UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

CCITT

COMITÉ CONSULTIVO
INTERNACIONAL
TELEGRÁFICO Y TELEFÓNICO

Z.100

(11/1988)

SERIE Z: LENGUAJES Y ASPECTOS GENERALES DE SOPORTE LÓGICO PARA SISTEMAS DE TELECOMUNICACIÓN

Lenguaje de especificación y descripción funcionales (LED)
Criterios para la utilización de técnicas de descripción formal (TDF)

LENGUAJE DE ESPECIFICACIÓN Y DESCRIPCIÓN (LED)

Reedición de la Recomendación Z.100 del CCITT publicada en el Libro Azul, Fascículo X.1 (1988)

NOTAS

1 La Recomendación Z.100 del CCITT se publicó en el fascículo X.1 del Libro Azul. Este fichero es un extracto del Libro Azul. Aunque la presentación y disposición del texto son ligeramente diferentes de la versión del Libro Azul, el contenido del fichero es idéntico a la citada versión y los derechos de autor siguen siendo los mismos (véase a continuación).

2 Por razones de concisión, el término «Administración» se utiliza en la presente Recomendación para designar a una administración de telecomunicaciones y a una empresa de explotación reconocida.

LENGUAJE DE ESPECIFICACIÓN Y DESCRIPCIÓN (LED)

ÍNDICE

NOTA PRELIMINAR

Esta Recomendación remplaza las Recomendaciones Z.100 a Z.104 y la Recomendación X.250 del Libro Rojo del CCITT.

1 Introducción al LED

1.1 *Introducción*

La recomendación sobre el LED (lenguaje de especificación y descripción) tiene por finalidad proporcionar un lenguaje que permita una especificación y descripción inequívocas del comportamiento de sistemas de telecomunicaciones. Se tiene el propósito de que las especificaciones y descripciones escritas en LED sean formales en el sentido de que sea posible analizarlas e interpretarlas inequívocamente.

Los términos especificación y descripción se usan con el significado siguiente:

a) una especificación de un sistema es la descripción de su comportamiento nominal, y

b) una descripción de un sistema es la descripción de su comportamiento efectivo.

Nota – Como no se hace una distinción entre el uso del LED para especificación y su uso para descripción, el término especificación se utiliza en el texto que sigue tanto para el comportamiento nominal como para el comportamiento efectivo.

En sentido general, una especificación de sistema es la especificación del comportamiento y del conjunto de parámetros generales del sistema. Sin embargo el LED sólo tiene por objetivo especificar los aspectos relativos al comportamiento de un sistema; los parámetros generales que describen propiedades como la capacidad y el peso deben describirse mediante técnicas diferentes.

1.1.1 *Objetivos*

Los objetivos generales al definir el LED han sido los de proporcionar un lenguaje que:

a) sea fácil de aprender, utilizar e interpretar;

b) proporcione una especificación inequívoca, apropiada para pedidos y ofertas;

c) pueda ampliarse de modo que sea aplicable a nuevos desarrollos;

d) admita diversas metodologías de especificación y diseño de sistemas, sin presuponer la utilización de una de éstas.

1.1.2 *Aplicaciones*

El área principal de aplicación del LED es la especificación del comportamiento de aspectos de sistemas que funcionan en tiempo real. Entre estas aplicaciones están:

a) procesamiento de llamadas (por ejemplo tratamiento de llamadas, señalización telefónica, determinación del importe de las comunicaciones) en sistemas de conmutación;

b) mantenimiento y tratamiento de los fallos (por ejemplo, eliminación automática de fallos, pruebas de rutina) en sistemas generales de telecomunicaciones;

c) control de sistemas (por ejemplo control de sobrecarga, procedimientos de modificación y ampliación);

d) funciones de operación y mantenimiento, gestión de la red;

e) protocolos de comunicación de datos,

El LED puede utilizarse, desde luego, para la especificación funcional del comportamiento de cualquier objeto que pueda especificarse utilizando un modelo discreto, por lo que ha de entenderse que el objeto comunica con su entorno por mensajes discretos.

El LED es un lenguaje rico y puede utilizarse para especificaciones informales de alto nivel (y/o formalmente incompletas), para especificaciones semiformales, y para especificaciones detalladas. El usuario debe elegir las partes apropiadas del LED para el nivel deseado de comunicación y el entorno en que se utiliza el lenguaje. Según el entorno en que se utiliza una especificación, muchos aspectos pueden dejarse para que sean definidos de común acuerdo entre el origen y el destino de la especificación.

Así, el LED puede utilizarse para producir:

a) requisitos de facilidades,

b) especificaciones de sistemas,

c) Recomendaciones del CCITT

d) especificaciones de diseño de sistemas,

e) especificaciones detalladas

f) diseño de sistemas (tanto de alto nivel como detallado),

g) prueba de sistemas

y la organización usuaria puede elegir el nivel apropiado de aplicación del LED.

1.1.3 *Especificación de sistema*

Una especificación LED define un comportamiento de sistema en forma de estímulo/respuesta, suponiendo que tanto los estímulos como las respuestas son discretos y dan información. En particular, una especificación de sistema se percibe como la secuencia de respuestas a cualquier secuencia dada de estímulos.

El modelo de especificación de sistema se basa en el concepto de máquinas de estados finitos ampliados, que comunican.

El LED proporciona también conceptos estructurales que facilitan la especificación de sistemas grandes y/o complejos. Estas construcciones permiten la partición de la especificación de sistema en unidades manejables que pueden ser tratadas y comprendidas independientemente. La partición puede efectuarse en cierto número de pasos como resultado de los cuales se obtiene una estructura jerárquica de unidades que definen el sistema a diferentes niveles.

1.2 *Gramáticas LED*

El LED permite elegir entre dos formas sintácticas diferentes para representar un sistema: una representación gráfica (LED/GR) y una representación con frases textuales (LED/PR). Ambas formas son equivalentes, pues son representaciones concretas de la misma semántica LED. En particular, son equivalentes a una gramática abstracta, para los conceptos correspondientes.

Existe un subconjunto común al LED/PR y al LED/GR. Este subconjunto se denomina gramática textual común.

La figura 1.1 muestra las relaciones entre LED/PR, LED/GR, las gramáticas concretas y la gramática abstracta.

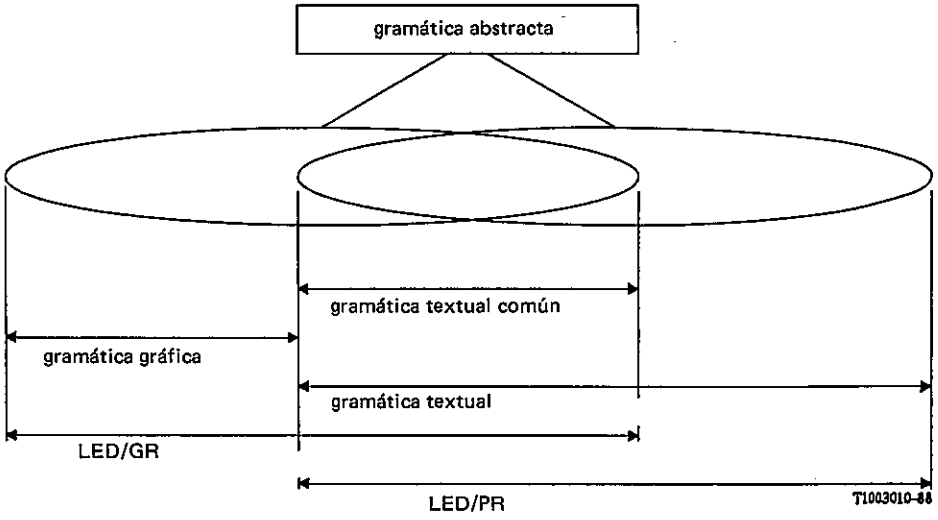


FIGURA 1.1

Gramáticas LED

Cada una de las gramáticas concretas tiene una definición de su propia sintaxis y de su relación con la gramática abstracta (es decir, una definición de cómo transformar a la sintaxis abstracta). Siguiendo este enfoque, sólo hay una definición de la semántica LED; cada una de las gramáticas concretas heredará la semántica vía sus relaciones con la gramática abstracta. Este enfoque asegura también que LED/PR y LED/GR sean equivalentes.

Se proporciona también una definición formal del LED que define cómo transformar una especificación de sistema a la sintaxis abstracta, y cómo interpretar una especificación dada en términos de la sintaxis abstracta.

1.3 *Definiciones básicas*

En esta Recomendación se utilizan algunos conceptos generales y convenios cuyas definiciones se presentan a continuación.

1.3.1 *Tipo, definición e instancia*

En esta Recomendación, los conceptos de tipo, instancia de tipo y las relaciones entre los mismos son fundamentales. Se utilizan el esquema y la terminología definidos a continuación y representados en la figura 1.2.

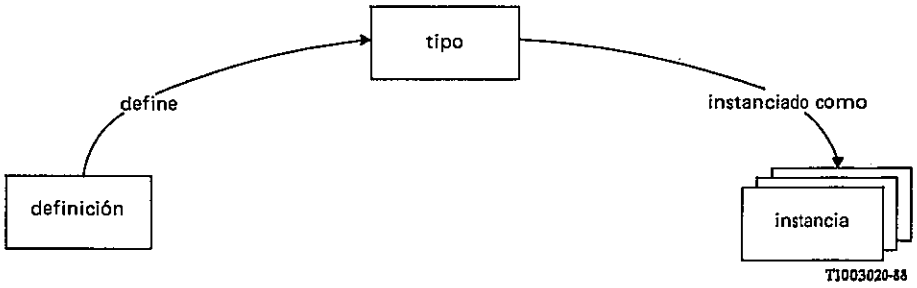


FIGURA 1.2

Concepto de tipo

Los tipos se definen por medio de definiciones. Una definición de un tipo define todas las propiedades asociadas con ese tipo. Un tipo puede ser instanciado en cualquier número de instancias. Una instancia de un tipo particular tiene todas las propiedades definidas para ese tipo.

Este esquema se aplica a varios conceptos LED, por ejemplo definiciones de sistema e instancias de sistema, definiciones de proceso e instancias de proceso.

Tipo de datos es una clase especial de tipo (véanse § 2.3 y § 5).

Nota – Para evitar textos recargados, el término instancia puede omitirse. Por ejemplo «un sistema se interpreta . . .» significa «una instancia de sistema se interpreta . . . ».

1.3.2 *Entorno*

Los sistemas especificados en LED se comportan según los estímulos recibidos del mundo exterior. Este mundo exterior se denomina entorno del sistema que se especifica.

Se supone que hay una o más instancias de proceso en el entorno y, por lo tanto, las señales que pasan del entorno al sistema tienen asociadas identidades de estas instancias de proceso. Estos procesos tienen valores PId diferentes de cualquier valor PId del sistema (véase el § 5.6.10).

Aunque el comportamiento del entorno es no determinista, debe obedecer a las limitaciones impuestas por la especificación del sistema.

1.3.3 *Errores*

Una especificación de sistema es una especificación de sistema LED válida, únicamente si satisface las reglas sintácticas y las condiciones estáticas de LED.

Si al interpretar una especificación LED válida se viola una condición dinámica, ocurre un error. Una interpretación de una especificación de sistema que conduce a un error significa que el comportamiento posterior del sistema no puede ser derivado de la especificación.

1.4 *Estilo de presentación*

1.4.1 *División del texto*

En los § 2, 3, 4, 5 la Recomendación está organizada por temas descritos en una introducción (opcional) seguida por ítems de enumeración titulados sobre:

a) *Gramática abstracta* – descrita por la sintaxis abstracta y las condiciones estáticas para la formación correcta.

b) *Gramática textual concreta* – tanto la gramática textual común, utilizada para LED/PR y LED/GR, como la gramática utilizada solamente para LED/PR. Esta gramática se describe por la sintaxis textual, las condiciones estáticas y las reglas de formación correcta para la sintaxis textual, y por la relación de la sintaxis textual con la sintaxis abstracta.

c) *Gramática gráfica concreta* – descrita por la sintaxis gráfica, las condiciones estáticas y las reglas de formación correcta para la sintaxis gráfica, la relación de esta sintaxis con la sintaxis abstracta, y algunas reglas de dibujo adicionales (a las indicadas en § 2.2.4).

d) *Semántica* – da significado a un tipo, establece sus propiedades, la forma en que se interpreta una instancia de ese tipo y las eventuales condiciones dinámicas que deben cumplirse para que la instancia de ese tipo tenga un comportamiento correcto en el sentido LED.

e) *Modelo* – da la relación de correspondencia para notaciones taquigráficas expresadas en términos de construcciones de sintaxis concreta, estrictas, previamente definidas.

f) *Ejemplos*

1.4.2 *Ítems de enumeración titulados*

Cuando un tema tiene una introducción seguida de un ítem de enumeración titulado, se considera que la introducción es una parte informal de la Recomendación, presentada solamente para facilitar la comprensión y no para completar la Recomendación.

Si un ítem de numeración titulado no tiene texto, se omite por completo.

La parte restante de esta sección describe los otros formalismos especiales utilizados en cada ítem de enumeración titulado y los títulos utilizados. Puede considerarse también como un ejemplo de la disposición tipográfica de los ítems de enumeración titulados del primer nivel, definidos más arriba, donde este texto formaría parte de una sección de introducción.

Gramática abstracta

La notación de sintaxis abstracta se define en § 1.5.1.

Si se omite el ítem de enumeración titulado *gramática abstracta*, no habrá entonces sintaxis abstracta adicional para el tema que se está presentando, y la sintaxis concreta corresponderá con la sintaxis abstracta definida por otra sección numerada de texto.

Se podrá hacer referencia a las reglas de la sintaxis abstracta a partir de cualesquiera ítems de enumeración titulados, utilizando el nombre de regla escrito en cursiva.

Las reglas en la notación formal pueden ir seguidas de párrafos que definen condiciones que deben ser satisfechas por una definición LED formada correctamente y que pueden ser verificadas sin interpretación de una instancia. Las condiciones estáticas en este punto se refieren solamente a la sintaxis abstracta. Las condiciones estáticas que sólo son importantes para la sintaxis concreta se definen después de la sintaxis concreta. Las condiciones estáticas de la sintaxis abstracta, junto con la sintaxis abstracta, definen la gramática abstracta del lenguaje.

Gramática textual concreta

La sintaxis textual concreta se especifica en la forma Backus-Naur ampliada de la descripción de sintaxis definida en la Recomendación Z.200, § 2.1 (véase también § 1.5.2).

La sintaxis textual va seguida de párrafos que definen las condiciones estáticas que deben ser satisfechas en un texto formado correctamente y que deben ser verificadas sin interpretación de una instancia. Son también aplicables las condiciones estáticas (si existen) de la gramática abstracta.

En muchos casos hay una relación simple entre la sintaxis concreta y la abstracta, pues una regla de sintaxis concreta se representa simplemente por una sola regla en la sintaxis abstracta. Cuando el mismo nombre se utiliza en la sintaxis abstracta y en la concreta para representar el mismo concepto, el texto «<x> en la sintaxis concreta representa *X* en la sintaxis abstracta» está implícito en la descripción del lenguaje y suele omitirse. En este contexto, no se distingue entre mayúsculas y minúsculas pero las subcategorías semánticas subrayadas son significativas.

La sintaxis textual concreta que no es una forma taquigráfica (sintaxis derivada, modelada por otras construcciones LED) es sintaxis textual concreta estricta. La relación de una sintaxis textual concreta con una sintaxis abstracta está definida solamente para la sintaxis textual concreta estricta.

La relación entre sintaxis textual concreta y sintaxis abstracta se omite si el tema que se está definiendo es una forma taquigráfica que está modelada por otras construcciones LED (véase *modelo*, más adelante).

Gramática gráfica concreta

La sintaxis gráfica concreta se especifica en la forma Backus-Naur ampliada de descripción de sintaxis definida en § 1.5.3.

La sintaxis gráfica va seguida de párrafos que definen las condiciones estáticas que deben cumplirse en LED/GR formado correctamente y que pueden verificarse sin interpretación de una instancia. Son también aplicables las condiciones estáticas (si existen) de la gramática abstracta.

La relación entre sintaxis gráfica concreta y sintaxis abstracta se omite si el tema que se está definiendo es una forma taquigráfica que está modelada por otras construcciones LED (véase *modelo*, más adelante).

En muchos casos hay una relación simple entre diagramas de gramática gráfica concreta y definiciones de sintaxis abstracta. Cuando el nombre de un símbolo no terminal comienza en la gramática concreta por la palabra «diagrama» y hay un nombre en la gramática abstracta que sólo difiere en que comienza por la palabra *definición*, las dos reglas representan el mismo concepto. Por ejemplo, <diagrama de sistema> en la gramática concreta y *definición-de-sistema* en la gramática abstracta son correspondientes.

La expansión en la sintaxis concreta, proveniente de facilidades tales como definiciones remotas (§ 2.4.1), macros (§ 4.2) y correspondencias de literales (§ 5.4.1.15), etc., debe considerarse antes que la correspondencia entre la sintaxis concreta y la abstracta.

Semántica

Se utilizan propiedades en las reglas de formación correcta que implican bien el tipo, bien otros tipos que se refieren a ese tipo.

Un ejemplo de propiedad es el conjunto de identificadores de señal de entrada válida de un proceso. Esta propiedad se utiliza en la condición estática «Para cada *nodo-de-estado*, todos los *identificadores-de-señal* de entrada (en el conjunto de señales de entrada válidas) aparecen, bien en un *conjunto-de-señales-de-conservación*, bien en un *nodo-de-entrada*.».

Todas las instancias tienen una propiedad de identidad pero, a menos que esté formada de alguna manera poco usual, esta propiedad se determina como se define en la sección general sobre identidades en el § 2. Por consiguiente, esto generalmente no se menciona como una propiedad de identidad. Además, no ha sido necesario mencionar subcomponentes de definiciones contenidos en la definición, pues la pertenencia de tales subcomponentes es evidente en la sintaxis abstracta. Por ejemplo, es evidente que una definición de bloque «tiene» encerradas definiciones de proceso y/o una definición de subestructura de bloque.

Una propiedad es estática si puede determinarse sin interpretación de una especificación de sistema LED, y es dinámica si se requiere una interpretación de la misma para determinarla.

La interpretación se describe de una manera operacional. Toda lista en la sintaxis abstracta deberá interpretarse en el orden dado. Esto es, la Recomendación describe cómo se crean las instancias a partir de la definición de sistema y cómo estas instancias se interpretan en una «máquina LED abstracta».

Son condiciones dinámicas aquéllas que deben cumplirse durante la interpretación y no pueden verificarse sin interpretación. Las condiciones dinámicas pueden conducir a errores (véase § 1.3.3).

Modelo

Se considera que algunas construcciones son «sintaxis concretas derivadas» (o notaciones taquigráficas) de otras construcciones de sintaxis concreta equivalentes. Por ejemplo, omitir una entrada de una señal es sintaxis concreta derivada de una entrada de esa señal seguida de una transición nula que retorna al mismo estado.

Algunas veces esta «sintaxis concreta derivada», si fuese expandida, daría lugar a una representación extremadamente grande (quizá infinita). Sin embargo, la semántica de tal especificación puede ser determinada.

Ejemplos

El ítem de enumeración titulado *Ejemplos* contiene ejemplos.

1.5 *Metalenguajes*

Para la definición de propiedades y sintaxis de LED se han utilizado diferentes metalenguajes según las necesidades particulares.

A continuación se presenta una introducción a los metalenguajes utilizados; cuando procede, se hace referencia solamente a libros de texto o publicaciones específicas de la UIT.

1.5.1 *Meta IV*

Se utiliza el siguiente subconjunto de Meta IV para describir la sintaxis abstracta de LED.

Una definición en la sintaxis abstracta puede considerarse como un objeto compuesto (un árbol) denominado que define un conjunto de subcomponentes.

Por ejemplo, la sintaxis abstracta para definición de variable es

Definición-de-variable :: *Nombre de variable Identificador-de-referencia-de-género*

que define el dominio del objeto compuesto (árbol) denominado *definición-de-variable*. Este objeto consiste en dos subcomponentes, que a su vez podrían ser árboles.

La definición Meta IV

Identificador-de-referencia-de-género = *Identificador*

expresa que un *identificador-de-referencia-de-género* es un *identificador*, por lo que no puede distinguirse sintácticamente de otros identificadores.

Un objeto podría también pertenecer a algunos dominios elementales (no compuestos). En el contexto de LED, éstos son:

a) Objetos enteros

Ejemplo:

Número-de-instancias :: *Ento Ento*

Número-de-instancias denota un dominio compuesto que contiene dos valores enteros (*Ento*) que denotan el número inicial y el número máximo de instancias.

b) Objetos citación

Se representan por una secuencia cualquiera, en negrilla, de letras mayúsculas y dígitos.

Ejemplo:

Proceso-de-destino = *Identificador-de-proceso* | **ENVIRONMENT**

El *proceso-de-destino* es bien un *identificador-de-proceso*, bien el entorno que está denotado por la citación **ENVIRONMENT**.

c) Objetos testigo

Testigo denota el dominio de testigos. Puede considerarse que este dominio consiste en un conjunto potencialmente infinito de objetos atómicos distintos que no requieren una representación.

