:
15 PROC (bd board INOUT);
16    bd :=[(1) : [(a,h):[.status: occupied, .p : [white,rook]],
17          (b,g):[.status: occupied, .p : [white,knight]],
18          (c,f):[.status: occupied, .p : [white,bishop]],
19            (d):[.status: occupied, .p : [white,queen]],
20            (e):[.status: occupied, .p : [white,king]]],
21   (2):[(ELSE) : [.status: occupied, .p : [white,pawn]]],
22  (3:6):[(ELSE) : [.status: free]],
23    (7):[(ELSE) : [.status: occupied, .p : [black,pawn]]],
24    (8):[(a,h) : [.status: occupied, .p : [black,rook]],
25       (b,g):[.status: occupied, .p : [black,knight]],
26              (e,f) : [.status: occupied, .p : [black,bishop]],
27                (d) : [.status: occupied, .p : [black,king]],
28                (e) : [.status: occupied, .p : [black,queen]]]];
29    RETURN;
30
31 END initialise;
```

```
32  register_move:
33  PROC (b board LOC,m move) EXCEPTIONS (illegal);
34     DCL starting square LOC :=b (m.lin_1)(m.col_2),
35         arriving square LOC :=b (m.lin_1)(m.col_2);
36
37     DO WITH m;
38        IF starting.status=free
39           OR (lin_2<1 OR lin_2>8 OR col_2<a OR col_2>h)
40           OR (arriving.status/=free AND arriving.p.kind=king)
41           THEN
42              CAUSE illegal;
43        FI;
44        CASE starting.p.kind, starting.p.color OF
45
46        (pawn),(white):
47           IF col_1 = col_2 AND (arriving.status/=free
48              OR NOT (lin_2=lin_1+1 OR lin_2=lin_1+2 AND lin_2=2))
49              OR (col_2=PRED(col_1) OR col_2=SUCC(col_1))
50              AND arriving.status=free OR arriving.p.color=white
51           THEN
52              CAUSE illegal; /*capturing en passant not implemented*/
53           FI;
54        (pawn),(black):
55           IF col_1=col_2 AND (arriving.status/=free
56              OR NOT (lin_2=lin_1-1 OR lin_2=lin_1-2 AND lin_1=7))
57              OR (col_2=PRED(col_1) OR col_2=SUCC(col_1))
58              AND arriving.status=free OR arriving.p.color=black
59              THEN
60                 CAUSE illegal; /* same remark */
61           FI;
62        (rook),(*):
63           IF NOT ok_rook (b,m)
64              THEN
65                 CAUSE illegal;
66           FI;
67        (bishop),(*):
68           IF NOT ok_bishop (b,m)
69              THEN
70                 CAUSE illegal;
71           FI;
72        (queen),(*):
73           IF NOT ok_rook (b,m)
74              THEN
75                 IF NOT ok_bishop (b,m)
76                    THEN
77                       CAUSE illegal
78                 FI;
79          FI;
80        (knight,*):
81          IF ABS(ABS(NUM(col_2)-NUM(col_1))
82             -ABS(lin_2-lin_1)) /= 1
83             OR ABS(NUM(col_2)-NUM((col_1))
84                +ABS(lin_2-lin_1) =/ 3
85             OR arriving.status/=free AND
```

```
86                arriving.p.color=starting.p.color
87            THEN CAUSE illegal;
88            FI;
89              (king),(*):
90               IF ABS(NUM(col_2)-NUM(col_1)) > 1
91                  OR ABS(lin_2-lin_1) > 1
92                  OR lin_2=lin_1 AND col_2=col_1
93                  OR arriving.status/=free AND
94                     arriving.p.color=starting.p.color
95               THEN CAUSE illegal;
96               FI; /*checking king moving to check not implemented*/
97          ESAC;
98       OD;
99       arriving :=starting;
100      RETURN;
101   END register_move;
102   ok_rook:
103   PROC (b board,m move)(BOOL);
104      DO WITH m;
105          IF NOT (col_2=col_1 OR lin_1=lin_2)
106             OR arriving.status/=free AND
107             arriving.p.color=starting.p.color
108          THEN RETURN FALSE;
109          FI;
110          IF col_1=col_2
111             THEN IF lin_1<lin_2
112                     THEN DO FOR l := lin_1+1 TO lin_2-1;
113                             IF board (l)(col_1).status/=free
114                                THEN RETURN FALSE;
115                             FI;
116                          OD;
117                     ELSE DO FOR l := lin_1-1 DOWN TO lin_2+1;
118                             IF board (l)(col_1).status/=free
119                                THEN RETURN FALSE;
120                             FI;
121                          OD;
122                FI;
123             ELSE IF col_1<col_2
124                     THEN DO FOR c := SUCC(col_1) TO PRED(col_2);
125                             IF board (lin_1)(c).status/=free
126                                THEN RETURN FALSE;
127                             FI;
128                          OD;
129                     ELSE DO FOR c := SUCC(col_2) DOWN TO PRED(col_1);
130                             IF board (lin_1)(c).status/=free
131                                THEN RETURN FALSE;
132                             FI;
133                          OD;
134                FI;
135          FI;
136          RETURN TRUE;
137       OD;
138   END ok_rook;
```

```
139  ok_bishop:
140  PROC (b board,m move)(BOOL);
141     DO WITH m;
142        CASE lin_2>lin_1,col_2>col_1 OF
143           (TRUE),(TRUE): c := col_1
144                       DO FOR  l := lin_1+1 TO lin_2-1;
145                          c := SUCC(c);
146                          IF board (l)(c).status/=free
147                              THEN RETURN FALSE;
148                          FI;
149                       OD;
150                       IF SUCC(c)/=col_2
151                          THEN RETURN FALSE;
152                       FI;
153           (TRUE),(FALSE): c := col_1
154                        DO FOR l := lin_1+1 TO lin_2-1;
155                           c := PRED(c);
156                           IF board (l)(c).status/=free
157                               THE RETURN FALSE;
158                           FI;
159                        OD;
160                        IF PRED(c)/=col_2
161                           THEN RETURN FALSE;
162                        FI;
163           (FALSE),(TRUE): c := col_1
164                       DO FOR l := lin_1-1 DOWN TO lin_2+1;
165                          c := SUCC(c)
166                          IF board (l)(c).status/=free
167                              THEN RETURN FALSE;
168                          FI;
169                       OD;
170                       IF SUCC(c)/=col_2
171                          THEN RETURN FALSE;
172                       FI:
173           (FALSE),(FALSE): c := col_1;
174                        DO FOR l := lin_1-1 DOWN TO lin_2+1;
175                           c := PRED(c);
176                           IF board (l)(c).status/=free
177                               THEN RETURN FALSE;
178                           FI;
179                        OD;
180                        IF PRED (c)/=col_2
181                           THEN RETURN FALSE;
182                        FI;
183        ESAC;
184        RETURN arriving.status=free OR
185              arriving.p.color/=starting.p.color;
186     OD;
187  END ok_bishop;
```

12. Construcción y manejo de una lista enlazada circularmente

```
1   CIRCULAR_LIST:
2   MODULE

3       HANDLE_LIST:
4       MODULE
5           GRANT INSERT, REMOVE, NODE;
6           NEWMODE NODE=STRUCT(PRED, SUC REF NODE, VALUE INT);
7           DCL POOL ARRAY(1:1000)NODE;
8           DCL HEAD NODE:=(: NULL,NULL,0 :);
9           INSERT:
            PROC(NEW NODE);
10              /* INSERT ACTIONS */
11          END INSERT;

12          REMOVE:
            PROC();
13              /* REMOVE ACTIONS */
14          END REMOVE;

15          INITIALIZE_LIST:
16          BEGIN
17              DCL LAST REF NODE:= ->HEAD;
18              DO FOR NEW IN POOL;
19                  NEW.PRED := LAST;
20                  LAST->.SUC:= ->NEW;
21                  LAST:= ->NEW;
22                  NEW.VALUE:=0;
23              OD;
24              HEAD.PRED:=LAST;
25              LAST->.SUC:= ->HEAD;
26          END INITIALIZE_LIST

27      END HANDLE_LIST;

28      DCL NODE_A NODE:=(: NULL,NULL,536 :);
29      REMOVE();
30      REMOVE();
31      INSERT(NODE_A);
32  END CIRCULAR_LIST;
```

13. Una región para gestionar accesos convenientes a un recurso

```
1   ALLOCATE_RESOURCES:
2   REGION
3       GRANT ALLOCATE, DEALLOCATE;
4       NEWMODE RESOURCE_SET = INT(0:9);
5       DCL ALLOCATED ARRAY(RESOURCE_SET)BOOL :=
                    (: (RESOURCE_SET): FALSE :);
6       DCL RESOURCE_FREED EVENT;

7       ALLOCATE:
8       PROC()(INT);
9           DO FOR EVER;
10              DO FOR I IN RESOURCE_SET;
11                  IF NOT ALLOCATED(I)
12                  THEN
13                      ALLOCATED(I) := TRUE;
14                      RETURN I;
15                  FI;
16              OD;
17              DELAY RESOURCE_FREED;
18          OD;
19      END ALLOCATE;

20      DEALLOCATE:
21      PROC(I INT);
22          ALLOCATE(I) := FALSE;
23          CONTINUE RESOURCE_FREED;
24      END DEALLOCATE;

25  END ALLOCATE_RESOURCES;
```

14. Llamadas en cola de espera ante un cuadro de conmutación

```
1   SWITCHBOARD:
2   MODULE
3   /* Este ejemplo ilustra un cuadro de conexión que pone en cola las
       llamadas entrantes y las presenta al operador con una velocidad
       constante.  Cada vez que el operador está listo se deja pasar una
       y sólo una llamada, la cual es tratada por un distribuidor de
       llamadas que deja pasar una llamada a intervalos fijos. Si el
       operador no está presto o hay otras llamadas esperando, se pone
       en la cola de espera una nueva llamada para que espere su turno. */

9       DCL OPERATOR_IS_READY,
10          SWITCH_IS_CLOSED EVENT;

11      CALL_DISTRIBUTOR:
12      PROCESS();
13          DO FOR EVER;
14              WAIT(10 /*seconds*/);
15              CONTINUE OPERATOR_IS_READY;
16          OD;
17      END CALL_DISTRIBUTOR;

18      CALL:
19      PROCESS();
20          DELAY CASE
21          (OPERATOR_IS_READY): /* some actions */
22          (SWITCH_IS_CLOSED): DO FOR I IN INT(1:100);
23                                  CONTINUE OPERATOR_IS_READY;
24                                  /*empty the queue*/
25                              OD;
26          ESAC;
27      END CALL;

28      OPERATOR:
29      PROCESS();
30          DO FOR_EVER;
31              IF TIME = 1700
32                  THEN
33                      CONTINUE SWITCH_IS_CLOSED;
34              FI;
35          OD;
36      END OPERATOR;

37      START CALL_DISTRIBUTOR();
38      START OPERATOR();
39      DO FOR I INT(1:100);
40          START CALL();
41      OD;
42  END SWITCHBOARD;
```

15. Atribución y desatribución de un conjunto de recursos

```
1   <> FREE (STEP);
2   COUNTER MANAGER:
3   MODULE
4   /* To illustrate the use of signals and the receive case, (buffers
5      might have been instead) we will look at an example where an
6      ALLOCATOR manages a set of resources, in this case a set of
7      COUNTERs. The module is part of a larger system where there are
8      USERs, that can request the services of the COUNTER_MANAGER. The
9      module is made to consistof two process definitions, one for the
10     ALLOCATION and one for the COUNTERS. INITIATE and TERMINATE
11     are internal signals sent from the ALLOCATOR
12     to the COUNTERs. All the other signals are external, being sent
13     from or to the USERs. */

14     SEIZE /* external signals */
15        ACQUIRE, RELEASE, CONGESTED,STEP,READOUT;
16     SIGNAL INITIATE = (INSTANCE),
17           TERMINATE;

18     ALLOCATOR:
19     PROCESS();
20        NEWMODE NO_OF_COUNTERS = INT(1:100);
21     DCL COUNTERS ARRAY (NO_OF_COUNTERS)
22              STRUCT (COUNTER INSTANCE,
23                      STATUS SET (BUSY,IDLE));
24     DO FOR EACH IN COUNTERS;
25     EACH:= (: START COUNTER(), IDLE :);
26     OD;

27     DO FOR EVER;
28     BEGIN
29        DCL USER INSTANCE;
30        AWAIT_SIGNALS:
31        RECEIVE CASE SET USER;
32        (ACQUIRE):
33           DO FOR EACH IN COUNTERS;
34              DO WITH EACH;
35                 IF STATUS = IDLE
36                 THEN
37                    STATUS:=BUSY;
38                    SEND INITIATE (USER) TO COUNTER;
39                    EXIT AWAIT_SIGNALS;
40                 FI;
41              OD;
42           OD;
43           SEND CONGESTED TO USER;
44        (RELEASE IN THIS_COUNTER);
45           SEND TERMINATE TO THIS_COUNTER;
```

```
46              FIND_COUNTER:
47              DO FOR EACH IN COUNTERS;
48                 DO WITH EACH;
49                    IF THIS_COUNTER = COUNTER
50                    THEN
51                       STATUS:= IDLE;
52                       EXIT FIND_COUNTER;
53                    FI;
54                 OD;
55              OD FIND_COUNTER;
56           ESAC AWAIT_SIGNALS;
57        END;
58        OD;
59        END ALLOCATOR;

60        COUNTER:
61        PROCESS();
62           DO FOR EVER;
63           BEGIN
64              DCL USER INSTANCE;
65                 COUNT:= 0;
66              RECEIVE CASE
67                 (INITIATE IN RECEIVED_USER):
68                    SEND READY TO RECEIVED_USER;
69                    USER:= RECEIVED_USER;
70              ESAC;
71              WORK_LOOP:
72              DO FOR EVER;
73                 RECEIVE CASE
74                    (STEP): COUNT +:=1;
75                    (TERMINATE):
76                       SEND READOUT(COUNT) TO USER;
77                       EXIT WORK_LOOP;
78                 ESAC;
79              OD WORK_LOOP;
80           END;
81           OD;
82        END COUNTER;

83        START ALLOCATOR();
84     END COUNTER_MANAGER;
```

16. Atribución y desatribución de un conjunto de recursos utilizando tampones

```
1  <> FREE(STEP);
2  USER_WORLD:
3  MODULE
4  /* This example is the same as no.15 except that buffers are
5     used for communication in stead of signals.
6     The main difference is that processes are now identified
7     by means of references to local message buffers rather than
8     by instance values. There is one message buffer declared
9     local to each process. There is one set of message types
10    for each process definition. When started each process must
11    identify its buffer address to the starting process.
12    The USER_WORLD module sketches some of the environment in
13    which the COUNTER_MANAGER is used. */
14
15 GRANT USER_BUFFERS,
16       ALLOCATOR_MESSAGES, ALLOCATOR_BUFFERS,
17       COUNTER_MESSAGES, COUNTER_BUFFERS;
18 NEWMODE
19   USER_MESSAGES =
20       STRUCT(TYPE SET(CONGESTED, READY,
21                       READOUT, ALLOCATOR_ID),
22              CASE TYPE OF
23              (CONGESTED)   :
24              (READY)       : COUNTER REF COUNTER_BUFFERS,
25              (READOUT)     : COUNT INT,
26              (ALLOCATOR_ID): ALLOCATOR REF ALLOCATOR_BUFFERS
27              ESAC),
28   USER_BUFFERS = BUFFER(1) USER_MESSAGES,
29   ALLOCATOR_MESSAGES =
30       STRUCT(TYPE SET(ACQUIRE, RELEASE, COUNTER_ID),
31              CASE TYPE OF
32              (ACQUIRE)   : USER REF USER_BUFFERS,
33              (RELEASE,
34               COUNTER_ID): COUNTER REF COUNTER_BUFFERS
35              ESAC),
36   ALLOCATOR_BUFFERS = BUFFER(1) ALLOCATOR_MESSAGES,
37   COUNTER_MESSAGES =
38       STRUCT(TYPE SET(INITIATE, STEP, TERMINATE),
39              CASE TYPE OF
40              (INITIATE) : USER REF USER_BUFFERS,
41              (STEP,
42               TERMINATE):
43              ESAC,
44   COUNTER_BUFFERS = BUFFER(1) COUNTER_MESSAGES;
45 DCL USER_BUFFER USER_BUFFERS,
46     ALLOCATOR_BUF REF ALLOCATOR_BUFFERS,
47     COUNTER_BUF REF COUNTER_BUFFERS;
48 START ALLOCATOR(->USER_BUFFER);
49 ALLOCATOR_BUF := (RECEIVE USER_BUFFER).ALLOCATOR;
50 END_USER_WORLD;
```

```
COUNTER_MANAGER:
MODULE
SEIZE USER_BUFFERS,
      ALLOCATOR_MESSAGES, ALLOCATOR_BUFFERS,
      COUNTER_MESSAGES, COUNTER_BUFFERS;

ALLOCATOR:
PROCESS(STARTER REF USER_BUFFERS);
   DCL ALLOCATOR_BUFFER ALLOCATOR_BUFFERS;
   NEWMODE NO_OF_COUNTERS = INT(1:10);
   DCL COUNTERS ARRAY(NO_OF_COUNTERS)
         STRUCT(COUNTER REF COUNTER_BUFFERS,
               STATUS SET(BUSY, IDLE)),
       MESSAGE ALLOCATOR_MESSAGES;
   SEND STARTER->([ALLOCATOR_ID, ->ALLOCATOR_BUFFER]);
   DO FOR EACH IN COUNTERS;
      START COUNTER(->ALLOCATOR_BUFFER);
      EACH := [(RECEIVE ALLOCATOR_BUFFER).COUNTER, IDLE];
   OD;
   DO FOR EVER;
      BEGIN
      DCL USER REF USER_BUFFERS;
      MESSAGE := RECEIVE ALLOCATOR_BUFFER;
      HANDLE_MESSAGES:
      CASE MESSAGE.TYPE OF
      (ACQUIRE):
         USER := MESSAGE.USER;
         DO FOR EACH IN COUNTERS;
            DO WITH EACH;
               IF STATUS= IDLE
               THEN STATUS := BUSY;
                    SEND COUNTER->([INITIATE, USER]);
                    EXIT HANDLE_MESSAGES;
               FI;'
            OD;
         OD;
         SEND USER->([CONGESTED]);
      (RELEASE):
         SEND MESSAGE.COUNTER([TERMINATE]);
         FIND_COUNTER:
         DO FOR EACH IN COUNTERS;
            DO WITH EACH;
               IF MESSAGE.COUNTER = COUNTER
               THEN STATUS := IDLE;
                    EXIT FIND_COUNTER;
               FI;
            OD;
         OD FIND_COUNTER;
      ESAC HANDLE_MESSAGES;
      END;
   OD;
END ALLOCATOR;
```