Ejemplo:

Nombre :: *Testigo*

Un nombre consiste en un objeto atómico tal que cualquier *nombre* puede distinguirse de cualquier otro nombre.

d) Objetos no especificados

Un objeto no especificado denota dominios que podrían tener alguna representación, pero esa representación no concierne a esta Recomendación.

Ejemplo:

Texto-informal :: ...

Texto-informal contiene un objeto que no es interpretado.

Los siguientes operadores (constructores) en FBN (véase § 1.5.2) se utilizan también en la sintaxis abstracta: «*» para una lista que puede estar vacía, «+» para una lista no vacía, «|» para alternativa y «[» «]» para opcional.

Se utilizan paréntesis para la agrupación de dominios que están lógicamente relacionados.

Por último, la sintaxis abstracta utiliza otro operador postfijo «**-set**» que da un conjunto (colección no ordenada de objetos distintos).

Ejemplo:

Gráfico-de-proceso :: *Nodo-de-arranque-de-proceso Nodo-de-estado*-**set**

Un *gráfico-de-proceso* consiste en un *nodo-de-arranque-de-proceso* y un conjunto de *nodos-de-estado.*

1.5.2 *FBN*

En la forma Backus-Naur un símbolo terminal, bien se indica sin encerrarlo entre paréntesis angulares (es decir, el signo menor que y el signo mayor que, <y>), o bien es una de las dos representaciones <nombre> y <cadena de caracteres>. Obsérvese que los dos terminales especiales <nombre> y <cadena de caracteres> pueden tener también semánticas resaltadas como se define más adelante.

Los paréntesis angulares y la palabra o palabras encerradas son, bien un símbolo no terminal, bien uno de los dos símbolos terminales <cadena de caracteres> o <nombre>. Categorías sintácticas son los no terminales indicados por una o más palabras encerradas entre paréntesis angulares. Para cada símbolo no terminal se da una regla de producción sea en gramática textual concreta, sea en gramática gráfica. Por ejemplo

<expresión de visión> ::=

VIEW (<identificador de variable>, <expresión>)

Una regla de producción para un símbolo no terminal consiste en el símbolo no terminal a la izquierda del símbolo ::=, y una o más construcciones, que consisten en uno o más símbolos no terminales y/o terminales en el lado derecho. Por ejemplo <expresión de visión>, <identificador de variable> y <expresión> en el ejemplo anterior son no terminales; VIEW, los paréntesis y la coma son símbolos terminales.

Algunas veces el símbolo incluye una parte subrayada. Esta parte subrayada resalta un aspecto semántico de ese símbolo. Por ejemplo <identificador de variable> es sintácticamente idéntico a <identificador>, pero semánticamente requiere que el identificador sea un identificador de variable.

En el lado derecho del símbolo ::= puede haber varias producciones alternativas para el no terminal, separadas por barras verticales (|). Por ejemplo:

<área de bloque> ::=

<referencia gráfica de bloque>

| <diagrama de bloque>

expresa que un <área de bloque> es, bien una <referencia gráfica de bloque>, bien un <diagrama de bloque>.

Los elementos sintácticos pueden agruparse utilizando llaves ({ y }), similares a los paréntesis en Meta IV (véase § 1.5.1). Un grupo encerrado entre llaves puede contener una o más barras verticales, que indican elementos sintácticos alternativos. Por ejemplo:

<área de interacción de bloques> ::=

{<área de bloque> | <área de definición de canal>}+

La repetición de grupos encerrados en llaves se indica por un asterisco (*) o signo más (+). Un asterisco indica que el grupo es opcional y puede repetirse cualquier número de veces; un signo más indica que el grupo tiene que estar presente y puede repetirse cualquier número de veces. El ejemplo anterior expresa que un <área de interacción de bloques> contiene por lo menos un <área de bloque> o <área de definición de canal> y puede contener más <área de bloque>s y <área de definición de canal>s.

Si se agrupan elementos sintácticos por medio de corchetes ([y]), el grupo es opcional. Por ejemplo:

<encabezamiento de proceso> ::=

PROCESS <identificador de proceso> [<parámetros formales>]

expresa que un <encabezamiento de proceso> puede, pero no tiene necesariamente que, contener <parámetros formales>.

1.5.3 *Metalenguaje para gramática gráfica*

Para la gramática gráfica el metalenguaje descrito en § 1.5.2 se amplía con los siguientes metasímbolos:

a) **contains**

b) **is associated with**

c) **is followed by**

d) **is connected to**

e) **set**

El metasímbolo **set** es un operador postfijo que actúa sobre los elementos sintácticos que le preceden inmediatamente, dentro de llaves, e indica un conjunto (no ordenado) de ítems. Cada ítem puede ser un grupo cualquiera de elementos sintácticos, en cuyo caso debe ser ampliado antes de aplicar el metasímbolo **set**.

Ejemplo:

{{<área de texto de sistema>}* {<diagrama de macro>}* <área de interacción de bloques>} **set**

es un conjunto de cero o más <área de texto de sistema>s, cero o más <diagrama de macro>s y un <área de interacción de bloques>.

Todos los demás metasímbolos son operadores infijos, que tienen un símbolo gráfico no terminal como argumento de la izquierda. El argumento de la derecha es, bien un grupo de elementos sintácticos encerrados por llaves, bien un elemento sintáctico único. Si el lado derecho de una regla de producción tiene un símbolo gráfico no terminal como primer elemento y contiene uno o más de estos operadores infijos, el símbolo gráfico no terminal es el argumento de la izquierda de cada uno de estos operadores infijos. Un símbolo gráfico no terminal es un no terminal que tiene la palabra «símbolo» inmediatamente después del signo <.

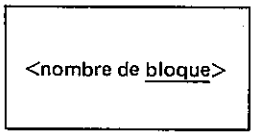
El metasímbolo **contains** indica que su argumento de la derecha debe situarse dentro de su argumento de la izquierda y del <símbolo de ampliación de texto> asociado, si existe. Ejemplo:

<referencia gráfica de bloque> ::=

<símbolo de bloque> **contains** <nombre de bloque>

<símbolo de bloque> ::=

significa lo siguiente



El metasímbolo **is associated with** indica que su argumento de la derecha está lógicamente asociado a su argumento de la izquierda (como si estuviese «contenido» en ese argumento; la asociación inequívoca se asegura por reglas de dibujo adecuadas).

El metasímbolo **is followed by** significa que su argumento de la derecha sigue (tanto lógicamente como en el dibujo) a su argumento de la izquierda.

El metasímbolo **is connected to** significa que su argumento de la derecha está conectado (tanto lógicamente como en el dibujo) a su argumento de la izquierda.

2 LED básico

2.1 *Introducción*

Un sistema LED consta de un conjunto de bloques. Los bloques están conectados entre sí y con el entorno por canales. Dentro de cada bloque hay uno o más procesos. Estos procesos comunican entre sí por señales y se supone que actúan concurrentemente.

El § 2 se ha dividido en ocho temas principales:

a) *Reglas generales*

conceptos de LED básico tales como reglas léxicas e identificadores, reglas de visibilidad, texto informal, partición de diagramas, reglas de dibujo, comentarios, ampliaciones de texto, símbolos de texto.

b) *Conceptos básicos de datos*

conceptos de datos del LED básico tales como valores, variables, expresiones.

c) *Estructura de sistema*

contiene conceptos LED referentes a los conceptos estructurales generales del lenguaje. Estos conceptos son sistema, bloque, proceso, procedimiento.

d) *Comunicación*

contiene mecanismos de comunicación utilizados en LED tales como canal, ruta de señales, señal.

e) *Comportamiento*

las construcciones relevantes para el comportamiento de un proceso: reglas generales de conectividad de un gráfico de proceso o de procedimiento, definición de variable, arranque, estado, entrada, conservación, etiqueta, transición.

f) *Acción*

construcciones activas tales como tarea, crear proceso, llamada a procedimiento, salida, decisión.

g) *Temporizadores*

definición de temporizador y primitivas de temporizador.

h) *Ejemplos*

ejemplos a los cuales se hace referencia en los demás temas.

2.2 *Reglas generales*

2.2.1 *Reglas léxicas*

Las reglas léxicas definen unidades léxicas. Las unidades léxicas son los símbolos terminales de la *sintaxis textual concreta.*

```
<unidad léxica> ::=
        <nombre>
    |   <cadena de caracteres>
    |   <especial>
    |   <especial compuesto>
    |   <nota>
    |   <palabra clave>

<nombre> ::=
        <palabra> { <subrayado> <palabra> }*

<palabra> ::=
        {<alfanumérico> | <punto>}*
        <alfanumérico>
        {<alfanumérico> | <punto>}*

<alfanumérico> ::=
        <letra>
    |   <dígito decimal>
    |   <nacional>

<letra> ::=
        A | B | C | D | E | F | G | H | I | J | K | L | M
    |   N | O | P | Q | R | S | T | U | V | W | X | Y | Z
    |   a | b | c | d | e | f | g | h | i | j | k | l | m
    |   n | o | p | q | r | s | t | u | v | w | x | y | z
```

```
<dígito decimal> ::=
        0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<nacional> ::=
        #
    |   `
    |   ¤
    |   @
    |   <corchete izquierdo>
    |   \
    |   <corchete derecho>
    |   <llave izquierda>
    |   <línea vertical>
    |   <llave derecha>
    |   <tilde>
    |   <punta de flecha hacia arriba>

<corchete izquierdo> ::=
        [

<corchete derecho> ::=
        ]

<llave izquierda> ::=
        {

<línea vertical> ::=
        |

<llave derecha> ::=
        }

<tilde> ::=
        ~

<punta de flecha hacia arriba> ::=
        ^

<punto> ::=
        .

<subrayado> ::=
        _

<cadena de caracteres> ::=
            <apóstrofo> {<alfanumérico>
                        |   <otro carácter>
                        |   <especial>
                        |   <punto>
                        |   <subrayado>
                        |   <espacio>
                        |   <apóstrofo> <apóstrofo>}* <apóstrofo>

<texto> ::=
            {<alfanumérico>
        |       <otro carácter>
        |       <especial>
        |       <punto>
        |       <subrayado>
        |       <espacio>
        |       <apóstrofo>}*

<apóstrofo> ::=
        '

<otro carácter> ::=
        ? | & | %

<especial> ::=
        + | - | ! | / | > | * | ( | ) | " | , | ; | < | = | :
```

<especial compuesto> ::=
    ==
  | ==>
  | /=
  | <=
  | >=
  | //
  | :=
  | =>
  | ->
  | (.
  | .)

<nota> ::=
    /* <texto> */

<palabra clave> ::=
    ACTIVE
  | ADDING
  | ALL
  | ALTERNATIVE
  | AND
  | AXIOMS
  | BLOCK
  | CALL
  | CHANNEL
  | COMMENT
  | CONNECT
  | CONSTANT
  | CONSTANTS
  | CREATE
  | DCL
  | DECISION
  | DEFAULT
  | ELSE
  | ENDALTERNATIVE
  | ENDBLOCK
  | ENDCHANNEL
  | ENDDECISION
  | ENDGENERATOR
  | ENDMACRO
  | ENDNEWTYPE
  | ENDPROCEDURE
  | ENDPROCESS
  | ENDREFINEMENT
  | ENDSELECT
  | ENDSERVICE
  | ENDSTATE
  | ENDSUBSTRUCTURE
  | ENDSYNTYPE
  | ENDSYSTEM
  | ENV
  | ERROR
  | EXPORT
  | EXPORTED
  | EXTERNAL
  | FI
  | FOR
  | FPAR
  | FROM
  | GENERATOR
  | IF
  | IMPORT
  | IMPORTED
  | IN
  | INHERITS
  | INPUT
  | JOIN

LITERAL
LITERALS
MACRO
MACRODEFINITION
MACROID
MAP
MOD
NAMECLASS
NEWTYPE
NEXTSTATE
NOT
NOW
OFFSPRING
OPERATOR
OPERATORS
OR
ORDERING
OUT
OUTPUT
PARENT
PRIORITY
PROCEDURE
PROCESS
PROVIDED
REFERENCED
REFINEMENT
REM
RESET
RETURN
REVEALED
REVERSE
SAVE
SELECT
SELF
SENDER
SERVICE
SET
SIGNAL
SIGNALLIST
SIGNALROUTE
SIGNALSET
SPELLING
START
STATE
STOP
STRUCT
SUBSTRUCTURE
SYNONYM
SYNTYPE
SYSTEM
TASK
THEN
TIMER
TO
TYPE
VIA
VIEW
VIEWED
WITH
XOR

<espacio> representa el carácter del Alfabeto No. 5 del CCITT para espacio.

Los caracteres <nacional> se han representado según figuran en la versión internacional de referencia del Alfabeto No. 5 del CCITT (Recomendación T.50). La responsabilidad de la definición de las representaciones nacionales de estos caracteres incumbe a los organismos nacionales de normalización.

Todas las <letra>s se tratan siempre como si fuesen mayúsculas, a menos que se encuentren dentro de <cadena de caracteres>. (El tratamiento de los <nacional>s pueden definirlo los organismos nacionales de normalización.)

Una <unidad léxica> es terminada por el primer carácter que no puede formar parte de la <unidad léxica> de acuerdo con la sintaxis especificada anteriormente. Cuando un carácter <subrayado> va seguido de uno o más caracteres de control (los caracteres de control se definen en la Recomendación T.50) o espacios, todos esos caracteres (incluido el <subrayado>) se ignoran; por ejemplo, A_B denota el mismo <nombre> que AB. Este uso de <subrayado> permite dividir <unidad léxica>s en más de una línea.

Cuando un carácter <subrayado> va seguido por una <palabra> en un <nombre>, puede especificarse uno o más caracteres de control o espacios en lugar del carácter <subrayado>, siempre y cuando una de las <palabras>s que incluyen al carácter <subrayado> no constituyan una <palabra clave>; por ejemplo A B denota el mismo <nombre> que A_B.

Sin embargo, en algunos casos la ausencia de <subrayado> en <nombre>s es ambigua, por lo que se aplican las reglas siguientes:

1. Los <subrayado>s en un <nombre> en un <ítem de trayecto> deben especificarse explícitamente.
2. Cuando uno o más <nombre>s o <identificador>s pueden ir seguidos directamente por un <género> (por ejemplo <definición de variable>s, <definición de visión>s), deben especificarse explícitamente los <subrayado>s en esos <nombre>s o <identificador>s.
3. Cuando una <definición de datos> contiene <instanciaciones de generador>, los <subrayado>s en el <nombre de género> que siguen a la palabra clave NEWTYPE deben especificarse explícitamente.

Un carácter de control tiene el mismo significado que un espacio.

Los caracteres de control y espacios pueden aparecer cualquier número de veces entre dos <unidad léxica>s. Cualquier número de caracteres de control y espacio entre dos <unidad léxica>s tiene el mismo significado que un espacio.

El carácter / seguido inmediatamente del carácter * siempre comienza una <nota>. El carácter * seguido inmediatamente por el carácter / en una <nota> siempre termina la <nota>. Una <nota> puede insertarse antes o después de cualquier <unidad léxica>.

Dentro de un <cuerpo de macro> se aplican reglas léxicas especiales (véase § 4.2.1).

2.2.2 *Reglas de visibilidad e identificadores*

Gramática abstracta

Identificador	::	*Nombre de calificador*
Calificador	=	*Ítem-de-trayecto* +
Ítem-de-trayecto	=	*Calificador-de-sistema* |
		Calificador-de-bloque |
		Calificador-de-subestructura-de-bloque |
		Calificador-de-señal |
		Calificador-de-proceso |
		Calificador-de-procedimiento |
		Calificador-de-género |
Calificador-de-sistema	::	*Nombre-de-sistema*
Calificador-de-bloque	::	*Nombre-de-bloque*
Calificador-de-subestructura-de-bloque	::	*Nombre-de-subestructura-de-bloque*
Calificador-de-proceso	::	*Nombre-de-proceso*
Calificador-de-procedimiento	::	*Nombre-de-procedimiento*
Calificador-de-señal	::	*Nombre-de-señal*
Calificador-de-género	::	*Nombre-de-género*
Nombre	::	*Testigo*

Gramática textual concreta

<identificador> ::=
[<calificador>] <nombre>

<calificador> ::=
<ítem de trayecto> {/<ítem de trayecto>}*

<ítem de trayecto> ::=
<clase de unidad de ámbito> <nombre>

<clase de unidad de ámbito> ::=

SYSTEM
| BLOCK
| SUBSTRUCTURE
| SIGNAL
| PROCESS
| PROCEDURE
| TYPE
| SERVICE

No hay sintaxis abstracta correspondiente para la <clase de unidad de ámbito> denotada por servicio. Los <nombre>s e <identificador>s de entidades definidos en una <definición de servicio> son transformados en <nombre>s respectivamente <identificador>s únicos definidos en la <definición de proceso> que contiene la <definición de servicio>.

El <calificador> refleja la estructura jerárquica desde el nivel de sistema hasta el contexto definidor, y de tal modo que el nivel de sistema es la parte textual más a la izquierda.

Se permite omitir algunos de los <ítem de trayecto>s más a la izquierda (excepto para <definición remota>s, véase § 2.4.1), o todo el <calificador>. Cuando se omite todo el <calificador> y el <nombre> denota una entidad de la clase de entidad que contiene variables, sinónimos, literales y operadores (véase *Semántica* más adelante), la vinculación del <nombre> a una definición debe poder ser resuelta por el contexto efectivo. En otros casos el <identificador> está vinculado a una entidad que tiene su contexto de definición en la unidad de ámbito circundante más cercana en que el <calificador> del <identificador> es el mismo que la parte más a la derecha del <calificador> completo que denota esa unidad de ámbito. Si el <identificador> no contiene un <calificador>, se omite el requisito de concordancia de los <calificador>s.

Una subseñal tiene que estar calificada por su señal progenitora a menos que no exista otra señal visible en ese lugar que tenga el mismo <nombre>.

La resolución por el contexto es posible en los casos siguientes:

a) La unidad de ámbito en que se utiliza el <nombre> no es una <definición parcial de tipo> y contiene una definición que tiene ese <nombre>. El <nombre> estará vinculado a esa definición.

b) La unidad de ámbito en la cual se utiliza el <nombre> no contiene ninguna definición que tenga ese <nombre> o la unidad de ámbito es una <definición parcial de tipo>, y en la totalidad de la <definición de sistema> existe exactamente una definición visible de una entidad que tiene el mismo <nombre> y a la cual se puede vincular el <nombre> sin violar ninguna de las propiedades estáticas (compatibilidad de géneros, etc.) de la construcción en la que ocurre el <nombre>. El <nombre> estará vinculado a esa definición.

Sólo podrán utilizarse identificadores visibles, excepto el <identificador de variable> en una <definición de visión> y el <identificador> utilizado en lugar de un <nombre> en una definición referenciada (es decir, una definición tomada de la <definición de sistema>).

Semántica

Las unidades de ámbito se definen por el siguiente esquema:

Gramática textual concreta	*Gramática gráfica concreta*
<definición de sistema>	<diagrama de sistema>
<definición de bloque>	<diagrama de bloque>
<definición de proceso>	<diagrama de proceso>
<definición de procedimiento>	<diagrama de procedimiento>
<definición de subestructura de bloque>	<diagrama de subestructura de bloque>
<definición de subestructura de canal>	<diagrama de subestructura de canal>
<definición de servicio>	<diagrama de servicio>
<definición parcial de tipo>	
<refinamiento de señal>	

Una unidad de ámbito tiene asociada una lista de definiciones. Cada una de las definiciones define una entidad que pertenece a cierta clase de entidad y tiene un nombre asociado. Para una <definición parcial de tipo>, la lista asociada de definiciones consiste en las <signatura de operador>s, las <signatura de literal>s y cualesquiera <signatura de operador>s y <signatura de literal>s heredadas de un género progenitor, de una instancia de generador, o implicada por el uso de notaciones taquigráficas tales como la palabra clave ORDERING (véase § 5.4.1.8). Obsérvese que una <definición de visión> no define una entidad.

Aunque los <operador infijo>s, <operador>s con signo de admiración y <cadena de caracteres>s tienen su notación sintáctica propia, son en efecto <nombre>s, y se representan en sintaxis abstracta por un nombre. En lo sucesivo se tratarán como si fuesen (también sintácticamente) <nombre>s. Sin embargo, los <nombre de estado>s, <nombre de conector>s, <nombre formal de generador>s, <identificador de valor>s en ecuaciones, <nombre formal de macro>s y <nombre de macro>s tienen reglas de visibilidad especiales y por lo tanto no pueden ser calificados. Los <nombre de estado>s y <nombre de conector>s no son visibles fuera de un <cuerpo de proceso>, <cuerpo de procedimiento> o <cuerpo de servicio> respectivamente. En las secciones apropiadas se explican otras reglas de visibilidad.

Se dice que cada entidad tiene su contexto definidor en la unidad de ámbito que la define. Las entidades son referenciadas por medio de <identificador>s.

El <calificador> dentro de un <identificador> especifica unívocamente el contexto definidor del <nombre>.