```
103 COUNTER:
104 PROCESS(STARTER REF ALLOCATOR_BUFFERS);
105    DCL COUNTER_BUFFER ALLOCATOR_BUFFERS;
106    SEND STARTER->([COUNTER_ID, ->COUNTER_BUFFER]);
107    DO FOR EVER;
108       BEGIN
109         DCL USER REF USER_BUFFERS,
110             COUNT INT := 0,
111             MESSAGE COUNTER_MESSAGES;
112         MESSAGE := RECEIVE COUNTER_BUFFER;
113         CASE MESSAGE.TYPE OF
114         (INITIATE): USER := MESSAGE.USER;
115                     SEND USER->([READY, ->COUNTER_BUFFER]);
116         ELSE /* some error action */
117         ESAC;
118         WORK_LOOP:
119         DO FOR EVER;
120            MESSAGE := RECEIVE COUNTER_BUFFER;
121            CASE MESSAGE.TYPE OF
122            (STEP)      : COUNT +:= 1;
123            (TERMINATE):SEND USER->([READOUT, COUNT]);
124                        EXIT WORK_LOOP;
125            ELSE /* some error action */
126            ESAC;
127         OD WORK_LOOP;
128    END;
129    OD;
130 END COUNTER;
131 END COUNTER_MANAGER;
```

17. Explorador de cadenas

```
1  string_scanner1: /* This program implements strings by means
2                      of packed arrays of characters.*/
3  MODULE
4  SYN
5    blanks ARRAY(0:9)CHAR PACK = [(*):' '], linelength = 132;
6  SYNMODE
7    stringptr = ROW ARRAY(lineindex)CHAR PACK,
8    lineindex = INT(0:linelength-1);
9
10 scanner:
11 PROC(string stringptr, scanstart lineindex INOUT,
12      scanstop lineindex, stopset POWERSET CHAR)
13     RETURNS(ARRAY(0:9)CHAR PACK);
14     DCL count INT:=0,
15         res ARRAY(0:9)CHAR PACK:=blanks;
16     DO
17       FOR c IN string->(scanstart:scanstop)
18       WHILE NOT (c IN stopset);
19         count+:=1;
20     OD;
21     IF count>0
22       THEN
23          IF count>10
24            THEN
25              count:=10;
26           FI;
27           res(0:count-1):=string->(scanstart:scanstart+count-1);
28     FI;
29     RESULT res;
30     IF scanstart+count < scanstop
31       THEN
32          scanstart:=scanstart+count+1;
33     FI;
34     END scanner;
35
36 GRANT
37    scanner;
38
39 END string_scanner;
```

18. Explorador de cadenas

```
1  string_scanner2: /* This example is the same as no.18 but it uses
2                      character string in stead of packed arrays.*/
3  MODULE
4    SYN
5      blanks = (10)' ', linelength = 132;
6    SYNMODE
7      stringptr = ROW CHAR(linelength),
8      lineindex = INT(0:linelength-1);
9
10   scanner:
11     PROC(string stringptr, scanstart lineindex INOUT,
12         scanstop lineindex, stopset POWERSET CHAR)
13        RETURNS (CHAR(10));
14       DCL count INT:=0;
15       DO FOR i := scanstart TO scanstop
16          WHILE NOT (string->(i) IN stopset);
17            count+:=1;
18       OD;
19       IF count>0
20         THEN
21           IF count>=10
22             THEN
23               RESULT string->(scanstart UP 10);
24             ELSE
25               RESULT string->(scanstart:scanstart+count-1)
26                     //blanks(count:9);
27           FI;
28         ELSE
29           RESULT blanks;
30       FI;
31       IF scanstart+count < scanstop
32         THEN
33           scanstart:=scanstart+count+1;
34       FI;
35     END scanner;
36
37   GRANT
38     scanner;
39
40 END string_scanner;
```

19. Supresión de un elemento en una lista doblemente enlazada

```
1  QUEUE_REMOVAL:
2  MODULE
3     SEIZE INFO;
4     GRANT REMOVE;
5     REMOVE: PROC(P PTR) RETURNS(INFO) EXCEPTIONS(EMPTY);
6     /* This procedure removes the item referred to by
7        P from a queue and returns the information contents
8        of that queue element.*/
9          DCL 1 X BASED (P),
10              2 I INFO POS(0,8:31),
11              2 PREV PTR POS(1,0:15),
12              2 NEXT PTR POS(1,16:31);
13         DCL PREV, NEXT PTR;
14         PREV := X.PREV;
15         NEXT := X.NEXT;
16         X.PREV, X.NEXT := NULL;
17         RESULT X.INFO;
18         P := PREV;
19         X.NEXT := NEXT;
20         P := NEXT;
21         X.PREV := PREV;
22    END REMOVE;
23
24 END QUEUE_REMOVAL;
```

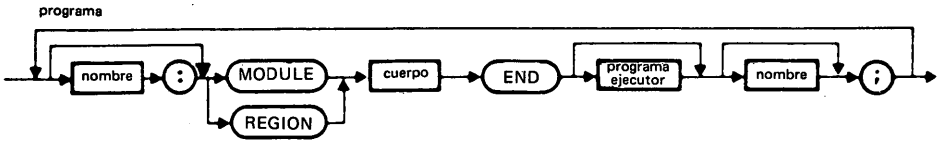
APÉNDICE E: DIAGRAMAS DE SINTAXIS

Los diagramas de este apéndice describen la sintaxis del CHILL.

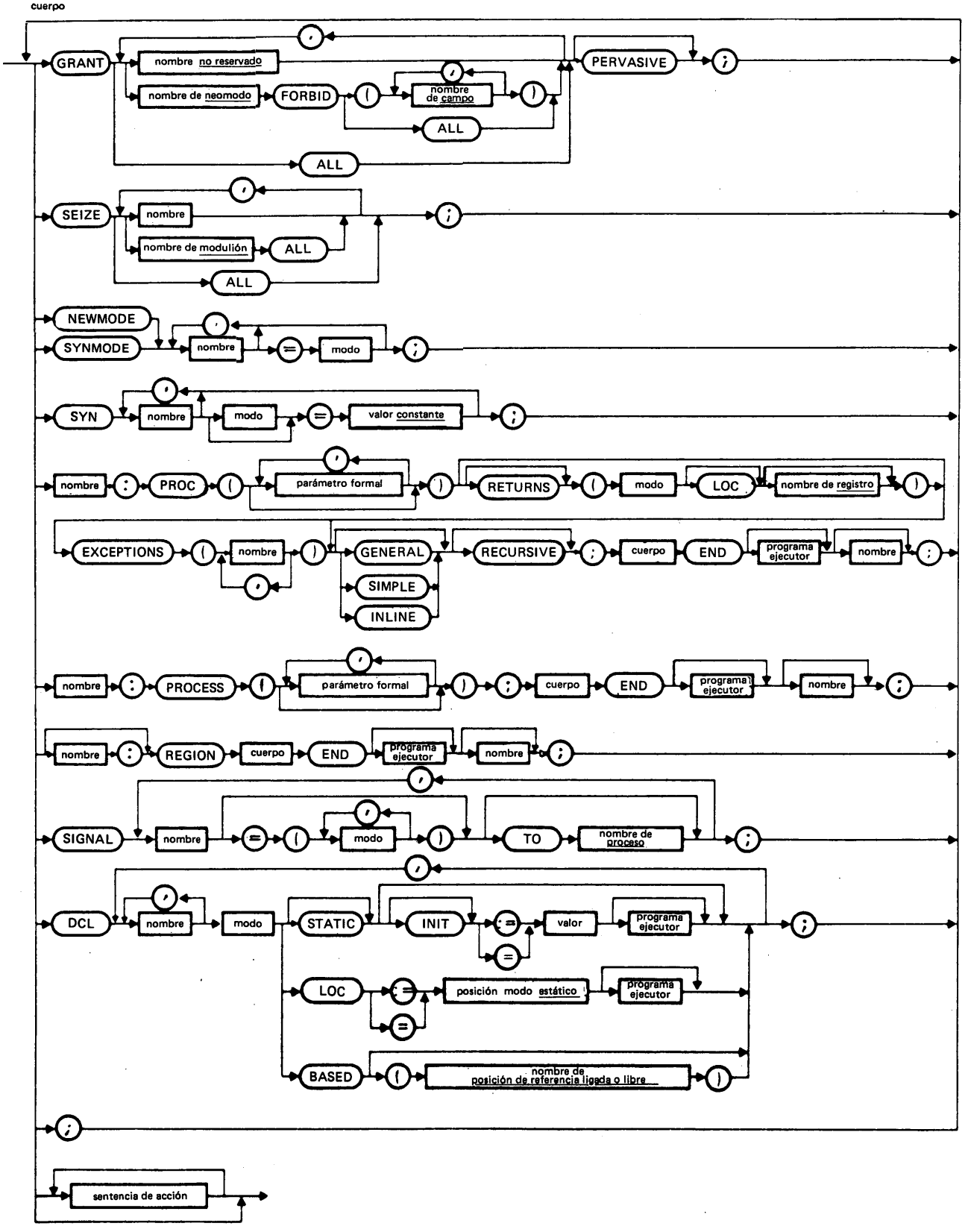
Se han diseñado los diagramas para la comprensión humana, no como base para el análisis.

Se han realizado simplificaciones con el fin de aumentar la comprensión y por tanto no pueden considerarse como definitivos, sólo como una ayuda para entender el CHILL. La definición de sintaxis libre de contexto se especifica bajo la forma de Backus-Naur en otra parte de este documento.

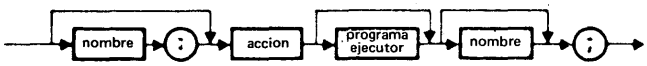
programa



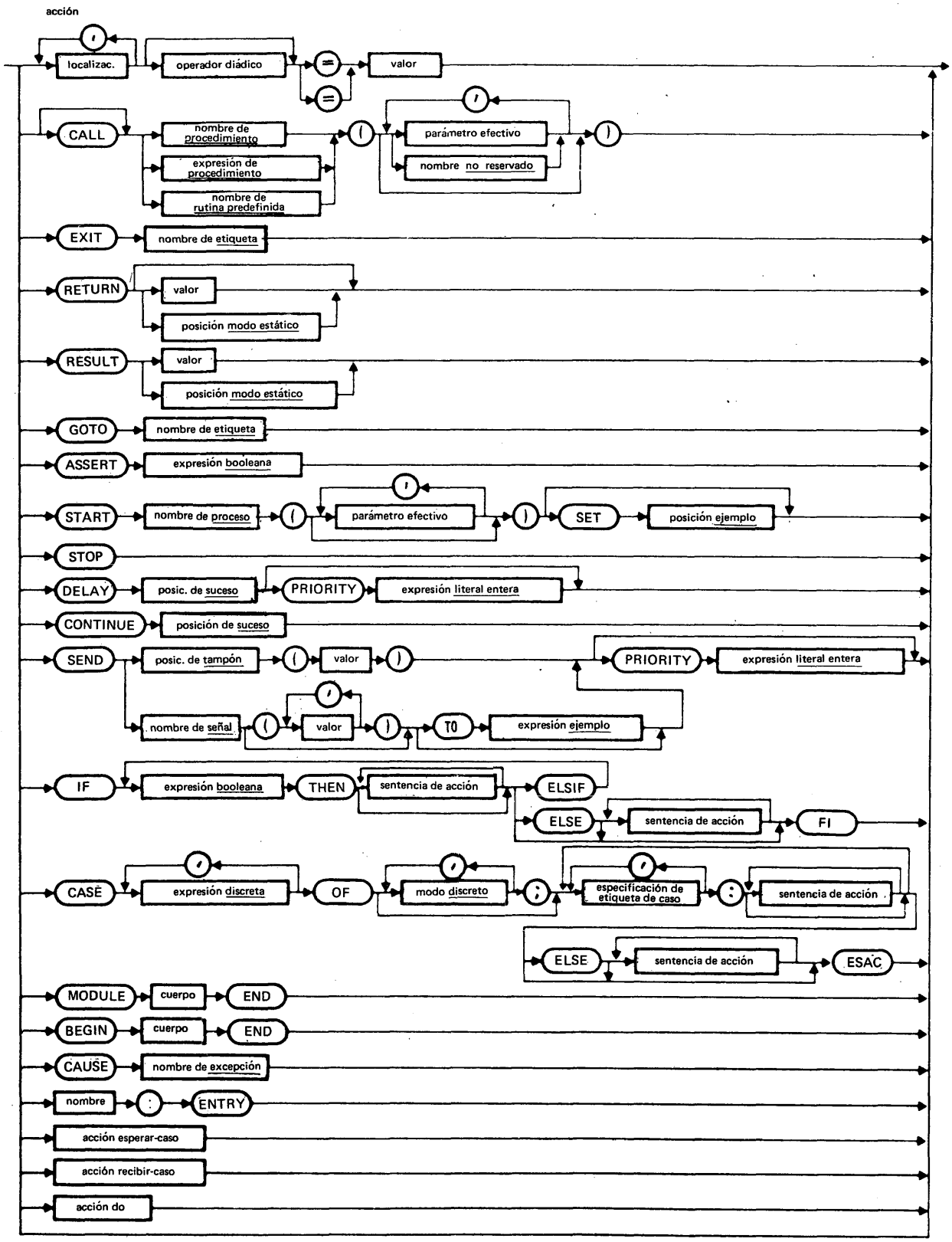
cuerpo



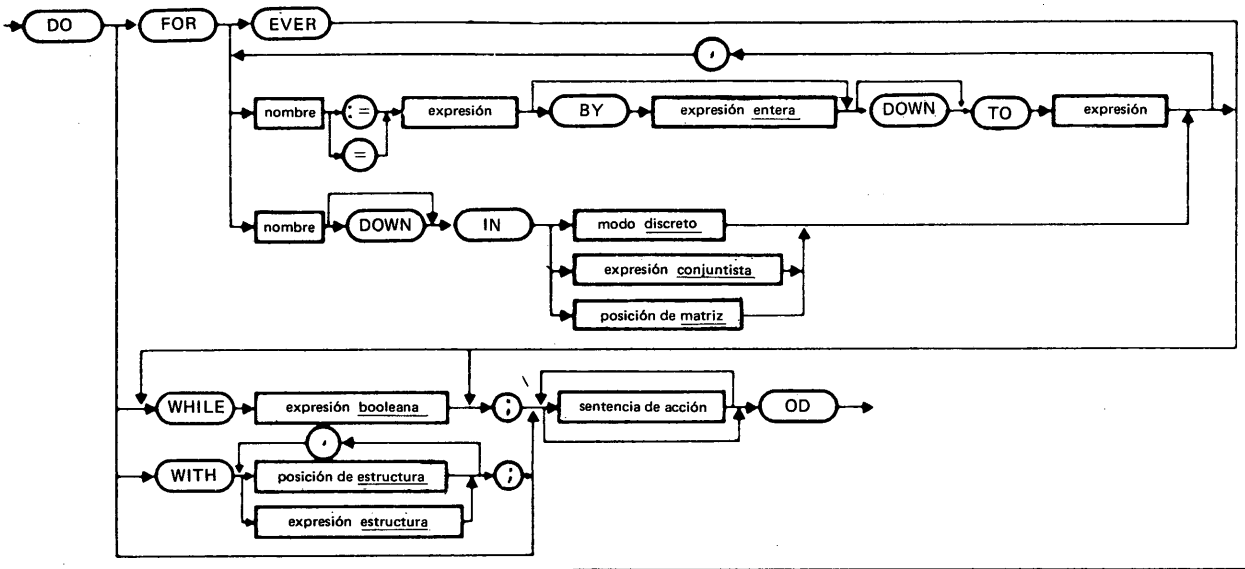
sentencia de acción



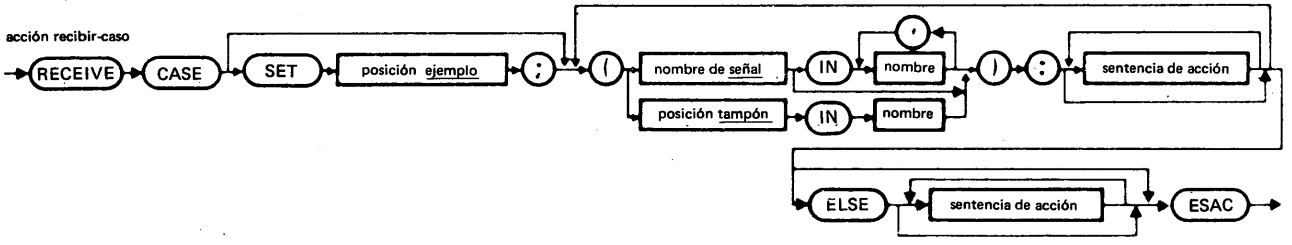
acción



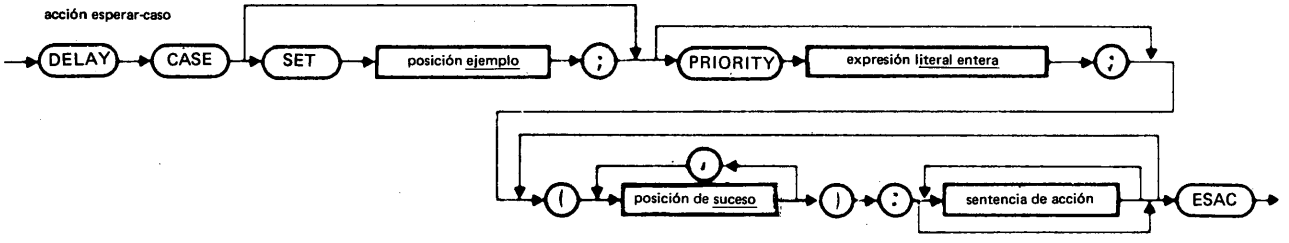
acción hacer



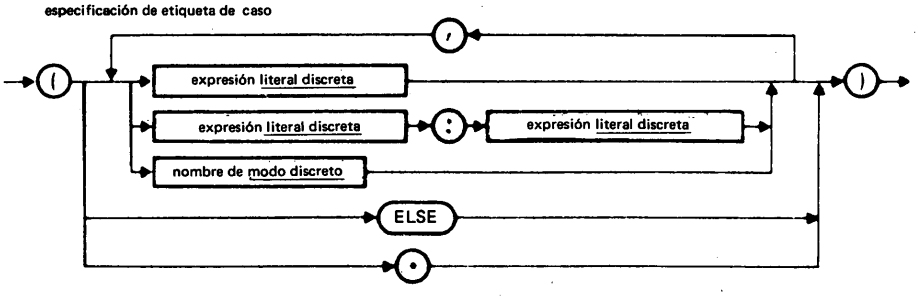
acción recibir-caso



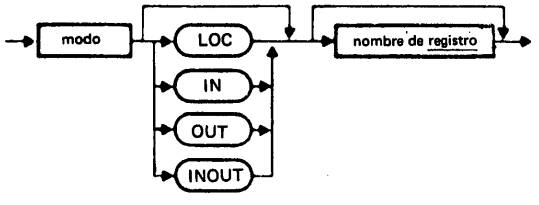
acción esperar-caso



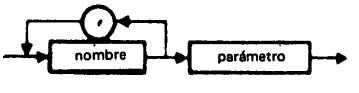
especificación de etiqueta de caso



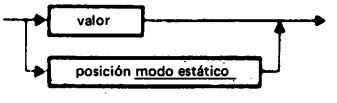
parámetro

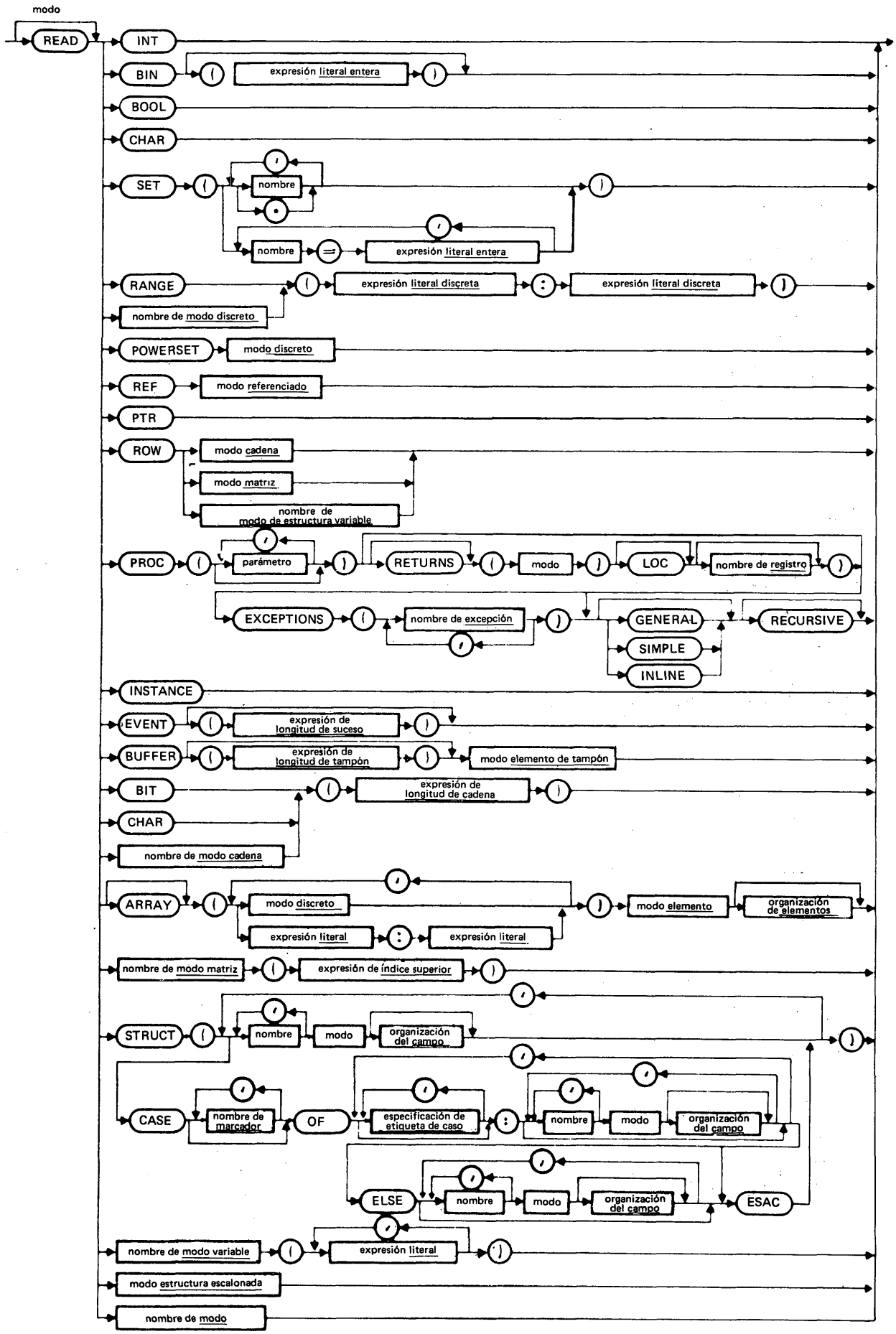


parámetro formal

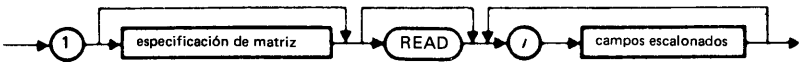


parámetro efectivo

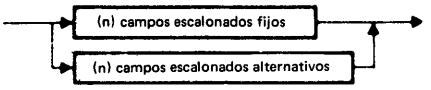



modo
READ
INT
BIN
expresión literal entera
BOOL
CHAR
SET
nombre
nombre
expresión literal entera
RANGE
expresión literal discreta
expresión literal discreta
nombre de modo discreto
POWERSET
modo discreto
REF
modo referenciado
PTR
ROW
modo cadena
modo matriz
nombre de modo de estructura variable
PROC
parámetro
RETURNS
modo
LOC
nombre de registro
EXCEPTIONS
nombre de excepción
GENERAL
SIMPLE
INLINE
RECURSIVE
INSTANCE
EVENT
expresión de longitud de suceso
BUFFER
expresión de longitud de tampón
modo elemento de tampón
BIT
expresión de longitud de cadena
CHAR
nombre de modo cadena
ARRAY
modo discreto
expresión literal
expresión literal
modo elemento
organización de elementos
nombre de modo matriz
expresión de índice superior
STRUCT
nombre
modo
organización del campo
CASE
nombre de marcador
OF
especificación de etiqueta de caso
nombre
modo
organización del campo
ELSE
nombre
modo
organización del campo
ESAC
nombre de modo variable
expresión literal
modo estructura escalonada
nombre de modo

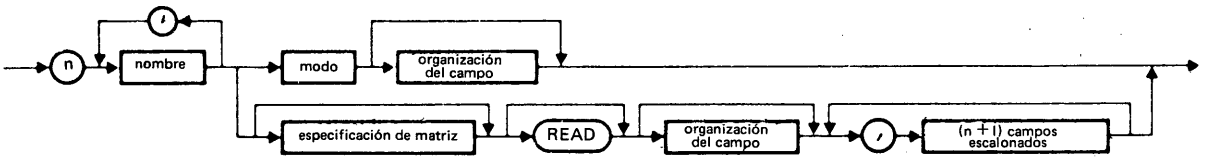
modo estructura escalonada



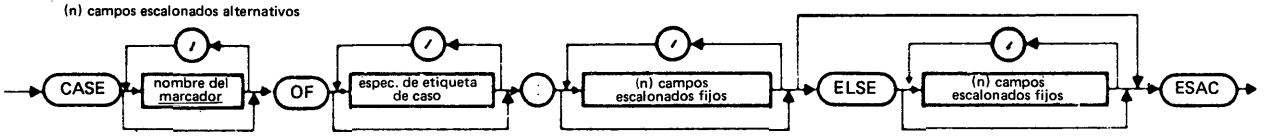
(n) campos escalonados



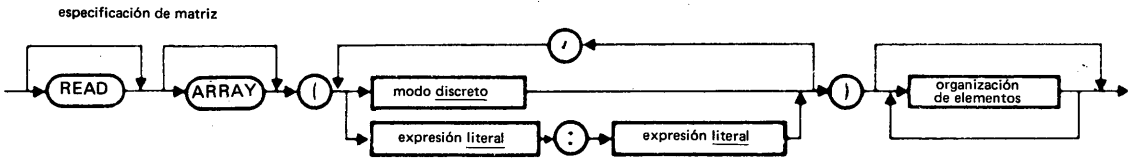
(n) campos escalonados fijos