Existen las siguientes clases de entidades:

a) sistema
b) bloques
c) canales, rutas de señales
d) señales, temporizadores
e) procesos
f) procedimientos
g) variables (incluidos parámetros formales), sinónimos, literales, operadores
h) géneros
i) generadores
j) entidades importadas
k) listas de señales
l) servicios
m) subestructuras de bloque, subestructuras de canal

Se dice que un identificador es visible en una unidad de ámbito

a) si la parte nombre del <identificador> tiene su contexto definidor en esa unidad de ámbito, o
b) si es visible en la unidad de ámbito que definió esa unidad de ámbito, o
c) si la unidad de ámbito contiene una <definición parcial de tipo> en la que se define el <identificador>, o
d) si la unidad de ámbito contiene una <definición de señal> en la que se define el <identificador>.

Dos definiciones en la misma unidad de ámbito y pertenecientes a la misma clase de entidad no pueden tener el mismo <nombre>. Una excepción la constituyen las definiciones de <signatura de operador> y <signatura de literal> en la misma <definición parcial de tipo> (véase § 5.2.2): dos o más operadores y/o literales pueden tener el mismo <nombre> con diferentes <género de argumento>s o diferente género <resultado>.

Otra excepción la constituyen las entidades importadas. Para esta clase de entidad los pares de (<nombre de importación>, <género>) en las <definición de importación>s en la unidad de ámbito tienen que ser distintos.

En la gramática textual concreta, el nombre o identificador opcional en una definición después de la palabra clave de terminación (ENDSYSTEM, ENDBLOCK, etc.) debe ser sintácticamente el mismo que el nombre o identificador que sigue a la correspondiente palabra clave de comienzo (SYSTEM, BLOCK, etc., respectivamente).

2.2.3 *Texto informal*

Gramática abstracta

Texto-informal :: ...

Gramática textual concreta

<texto informal> ::=
<cadena de caracteres>

Semántica

Si se utiliza texto informal en una especificación de sistema LED, debe tenerse presente que este texto no es LED formal, es decir, LED no le da ninguna semántica. La semántica del texto informal puede definirse por otros medios.

2.2.4 *Reglas de dibujo*

El usuario podrá elegir el tamaño de los símbolos gráficos.

Los símbolos no deben superponerse ni cruzarse unos con otros. Una excepción a esta regla la constituyen los símbolos formados por líneas, por ejemplo <símbolo de canal>, <símbolo de ruta de señales>, <símbolo de línea de crear>, <símbolo de línea de flujo>, <símbolo de asociación de trazo continuo> y <símbolo de asociación de trazo discontinuo>, que pueden cruzarse unos con otros. El cruce de un símbolo con otro no conlleva una asociación lógica.

El metasímbolo **is followed by** implica un <símbolo de línea de flujo>.

Los símbolos formados por líneas pueden consistir en uno o más segmentos de recta.

Un <símbolo de línea de flujo> debe tener una flecha cuando entra en otro <símbolo de línea de flujo>, en un <símbolo de conector de salida> o en un <símbolo de estado siguiente>. En otros casos, las flechas son opcionales en los <símbolo de línea de flujo>s. Los <símbolo de línea de flujo>s son horizontales o verticales.

Se permiten imágenes especulares verticales de <símbolo de entrada>, <símbolo de salida>, <símbolo de comentario> y <símbolo de ampliación de texto>.

El argumento de la derecha del metasímbolo **is associated with** debe estar más cerca del argumento de la izquierda que de cualquier otro símbolo gráfico. Los elementos sintácticos del argumento de la derecha deben distinguirse unos de otros.

El texto dentro de un símbolo gráfico debe leerse de izquierda a derecha, comenzando por la esquina superior izquierda. El borde derecho del símbolo se interpreta como un carácter de nueva línea, indicativo de que la lectura debe continuar en el punto situado más a la izquierda de la línea siguiente (si existe).

2.2.5 *Partición de diagramas*

La siguiente definición de partición de diagrama no forma parte de la *gramática gráfica concreta*, pero se utiliza el mismo metalenguaje.

<página> ::=
<símbolo de casilla> **contains**
{<área de encabezamiento> <área de número de página>
{<unidad sintáctica>} *}

<área de encabezamiento> ::=
<símbolo de texto implícito> **contains** <encabezamiento>

<área de número de página> ::=
<símbolo de texto implícito> **contains** [<número de página> [(<número de páginas>)]]

<número de página> ::=
<nombre de literal>

<número de páginas> ::=

<nombre de literal natural>

La <página> es un no terminal de comienzo, por lo cual no se hace referencia al mismo en ninguna regla de producción. Un diagrama puede dividirse (particionarse) en varias <página>s, en cuyo caso el <símbolo de casilla> que delimita el diagrama y el <encabezamiento> del diagrama se reemplazan por un <símbolo de casilla> y un <encabezamiento> para cada <página>.

El usuario de LED puede decidir que el límite del medio en el que se reproducen los diagramas implique los <símbolo de casilla>s.

El <símbolo de texto implícito> no se muestra, pero está implícito para obtener una clara separación entre <área de encabezamiento> y <área de número de página>. El <área de encabezamiento> se sitúa en la esquina superior izquierda del <símbolo de casilla>. El <área de número de página> se coloca en la esquina superior derecha del <símbolo de casilla>. El <encabezamiento> y la <unidad sintáctica> dependen del tipo de diagrama.

2.2.6 *Comentario*

Un comentario es una notación para representar comentarios asociados con símbolos o texto.

En la *gramática textual concreta* se utilizan dos formas de comentarios. La primera es la <nota>, definida en § 2.2.1.

En la figura 2.9.1 y en la figura 2.9.3 se muestran ejemplos.

La sintaxis concreta de la segunda forma es:

<fin> ::=
[<comentario>];

<comentario> ::=
COMMENT <cadena de caracteres>

En la figura 2.9.2 se muestra un ejemplo.

En la *gramática gráfica concreta* se utiliza la sintaxis siguiente:

<área de comentario> ::=
<símbolo de comentario> **contains** <texto>
is connected to <símbolo de asociación de trazo discontinuo>

<símbolo de comentario> ::=

<símbolo de asociación de trazo discontinuo> ::=

Un extremo del <símbolo de asociación de trazo discontinuo> debe estar conectado al centro del segmento vertical del <símbolo de comentario>.

Un <símbolo de comentario> puede conectarse a cualquier símbolo gráfico por medio de un <símbolo de asociación de trazo discontinuo>. El <símbolo de comentario> se considera un símbolo cerrado completando (en la imaginación) el rectángulo para encerrar el texto. Contiene texto de comentario relacionado con el símbolo gráfico.

En la figura 2.9.4, en § 2.9, se muestra un ejemplo.

2.2.7 *Ampliación de texto*

<área de ampliación de texto> ::=
<símbolo de ampliación de texto> **contains** <texto>
is connected to <símbolo de asociación de trazo continuo>

<símbolo de ampliación de texto> ::=
<símbolo de comentario>

<símbolo de asociación de trazo continuo> ::=

Un extremo del <símbolo de asociación de trazo continuo> tiene que estar conectado con el centro del segmento vertical del <símbolo de ampliación de texto>.

Un <símbolo de ampliación de texto> puede conectarse con cualquier símbolo gráfico por medio de un <símbolo de asociación de trazo continuo>. El <símbolo de ampliación de texto> se considera un símbolo cerrado completando (en la imaginación) el rectángulo.

El texto contenido en el <símbolo de ampliación de texto> es una continuación del texto dentro del símbolo gráfico y se considera contenido en ese símbolo.

2.2.8 *Símbolo de texto*

<símbolo de texto> se utiliza en cualquier diagrama. El contenido depende del diagrama.

<símbolo de texto> ::=

2.3 *Conceptos básicos de datos*

El concepto de datos en LED se define en el § 5; comprende la terminología de datos LED, los medios para definir nuevos tipos de datos, y facilidades de datos predefinidas.

Aparecen datos en definiciones de tipo de datos, expresiones, la aplicación de operadores, variables, valores y literales.

2.3.1 *Definiciones de tipos de datos*

Los datos en LED están principalmente relacionados con los tipos de datos. Un tipo de datos define conjuntos de valores, un conjunto de operadores que puede aplicarse a estos valores y un conjunto de reglas algebraicas (ecuaciones) que definen el comportamiento de estos operadores cuando son aplicados a los valores. Los valores, los operadores y las reglas algebraicas definen colectivamente las propiedades del tipo de datos. Estas propiedades son definidas por definiciones de tipo de datos.

LED permite la definición de cualquier tipo de datos que se necesite, incluidos mecanismos de composición (tipos compuestos), con la sola condición de que tal definición pueda especificarse formalmente. En cambio, a los efectos de lenguajes de programación hay consideraciones de implementación que requieren que el conjunto de tipos de datos disponible y, en particular, el mecanismo de composición proporcionado (matriz, estructura, etc.), estén limitados.

2.3.2 *Variable*

Variables son objetos a los cuales puede asociarse un valor por asignación. Cuando se accede a la variable, se obtiene el valor asociado.

2.3.3 *Valores y literales*

Un conjunto de valores con ciertas características se denomina un género. Los operadores se definen para que actúen desde y hacia valores de géneros. Por ejemplo, la aplicación del operador de suma («+») desde y hacia valores del género entero es válida, en tanto que la suma de valores del género booleano no lo es.

Todos los géneros tienen por lo menos un valor. Todo valor pertenece a un género, y solamente a uno; es decir, los géneros no pueden tener valores en común.

La mayor parte de los géneros tienen formas literales para denotar valores del género (por ejemplo, para enteros se utiliza «2» más bien que «1 + 1». Puede haber más de un literal para denotar el mismo valor (por ejemplo 12 y 012 pueden utilizarse para denotar el mismo valor entero). La misma denotación literal puede utilizarse para más de un género; por ejemplo 'A' es un carácter, pero es también una cadena de caracteres de longitud 1. Algunos géneros pueden no tener literales; por ejemplo, un valor compuesto a menudo no tiene literales propios pero sí valores definidos por operaciones de composición sobre valores de sus componentes.

2.3.4 *Expresiones*

Una expresión denota un valor. Si una expresión no contiene una variable, por ejemplo si es un literal de un género dado, toda ocurrencia de la expresión denotará siempre el mismo valor. Una expresión que contiene variables puede interpretarse como valores diferentes durante la interpretación de un sistema LED según el valor asociado a las variables.

2.4 *Estructura de sistema*

2.4.1 *Definiciones remotas*

Una <definición remota> es una definición que ha sido sacada de su contexto definidor para obtener una mejor sinopsis. Es similar a una definición de macro (véase § 4.2), pero es «llamada» únicamente desde un solo lugar (el contexto definidor) utilizando una referencia.

Gramática concreta

```
<definición remota> ::=
        <definición> | <diagrama>

<definición de sistema> ::=
            {<definición textual de sistema> | <diagrama de sistema>}
            {<definición remota>}*

<definición> ::=
            <definición de bloque>
          | <definición de proceso>
          | <definición de procedimiento>
          | <definición de subestructura de bloque>
          | <definición de subestructura de canal>
          | <definición de servicio>
          | <definición de macro>

<diagrama> ::=
            <diagrama de bloque>
          | <diagrama de proceso>
          | <diagrama de procedimiento>
          | <diagrama de subestructura de bloque>
          | <diagrama de subestructura de canal>
          | <diagrama de servicio>
          | <diagrama de macro>
```

Para cada <definición remota>, excepto <definición de macro> y <diagrama de macro>, tiene que haber una referencia en la <definición de sistema>, el <diagrama de sistema>, u otra <definición remota>.

Para cada referencia tiene que haber una <definición remota> correspondiente.

En cada <definición remota> tiene que haber un <identificador> inmediatamente después de la palabra clave inicial. El <calificador> en este <identificador> debe estar completo, o debe omitirse. Si se omite el calificador, el nombre tiene que ser único en la definición de sistema, dentro de la clase de entidad de la <definición remota>. No está permitido especificar un <calificador> en el <identificador> después de la palabra clave inicial para definiciones que no son <definición remota>s (es decir, un <nombre> tiene que estar especificado para definiciones normales).

Semántica

Antes de que pueda analizarse una <definición de sistema concreto>, hay que sustituir cada referencia por la <definición remota> correspondiente. En esta sustitución, el <identificador> de la <definición remota> se sustituye por el <nombre> en la referencia.

2.4.2 *Sistema*

Gramática abstracta

Definición-de-sistema	::	*Nombre-de-sistema* *Definición-de-bloque*-**set** *Definición-de-canal*-**set** *Definición-de-señal*-**set** *Definición-de-tipo-de-datos* *Definición-de-tipo-de-sinónimo*-**set**
Nombre-de-sistema	=	*Nombre*

Una *definición-de-sistema* tiene un nombre que puede utilizarse en calificadores.

Una *definición-de-sistema* debe contener por lo menos una *definición-de-bloque.*

Las definiciones de todos los canales, señales, tipos de datos, sintipos utilizados en el interfaz con el entorno y entre bloques del sistema están contenidas en la *definición-de-sistema.* Todos los datos predefinidos se consideran definidos a nivel de sistema.

Gramática textual concreta

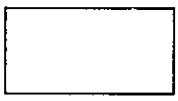
```
<definición textual de sistema> ::=
                    SYSTEM <nombre de sistema> <fin>
                    {<definición de bloque
                    |     <referencia textual de bloque>
                    |     <definición de canal>
                    |     <definición de señal>
                    |     <definición de lista de señales>
                    |     <definición de seleccionar>
                    |     <definición de macro>
                    |     <definición de datos> } +
                              ENDSYSTEM [<nombre de sistema>] <fin>

<referencia textual de bloque>    ::=
                    BLOCK <nombre de bloque> REFERENCED <fin>
```

<definición de seleccionar> se define en § 4.3.3, <definición de macro> en § 4.2, <definición de datos> en § 5.5.1, <definición de bloque> en § 2.4.3, <definición de canal> en § 2.5.1, <definición de señal> en § 2.5.4, <definición de lista de señales> en § 2.5.5.

Un ejemplo de <definición de sistema> se muestra en la figura 2.9.5, en § 2.9.

Gramática gráfica concreta

```
<diagrama de sistema> ::=
                    <símbolo de casilla> contains
                         {<encabezamiento de sistema>
                         {     {<área de texto de sistema>}*
                               {<diagrama de macro>}*
                                <área de interacción de bloques>} set}

<símbolo de casilla> ::=
```



```
<encabezamiento de sistema> ::=
                    SYSTEM <nombre de sistema>

<área de texto de sistema> ::=
                    <símbolo de texto> contains
                         {<definición de señal>
                         |     <definición de lista de señales>
                         |     <definición de datos>
                         |     <definición de macro>
                         |     <definición de seleccionar>}*
```

<área de interacción de bloque> ::=
{<área de bloque>
| <área de definición de canal>}+

<área de bloque> ::=
<referencia gráfica de bloque>
| <diagrama de bloque>

<referencia gráfica de bloque> ::=
<símbolo de bloque> **contains** <nombre de bloque>

<símbolo de bloque> ::=

<definición de seleccionar> se define en § 4.3.3, <definición de macro> y <diagrama de macro> en § 4.2, <definición de datos> en § 5.5.1, <diagrama de bloque> en § 2.4.3, <área de definición de canal> en § 2.5.1, <definición de señal> en § 2.5.4, <definición de lista de señales> en § 2.5.5.

Definición-de-bloque-**set** en la *gramática abstracta* corresponde a las <área de bloque>s, *definición-de-canal*-**set** corresponde al <área de definición de canal>.

En la figura 2.9.6 se muestra un ejemplo de un <diagrama de sistema>.

Semántica

Una *definición-de-sistema* es la representación LED de una especificación o descripción de un sistema.

Un sistema está separado de su entorno por la frontera del sistema y contiene un conjunto de bloques. La comunicación entre el sistema y el entorno o entre bloques dentro del sistema sólo puede efectuarse mediante señales. Dentro de un sistema, estas señales son transportadas por canales. Los canales conectan bloques entre sí o con la frontera del sistema.

Antes de interpretar una *definición-de-sistema* se elige un subconjunto consistente (véase § 3.2.1). Este subconjunto se denomina una instancia de la *definición-de-sistema.* Una instancia de un sistema es una instanciación de un tipo de sistema definido por una *definición-de-sistema.* La interpretación de una instancia de una *definición-de-sistema* es efectuada por una máquina LED abstracta que, al hacerlo, da semántica a los conceptos LED. Interpretar una instancia de una *definición-de-sistema* es:

a) iniciar el tiempo del sistema,
b) interpretar los bloques y sus canales conectados que están contenidos en el subconjunto de partición consistente seleccionado.

2.4.3 *Bloque*

Gramática abstracta

Definición-de-bloque	::	*Nombre-de-bloque* *Definición-de-proceso*-**set** *Definición-de-señal*-**set** *Conexión-de-canal-a-ruta*-**set** *Definición-de-ruta-de-señales*-**set** *Definición-de-tipo-de-datos* *Definición-de-tipo-de-sinónimo*-**set** [*Definición-de-subestructura-de-bloque*]
Nombre-de-bloque	=	*Nombre*

A menos que una *definición-de-bloque* contenga una *definición-de-subestructura-de-bloque*, debe haber por lo menos una *definición-de-proceso* y una *definición-de-ruta-de-señales* dentro del bloque.

Es posible efectuar actividades de partición sobre los bloques que especifican *definición-de-subestructura-de-bloque*; esta característica del lenguaje se trata en § 3.2.2.

Gramática textual concreta

<definición de bloque> ::=

BLOCK {<nombre de bloque>|<identificador de bloque>}<fin>
{<definición de señal>
| <definición de lista de señales>
| <definición de proceso>
| <referencia textual de proceso>
| <definición de ruta de señales>
| <definición de macro>
| <definición de datos>
| <definición de seleccionar>
| <conexión de canal a ruta>}*
[<definición de subestructura de bloque>|<referencia textual de subestructura de bloque>]
ENDBLOCK [<nombre de bloque>|<identificador de bloque>] <fin>

<referencia textual de proceso> ::=

PROCESS <nombre de proceso>[<número de instancias>] REFERENCED <fin>

<definición de señal> se define en § 2.5.4, <definición de lista de señales> en § 2.5.5, <definición de proceso> en § 2.4.4, <definición de ruta de señales> en § 2.5.2, <conexión de canal a ruta> en § 2.5.3, <definición de subestructura de bloque> y <referencia textual de subestructura de bloque> se definen en § 3.2.2, <definición de macro> en § 4.2.2 y <definición de datos> en § 5.5.1.

En la figura 2.9.7, en § 2.9, se muestra un ejemplo de <definición de bloque>.

Gramática gráfica concreta

<diagrama de bloque> ::=
<símbolo de casilla>
contains {<encabezamiento de bloque>
{{<área de texto de bloque>}*{<diagrama de macro>}*
[<área de interacción de procesos>] [<área de subestructura de bloque>]} **set**}
is associated with {<identificador de canal>}*

El <identificador de canal> identifica un canal conectado a una ruta de señales en el <diagrama de bloque>. Se coloca fuera del <símbolo de casilla>, próximo al punto extremo de la ruta de señales en el <símbolo de casilla>. Si el <diagrama de bloque> no contiene un <área de interacción de procesos>, deberá contener un <área de subestructura de bloque>.

<encabezamiento de bloque> ::=
BLOCK {<nombre de bloque>|<identificador de bloque>}

<área de texto de bloque> ::=
<área de texto de sistema>

<área de interacción de procesos> ::=
{<área de proceso>
| <área de línea de crear>
| <área de definición de ruta de señales>}+

<área de proceso> ::=
<referencia gráfica de proceso>|<diagrama de proceso>

<referencia gráfica de proceso> ::=
<símbolo de proceso> **contains** <nombre de proceso>[<número de instancias>]}

<símbolo de proceso> ::=

<número de instancias> se define en § 2.4.4.

<área de línea de crear> ::=
<símbolo de línea de crear>
is connected to {<área de proceso> <área de proceso>}

<símbolo de línea de crear> ::=

La punta de flecha en el <símbolo de línea de crear> indica el <área de proceso> sobre la cual se ejecuta la acción de crear.

<diagrama de proceso> se define en § 2.4.4, <área de definición de ruta de señales> en § 2.5.2, <área de subestructura de bloque> en § 3.2.2, <diagrama de macro> en § 4.2.2.

En la figura 2.9.8, en § 2.9, se muestra un ejemplo de <diagrama de bloque>.

Semántica

Una definición de bloque es un contenedor para una o más definiciones de proceso de un sistema y/o una subestructura de bloque. La definición de bloque tiene por finalidad agrupar procesos que, como un todo, realizan cierta función, sea directamente, sea mediante una subestructura de bloque.

Una definición de bloque proporciona un interfaz de comunicación estático en virtud del cual sus procesos pueden comunicar con otros procesos. Además, establece el ámbito para definiciones de proceso.

Interpretar un bloque es crear los procesos iniciales en el bloque.

2.4.4 *Proceso*

Gramática abstracta

Definición-de-proceso	::	*Nombre-de-proceso* *Número-de-instancias* *Parámetro-formal-de-proceso* * *Definición-de-procedimiento*-**set** *Definición-de-señal*-**set** *Definición-de-tipo-de-datos* *Definición-de-tipo-de-sinónimo*-**set** *Definición-de-variable*-**set** *Definición-de-visión*-**set** *Definición-de-temporizador*-**set** *Gráfico-de-proceso*
Número-de-instancias	::	*Ento Ento*
Nombre-de-proceso	=	*Nombre*
Gráfico-de-proceso	::	*Nodo-de-arranque-de-proceso* *Nodo-de-estado*-**set**
Parámetro-formal-de-proceso	::	*Nombre-de-variable* *Identificador-de-referencia-de-género*

Gramática textual concreta

```
<definición de proceso> ::=
        PROCESS {<identificador de proceso>|<nombre de proceso>}
            [<número de instancias>] <fin>
                [<parámetros formales> <fin>] [<conjunto de señales de entrada
                válidas>]
                    {<definición de señal>
                    | <definición de lista de señales>
                    | <definición de procedimiento>
                    | <referencia textual de procedimiento>
                    | <definición de macro>
                    | <definición de datos>
                    | <definición de variable>
                    | <definición de visión>
                    | <definición de seleccionar>
                    | <definición de importación>
                    | <definición de temporizador>}*
                            {<cuerpo de proceso>
                            | <descomposición de servicio>}
        ENDPROCESS [<nombre de proceso>|<identificador de proceso>] <fin>

<referencia textual de procedimiento> ::=
        PROCEDURE <nombre de procedimiento> REFERENCED <fin>

<conjunto de señales de entrada válidas> ::=
        SIGNALSET [<lista de señales>] <fin>

<cuerpo de proceso> ::=
            <arranque>{<estado>}*
```

<parámetros formales> ::=
FPAR <nombre de variable>{,<nombre de variable>}*<género>
{,<nombre de variable>{,<nombre de variable>}*<género>}*

<número de instancias> ::=
([<número inicial>], [<número máximo>])

<número inicial> ::=
<expresión simple natural>

<número máximo> ::=
<expresión simple natural>

El número inicial de instancias y el número máximo de instancias contenidos en *número-de-instancias* se derivan de <número de instancias>. Si <número inicial> se omite, el <número inicial> es 1. Si <número máximo> se omite, entonces el <número máximo> no está acotado.

El <número de instancias> utilizado en la derivación es el siguiente:

a) Si no hay <referencia textual de proceso> para el proceso, se utiliza el <número de instancias> en la <definición de proceso>. Si no contiene un <número de instancias>, se utiliza el <número de instancias> en que se omiten el <número inicial> y el <número máximo>.

b) Si se omite el <número de instancias> en la <definición de proceso> y en una <referencia textual de proceso>, se utiliza el <número de instancias> en que se omiten el <número inicial> y el <número máximo>.

c) Si se omite el <número de instancias> en la <definición de proceso> o en una <referencia textual de proceso>, el <número de instancias> es el que está presente.

d) Si se especifica el <número de instancias> en la <definición de proceso> y en una <referencia textual de proceso>, los dos <número de instancias> deben ser léxicamente iguales, y se utiliza ese <número de instancias>.

Se aplica una relación similar para el <número de instancias> especificado en el <diagrama de proceso> y en la <referencia gráfica de proceso>, como se define a continuación.

<definición de señal> se define en § 2.5.4, <definición de lista de señales> en § 2.5.5, <definición de visión> en § 2.6.1.2, <definición de variable> en § 2.6.1.1, <definición de procedimiento> en § 2.4.5, <definición de temporizador> en § 2.8, <definición de macro> en § 4.2.2, <definición de importación> en § 4.1.3, <definición de seleccionar> en § 4.3.3, <expresión simple> en § 4.3.2, <descomposición de servicio> en § 4.10.1, <definición de datos> en § 5.5.1.

El <número inicial> de instancias tiene que ser inferior o igual al <número máximo> y el <número máximo> tiene que ser mayor que cero.

La utilización de <conjunto de señales de entrada válidas> se define en § 2.5.2, *modelo.*

Un ejemplo de <definición de proceso> se muestra en la figura 2.9.9, en § 2.9.

Gramática gráfica concreta

<diagrama de proceso> ::=
<símbolo de casilla>
contains {<encabezamiento de proceso>
{{<área de texto de proceso>}*
{<área de procedimiento>}*
{<diagrama de macro>}*
{<área de gráfico de proceso>|<área de interacción de servicios>}} **set**}
[**is associated with** {<identificador ruta de señales>}+]

El <identificador de ruta de señales> identifica una ruta externa de señales conectada a una ruta de señales en el <diagrama de proceso>. Se coloca fuera del <símbolo de casilla>, próximo al punto extremo de la ruta de señales en el <símbolo de casilla>.

<área de texto de proceso> ::=
<símbolo de texto> **contains** {
[<conjunto de señales de entrada válidas>]
{<definición de señal>
| <definición de lista de señales>
| <definición de variable>
| <definición de visión>
| <definición de importación>
| <definición de datos>
| <definición de macro>
| <definición de temporizador>
| <definición de seleccionar> }*}

<encabezamiento de proceso> ::=
PROCESS {<nombre de proceso>|<identificador de proceso>}
[<número de instancias> [<fin>]]
[<parámetros formales>]

<área de gráfico de proceso> ::=
<área de arranque>{<área de estado>|<área de conector de entrada> }*

<definición de señal> se define en § 2.5.4, <definición de lista de señales> en § 2.5.5, <definición de visión> en § 2.6.1.2, <definición de variable> en § 2.6.1.1, <área de procedimiento> en § 2.4.5, <definición de temporizador> en § 2.8, <definición de macro> y <diagrama de macro> en § 4.2.2, <definición de importación> en § 4.1.3, <definición de seleccionar> en § 4.3.3, <definición de datos> en § 5.5.1, <área de arranque> en § 2.6.2, <área de estado> en § 2.6.3, <área de conector de entrada> en § 2.6.6 y <área de interacción de servicios> en § 4.10.1.

Un ejemplo de <diagrama de proceso> se muestra en la figura 2.9.10, en § 2.9.

Semántica

Una definición de proceso introduce el tipo de un proceso con que se desea representar un comportamiento dinámico.

En el *número-de-instancias*, el primer valor representa el *número-de-instancias* del proceso que existe cuando se crea el sistema, y el segundo valor representa el número máximo de instancias simultáneas del tipo de proceso.

Una instancia de proceso es una máquina de estados finitos ampliada, que comunica, y que realiza cierto conjunto de acciones, denominadas como transiciones, en función de la recepción de una señal determinada, cuando se encuentra en un estado. Después de efectuada una transición, el proceso se encontrará en espera en otro estado, que no tiene que ser necesariamente diferente del precedente.

El concepto de máquina de estados finitos se ha ampliado en el sentido de que el estado resultante después de una transición puede ser afectado, no sólo por la señal que causa la transición, sino también por decisiones tomadas sobre variables conocidas por el proceso.

Varias instancias del mismo tipo de proceso pueden existir al mismo tiempo y actuar asíncronamente y en paralelo, entre sí y con otras instancias de un tipo de proceso diferente en el sistema.

Cuando se crea un sistema, los procesos iniciales se crean en un orden aleatorio. La comunicación de señales entre los procesos sólo comienza después de que se hayan creado todos los procesos iniciales. Los parámetros formales de estos procesos iniciales se inicializan a un valor indefinido.

Las instancias de proceso existen desde el instante en que un sistema es creado o puede ser creado por acciones de petición de crear que arrancan los procesos que se están interpretando; la interpretación comienza cuando se interpreta la acción de arrancar; las instancias de proceso pueden dejar de existir ejecutando acciones de parada.

Las señales recibidas por instancias de proceso se denominan señales de entrada, y las señales enviadas a instancias de proceso se denominan señales de salida.

Las señales sólo pueden ser consumidas por una instancia de proceso cuando ésta se encuentra en un estado. El conjunto completo de señales de entrada válidas es la unión del conjunto de señales en todas las rutas de señales que conducen al proceso, el <conjunto de señales de entrada válidas>, las señales implícitas y las señales de temporizador.

Un puerto de entrada, y sólo uno, está asociado a cada instancia de proceso. Cuando una señal de entrada llega al proceso, se aplica al puerto de entrada de la instancia de proceso.

El proceso está, bien en espera en un estado, bien efectuando una transición. Para cada estado hay un conjunto de señales de conservación (véase también § 2.6.5). Cuando el proceso se encuentra en espera en un estado, la primera señal de entrada cuyo identificador no forme parte del conjunto de señales de conservación es extraída de la cola y consumida por el proceso.

El puerto de entrada puede retener cualquier número de señales de entrada, de modo que varias señales de entrada formen cola para la instancia de proceso. El conjunto de señales retenidas son ordenadas en la cola según el instante de llegada. Si dos o más señales llegan «simultáneamente» por trayectos diferentes, se ordenan arbitrariamente.

Cuando se crea el proceso, se le da un puerto de entrada vacío, y se crean variables locales a las cuales se asignan valores.

Los parámetros formales son variables creadas, bien cuando el sistema es creado (pero no se le han pasado parámetros efectivos, por lo cual aún no están inicializados), bien cuando la instancia del proceso es creada dinámicamente.

Para todas las instancias de proceso pueden utilizarse cuatro expresiones que dan un valor de PId (véase § 5.6.10): SELF, PARENT, OFFSPRING y SENDER. Estas expresiones dan un resultado para:

a) la instancia de proceso (SELF);

b) la instancia del proceso creador (PARENT);

c) la última instancia de proceso creada por el proceso (OFFSPRING);

d) la instancia de proceso a partir de la cual se ha consumido la última señal de entrada (SENDER) (véase también § 2.6.4).

Estas expresiones se explican con más detalle en § 5.5.4.3.

SELF (MISMO), PARENT (PROGENITOR), OFFSPRING (VÁSTAGO) y SENDER (EMISOR) pueden utilizarse en expresiones dentro de las instancias de proceso.

Para todas las instancias de proceso presentes en la inicialización del sistema, la expresión predefinida PARENT siempre tiene el valor NULL.

Para todas las instancias de proceso de nueva creación, las expresiones predefinidas SENDER y OFFSPRING tienen el valor NULL.

2.4.5 *Procedimiento*

Los procedimientos se definen por medio de definiciones de procedimiento. El procedimiento es invocado mediante una llamada a procedimiento que hace referencia a la definición de procedimiento. Una llamada a procedimiento tiene asociados parámetros; éstos se utilizan para pasar valores y también para controlar el ámbito de las variables para la ejecución del procedimiento. El mecanismo de paso de los parámetros controla qué variables son afectadas por la interpretación de un procedimiento.

Gramática abstracta

Definición-de-procedimiento	::	*Nombre-de-procedimiento* *Parámetro-formal-de-procedimiento* * *Definición-de-procedimiento*-**set** *Definición-de-tipo-de-datos* *Definición-de-tipo-de-sinónimo*-**set** *Definición-de-variable*-**set** *Gráfico-de-procedimiento*
Nombre-de-procedimiento	=	*Nombre*
Parámetro-formal-de-procedimiento	=	*Parámetro-de-entrada* | *Parámetro-de-entrada-salida*
Parámetro-de-entrada	::	*Nombre-de-variable* *Identificador-de-referencia-de-género*

Parámetro-de-entrada-salida :: *Nombre-de-variable*
Identificador-de-referencia-de-género

Gráfico-de-procedimiento :: *Nodo-de-arranque-de-procedimiento*
Nodo-de-estado-**set**

Nodo-de-arranque-de-procedimiento :: *Transición*

Gramática textual concreta

```
<definición de procedimiento> ::=
        PROCEDURE {<identificador de procedimiento>|<nombre de procedimiento> }
        <fin>
            [parámetros formales de procedimiento> <fin>]
                {<definición de datos>
                |  <definición de variable>
                |  <referencia textual de procedimiento>
                |  <definición de procedimiento>
                |  <definición de seleccionar>
                |  <definición de macro> }*
                    <cuerpo de procedimiento>
        ENDPROCEDURE [<nombre de procedimiento>|<identificador de procedi-
        miento>] <fin>

<parámetros formales de procedimiento> ::=
                    FPAR <parámetro formal de variable>
                        {,<parámetro formal de variable> }*

<parámetro formal de variable>::=
                    [IN/OUT
                |   IN]
                <nombre de variable>{,<nombre de variable>}*<género>

<cuerpo de procedimiento> ::=
                <cuerpo de proceso>
```

<definición de variable> se define en § 2.6.1.1, <referencia textual de procedimiento> en § 2.4.4, <definición de macro> en § 4.2, <definición de seleccionar> en § 4.3.3, <definición de datos> en § 5.5.1, <género> en § 5.2.2.

En una <definición de procedimiento>, <definición de variable> no puede contener <nombre de variable>s REVEALED, EXPORTED, REVEALED EXPORTED, EXPORTED REVEALED (véase § 2.6.1). La figura 2.9.11 muestra un ejemplo de <definición de procedimiento>.

Gramática gráfica concreta

```
<diagrama de procedimiento> ::=
            <símbolo de casilla> contains {<encabezamiento de procedimiento>
                {{<área de texto de procedimiento>
                |  <área de procedimiento>
                |  <diagrama de macro> }*
                        <área de gráfico de procedimiento> } set }

<área de procedimiento> ::=
            <referencia gráfica de procedimiento>
            |  <diagrama de procedimiento>

<área de texto de procedimiento> ::=
            <símbolo de texto> contains
                {<definición de variable>
                |  <definición de datos>
                |  <definición de seleccionar>
                |  <definición de macro> }*
```

<referencia gráfica de procedimiento> ::=
<símbolo de procedimiento> **contains** <nombre de procedimiento>

<símbolo de procedimiento> ::=

<encabezamiento de procedimiento> ::=
PROCEDURE {<nombre de procedimiento>|<identificador de procedimiento>}
[<parámetros formales de procedimiento>]

<área de gráfico de procedimiento> ::=
<área de arranque de procedimiento>
{<área de arranque>|<área de conector de entrada>}*

<área de arranque de procedimiento> ::=
<símbolo de arranque de procedimiento> **is followed by** <área de transición>

<símbolo de arranque de procedimiento> ::=

<definición de variable> se define en § 2.6.1.1, <área de transición> en § 2.6.7.1, <área de estado> en § 2.6.3, <área de conector de entrada> en § 2.6.6, <definición de macro> y <diagrama de macro> en § 4.2, <definición de seleccionar> en § 4.3.3 y <definición de datos> en § 5.5.1.

La figura 2.9.12, en § 2.9, muestra un ejemplo de <diagrama de procedimiento>.

Semántica

Un procedimiento es un medio de dar un nombre a un grupo de ítems y representar este grupo por una sola referencia. Las reglas para los procedimientos prescriben la forma de elegir el grupo de ítems, y limitan el ámbito del nombre de variables definidas en el procedimiento.

Una variable de procedimiento es una variable local dentro del procedimiento, que no puede ser ni revelada, ni vista, ni exportada, ni importada. Se crea cuando se interpreta el nodo de arranque de procedimiento, y deja de existir cuando se interpreta el nodo de retorno del gráfico de procedimiento.

Cuando se interpreta una definición, se interpreta su gráfico de procedimiento.

Una definición de procedimiento sólo es interpretada cuando la llama una instancia de proceso, y es interpretada por esa instancia de proceso.

La interpretación de una definición de procedimiento causa la creación de una instancia de procedimiento y hace que la interpretación comience de la manera siguiente:

a) Para cada *parámetro-de-entrada* se crea una variable local que tiene el *nombre* y el *género* del *parámetro-de-entrada*. Se asigna a la variable el valor de la expresión dada por el parámetro efectivo correspondiente, que puede ser indefinido.

b) Si un parámetro efectivo está vacío, al parámetro formal correspondiente se le da el valor indefinido.

c) Un parámetro formal sin atributo explícito tiene el atributo IN implícito.

d) Para cada *definición-de-variable* en la *definición-de- procedimiento* se crea una variable local que tiene el mismo *nombre* y *género* de la *definición-de-variable.*

e) Cada *parámetro-de-entrada-salida* denota un nombre sinónimo para la variable que se da en la expresión del parámetro efectivo. Este nombre sinónimo se utiliza durante toda la interpretación del *gráfico-de-procedimiento* cuando se hace referencia al valor de la variable o cuando se asigna un nuevo valor a la variable.

f) La *transición* contenida en el *nodo-de-arranque-de-procedimiento* se interpreta.

2.5 *Comunicación*

2.5.1 *Canal*

Gramática abstracta

Definición-de-canal	::	*Nombre-canal* *Trayecto-de-canal* [*Trayecto-de-canal*]
Trayecto-de-canal	::	*Bloque-de-origen* *Bloque-de-destino* *Identificador-de-señal*-**set**
Bloque-de-origen	=	*Identificador-de-bloque* | ENVIRONMENT
Bloque-de-destino	=	*Identificador-de-bloque* | ENVIRONMENT
Identificador-de-bloque	=	*Identificador*
Identificador-de-señal	=	*Identificador*
Nombre-de-canal	=	*Nombre*

El *identificador-de-señal*-**set** debe contener la lista de todas las señales que pueden ser transportadas por el(los) trayecto(s) de canal definidos por el canal.

Por lo menos uno de los puntos extremos del canal tiene que ser un bloque.

Si los dos puntos extremos son bloques, éstos tienen que ser diferentes.

El(los) punto(s) extremo(s) de bloque deben estar definidos en la misma unidad de ámbito en que está definido el canal.

Gramática textual concreta

```
<definición de canal> ::=
        CHANNEL <nombre de canal>
            <trayecto de canal>
            [<trayecto de canal>]
                [<definición de subestructura de canal>
            |    <referencia textual de subestructura de canal>]
        ENDCHANNEL [<nombre de canal>] <fin>

<trayecto de canal> ::=
        {   FROM <identificador de bloque> TO <identificador de bloque>
        |   FROM <identificador de bloque> TO ENV
        |   FROM ENV TO <identificador de bloque>}
        WITH <lista de señales> <fin>
```

<lista de señales> se define en § 2.5.5, <definición de subestructura de canal> y <referencia textual de subestructura de canal> en § 3.2.3.

Cuando se definen dos <trayecto de canal>s, debe ser de sentidos opuestos.

Gramática gráfica concreta

<área de definición de canal> ::=
<símbolo de canal>
is associated with {<nombre de canal>
{ [[<identificador de canal>|<identificador de bloque>]]
<área de lista de señales>[<área de lista de señales>] } **set** }
is connected to {<área de bloque>{<área de bloque>|<símbolo de casilla>}
[<área de asociación de subestructura de canal>] } **set**

El <identificador de canal> identifica un canal externo conectado al <diagrama de subestructura de bloque> delimitado por el <símbolo de casilla>. El <identificador de bloque> identifica un bloque externo que es un punto extremo de canal para el <diagrama de subestructura de canal> delimitado por el <símbolo de casilla>.

<símbolo de canal> ::=
<símbolo de canal 1>
| <símbolo de canal 2>
| <símbolo de canal 3>

<símbolo de canal 1> ::=

<símbolo de canal 2> ::=

<símbolo de canal 3> ::=

<área de lista de señales> se define en § 2.5.5, <área de bloque> y <símbolo de casilla> en § 2.4.1 y <área de asociación de subestructura de canal> en § 3.2.3.

Para cada flecha en el <símbolo de canal> tiene que haber un <área de lista de señales>. Un <área de lista de señales> tiene que estar suficientemente próxima a la flecha con el que está asociada para que tal asociación sea inequívoca.

Semántica

Un canal representa una ruta de transporte de señales. Un canal puede considerarse como uno o dos trayectos de canal unidireccionales e independientes entre dos bloques o entre un bloque y su entorno.

Las señales transportadas por canales se entregan al punto extremo de destino.

Las señales son presentadas en el punto extremo de destino de un canal en el mismo orden en que fueron presentadas en el punto de origen. Si dos o más señales son presentadas simultáneamente al canal, serán ordenadas arbitrariamente.

Un canal puede demorar las señales que transporta. Esto significa que hay una cola de espera de tipo primero-en-entrar-primero-en-salir FIFO («First-In-First-Out») asociada a cada sentido de transmisión de un canal. Una señal presentada al canal es introducida en la cola de espera. Tras un intervalo de tiempo indeterminado y no constante, la primera instancia de señal es liberada de la cola y aplicada a uno de los canales o a una de las rutas de señales que está conectado(a) al canal.

Pueden existir varios canales entre los dos mismos puntos extremos. Canales diferentes pueden transportar señales del mismo tipo.

2.5.2 *Ruta de señales*

Gramática abstracta

Definición-de-ruta-de-señales	::	*Nombre-de-ruta-de-señales* *Trayecto-de-ruta-de-señales* [*Trayecto-de-ruta-de-señales*]
Trayecto-de-ruta-de-señales	::	*Proceso-de-origen* *Proceso-de-destino* *Identificador-de-señal*-**set**
Proceso-de-origen	=	*Identificador-de-proceso* | ENVIRONMENT
Proceso-de-destino	=	*Identificador-de-proceso* | ENVIRONMENT
Nombre-de-ruta-de-señales	=	*Nombre*

Por lo menos uno de los puntos extremos del *trayecto-de-ruta-de-señales* tiene que ser un proceso.

Si los dos puntos extremos son procesos, los *identificadores-de-proceso* tienen que ser diferentes.

El(los) punto(s) extremo(s) de proceso tienen que estar definidos en la misma unidad de ámbito en que está definida la ruta de señales.

Gramática textual concreta

<definición de ruta de señales> ::=
SIGNALROUTE <nombre de ruta de señales>
<trayecto de ruta de señales>
[<trayecto de ruta de señales>]

<trayecto de ruta de señales> ::=
{ FROM <identificador de proceso> TO <identificador de proceso>
| FROM <identificador de proceso> TO ENV
| FROM ENV TO <identificador de proceso> }
WITH <lista de señales> <fin>

La <lista de señales> se define en § 2.5.5.