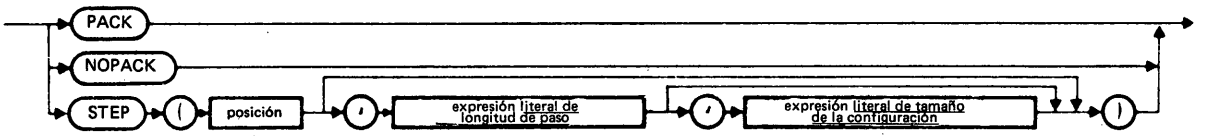
(n) campos escalonados alternativos



especificación de matriz

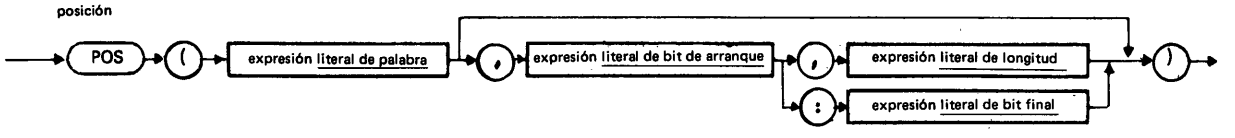


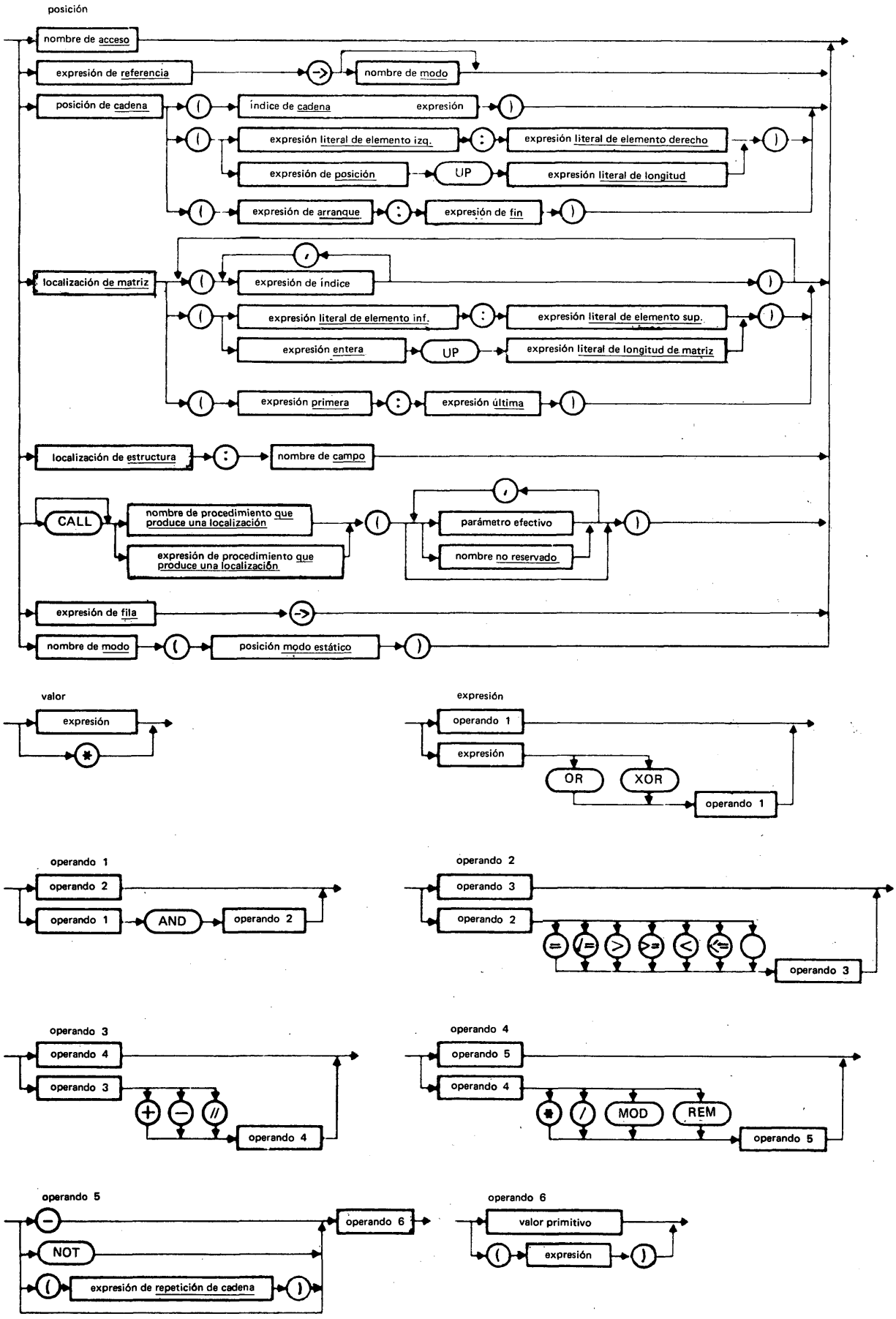
organización de elementos

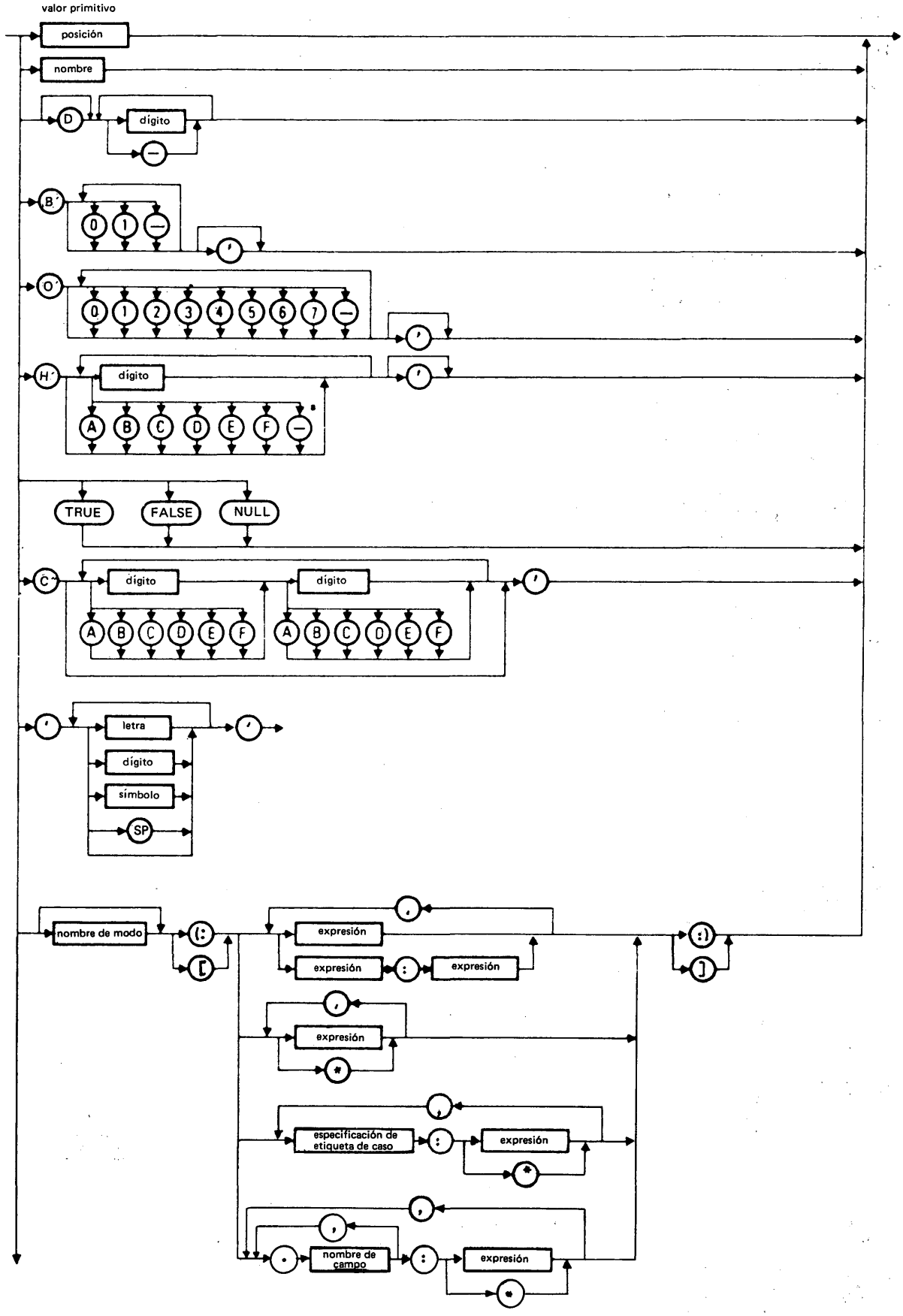


organización del campo

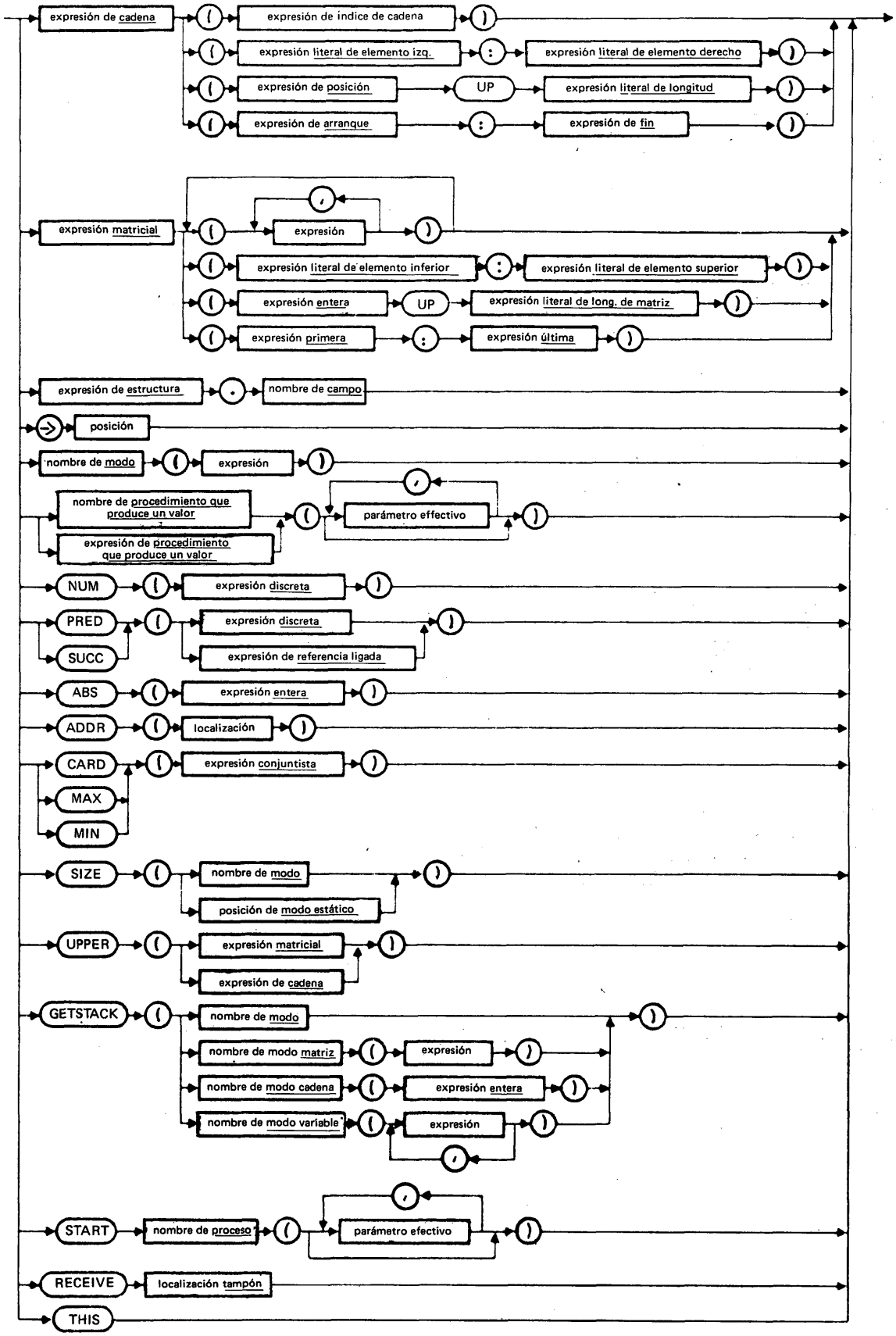
posición



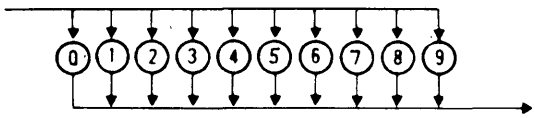

posición
nombre de acceso
expresión de referencia
nombre de modo
posición de cadena
índice de cadena
expresión
expresión literal de elemento izq.
expresión literal de elemento derecho
expresión de posición
UP
expresión literal de longitud
expresión de arranque
expresión de fin
localización de matriz
expresión de índice
expresión literal de elemento inf.
expresión literal de elemento sup.
expresión entera
UP
expresión literal de longitud de matriz
expresión primera
expresión última
localización de estructura
nombre de campo
CALL
nombre de procedimiento que produce una localización
expresión de procedimiento que produce una localización
parámetro efectivo
nombre no reservado
expresión de fila
nombre de modo
posición modo estático
valor
expresión
expresión
operando 1
expresión
OR
XOR
operando 1
operando 1
operando 2
operando 1
AND
operando 2
operando 2
operando 3
operando 2
operando 3
operando 3
operando 4
operando 3
operando 4
operando 4
operando 5
operando 4
MOD
REM
operando 5
operando 5
operando 6
NOT
expresión de repetición de cadena
operando 6
valor primitivo
expresión


valor primitivo
posición
nombre
D
dígito
B'
0
1
'
O'
0
1
2
3
4
5
6
7
'
H'
dígito
A
B
C
D
E
F
'
TRUE
FALSE
NULL
C'
dígito
dígito
A
B
C
D
E
F
A
B
C
D
E
F
'
'
letra
dígito
símbolo
SP
'
nombre de modo
(:
[
expresión
,
expresión
:
expresión
:)
]
expresión
,
*
especificación de etiqueta de caso
:
expresión
,
*
.
nombre de campo
:
expresión
,
*

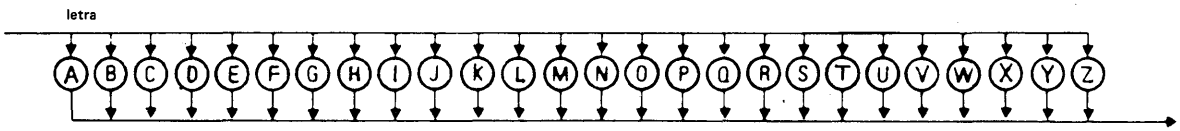
valor primitivo (continuación)



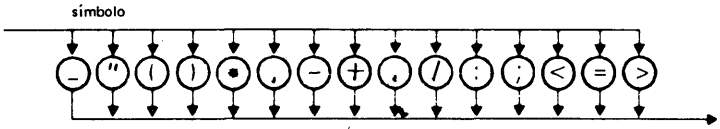
dígito



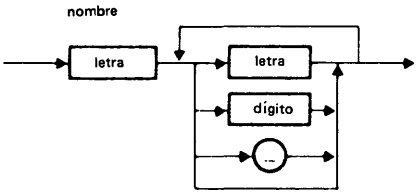
letra



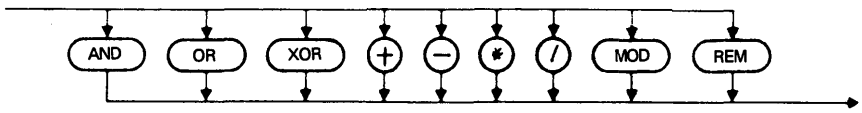
símbolo



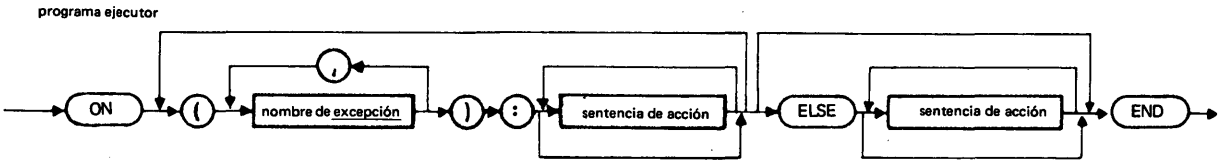
nombre



operador diádico



programa ejecutor



APÉNDICE F: ÍNDICE DE REGLAS DE PRODUCCIÓN

no terminal	definido en el punto	página	utilizado en la (o las) página(s)
<acción>	6.1	97	97
<acción arrancar>	6.13	114	97
<acción caso>	6.4	100	97
<acción causa>	6.12	113	97
<acción continuar>	6.15	114	97
<acción corchetada>	6.1	97	97
<acción de asignación>	6.2	98	97
<acción de asignación múltiple>	6.2	98	98
<acción de asignación simple>	6.2	98	98
<acción dirigirse a>	6.9	112	97
<acción enviar>	6.18.1	116	97
<acción enviar señal>	6.18.2	116	115
<acción enviar tampón>	6.18.3	117	115
<acción hacer>	6.5.1	102	97
<acción llamar>	6.7	109	97
<acción parar>	6.14	114	97
<acción poner en espera>	6.16	115	97
<acción poner en espera y elegir>	6.17	115	97
<acción predicado>	6.10	113	97
<acción recibir señal y elegir>	6.19.2	118	118
<acción recibir tampón y elegir>	6.19.3	120	118
<acción recibir y elegir>	6.19.1	118	97
<acción resultado>	6.8	111	97
<acción retornar>	6.8	111	97
<acción salir>	6.6	108	97
<acción si>	6.3	99	97
<acción vacía>	6.11	113	97
<alternativa de caso>	6.4	100	100
<alternativa de escalón (n)>	3.10.5	37	37