Cuando hay definidos dos <trayecto de ruta de señales>s, deben ser de sentidos opuestos.

Gramática gráfica concreta

<área de definición de ruta de señales> ::=
<símbolo de ruta de señales>
is associated with {<nombre de ruta de señales>
{[<identificador de canal>] <área de lista de señales> [<área de lista de señales>] }**set** }
is connected to
{<área de proceso> {<área de proceso> | <símbolo de casilla>} }**set**

<símbolo de ruta de señales> ::=
<símbolo de ruta de señales 1> | <símbolo de ruta de señales 2>

<símbolo de ruta de señales 1> ::=

<símbolo de ruta de señales 2> ::=

Un símbolo de ruta de señales incluye una punta de flecha en un extremo (un solo sentido) o una punta de flecha en cada extremo (ambos sentidos), que indica el sentido del flujo de las señales.

Para cada punta de flecha en el <símbolo de ruta de señales> debe haber un <área de lista de señales>. Un <área de lista de señales> tiene que estar lo suficientemente próxima a la punta de flecha a que está asociada para que tal asociación sea inequívoca.

Cuando el <símbolo de ruta de señales> está conectado al <símbolo de casilla>, el <identificador de canal> identifica un canal al cual está conectada la ruta de señales.

Semántica

Una ruta de señales representa una ruta de transporte de señales. Una ruta de señales puede considerarse como uno o dos trayectos unidireccionales independientes de ruta de señales entre dos procesos o entre un proceso y su entorno.

Las señales transportadas por rutas de señales se entregan a los puntos extremos de destino.

Una ruta de señales no introduce ningún retardo en el transporte de las señales.

Ninguna ruta de señales conecta instancias de proceso del mismo tipo. En este caso, la interpretación del *nodo-de-salida* implica que la señal se pone directamente en el puerto de entrada del proceso de destino.

Pueden existir varias rutas de señales entre los mismos dos puntos extremos. Diferentes rutas de señales pueden transportar señales del mismo tipo.

Modelo

Un <conjunto de señales de entrada válidas> contiene señales que el proceso puede recibir. Sin embargo, un <conjunto de señales de entrada válidas> no debe contener señales de temporizador. Si una <definición de bloque> contiene <definición de ruta de señales>s, el <conjunto de señales de entrada válidas>, de haberlo, no necesita contener señales en las rutas de señales que conducen al proceso.

Si una <definición de bloque> no contiene <definición de ruta de señales>s, todas las <definición de proceso>s en esa <definición de bloque> deben contener un <conjunto de señales de entrada válidas>. En ese caso, las <definición de ruta de señales>s y las <conexión de canal a ruta>s se derivan de los <conjunto de señales de llamada válida>s, las <salida>s y los canales que terminan en el límite de los bloques. Las señales que corresponden a un determinado sentido de flujo entre dos procesos en la ruta de señales implicada constituyen la intersección de las señales especificadas en el <conjunto de señales de entrada válidas> del proceso de destino y las señales mencionadas en una salida del proceso de origen. Si uno de los puntos extremos es el entorno, el conjunto de entrada/conjunto de salida para ese punto extremo está constituido por las señales transportadas por el canal en el sentido dado.

2.5.3 *Conexión*

Gramática abstracta

Conexión-de-canal-a-ruta :: *Identificador-de-canal*
Identificador-de-ruta-de-señales-**set**

Identificador-de-ruta-de-señales = *Identificador*

En § 3 figuran otras construcciones de conexión.

Cada *identificador-de-canal* conectado al bloque circundante debe ser mencionado exactamente en una *conexión-de-canal-a-ruta*. El *identificador-de-canal* en una *conexión-de-canal-a-ruta* debe denotar un canal conectado al bloque circundante.

Cada *identificador-de-ruta-de-señales* en una *conexión-de-canal-a-ruta* tiene que estar definido en el mismo bloque en que está definida la *conexión-de-canal-a-ruta* y tiene que tener la frontera de ese bloque como uno de sus puntos extremos. Cada *identificador-de-ruta-de-señales* definido en el bloque circundante y que tiene el entorno como uno de sus puntos extremos, debe mencionarse en una *conexión-de-canal-a-ruta*, y sólo en una.

Para un determinado sentido de flujo, la unión de los conjuntos *identificador-de-señal* en las rutas de señales en una *conexión-de-canal-a-ruta* tiene que ser igual al conjunto de las señales transportadas por el *identificador-de-canal* en la misma *conexión-de-canal-a-ruta* y correspondientes al mismo sentido de flujo.

Gramática textual concreta

<conexión de canal a ruta> ::=
CONNECT <identificador de canal>
AND <identificador de ruta de señales> {,<identificador de ruta de señales>}* <fin>

Ningún <identificador de ruta de señales> en una <conexión de canal a ruta> puede ser mencionado dos veces.

Gramática gráfica concreta

Gráficamente, la construcción de conectar está representada por el <identificador de canal> asociado a la ruta de señales y contenido en el <área de definición de ruta de señales> (véase § 2.5.2 *Gramática gráfica concreta*).

2.5.4 *Señal*

Gramática abstracta

Definición-de-señal :: *Nombre-de-señal*
*Identificador-de-referencia-de-género**
[*Refinamiento-de-señal*]

Nombre-de-señal :: *Nombre*

El *identificador-de-referencia-de-género* se define en § 5.2.2.

Gramática textual concreta

<definición de señal> ::=
SIGNAL {<nombre de señal> [<lista de géneros>][<refinamiento de señal>]}
{,<nombre de señal> [<lista de géneros>][<refinamiento de señal>]}* <fin>

<lista de géneros> ::=
(<género> {, <género>}*)

<refinamiento de señal> se define en § 3.3 y <género> en § 5.2.2.

Semántica

Una instancia de señal es un flujo de información entre procesos, siendo también una instanciación de un tipo de señal definido por una definición de señal. Una instancia de señal puede ser enviada, bien por el entorno, bien por un proceso, y siempre está dirigida, sea hacia un proceso, sea hacia el entorno.

Con cada instancia de señal están asociados dos valores PId (véase el § 5.6.10) que denotan los procesos de origen y de destino, el <identificador de señal> especificado en la salida correspondiente y otros valores, cuyos géneros se definen en la definición de señal.

2.5.5 *Definición de lista de señales*

Un <identificador de lista de señales> puede utilizarse en <definición de canal>, <definición de ruta de señales>, <definición de lista de señales>, <conjunto de señales de entrada válidas> y <lista de conservaciones> como una notación taquigráfica para enumerar identificadores de señal y señales de temporizador.

Gramática textual concreta

<definición de lista de señales> ::=
SIGNALLIST <nombre de lista de señales> = <lista de señales> <fin>

<lista de señales> ::=
<ítem de señal> {,<ítem de señal>}*

<ítem de señal> ::=
<identificador de señal> | <identificador de señal prioritaria> (<identificador de lista de señales>) | <identificador de temporizador>

La <lista de señales> que se construye reemplazando todos los <identificador de lista de señales>s de la lista por los <identificador de señal>s que éstos denotan, corresponde a un *identificador-de-señal*-**set** en la *gramática abstracta.* En toda <lista de señales> así construida, cada <identificador de señal> tiene que ser distinto.

Gramática gráfica concreta

<área de lista de señales> ::=
<símbolo de lista de señales> **contains** <lista de señales>

<símbolo de lista de señales> ::=

2.6 *Comportamiento*

2.6.1 *Variables*

2.6.1.1 *Definición de variable*

Gramática abstracta

Definición-de-variable	::	*Nombre-de-variable* *Identificador-de-referencia-de-género* [REVEALED]
Nombre-de-variable	=	*Nombre*

Gramática textual concreta

<definición de variable> ::=
DCL [REVEALED|EXPORTED|REVEALED EXPORTED|EXPORTED REVEALED]
<nombre de variable> {,<nombre de variable>}* <género> [:= <expresión fundamental>].
{,<nombre de variable> {,<nombre de variable>}* <género> [:= <expresión fundamental>] }* <fin>

Variable exportada se define en § 4.13.

Semántica

La semántica de variables se define en § 2.3.2. El valor de una variable sólo lo puede modificar el propietario. El propietario de una variable es el proceso (o procedimiento) donde la variable está declarada. El valor de una variable sólo lo conoce el propietario a menos que la variable tenga el atributo REVEALED. El atributo REVEALED permite a todos los otros procesos del mismo bloque ver la variable, siempre que tengan la definición de visión en la declaración de la variable.

Modelo

La <expresión fundamental> en una <definición de variable> o el valor por defecto en un <género> no tienen sintaxis abstracta correspondiente. Son sintaxis derivadas para especificar una secuencia de sentencias de asignación en la transición inicial de la unidad de ámbito circundante. Las sentencias de asignación asignan la <expresión fundamental> a todos los <nombre de variable>s mencionados en la <definición de variable>. Si se especifica un valor por defecto en un <género> y también una <expresión fundamental> en la <definición de variable>, se aplica la <expresión fundamental> de la <definición de variable>.

2.6.1.2 *Definición de visión*

Gramática abstracta

Definición-de-visión :: *Identificador-de-variable*
Identificador-de-referencia-de-género

La *definición-de-variable* designada por *identificador-de-variable* debe tener el atributo REVEALED, y tiene que ser del mismo género que el del *identificador-de-referencia-de-género* denotado.

Gramática textual concreta

<definición de visión> ::=
VIEWED
<identificador de variable> {, <identificador de variable>}* <género>
{,<identificador de variable>{, <identificador de variable>}* <género>}* <fin>

El calificador de <identificador de variable> en <definición de visión> sólo puede omitirse si existe en el bloque una, y sólo una, <definición de proceso> que tiene una <definición de variable> que define un <nombre de variable> que es el mismo que el <nombre de variable> mencionado en la <definición de visión> y que tiene el atributo REVEALED, y que es del mismo <género> que el denotado por el <género> en la <definición de visión>.

Semántica

El mecanismo de visión permite a una instancia de proceso ver continuamente el valor de la variable vista como si ésta estuviese definida localmente. La instancia de proceso que observa no tiene, sin embargo, derecho a modificarla.

2.6.2 *Arranque*

Gramática abstracta

Nodo-de-arranque-de-proceso :: *Transición*

Gramática textual concreta

<arranque> ::=
START <fin> <transición>

Gramática gráfica concreta

<área de arranque> ::=
<símbolo de arranque> **is followed by** <área de transición>

<símbolo de arranque> ::=

Semántica

La *transición* del *nodo-de-arranque-de-proceso* se interpreta.

2.6.3 *Estado*

Gramática abstracta

Nodo-de-estado :: *Nombre-de-estado*
Conjunto-de-señales-de-conservación
Nodo-de-entrada-**set**

Nombre-de-estado = *Nombre*

Dentro de un *gráfico-de-proceso* o *gráfico-de-procedimiento*, los *nodos-de-estado* tienen diferentes *nombres-de-estado.*

Para cada *nodo-de-estado*, todos los *identificadores-de-señal* (en el conjunto completo de señales de entrada válidas) aparecen, bien en un *conjunto-de-señales-de-conservación*, bien en un *nodo-de-entrada.*

El *identificador-de-señal* en el *nodo-de-entrada*-**set** tienen que ser distintos.

Gramática textual concreta

<estado> ::=
STATE <lista de estados> <fin>
{<parte entrada>
| <entrada prioritaria>
| <parte conservación>
| <señal continua>}*
[ENDSTATE [<nombre de estado>] <fin>]

<lista de estados> ::=
{<nombre de estado> {, <nombre de estado>}*}
| <lista de estados asterisco>

<parte entrada> se define en § 2.6.4, <parte conservación> en § 2.6.5, <señal continua> en § 4.11, <lista de estados asterisco> en § 4.4 y <entrada prioritaria> en § 4.10.2.

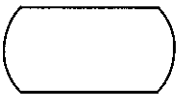
Cuando la <lista de estados> contiene un solo <nombre de estado>, el <nombre de estado> representa un *nodo-de-estado.* Para cada *nodo-de-estado,* el *conjunto-de-señales-de-conservación* se representa por la <parte conservación> y eventuales conservaciones implícitas de señales. Para cada *nodo-de-estado,* el *nodo-de-entrada*-set se representa por la <parte entrada> y las eventuales señales de entrada implícitas.

El <nombre de estado> opcional que termina un <estado> sólo puede especificarse si la <lista de estados> en el <estado> consiste en un solo <nombre de estado>, en cuyo caso éste debe ser el mismo <nombre de estado> que el contenido en la <lista de estados>.

Gramática gráfica concreta

<área de estado> ::=
<símbolo de estado> **contains** <lista de estados> **is associated with**
{<área de asociación de entrada>
| <área de asociación de entrada prioritaria>
| <área de asociación de señal continua>
| <área de asociación de conservación>}*

<símbolo de estado> ::=



<área de asociación de entrada> ::=
<símbolo de asociación de trazo continuo> **is connected to** <área de entrada>

<área de asociación de conservación> ::=
<símbolo de asociación de trazo continuo> **is connected to** <área de conservación>

<área de entrada> se define en § 2.6.4 <área de conservación> en § 2.6.5, <área de asociación de señal continua> en § 4.11, <área de asociación de entrada prioritaria> en § 4.10.2.

Un <área de estado> representa uno o más *nodos-de-estado.*

Los <símbolos de asociación de trazo continuo>s que tienen su origen en un <símbolo de estado> pueden tener un trayecto de origen común.

Semántica

Un estado representa una condición particular en la cual una instancia de proceso puede consumir una instancia de señal, lo que causa una transición. Si no hay instancias de señal retenidas, el proceso espera en el estado hasta que se reciba una instancia de señal.

Modelo

Cuando la <lista de estados> de cierto <estado> contiene más de un <nombre de estado>, se crea una copia del <estado> para cada uno de estos <nombre de estado>s. Entonces el <estado> es reemplazado por estas copias.

2.6.4 *Entrada*

Gramática abstracta

Nodo-de-entrada :: *Identificador-de-señal*
[*Identificador-de-variable*]*
Transición

Identificador-de-variable = *Identificador*

La longitud del [*identificador de variable*]* debe ser igual al número de *identificadores-de-referencia-de-género* de la *definición-de-señal* denotada por el *identificador-de-señal.*

Los géneros de las variables deben corresponder en posición a los géneros de los valores que pueden ser transportados por la señal.

No se permite especificar un número de variables, para ser recibidas, superior al número de valores transportados por la instancia de señal.

Gramática textual concreta

<parte entrada> ::=
INPUT <lista de entradas> <fin>
[<condición habilitante>]<transición>

<lista de entradas> ::=
<lista de entradas asterisco>
| <estímulo>{,<estímulo>}*

<estímulo> ::=
{<identificador de señal>
| <identificador de temporizador>} [([<identificador de variable>] {,[<identificador de variable>]}*)]

<transición> se define en § 2.6.7, <condición habilitante> en § 4.12, y <lista de entradas asterisco> en § 4.6.

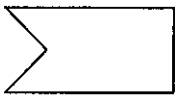
Cuando la <lista de entradas> contiene un solo <estímulo>, la <parte entrada> representa un <nodo de entrada>. En la *gramática abstracta,* las señales de temporizador (<identificador de temporizador>) son también representadas por *identificador-de-señal.* Las señales de temporizador y las señales ordinarias sólo se distinguen cuando proceda, pues en muchos aspectos tienen propiedades similares. Las propiedades exactas de las señales de temporizador se definen en § 2.8.

Una <transición> debe tener un terminador de transición, como se indica en § 2.6.7.2.

Gramática gráfica concreta

<área de entrada> ::=
<símbolo de entradas> **contains** <lista de entradas>
is followed by {[<área de condición habilitante>] <área de transición>}

<símbolo de entrada> ::=



<área de transición> se define en § 2.6.7, <área de condición habilitante> en § 4.12.

Un <área de entrada> cuya <lista de entradas> contiene un solo <estímulo> corresponde a un *nodo-de-entrada.* Cada uno de los <identificadores de señal> contenido en un <símbolo de entrada> da el nombre de uno de los *nodos-de-entrada* que este <símbolo de entrada> representa.

Semántica

Una entrada permite el consumo de la instancia de señal de entrada especificada. El consumo de la instancia de la señal de entrada pone a la disposición del proceso la información transportada por la señal. A las variables asociadas con la entrada se asignan valores transportados por la señal consumida. Si no hay variable asociada con la entrada para un género especificado en la señal, se descarta el valor de este género.

A la expresión SENDER del proceso consumidor se le da el valor PId de la instancia de proceso originadora, transportado por la instancia de señal.

Las instancias de señal que pasan del entorno a una instancia de proceso dentro del sistema tendrán siempre un valor PId diferente de cualquiera de los del sistema. El acceso se efectúa mediante la expresión SENDER.

Modelo

Cuando la lista de <estímulo>s de cierta <parte entrada> contiene más de un <estímulo>, se crea una copia de la <parte entrada> para cada uno de esos <estímulos>s. La <parte entrada> se reemplaza entonces por estas copias.

2.6.5 *Conservación*

Una conservación especifica un conjunto de identificadores de señal cuyas instancias no son relevantes para el proceso en el estado al cual se ha asociado la conservación, y que debe conservarse para un procesamiento futuro.

Gramática abstracta

Conjunto-de-señales-de-conservación :: *Identificador-de-señal*-**set**

En cada *nodo-de-estado,* los *identificadores-de-señal* contenidos en el *conjunto-de-señales-de-conservación* tienen que ser diferentes.

Gramática textual concreta

<parte conservación> ::=
SAVE <lista de conservaciones> <fin>

<lista de conservación> ::=
{<lista de señales> | <lista de conservaciones asterisco>}

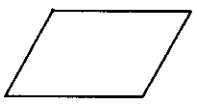
Una <lista de conservaciones> representa el *identificador-de-señal*-**set**. La <lista de conservaciones asterisco> es una notación taquigráfica que se explica en § 4.8.

Gramática gráfica concreta

<área de conservación> ::=

<símbolo de conservación> **contains** <lista de conservaciones>

<símbolo de conservación> ::=



Semántica

Las señales conservadas se retienen en el puerto de entrada en el orden de su llegada.

El efecto de la conservación es válido solamente para el estado al cual está asociada la conservación. En el estado siguiente, las instancias de señal que han sido «conservadas» se tratan como instancias de señal normales.

2.6.6 *Etiqueta*

Gramática textual concreta

<etiqueta> ::=
<nombre de conector>:

Todos los <nombre de conector>s definidos en un <cuerpo de proceso> tienen que ser distintos.

Una etiqueta representa el punto de entrada de un «salto» desde las sentencias de unión correspondientes con los mismos <nombre de conector>s en el mismo <cuerpo de proceso>.

Sólo se permiten «saltos» a etiquetas dentro del mismo <cuerpo de proceso>.

Gramática gráfica concreta

<área de conector de entrada> ::=
<símbolo de conector de entrada> **contains** <nombre de conector> **is followed by** <área de transición>

<símbolo de conector de entrada> ::=

<área de transición> se define en § 2.6.7.1.

Un <área de conector de entrada> representa la continuación de un <símbolo de línea de flujo> desde un <área de conector de salida> correspondiente, con el mismo <nombre de conector>, en la misma <área de gráfico de proceso> o <área de gráfico de procedimiento>.

2.6.7 *Transición*

2.6.7.1 *Cuerpo de transición*

Gramática abstracta

Transición	::	*Nodo-de-gráfico* * (*Terminador* | *Nodo-de-decisión*)
Nodo-de-gráfico	::	*Nodo-de-tarea* | *Nodo-de-salida* | *Nodo-de-petición-de-crear* | *Nodo-de-llamada* | *Nodo-de-inicializar* | *Nodo-de-reinicializar* |
Terminador	::	*Nodo-de-estado-siguiente* | *Nodo-de-parada* | *Nodo-de-retorno*

Gramática textual concreta

<transición> ::=
{<cadena de transición> [<sentencia de terminador>]}
| <sentencia de terminador>

<cadena de transición> ::=
{<sentencia de acción>}$^{+}$

<sentencia de acción> ::=
[<etiqueta>] <acción> <fin>

<acción> ::=
<tarea>
| <salida>
| <salida prioritaria>
| <petición de crear>
| <decisión>
| <opción de transición>
| <inicializar>
| <reinicializar>
| <exportación>
| <llamada a procedimiento>

<sentencia de terminador> ::=
[<etiqueta>]<terminador> <fin>

<terminador> ::=
<estado siguiente>
| <unión>
| <parada>
| <retorno>

<tarea> se define en § 2.7.1, <salida> en § 2.7.4, <petición de crear> en § 2.7.2, <decisión> en § 2.7.5, <inicializar> y <reinicializar> en § 2.8, <llamada a procedimiento> en § 2.7.3, <estado siguiente> en § 2.6.7.2.1, <unión> en § 2.6.7.2.2, <parada> en § 2.6.7.2.3, <retorno> en § 2.6.7.2.4, <salida prioritaria> en § 4.10.2, <opción de transición> en § 4.3.4, y <exportación> en § 4.13.

Si se omite el <terminador> de una <transición>, la última acción en la <transición> debe contener una <decisión> de terminación (véase el § 2.7.5) o una <opción de transición> de terminación, con excepción de todas las <transición>s contenidas en <decisión>s y <opción de transición>s (<opción de transición> se define en § 4.3.4).

Ningún <terminador> o <acción> puede seguir a un <terminador>, una <opción de transición> de terminación o una <decisión> de terminación.

Gramática gráfica concreta

<área de transición> ::=
[<área de cadena de transición>] **is followed by**
{<área de estado>
| <área de estado siguiente>
| <área de decisión>
| <símbolo de parada>
| <área de fusión>
| <área de conector de salida>
| <símbolo de retorno>
| <área de opción de transición>}

<área de cadena de transición> ::=
{<área de tarea>
|<área de salida>
|<área de salida prioritaria>
|<área de inicializar>
|<área de reinicializar>
|<área de exportación>
|<área de petición de crear>
|<área de llamada a procedimiento>}
[is followed by <área de cadena de transición>]

<área de tarea> se define en § 2.7.1, <área de salida> en § 2.7.4, <área de petición de crear> en § 2.7.2, <área de decisión> en § 2.7.5, <área de inicilizar> y <área de reinicializar> en § 2.8, <área de llamada a procedimiento> en § 2.7.3, <área de estado siguiente> en § 2.6.7.2.1, <área de fusión> en § 2.6.7.2.2, <símbolo de parada> en § 2.6.7.2.3, <símbolo de retorno> en § 2.6.7.2.4, <área de salida prioritaria> en § 4.10.2, <área de opción de transición> en § 4.3.4, <área de exportación> en § 4.13 y <área de conector de salida> en § 2.6.7.2.2.

Una transición consiste en una secuencia de acciones que serán ejecutadas por el proceso.

El <área de transición> corresponde a *transición* y el <área de cadena de transición> corresponde a *nodo-de-gráfico**.

Semántica

Una transición efectúa una secuencia de acciones. Durante una transición los datos de un proceso pueden manipularse y las señales pueden presentarse a la salida. Una transición terminará por la entrada del proceso en un estado, por una parada o por un retorno.

2.6.7.2 *Terminador de transición*

2.6.7.2.1 *Estado siguiente*

Gramática abstracta

Nodo-de-estado-siguiente :: *Nombre-de-estado*

El *nombre-de-estado* especificado en un estado siguiente tiene que ser el nombre de un estado dentro del mismo *gráfico-de-proceso* o *gráfico-de-procedimiento.*

Gramática textual concreta

<estado siguiente> ::=
NEXTSTATE <cuerpo de estado siguiente>

<cuerpo de estado siguiente> ::=
{<nombre de estado> | <estado siguiente indicado por guión> }

<estado siguiente indicado por guión> se define en § 4.9.

Gramática gráfica concreta

<área de estado siguiente> ::=
<símbolo de estado> **contains** <cuerpo de estado siguiente>

Semántica

Un estado siguiente representa un terminador de una transición. Especifica el estado a que pasará la instancia de proceso cuando termine la transición.

2.6.7.2.2 *Unión*

Una unión cambia el flujo en un <diagrama de proceso> o <cuerpo de proceso> expresando que la <sentencia de acción> siguiente a interpretar es la que contiene el mismo <nombre de conector>.

Gramática textual concreta

<unión> ::=
JOIN <nombre de conector>

Tiene que haber un <nombre de conector>, y sólo uno, correspondiente a una <unión> dentro del mismo <cuerpo de proceso>, <cuerpo de procedimiento> respectivamente <cuerpo de servicio>.

Gramática gráfica concreta

<área de fusión> ::=
<símbolo de fusión> **is connected to** <símbolo de línea de flujo>

<símbolo de fusión> ::=
<símbolo de línea de flujo>

<símbolo de línea de flujo> ::=

<área de conector de salida> ::=
<símbolo de conector de salida> **contains** <nombre de conector>

<símbolo de conector de salida> ::=
<símbolo de conector de entrada>

Para cada <área de conector de salida> en un <área de gráfico de proceso> o <área de gráfico de procedimiento> debe haber un <área de conector de entrada>, y solo una, respectivamente, en esa <área de gráfico de proceso> o <área de gráfico de procedimiento> con el mismo <nombre de conector>

Un <área de conector de salida> corresponde a una <unión> en la *gramática textual concreta.* Si un <área de fusión> está incluida en un <área de transición>, ello equivale a especificar un <área de conector de salida> en el <área de transición> que contiene un <nombre de conector> único y asociar un <área de conector de entrada>, con el mismo <nombre de conector>, al <símbolo de línea de flujo> en el <área de fusión>.

Modelo

En la sintaxis abstracta una <unión> o <área de conector de salida> se deriva de la <cadena de transición> en la cual la primera <sentencia de acción> o área tiene asociado el mismo <nombre de conector>.

2.6.7.2.3 *Parada*

Gramática abstracta

Nodo-de-parada :: *()*

Un *nodo-de-parada* no podrá estar contenido en un *gráfico-de-procedimiento.*

Gramática textual concreta

<parada> ::=
STOP

Gramática gráfica concreta

<símbolo de parada> ::=

Semántica

La parada causa la detención inmediata de la instancia de proceso que la emite. Esto significa que las señales retenidas en el puerto de entrada se descartan y que las variables y temporizadores creados para el proceso, el puerto de entrada y el proceso dejarán de existir.

2.6.7.2.4 *Retorno*

Gramática abstracta

Nodo-de-retorno :: *()*

Un *nodo-de-retorno* no podrá estar contenido en un *gráfico-de-proceso.*

Gramática textual concreta

<retorno> ::=
RETURN

Gramática gráfica concreta

<símbolo de retorno> ::=

Semántica

Un *nodo-de-retorno* se interpreta de la forma siguiente:

a) Todas las variables creadas por la interpretación del *nodo-de-arranque-de-procedimiento* dejarán de existir.

b) La interpretación del *nodo-de-retorno* completa la interpretación del *gráfico-de-procedimiento* y la instancia de procedimiento deja de existir.

c) En lo sucesivo, la interpretación del proceso (o procedimiento) llamante continúa en el nodo que sigue a la llamada.

2.7 *Acción*

2.7.1 *Tarea*

Gramática abstracta

Nodo-de-tarea :: *Sentencia-de-asignación* |
Texto-informal

Gramática textual concreta

<tarea> ::=
TASK <cuerpo de tarea>

<cuerpo de tarea> ::=
{<sentencia de asignación>{,<sentencia de asignación>}*}
| {<texto informal>{,<texto informal>}*}

<sentencia de asignación> se define en § 5.5.3.

Gramática gráfica concreta

<área de tarea> ::=
<símbolo de tarea> **contains** <cuerpo de tarea>

<símbolo de tarea> ::=

Semántica

La interpretación de un *nodo-de-tarea* es la interpretación de la *sentencia-de-asignación* que se explica en § 5.5.3, o la interpretación del *texto-informal* que se explica en § 2.2.3.

Modelo

Una <tarea> y un <área de tarea> pueden contener varias <sentencia de asignación>s o <texto informal>s. En ese caso es sintaxis derivada para especificar una secuencia de <tarea>s, una para cada <sentencia de asignación> o <texto informal> de modo que se mantenga el orden original en que fueron especificados en el <cuerpo de tarea>.

Esta notación taquigráfica se expande antes de expandir cualquier <expresión de importación> (véase el § 4.13).

2.7.2 *Crear*

Gramática abstracta

Nodo-de-petición-de-crear :: *Identificador-de-proceso*
[*Expresión*]*

Identificador-de-proceso = *Identificador*

El número de *expresiones* en la [*expresión*]* tiene que ser igual al número de *parámetros-formales-de-proceso* en la *definición-de-proceso* del *identificador-de-proceso.* Cada *expresión* tiene que tener el mismo género que el *parámetro-formal-de-proceso* que le corresponde en posición en la *definición-de-proceso* denotada por el *identificador-de-proceso.*

Gramática textual concreta

<petición de crear> ::=
CREATE <cuerpo de crear>

<cuerpo de crear> ::=
<identificador de proceso> [<parámetros efectivos>]

<parámetros efectivos> ::=
([<expresión>] {,[<expresión>]}*)

<expresión> se define en § 5.

Gramática gráfica concreta

<área de petición de crear> ::=
<símbolo de petición de crear> **contains** <cuerpo de crear>

<símbolo de petición de crear> ::=

Un <área de petición de crear> representa un *nodo-de-petición-de-crear.*

Semántica

Cuando se crea una instancia de proceso se le da un puerto de entrada vacío, se crean variables y las expresiones de parámetros efectivos se interpretan en el orden dado y se asignan (como se define en el § 5.5.3) a los parámetros formales correspondientes. Si un parámetro efectivo está vacío, se da al parámetro formal correspondiente el valor indefinido. Seguidamente, el proceso arranca interpretando el nodo de arranque en el gráfico de proceso.

El proceso creado actúa entonces asíncronamente y en paralelo con otros procesos.

La acción crear causa la creación de una instancia de proceso en el mismo bloque. El PARENT del proceso creado tiene el mismo valor PId que el SELF del proceso creador. Las expresiones SELF del proceso creado y OFFSPRING del proceso creador tienen ambas el último valor de PId creado (véase el § 5.6.10.1).

Si se intenta crear un número mayor de instancias de proceso que el especificado por el número máximo de instancias en la definición de proceso, no se crea ninguna nueva instancia, la expresión OFFSPRING del proceso creador tiene el valor NULL, y la interpretación continúa.

2.7.3 *Llamada a procedimiento*

Gramática abstracta

Nodo-de-llamada :: *Identificador-de-procedimiento*
[*Expresión*]*

Identificador-de-procedimiento = *Identificador*

La longitud de la [*expresión*]* tiene que ser igual al número de los *parámetros-formales-de-procedimiento* en la *definición-de-procedimiento* del *identificador-de-procedimiento.*

Cada *expresión* correspondiente en posición a un *parámetro-formal-de-proceso* IN tiene que tener el mismo género que el *parámetro-formal-de-proceso.*

Cada *expresión* correspondiente a un *parámetro-formal-de-proceso* IN/OUT tiene que ser un *identificador-de-variable* con el mismo *identificador-de-referencia-de-género* que el *parámetro-formal-de-proceso.*

Tiene que haber una *expresión* para cada *parámetro-formal-de-proceso* IN/OUT.

Gramática textual concreta

<llamada a procedimiento> ::=
CALL <cuerpo de llamada a procedimiento>

<cuerpo de llamada a procedimiento> ::=
<identificador de procedimiento> [<parámetros efectivos>]

<parámetros efectivos> se define en el § 2.7.2.

Un ejemplo de <llamada a procedimiento> se muestra en la figura 2.9.13, en el § 2.9.

Gramática gráfica concreta

<área de llamada a procedimiento> ::=
<símbolo de llamada a procedimiento> **contains** <cuerpo de llamada a procedimiento>

<símbolo de llamada a procedimiento> ::=

El <área de llamada a procedimiento> representa el *nodo-de-llamada.*

Un ejemplo de <área de llamada a procedimiento> se muestra en la figura 2.9.14, en el § 2.9.

Semántica

La interpretación de un nodo de llamada a procedimiento transfiere la interpretación a la definición de procedimiento referenciada en el nodo de llamada, y se interpreta ese gráfico de procedimiento. Los nodos del gráfico de procedimiento se interpretan de la misma manera que los nodos equivalentes de un gráfico de proceso.

La interpretación del proceso llamante se suspende hasta que haya terminado la interpretación del proceso llamado.

Las expresiones de parámetros efectivos se interpretan en el orden dado.

Se necesita una semántica especial en lo que respecta a la interpretación de datos y parámetros (véase la explicación en el § 2.4.4).

2.7.4 *Salida*

Gramática abstracta

Nodo-de-salida	::	*Identificador-de-señal* [*Expresión*]* [*Destino-de-señal*] *Dirigir-vía*
Destino-de-señal	=	*Expresión*
Dirigir-vía	=	*Identificador-de-ruta-de-señales*-**set**

La longitud de la [*expresión*]* tiene que ser igual al número de *identificadores-de-referencia-de-género* en la *definición-de-señal* denotada por el *identificador-de-señal.*

Cada *expresión* tiene que tener el mismo género que la *referencia-de-identificador-de-género* correspondiente (por posición) en la *definición-de-señal.*

Para cada subconjunto consistente posible (véase el § 3) tiene que existir por lo menos un trayecto de comunicación (ya sea implícito para el propio tipo de proceso, ya sea explícito vía rutas de señales y, posiblemente, canales) hacia el entorno o hacia un tipo de proceso que tenga *identificador-de-señal* en su conjunto de señales de entrada válidas y que se origine a partir del tipo de proceso en que se utilizó el *nodo-de-salida.*

Para cada *identificador-de-ruta-de-señales* en *dirigir-vía* debe cumplirse que el *proceso-de-origen* en (uno de) los *trayecto-de-ruta-de-señales* de la ruta de señales debe ser un proceso del mismo tipo que el del proceso que contiene el *nodo-de-salida* y que el *trayecto-de-ruta-de-señales* tiene que incluir el *identificador-de-señal* en su conjunto de *identificadores-de-señal.*

Si no se especifica ningún *identificador-de-ruta-de-señales* en *dirigir-vía,* todo proceso para el cual existe un trayecto de comunicación puede recibir la señal.

Gramática textual concreta

<salida> ::=
OUTPUT <cuerpo de salida>

<cuerpo de salida> ::=
<identificador de señal>
[<parámetros efectivos>] {, <identificador de señal> [<parámetros efectivos>]}*
[TO <expresión PId>]
[VIA { <identificador de ruta de señales>{,<identificador de ruta de señales>}*
| {<identificador de canal>{,<identificador de canal>}* }]

Los <parámetros efectivos> se definen en el § 2.7.2 y <expresión> en el § 5.4.2.1.

No se permite especificar un <identificador de canal> en la construcción VIA si hay rutas de señales especificadas para el bloque.

Para cada <identificador de canal> de una <salida> debe existir un canal proveniente del bloque circundante, y que pueda transportar las señales denotadas por los <identificador de señal>s contenidos en la <salida>.

TO <expresión PId> representa el *destino-de-señal.*

La construcción VIA representa el *dirigir-vía.*

Gramática gráfica concreta

<área de salida> ::=
<símbolo de salida> **contains** <cuerpo de salida>

<símbolo de salida> ::=

Semántica

La expresión PId de *destino-de-señal* se interpreta después de otras expresiones en el *nodo-de-salida.*

Los valores transportados por la instancia de señal son los valores de los parámetros efectivos en la salida. Si no hay ningún parámetro efectivo en la salida para un género en la definición de señal, la señal transporta el valor indefinido.

El valor PId de origen transportado por la instancia de señal es el valor asociado con SELF (del proceso que ejecuta la acción de salida). El valor PId de destino transportado por la instancia de señal es el valor de la expresión PId de destino de señal contenido en la salida.

La instancia de señal se pasa entonces a un trayecto de comunicación capaz de transportarla a la instancia de proceso de destino especificada.

Si no está especificado ningún *destino-de-señal,* tendrá entonces que existir un receptor, y sólo uno, que pueda recibir la señal de acuerdo con las rutas de señales o canales especificados en *dirigir-vía.* El valor PId de destino implícitamente transportado por la instancia de señal es el valor PId de este receptor.

El entorno puede siempre recibir cualquier señal del conjunto de señales de un canal que conduzca al entorno.

Obsérvese que el hecho de que se especifique el mismo *identificador-de-canal* o *identificador-de-ruta-de-señales* en el *dirigir-vía* de dos *nodos-de-salida* no significa automáticamente que las señales forman cola en el puerto de entrada en el mismo orden que se interpretan los *nodos-de-salida.* Sin embargo este orden se mantiene si las dos señales son transportadas por canales idénticos que conectan el *proceso-de-origen* con el *proceso-de-destino* o si los procesos se definen en el mismo bloque.

Si se especifica un sintipo en la definición de señal y se especifica una expresión en la salida, se aplica a la expresión la verificación de intervalo definida en el § 5.4.1.9.1. Si la verificación de intervalo es equivalente a Falso, la salida está en error y el comportamiento futuro del sistema está indefinido.

Una salida enviada a una instancia de proceso inexistente (tanto si nunca ha existido como si ha dejado de existir) causa un error de interpretación. La evaluación sobre la existencia de una instancia de proceso se hace al mismo tiempo que se interpreta la salida. Una detención ulterior de la instancia del proceso receptor hace que se descarte la señal en el puerto de entrada y no se informa una condición de error.

Modelo

Si se especifican varios pares de (<identificador de señal> <parámetros efectivos>) en un <cuerpo de salida>, esto es sintaxis derivada para especificar una secuencia de <salida>s o <área de salida>s en el mismo orden especificado en el <cuerpo de salida> inicial, cada uno de los cuales contiene un solo par de (<identificador de señal> <parámetros efectivos>). La cláusula TO y la cláusula VIA se repiten en cada una de las <salida>s o <área de salida>s. Esta notación taquigráfica se expande antes de expandir las eventuales notaciones taquigráficas en las expresiones contenidas.

2.7.5 *Decisión*

Gramática abstracta

Nodo-de-decisión	::	*Pregunta-de-decisión* *Respuesta-de-decisión*-**set** [*Respuesta-de-otro-caso*]
Pregunta-de-decisión	=	*Expresión* | *Texto-informal*
Respuesta-de-decisión	::	(*Condición-de-intervalo* | *Texto-informal*) *Transición*
Respuesta-de-otro-caso	::	*Transición*

Las *respuestas-de-decisión* tienen que ser mutuamente exclusivas.

Si la *pregunta-de-decisión* es una *expresión,* la *condición-de-intervalo* de las *respuestas-de-decisión* tiene que ser del mismo género que la *pregunta-de-decisión.*

Gramática textual concreta

<decisión> ::=
DECISION <pregunta> <fin> <cuerpo de decisión> ENDDECISION

<cuerpo de decisión> ::=
{<parte respuesta> <parte otro caso>}
| {<parte respuesta> {<parte respuesta>}$^{+}$ [<parte otro caso>]}

<parte respuesta> ::=
(<respuesta>) : [<transición>]

<respuesta> ::=
<condición de intervalo> | <texto informal>

<parte otro caso> ::=
ELSE:[<transición>]

<pregunta> ::=
<expresión de pregunta> | <texto informal>

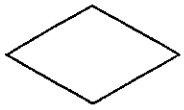
<condición de intervalo> se define en el § 5.4.1.9.1, <transición> en el § 2.6.7.1, <texto informal> en el § 2.2.3.

Una <decisión> u <opción de transición> (definida en el § 4.3.4) es de terminación si cada <parte respuesta> y <parte otro caso> en el <cuerpo de decisión> contiene una <transición> en que se especifica una <sentencia de terminador>, o contiene una <cadena de transición> cuya última <sentencia de acción> contiene una decisión de terminación o una opción.

Gramática gráfica concreta

<área de decisión> ::=
<símbolo de decisión> **contains** <pregunta>
is followed by
{{<parte respuesta gráfica> <parte otro caso gráfico>} **set**
| {<parte respuesta gráfica> {<parte respuesta gráfica>}+ [parte otro caso gráfico>]} **set** }

<símbolo de decisión> ::=



<parte respuesta gráfica> ::=
<símbolo de línea de flujo> **is associated with** <respuesta gráfica>
is followed by <área de transición>

<respuesta gráfica> ::=
<respuesta> | (<respuesta>)

<parte otro caso gráfico> ::=
<símbolo de línea de flujo> **is associated with** ELSE
is followed by <área de transición>

<área de transición> se define en el § 2.6.7.1 y <símbolo de línea de flujo> en el § 2.6.7.2.2.

La <respuesta gráfica> y ELSE pueden situarse a lo largo del <símbolo de línea de flujo> asociado, o en el <símbolo de línea de flujo> interrumpido.

Los <símbolo de línea de flujo>s que se originan en un <símbolo de decisión> pueden tener un trayecto de origen común.

Un <área de decisión> representa un *nodo-de-decisión.*

Semántica

Una decisión transfiere la interpretación al trayecto de salida cuya condición de intervalo contiene el valor dado por la interpretación de la pregunta. Se define un conjunto de respuestas posibles a la pregunta, cada una de las cuales especifica el conjunto de acciones a interpretar para esa elección de trayecto.

Una de las respuestas puede ser el complemento de las otras. Esto se consigue especificando la *respuesta-de-otro-caso,* que indica el conjunto de acciones a realizar cuando el valor de la expresión sobre la cual se plantea la pregunta no está cubierto por los valores o el conjunto de valores especificados en las otras respuestas.

En todos aquellos casos en que no se especifica la *respuesta-de-otro-caso,* el valor resultante de la evaluación de expresión de pregunta debe corresponder con una de las respuestas.

Hay una ambigüedad sintáctica entre <texto informal> y <cadena de caracteres> en <pregunta> y <respuesta>. Si la <pregunta> y todas las <respuesta>s son <cadena de caracteres>, todas estas se interpretan como <texto informal>. Si la <pregunta> o alguna <respuesta> es una <cadena de caracteres> que no encaja en el contexto de la decisión , la <cadena de caracteres> denota <texto informal>. El contexto de la decisión (es decir, el género) se determina sin tener en cuenta las <respuesta>s que son <cadena de caracteres>.