<alernativa de puesta en espera>	6.17	115	115
<alternativa de recepción de señal>	6.19.2	119	119
<alternativa de recepción de tampón>	6.19.3	120	120
<alternativa-on>	10.2	167	167
<alternativa variable>	3.10.4	32	32
<apóstrofo>	5.2.4.7	68	68
<atributo de parámetro>	3.7	25	25
<atributos de procedimiento>	7.4	127	127
<bit inicial>	3.10.6	39	39
<bit final>	3.10.6	39	39
<bloque principio-fin>	7.3	127	97

no terminal	definido en el punto	página	utilizado en la (o las) página(s)
<cadena de caracteres>	2.4	11	11
<campo de estructura>	4.2.9	56	50
<campo variable>	3.10.4	33	32
<campos>	3.10.4	32	32
<campos alternativos>	3.10.4	32	32
<campos alternativos de escalón (n)>	3.10.5	37	37
<campos de escalón (n)>	3.10.5	37	37
<campos fijos>	3.10.4	32	32
<campos fijos de escalón (n)>	3.10.5	37	37
<campos variables de escalón (n)>	3.10.5	37	37
<carácter>	5.2.4.7	67	11, 67
<cláusula directivo>	2.6	12	
<cláusula entonces>	6.3	100	99, 100
<cláusula prohibir>	9.2.6.2	163	163
<cláusula sino>	6.3	100	99, 100
<comentario>	2.4	11	
<componente de simultaneidad>	6.5.4	107	102
<contador de bucle>	6.5.2	103	103
<contenido de localización>	5.2.2	63	62
<control de iteración>	6.5.2	103	102
<control de simultaneidad>	6.5.4	107	107
<control mientras>	6.5.3	107	102
<conversión de expresión>	5.2.14	81	62
<conversión de localización>	4.2.12	57	50
<cuerpo de módulo>	7.2	124	132
<cuerpo de proceso>	7.2	124	132
<cuerpo de región>	7.2	125	133
<cuerpo principio-fin>	7.2	124	137
<cuerpo proc>	7.2	124	137
<declaración>	4.1.1	47	47
<declaración de identidad-loc>	4.1.3	48	47
<declaración de localización>	4.1.2	47	47
<declaración de entrada>	7.4	127	127
<declaración referida a una base>	4.1.4	49	47
<definición de modo>	3.2.1	15	17
<definición de procedimiento>	7.4	128	128
<definición de proceso>	7.5	132	132
<definición de señal>	8.5	140	140
<definición de sinónimo>	5.1	61	61
<descriptor desreferenciado>	4.2.15	59	50
<dígito>	5.2.4.2	65	11, 65, 67
<dígito hexadecimal>	5.2.4.2	65	65, 67, 68
<directivo>	2.6	12	12
<directivo CHILL>	2.6	12	12
<directivo de liberación>	2.6	12	12
<directivo de realización>			12

no terminal	definido en el punto	página	utilizado en la (o las) página(s)
<elemento cadena>	4.2.5	53	50
<elemento cedido>	9.2.6.2	163	163
<elemento de conjunto>	3.4.5	21	20
<elemento de conjunto numerado>	3.4.5	20	20
<elemento derecha>	4.2.6	53	53, 75
<elemento inferior>	4.2.8	55	55, 77
<elemento izquierda>	4.2.6	53	53, 75
<elemento matricial>	4.2.7	54	50
<elemento superior>	4.2.8	55	55, 77
<elemento tomado>	9.2.6.3	164	164
<enumeración conjuntista>	6.5.2	103	103
<enumeración de localización>	6.5.2	103	103
<enumeración de valores>	6.5.2	103	103
<enumeración por intervalo>	6.5.2	103	103
<enumeración por pasos>	6.5.2	103	103
<espacio>	5.2.4.7	67	67
<especificación de etiqueta de caso>	9.1.3	153	32, 37, 100
<especificación de la matriz>	3.10.5	37	37
<especificación de parámetro>	3.7	25	25, 127
<especificación de resultado>	3.7	25	26, 127
<etiqueta de caso>	9.1.3	154	154
<expresión>	5.3.2	88	20, 22, 27, 28, 29, 30, 32, 39, 45, 52, 53, 54, 55, 56, 58, 59, 62, 69, 75, 76, 77, 78, 79, 81, 82, 85, 86, 87, 88, 94, 95, 96, 99, 100, 103, 107, 109, 113, 115, 153
<expresión de arranque>	5.2.17	85	62, 114