Modelo

Si una <decisión> no es una decisión de terminación, es entonces sintaxis derivada para una <decisión> en la cual todas las <parte respuesta>s y la <parte otro caso> han insertado en su <transición> una <unión> con la primera <sentencia de acción> que sigue a la decisión o, si la decisión es la última <sentencia de acción> en una <cadena de transición>, con la siguiente <sentencia de terminador>.

2.8 *Temporizador*

Gramática abstracta

Definición-de-temporizador	::	*Nombre-de-temporizador* *Identificador-de-referencia-de-género**
Nombre-de-temporizador	=	*Nombre*
Nodo-de-inicializar	::	*Expresión-de-tiempo* *Identificador-de-temporizador* *Expresión**
Nodo-de-reinicializar	::	*Identificador-de-temporizador* *Expresión**
Identificador-de-temporizador	=	*Identificador*
Expresión-de-tiempo	=	*Expresión**

Los géneros de la *expresión** en el *nodo-de-inicializar* y *nodo-de-reinicializar* deben corresponder en posición con el *identificador-de-referencia-de-género** que sigue directamente al *nombre-de-temporizador* identificado por el *identificador-de-temporizador*

Las *expresiones* en un *nodo-de-inicializar* o *nodo-de-reinicializar* tienen que ser evaluadas en el orden dado.

Gramática textual concreta

<definición de temporizador> ::=
TIMER <nombre de temporizador> [<lista de géneros>]
{, <nombre de temporizador> [<lista de géneros>] }* <fin>

<reinicializar> ::=
RESET (<sentencia de reinicializar> { , <sentencia de reinicializar> }*)

<sentencia de reinicializar> ::=
<identificador de temporizador> [(<lista de expresiones>)]

<inicializar> ::=
SET <sentencia de inicializar> { , <sentencia de inicializar> }*

<sentencia de inicializar> ::=
(<expresión de tiempo>, <identificador de temporizador> [(<lista de expresiones>)])

<lista de géneros> y <lista de expresiones> se definen en § 2.5.4 y § 5.5.2.1 respectivamente.

Una <sentencia de reinicializar> representa un *nodo-de-reinicializar*; una <sentencia de inicializar> representa un *nodo-de-inicializar*. Si un <reinicializar> contiene varias <sentencia de reinicializar>s, éstas tienen que interpretarse en el orden dado. Si un <inicializar> contiene varias <sentencia de inicializar>s, éstas tienen que interpretarse en el orden dado.

Gramática gráfica concreta

<área de inicializar> ::=
<símbolo de tarea> **contains** <inicializar>

<área de reinicializar> ::=
<símbolo de tarea> **contains** <reinicializar>

Semántica

Una instancia de temporizador es un objeto, poseído por una instancia de proceso, que puede estar activo o inactivo. Dos ocurrencias de un identificador de temporización seguido por una lista de expresiones se refieren a la misma instancia de temporizador únicamente si las dos listas de expresiones tienen los mismos valores.

Cuando un temporizador inactivo es inicializado, se le asocia un valor de tiempo. Si este temporizador no es reinicializado, o si no es inicializado de nuevo, antes de que el tiempo de sistema llegue a este valor de tiempo, se aplica al puerto de entrada del proceso una señal con el mismo nombre que el temporizador. La misma acción se efectúa si el temporizador es inicializado con un valor de tiempo menor que NOW. Después del consumo de una señal de temporizador, la expresión SENDER da el mismo valor que la expresión SELF. Si se da una lista de expresiones cuando el temporizador está inicializado, los valores de estas expresiones (o expresión) están contenidos en la señal de temporizador en el mismo orden. Un temporizador está activo desde el momento de la inicialización hasta el momento del consumo de la señal de temporizador.

Si un género especificado en una definición de temporizador es un sintipo, la verificación de intervalo definida en § 5.4.19.1 aplicada a la expresión correspondiente en una inicialización o reinicialización debe ser Verdadero; de no ser así el sistema está en error y su comportamiento ulterior está indefinido.

Cuando un temporizador inactivo es reinicializado, sigue estando inactivo.

Cuando un temporizador activo es reinicializado, la asociación con el valor de tiempo se pierde; si hay una señal de temporización correspondiente retenida el puerto de entrada, se suprime y el temporizador pasa a inactivo.

La inicialización de un temporizador activo equivale a reinicializarlo e inicializarlo inmediatamente después. Entre los instantes de reinicialización e inicialización del temporizador, éste permanece activo.

Una instancia de temporizador está inactiva antes de ser inicializada por primera vez.

2.9 *Ejemplos*

```
----------
INPUT S1/*ejemplo*/;
TASK/*ejemplo*/T1:=0;
----------
```

FIGURA 2.9.1

Ejemplo de comentario (PR)

```
----------
INPUT I1 COMMENT 'ejemplo';
TASK T1:=0;
----------
```

FIGURA 2.9.2

Ejemplo de comentario (PR)

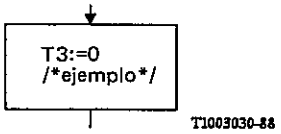


FIGURA 2.9.3

Ejemplo de comentario (GR)

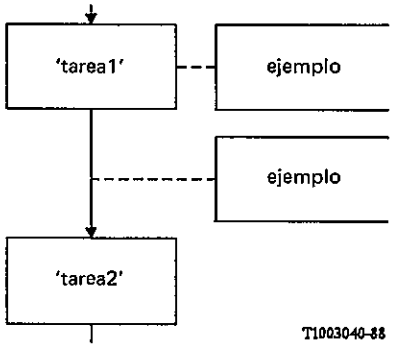


FIGURA 2.9.4

Ejemplo de comentario (GR)

```
SYSTEM DAEMON_GAME;

/* Este sistema es un juego ..... Un jugador lo termina por la señal Endgame */

SIGNAL Newgame, Probe, Result, Endgame, Gameid, Win, Lose, Score (Integer),
Subscr,Endsubscr, Bump;

CHANNEL C1
        FROM ENV TO Blockgame
        WITH Newgame, Probe, Result, Endgame;
        FROM Blockgame TO ENV
        WITH Gameid, Win, Lose, Score;
ENDCHANNEL C1;

CHANNEL C3 FROM Blockgame TO Blockdaemon
        WITH Subscr, Endsubscr;
ENDCHANNEL C3;

CHANNEL C4 FROM Blockdaemon TO Blockgame
        WITH Bump;
ENDCHANNEL C4;

BLOCK Blockgame REFERENCED;

BLOCK Blockdaemon REFERENCED;

ENDSYSTEM DAEMON_GAME;
```

FIGURA 2.9.5

Ejemplo de una especificación de sistema (PR)

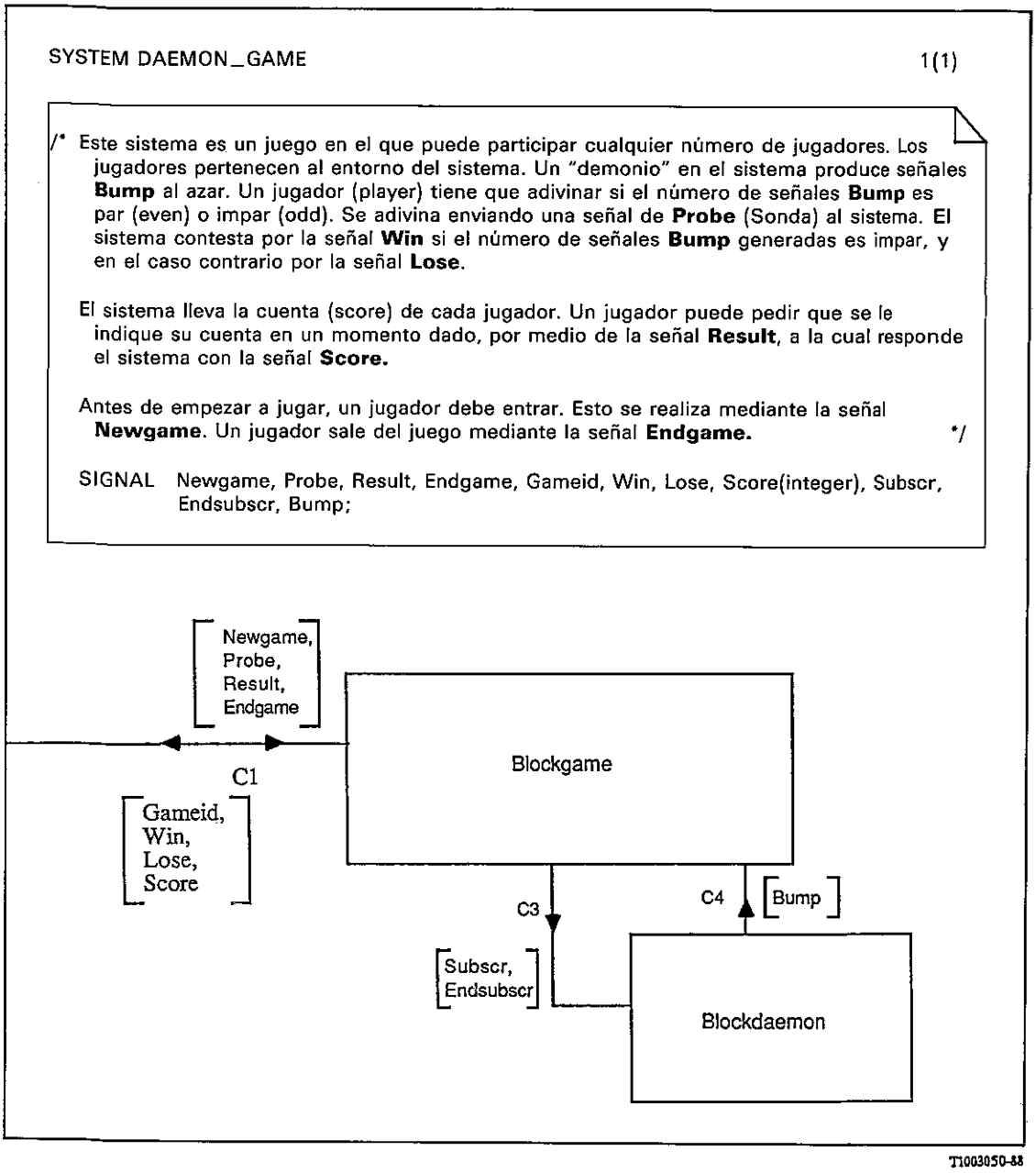


FIGURA 2.9.6

Ejemplo de una especificación de sistema (GR)

```
BLOCK Blockgame;


CONNECT C1 AND R1,R2,R3;
CONNECT C3 AND R4;
CONNECT C4 AND R5;
SIGNALROUTE R1 FROM ENV TO Monitor WITH Newgame;
SIGNALROUTE R2 FROM ENV TO Game
        WITH Probe, Result, Endgame;
SIGNALROUTE R3 FROM Game TO ENV
        WITH Gameid, Win, Lose, Score;
SIGNALROUTE R4 FROM Game TO ENV
        WITH Subscr, Endsubscr;
SIGNALROUTE R5 FROM ENV TO Game WITH Bump;


PROCESS Monitor (1,1) REFERENCED;


PROCESS Game (0,) REFERENCED;


ENDBLOCK Blockgame;
```

FIGURA 2.9.7

Ejemplo de especificación de bloque (PR)

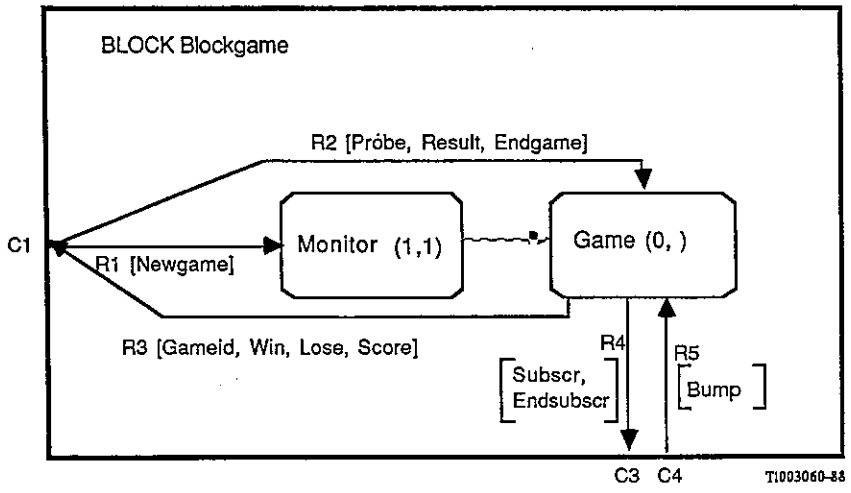


FIGURA 2.9.8

Ejemplo de un diagrama de bloque

```
PROCESS Game (0,);
FPAR Player Pid;


DCL
      Count Integer;/*contador para llevar la cuenta (score)*/


START;
   OUTPUT Subscr;
   OUTPUT Gameid TO Player;
   TASK Count:=0;
NEXTSTATE Even;


STATE Even;


INPUT Probe;
   OUTPUT Lose TO Player;
   TASK Count:=Count-1;
NEXTSTATE-;


INPUT Bump;
NEXSTATE Odd;


STATE Odd;


INPUT Bump;
NEXSTATE Even;


INPUT Probe;
   OUTPUT Win TO Player;
   TASK Count:=Count+1;
NEXTSTATE-;


STATE*;


INPUT Result;
   OUTPUT Score(Count) TO Player;
NEXTSTATE-;


INPUT Endgame;
   OUTPUT Endsubscr;
STOP;


ENDPROCESS Game;
```

FIGURA 2.9.9

Ejemplo de una especificación de proceso (PR)

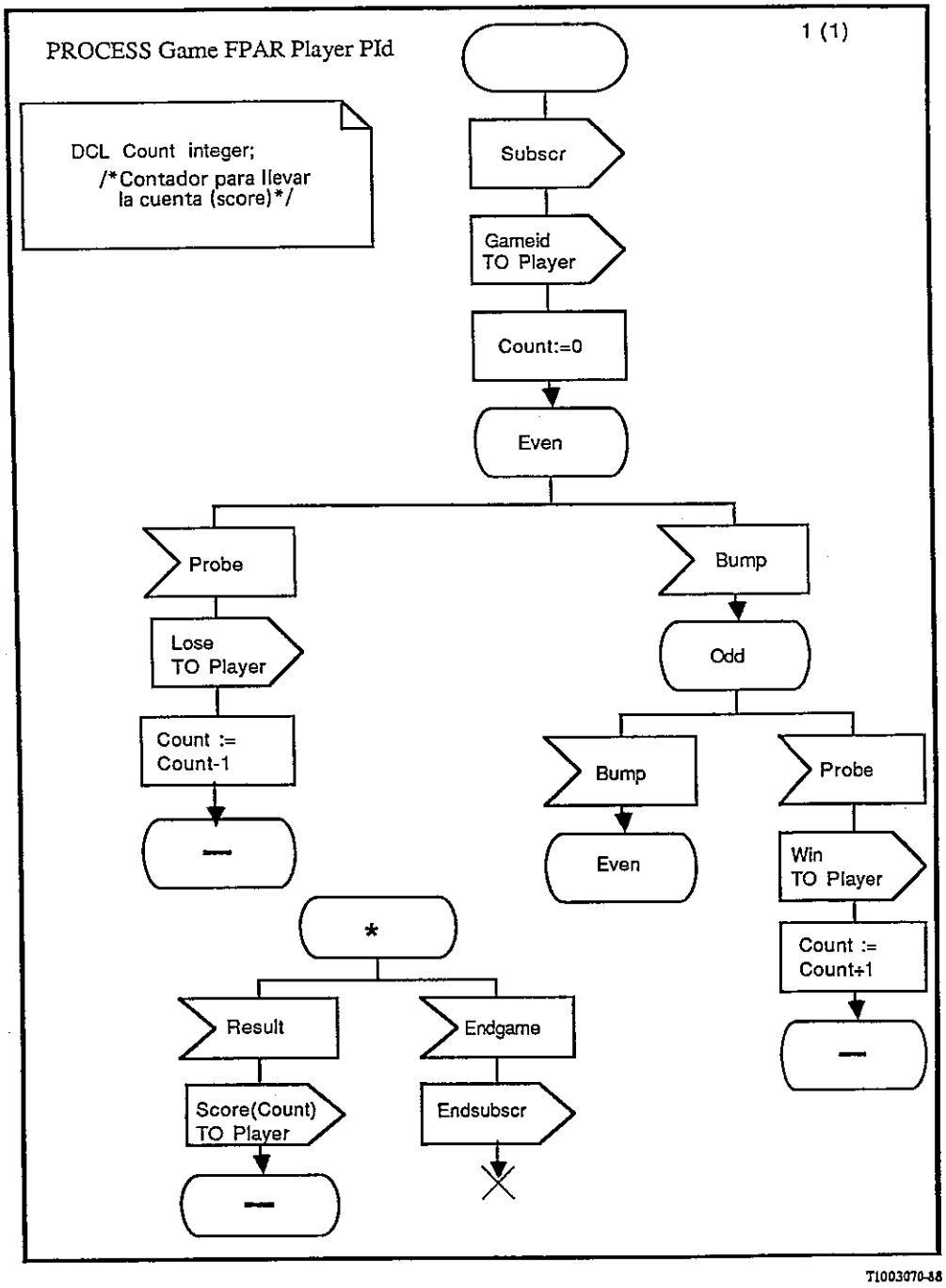


FIGURA 2.9.10

Ejemplo de una especificación de proceso (GR)

```
PROCEDURE check;
/* Se suponen las siguientes definiciones de señal:
   SIGNAL sig1(Boolean), sig2, sig3(Integer,Pid); */
      FPAR IN/OUT x, y Integer;
      DCL sum, index Integer,
            nice Boolean;
      START;
      TASK sum:=0,
            index:=1;
      NEXTSTATE idle;
      STATE idle;
            INPUT sig1(nice);
            DECISION nice;
                  (true): TASK 'Calculate sum';
                        OUTPUT sig3(sum, SENDER);
                        RETURN;
                  (false): NEXTSTATE Jaj;
            ENDDECISION;
            INPUT sig2;
            .....
      .........
      .........

ENDPROCEDURE check;
```

FIGURA 2.9.11

Ejemplo de un fragmento de una especificación de procedimiento (PR)

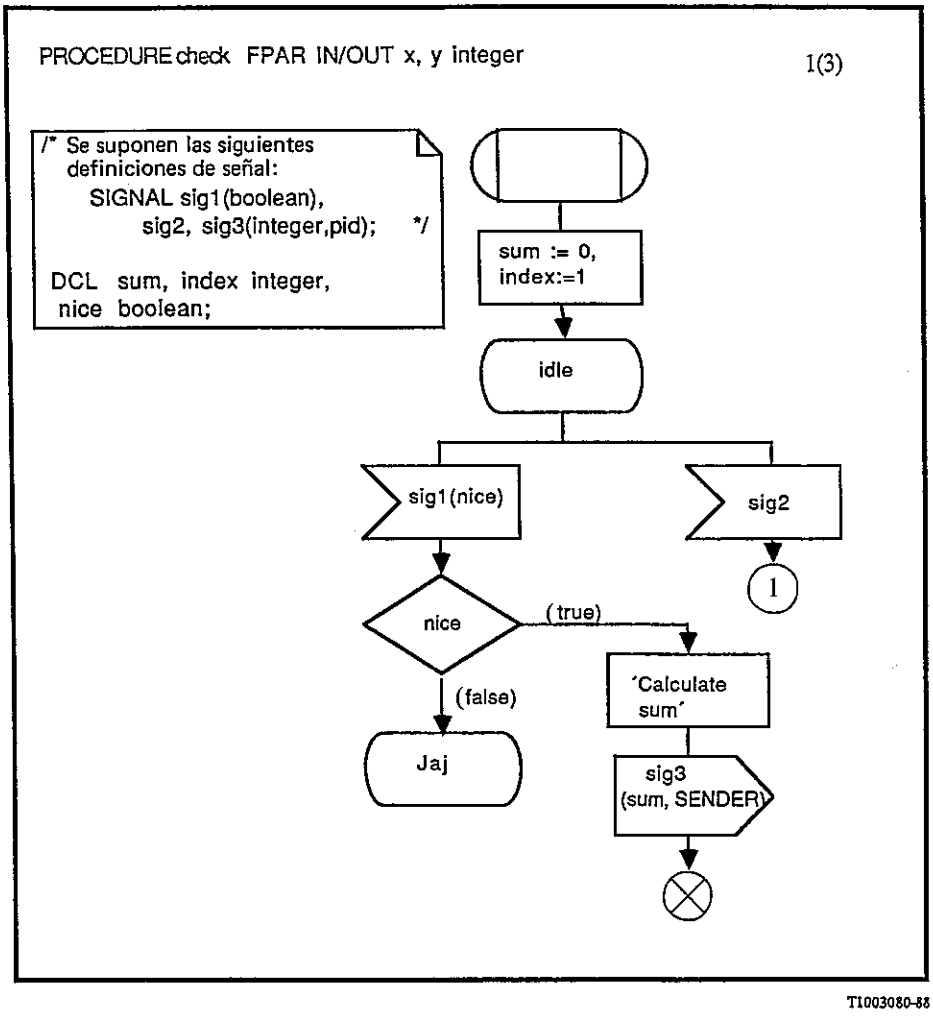


FIGURA 2.9.12

Ejemplo de un fragmento de una especificación de procedimiento (GR)

```
/* Se supone la siguiente definición de señal:
SIGNAL inquire(Integer,Integer,Integer); */
PROCESS alfa;
DCL a,b,c Integer;
        .....
        ......
        INPUT inquire (a,b,c);
        CALL check (a,b);
        ......
ENDPROCESS;
```

FIGURA 2.9.13

Ejemplo de una llamada a procedimiento en un fragmento de una definición de proceso (PR)

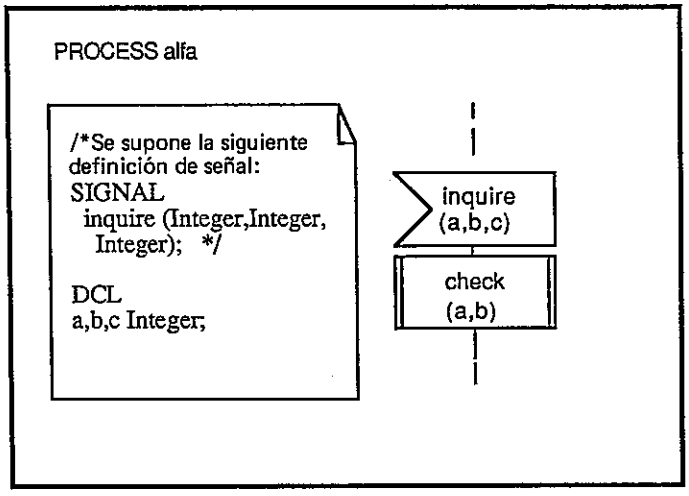


FIGURA 2.9.14

Ejemplo de una llamada a procedimiento en un fragmento de una definición de proceso (GR)

3 Conceptos estructurales en LED

3.1 *Introducción*

Esta sección define cierto número de conceptos necesarios para manejar estructuras jerárquicas en LED. La base de estos conceptos se define en § 2 y los conceptos definidos son adiciones estrictas a los definidos en § 2.