<expresión parentizada>	5.3.8	95	95
<expresión recibir>	5.2.18	86	62
<fin>	4.2.13	58	57, 76
<generalidad>	7.4	127	127
<índice superior>	3.10.3	30	30
<indiferente>	9.1.3	153	153
<inicialización>	4.1.2	47	47
<inicialización ligada al dominio>	4.1.2	47	47
<inicialización ligada al tiempo de vida>	4.1.2	47	47
<intervalo>	5.2.5	69	69
<intervalo literal>	3.4.6	22	22, 30, 153
<iteración>	6.5.2	103	103

no terminal	definido en el punto	página	utilizado en la (o las) página(s)
<letra>	5.2.4.7	67	11, 67
<límite inferior>	3.4.6	22	22
<límite superior>	3.4.6	22	22
<lista de conjunto>	3.4.5	20	20
<lista de conjunto no numerada>	3.4.5	20	20
<lista de conjunto numerada>	3.4.5	20	20
<lista de etiquetas de caso>	9.1.3	154	69, 154
<lista de excepciones>	3.7	25	25, 127, 167
<lista de expresiones>	4.2.7	54	54, 77, 82
<lista de expresiones literales>	3.10.4	32	32
<lista de intervalo>	6.4	100	100
<lista de nombres>	2.6	12	15, 32, 37, 47 48, 49, 61, 119, 127
<lista de nombres prohibidos>	9.2.6.2	163	163
<lista de parámetros>	3.7	25	25
<lista de parámetros de rutina predefinida>	11.1	170	170
<lista de parámetros efectivos>	6.7	109	85, 109
<lista de parámetros formales>	7.4	127	127, 132
<lista de selectores de caso>	6.4	100	100
<lista de sentencias de acción>	7.2	124	100, 102, 115 119, 120, 121
<lista de sentencias de datos>	7.2	124	124, 125
<lista de sucesos>	6.17	115	115
<literal>	5.2.4.1	64	62
<literal binario de cadena de bits>	5.2.4.8	68	68
<literal booleano>	5.2.4.3	66	64
<literal de cadena de bits>	5.2.4.8	68	64
<literal de cadena de caracteres>	5.2.4.7	67	64
<literal de conjunto>	5.2.4.4	66	64
<literal de procedimiento>	5.2.4.6	67	64
<literal de vacío>	5.2.4.5	66	64
<literal entero>	5.2.4.2	65	65
<literal entero binario>	5.2.4.2	65	65
<literal entero decimal>	5.2.4.2	65	65
<literal entero hexadecimal>	5.2.4.2	65	65
<literal entero octal>	5.2.4.2	65	65
<literal hexadecimal de cadena de bits>	5.2.4.8	68	68
<literal octal de cadena de bits>	5.2.4.8	68	68
<localización>	4.2.1	50	53, 54, 57, 58, 59, 63, 80, 82, 98, 107, 109, 111, 115, 119, 120, 170
<localización de modo dinámico>	4.2.1	50	50
<localización de modo estático>	4.2.1	50	48, 50, 57, 58, 82, 109, 111
<localización referenciada>	5.2.13	80	62
<longitud>	3.10.6	39	39
<longitud de cadena>	3.10.2	29	29, 53, 75

no terminal	definido en el punto	página	utilizado en la (o las) página(s)
<modo primitivo>	3.5	23	23
<modo procedimiento>	3.7	25	18
<modo referenciado>	3.6.2	24	24
<modo suceso>	3.9.2	27	27
<modo tampón>	3.9.3	28	27
<módulo>	7.6	132	97
<nombre>	4.2.2	11	12, 19, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 51, 52, 56, 57, 62, 63, 66, 67, 69, 79, 81, 82, 85, 87, 97, 103, 108, 112, 113, 117, 119, 120, 127, 131, 133, 140, 153, 163, 165, 170
<nombre de acceso>	4.2.2	51	50
<nombre de excepción>	3.7	26	26, 113
<nombre de modo cadena origen>	3.10.2	29	29
<nombre de modo estructura variable origen>	3.10.4	32	32
<nombre de modo matriz origen>	3.10.3	30	30
<nombre de modulión>	9.2.6.3	164	164
<nombre de valor>	5.2.3	63	62
<operador aritmético aditivo>	5.3.5	92	92, 98
<operador aritmético multiplicativo>	5.3.6	93	93, 98
<operador cero-ádico>	5.2.19	87	62
<operador de asignación>	6.2	98	98
<operador de concatenación de cadena>	5.3.5	92	92
<operador de inclusión conjuntista>	5.3.4	90	90
<operador de pertenencia>	5.3.4	90	90
<operador de relación>	5.3.4	90	90
<operador diádico cerrado>	6.2	98	98
<operador diferencia conjuntista>	5.3.5	92	92, 98
<operador monádico>	5.3.7	94	94
<operador repetición de cadena>	5.3.7	94	94
<operador-3>	5.3.4	90	90
<operador-4>	5.3.5	92	92
<operando-1>	5.3.3	89	88, 89
<operando-2>	5.3.4	90	89, 90
<operando-3>	5.3.5	91	90, 91
<operando-4>	5.3.6	93	91
<operando-5>	5.3.7	94	93
<operando-6>	5.3.8	95	94
<organización de campo>	3.10.6	39	32, 37
<organización de elementos>	3.10.6	39	30, 37

no terminal	definido en el punto	página	utilizado en la (o las) página(s)
<palabra>	3.10.6	39	39
<parámetro de rutina predefinida>	11	170	170
<parámetro efectivo>	6.7	109	109
<parámetro formal>	7.4	127	127
<parte de control>	6.5.1	102	102
<paso>	3.10.6	39	39
<pos>	3.10.6	39	39
<posición>	4.2.6	53	53, 75
<primero>	4.2.14	59	59, 79
<principio>	4.2.13	58	58, 76
<prioridad>	6.16	115	115, 117
<programa>	7.8	131	47, 48, 97, 127
<programa ejecutor>	10.2	167	131, 133
<referencia libre desreferenciada>	4.2.4	52	50
<referencia ligada desreferenciada>	4.2.3	52	50
<región>	7.7	133	124, 133
<resultado>	6.8	133	124, 133
<segmento de cadena>	4.2.13	58	50
<segmento de matriz>	4.2.14	59	50
<sentencia de acción>	6.1	97	125, 133
<sentencia de cesión>	9.2.6.2	163	162
<sentencia de datos>	7.2	124	124
<sentencia de declaración>	4.1.1	47	124
<sentencia de definición>	7.2	124	124
<sentencia de definición de neomodo>	3.2.3	17	125
<sentencia de definición de procedimiento>	7.4	127	125
<sentencia de definición de proceso>	7.5	132	125
<sentencia de definición de señal>	8.5	140	125
<sentencia de definición de sínmodo>	3.2.2	17	125
<sentencia de definición de sinónimo>	5.1	61	125
<sentencia de entrada>	7.4	132	124
<sentencia de toma>	9.2.6.3	164	162
<sentencia de visibilidad>	9.2.6.1	163	125
<símbolo>	5.2.4.7	67	67
<símbolo de asignación>	6.2	98	47, 48, 98, 103
<subcadena>	4.2.6	53	50
<subexpresión>	5.3.2	88	88
<submatriz>	4.2.8	55	50
<suboperando-1>	5.3.3	89	89
<suboperando-2>	5.3.4	90	90
<suboperando-3>	5.3.5	91	92
<suboperando-4>	5.3.6	93	93

no terminal	definido en el punto	página	utilizado en la (o las) página(s)
<tipo de cadena>	3.10.2	29	29
<tupla>	5.2.5	69	62
<tupla conjuntista>	5.2.5	69	62
<tupla de estructura etiquetada>	5.2.5	69	69
<tupla estructura>	5.2.5	69	69
<tupla estructura no etiquetada>	5.2.5	69	69
<tupla matricial>	5.2.5	69	69
<tupla matricial etiquetada>	5.2.5	69	69
<tupla matricial no etiquetada>	5.2.5	69	69
<último>	4.2.14	59	59, 77
<vacío>	6.11	113	113, 125
<valor>	5.3.1	87	47, 61, 69, 98, 111, 117, 170
<valor campo de estructura>	5.2.12	79	62
<valor de arranque>	6.5.2	103	103
<valor del paso>	6.5.2	103	103
<valor elemento cadena>	5.2.6	75	62
<valor elemento matricial>	5.2.9	77	59
<valor indefinido>	5.3.1	87	87
<valor innominado>	3.4.5	20	20
<valor primitivo>	5.2.1	62	95
<valor segmento de cadena>	5.2.8	76	62
<valor segmento de matriz>	5.2.11	79	62
<valor subcadena>	5.2.7	75	62
<valor submatricial>	5.2.10	77	62
<ventana de cesión>	9.2.6.2	163	163
<ventana de toma>	9.2.6.3	164	164

APÉNDICE G: ÍNDICE

Impreso en Suiza — ISBN 92-61-01123-3