Los conceptos introducidos en esta sección tienen por finalidad proporcionar al usuario del LED medios para especificar sistemas grandes y/o complejos. Los conceptos definidos en § 2 son adecuados para especificar sistemas relativamente pequeños que pueden ser comprendidos y manejados a un nivel único de bloques. Cuando se especifican sistemas grandes, o complejos, es necesario particionar la especificación del sistema en unidades manejables, que puedan ser tratadas y comprendidas independientemente. A menudo conviene efectuar la partición en varios pasos con lo que se obtiene una estructura jarárquica de unidades que especifican el sistema.

El término partición (y su derivado particionar) significa división de una unidad en subunidades más pequeñas, que son sus componentes. La partición no afecta al interfaz estático de una unidad. Además de efectuar la partición, es necesario también añadir nuevos detalles al comportamiento de un sistema cuando se desciende a niveles inferiores en la estructura jerárquica de la especificación del sistema. Esto se denota por el término refinamiento.

3.2 *Partición (o fraccionamiento)*

3.2.1 *Generalidades*

Una definición de bloque puede ser fraccionada (o subdivida, o «particionada») en un conjunto de definiciones de subbloque, definiciones de canal y definiciones de subcanal. De manera similar, una definición de canal puede ser fraccionada en un conjunto de definiciones de bloque, definiciones de canal y definiciones de subcanal. Así, cada definición de bloque y definición de canal puede tener dos versiones: una versión no fraccionada y una versión fraccionada en las sintaxis concretas. Sin embargo, las estructuras de canal se transforman cuando se establece una relación de correspondencia con la sintaxis abstracta. Estas dos versiones tienen el mismo interfaz estático, pero su comportamiento puede diferir hasta cierto punto, porque el orden de las señales de salida puede no ser el mismo. Una definición de subbloque es una definición de bloque, y una definición de subcanal es una definición de canal.

En la definición de un sistema concreto, así como en la definición de un sistema abstracto, puede aparecer tanto la versión no fraccionada como la versión fraccionada de un bloque. En este caso, una definición de sistema concreto contiene varios subconjuntos de partición consistentes, correspondiendo cada subconjunto a una instancia de sistema. Un subconjunto de partición consistente es una selección de las *definiciones de bloque* en una *definición de sistema* de tal modo que:

a) si contiene una *definición-de-bloque*, tiene entonces que contener la definición de la unidad de ámbito circundante, si existe;

b) debe contener todas las *definiciones-de-bloque* definidas al nivel del sistema y, si contiene una *definición-de-subbloque* de una *definición-de-bloque*, tendrá entonces que contener todas las otras *definiciones-de-subbloque* de esa *definición-de-bloque*;

c) todas las *definiciones-de-bloque* «constitutivo de hoja» en la estructura resultante contienen *definiciones-de-proceso.*

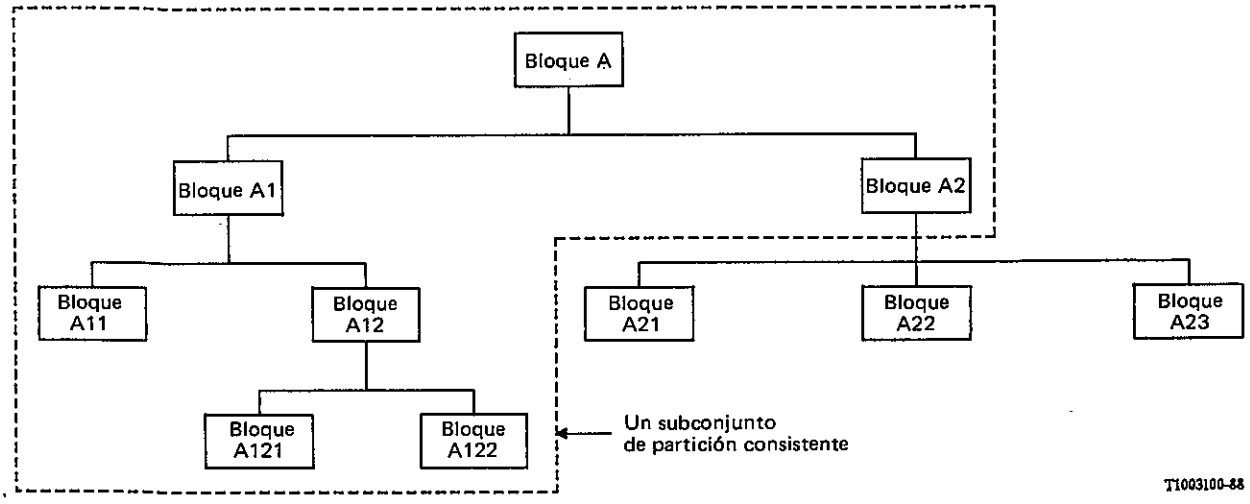


FIGURA § 3.2.1

Subconjunto de partición consistente ilustrado en un diagrama auxiliar

En el momento de la interpretación del sistema, se interpreta un subconjunto de partición consistente.

Se interpretan los procesos en cada uno de los bloques constitutivos de hoja en el subconjunto de partición consistente. Si estos bloques constitutivos de hoja contienen también subestructuras, no producen efecto alguno. Las subestructuras en los bloques no constitutivos de hoja producen efecto sobre la visibilidad, y los procesos en estos bloques no se interpretan.

3.2.2 *Partición de bloque*

Gramática abstracta

Definición-de-subestructura-de-bloque	::	*Nombre-de-subestructura-de-bloque* *Definición-de-subbloque*-**set** *Conexión-de-canal*-**set** *Definición-de-canal*-**set** *Definición-se-señal*-**set** *Definición-de-tipo-de-datos* *Definición-de-sintipo*-**set**
Nombre-de-subestructura-de-bloque	=	*Nombre*
Definición-de-subbloque	=	*Definición-de-bloque*
Conexión-de-canal	::	*Identificador-de-canal* *Identificador-de-subcanal*-**set**
Identificador-de-subcanal	=	*Identificador-de-canal*
Identificador-de-canal	=	*Identificador*

La *definición-de-subestructura-de-bloque* tiene que contener por lo menos una *definición-de-subbloque.* Queda entendido que, en lo sucesivo, un término de sintaxis abstracta está contenido en la *definición-de-subestructura-de-bloque,* si no se indica otra cosa.

Un *identificador-de-bloque* contenido en una *definición-de-canal* tiene que denotar una *definición-de-subbloque.* Una *definición-de-canal* que conecta una *definición-de-subbloque* a la frontera de la *definición-de-subestructura-de-bloque* se denomina definición de subcanal.

Para cada *definición-de-canal* externa conectada a la *definición-de-subestructura-de-bloque* tiene que hacer exactamente una *conexión-de-canal.* El *identificador-de-canal* en la *conexión-de-canal* tiene que identificar esta *definición-de-canal* externa.

Para señales que salen de la *definición-de-subestructura-de-bloque*, la unión de los *identificadores-de-señal* asociados al *identificador-de-subcanal*-set contenido en una *conexión-de-canal* debe ser idéntica a los *identificadores-de-señal* asociados al *identificador-de-canal* contenido en la *conexión-de-canal*. La misma regla es válida para las señales que entran en la *definición-de-subestructura-de-bloque*. Sin embargo, esta regla se modifica en el caso de refinamiento de señal (véase § 3.3).

Cada *identificador-de-subcanal* tiene que aparecer en una, y sólo una, *conexión-de-canal*.

Puesto que una *definición-de-subbloque* es una *definición-de-bloque*, puede ser fraccionada. Esta partición puede repetirse cualquier número de veces, de lo que resulta una estructura jerárquica en árbol de *definiciones-de-bloque* y sus *definiciones-de-subbloque*. Se dice que las *definiciones-de-subbloque* de una *definición-de-bloque* existen en el nivel inmediato inferior del árbol de bloques (véase también la figura que sigue).

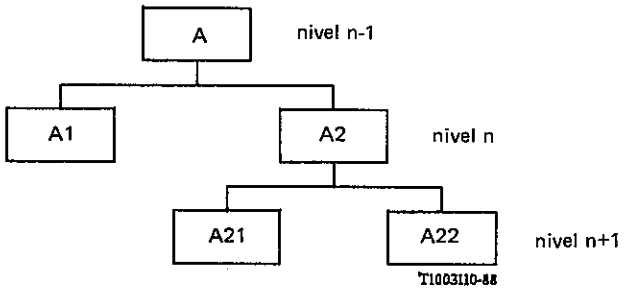


FIGURA 1 § 3.2.2

Diagrama de árbol de bloques

El diagrama de árbol de bloques es un diagrama auxiliar.

Gramática textual concreta

<definición de subestructura de bloque> ::=
SUBSTRUCTURE {[<nombre de subestructura de bloque>]
| <identificador de subestructura de bloque> } <fin>
{ <definición de bloque>
| <referencia textual de bloque>
| <definición de canal>
| <conexión de canal>
| <definición de señal>
| <definición de lista de señales>
| <definición de datos>
| <definición de seleccionar>
| <definición de macro> }+
ENDSUBSTRUCTURE [{<nombre de subestructura de bloque>
| <identificador de subestructura de bloque>}] <fin>

El <nombre de subestructura de bloque> después de la palabra clave SUBSTRUCTURE sólo puede omitirse si es igual al <nombre de bloque> en la <definición de bloque> circundante.

<referencia textual de subestructura de bloque> ::=
SUBSTRUCTURE <nombre de subestructura de bloque> REFERENCED <fin>

<conexión de canal> ::=
CONNECT <identificador de canal> AND <identificador de subcanal>
{, <identificador de subcanal>}* <fin>

Gramática gráfica concreta

<diagrama de subestructura de bloque> ::=
<símbolo de casilla>
contains {<encabezamiento de subestructura de bloque>
{{<área de texto de subestructura de bloque>}*
{<diagrama de macro>}*
<área de interacción de bloque> } **set** }
is associated with {<identificador de canal>}*

El <identificador de canal> identifica un canal conectado a un subcanal en el <diagrama de subestructura de bloque>. Se coloca fuera del <símbolo de casilla> próximo al punto extremo del subcanal en el <símbolo de casilla>

Un <símbolo de canal> dentro del <símbolo de casilla> y conectado a éste indica un subcanal.

<encabezamiento de subestructura de bloque> ::=
SUBSTRUCTURE {<nombre de subestructura de bloque> | <identificador de subestructura de bloque>}

<área de texto de subestructura de bloque> ::=
<área de texto de sistema>

<área de subestructura de bloque> ::=
<referencia gráfica de subestructura de bloque>
| <diagrama de subestructura de bloque>
| <diagrama de subestructura de bloque abierto>

<referencia gráfica de subestructura de bloque> ::=
<símbolo de subestructura de bloque> **contains** <nombre de subestructura de bloque>

<símbolo de subestructura de bloque> ::=
<símbolo de bloque>

<diagrama de subestructura de bloque abierto> ::=
{{<área de texto de subestructura de bloque>}*
{<diagrama de macro>}*
<área de interacción de bloques>} **set**

Cuando un <área de subestructura de bloque> es un <diagrama de subestructura de bloque abierto>, el <diagrama de bloque> circundante no puede contener <área de texto de bloque>, <diagrama de macro>, ni <área de interacción de procesos>.

Semántica

Véase § 3.2.1.

Modelo

Un <diagrama de subestructura de bloque abierto> se transforma en un <diagrama de subestructura de bloque> de tal manera que en el <encabezamiento de subestructura de bloque> el <nombre de subestructura de bloque> o el <identificador de subestructura de bloque> es el mismo que el <nombre de bloque> respectivamente <identificador de bloque> en el <diagrama de bloque> circundante.

Ejemplo

A continuación se presenta un ejemplo de una <definición de subestructura de bloque>.

```
BLOCK A;
    SUBSTRUCTURE A;
        SIGNAL s5(nat), s6, s8, s9(min);
        BLOCK a1 REFERENCED;
        BLOCK a2 REFERENCED;
        BLOCK a3 REFERENCED;
        CHANNEL c1 FROM a2 TO ENV WITH s1, s2; ENDCHANNEL c1;
        CHANNEL c2 FROM ENV TO a1 WITH s3;
            FROM a1 TO ENV WITH s1; ENDCHANNEL c2;
        CHANNEL d1 FROM a2 TO ENV WITH s7; ENDCHANNEL d1;
        CHANNEL d2 FROM a3 TO ENV WITH s10; ENDCHANNEL d2;
        CHANNEL e1 FROM a1 TO a2 WITH s5, s6; ENDCHANNEL e1;
        CHANNEL e2 FROM a3 TO a1 WITH s8; ENDCHANNEL e2;
        CHANNEL e3 FROM a2 TO a3 WITH s9; ENDCHANNEL e3;
        CONNECT c AND c1, c2;
        CONNEXT d AND d1, d2;
    ENDSUBSTRUCTURE A;
ENDBLOCK A;
```

El <diagrama de subestructura de bloque> para el mismo ejemplo se presenta a continuación.

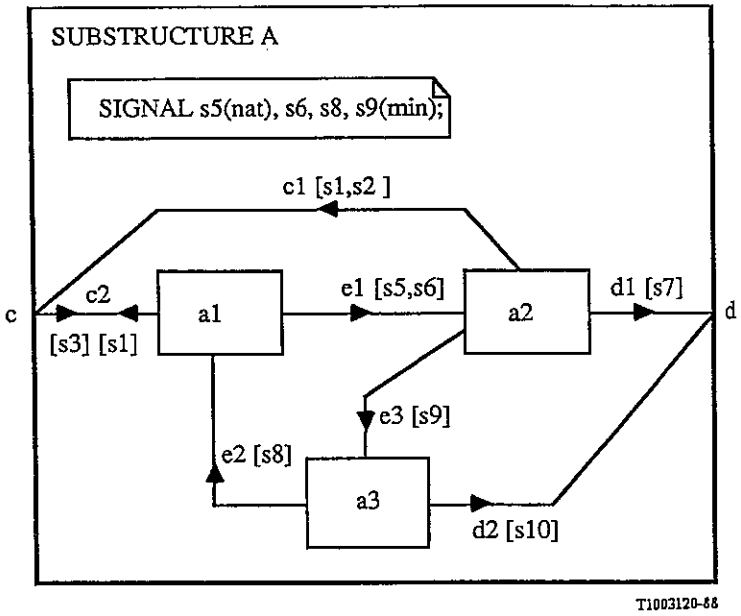


FIGURA 2/§ 3.2.2

Diagrama de subestructura de bloque para un bloque A

3.2.3 *Partición de canal*

Todas las condiciones estáticas se enuncian mediante una gramática textual concreta. Condiciones análogas se cumplen para la gramática gráfica concreta.

Gramática textual concreta

<definición de subestructura de canal> ::=
SUBSTRUCTURE {[<nombre de subestructura de canal>]
| <identificador de subestructura de canal> } <fin>
{ <definición de bloque>
| <referencia textual de bloque>
| <definición de canal>
| <conexión de punto extremo de canal>
| <definición de señal>
| <definición de lista de señales>
| <definición de datos>
| <definición de seleccionar>
| <definición de macro> }+
ENDSUBSTRUCTURE [{ <nombre de subestructura de canal>
| <identificador de subestructura de canal>}] <fin>

El <nombre de subestructura de canal> que sigue a la palabra clave SUBSTRUCTURE sólo puede omitirse si es el mismo que el <nombre de canal> en la <definición de canal> circundante.

<referencia textual de subestructura de canal> ::=
SUBSTRUCTURE <nombre de subestructura de canal> REFERENCED <fin>

<conexión de punto extremo de canal> ::=
CONNECT {<identificador de bloque | ENV} AND <identificador de subcanal>
{, <identificador de subcanal>}* <fin>

Para cada punto extremo de la <definición de canal> fraccionada tiene que haber exactamente una <conexión de punto extremo de canal>. El <identificador de bloque> o ENVIRONMENT en una <conexión de punto extremo de canal> tiene que identificar uno de los puntos extremos de la <definición de canal> fraccionada.

Gramática gráfica concreta

<diagrama de subestructura concreta> ::=
<símbolo de casilla>
contains {<encabezamiento de subestructura de canal>
{{ <área de texto de subestructura de canal>}*
{<diagrama de macro>}*
<área de interacción de bloques> } **set** }
is associated with {<identificador de bloque> | ENV}+

El <identificador de bloque> o ENV identifica un punto extremo del canal fraccionado. El <identificador de bloque> se coloca fuera del <símbolo de casilla> próximo al punto extremo del subcanal asociado en el <símbolo de casilla>. El <símbolo de canal> dentro del <símbolo de casilla> y conectado a éste indica un subcanal.

<encabezamiento de subestructura de canal> ::=
SUBSTRUCTURE { <nombre de subestructura de canal>
| <identificador de subestructura de canal>}

<área de texto de subestructura de canal> ::=
<área de texto de sistema>

<área de asociación de subestructura de canal> ::=
<símbolo de asociación de trazo discontinuo>
is connected to <área de subestructura de canal>

<área de subestructura de canal> ::=
<referencia gráfica de subestructura de canal>
| <diagrama de subestructura de canal>

<referencia gráfica de subestructura de canal> ::=
<símbolo de subestructura de canal> **contains** <nombre de subestructura de canal>

<símbolo de subestructura de canal> ::=
<símbolo de bloque>

Modelo

Una <definición de canal> que contiene una <definición de subestructura de canal> se transforma en una <definición de bloque> y dos <definición de canal>s de tal modo que:

a) Cada una de las dos <definición de canal>s está conectada al bloque y a un punto extremo del canal original. Las <definición de canal>s tienen nombres nuevos distintos y toda referencia al canal original en las construcciones VIA se reemplaza por una referencia al nuevo canal apropiado.

b) La <definición de bloque> tiene un nombre nuevo distinto y contiene solamente una <definición de subestructura de bloque> que tiene el mismo nombre y contiene las mismas definiciones que la <definición de subestructura de canal> original. Los calificadores en la nueva <definición de bloque> se cambian para incluir el nombre de bloque. Las dos <conexión de punto extremo de canal>s provenientes de la <definición de subestructura de canal> original se representan por dos <conexión de canal>s en las cuales el <identificador de bloque> o ENV se reemplaza por el nuevo canal apropiado.

Esta transformación debe efectuarse inmediatamente después de las de un sistema genérico. Véase § 4.3.

Ejemplo

A continuación se presenta un ejemplo de una <definición de subestructura de canal>.

```
CHANNEL C FROM A TO B WITH s1;
    FROM B TO A WITH s2;

    SUBSTRUCTURE C;

        SIGNAL s3(hel), s4(boo), s5;

        BLOCK b1 REFERENCED;
        BLOCK b2 REFERENCED;

        CHANNEL c1 FROM ENV TO b1 WITH s1;
            FROM b1 TO ENV WITH s2; ENDCHANNEL c1;

        CHANNEL c2 FROM b2 TO ENV WITH s1;
            FROM ENV TO b2 WITH s2; ENDCHANNEL c2;

        CHANNEL e1 FROM b1 TO b2 WITH s3; ENDCHANNEL e1;
        CHANNEL e2 FROM b2 TO b1 WITH s4, s5; ENDCHANNEL e2;
```

CONNECT A AND c1;
CONNECT B AND c2;

ENDSUBSTRUCTURE C;

ENDCHANNEL C;

El <diagrama de subestructura de canal> para el mismo ejemplo se presenta a continuación.

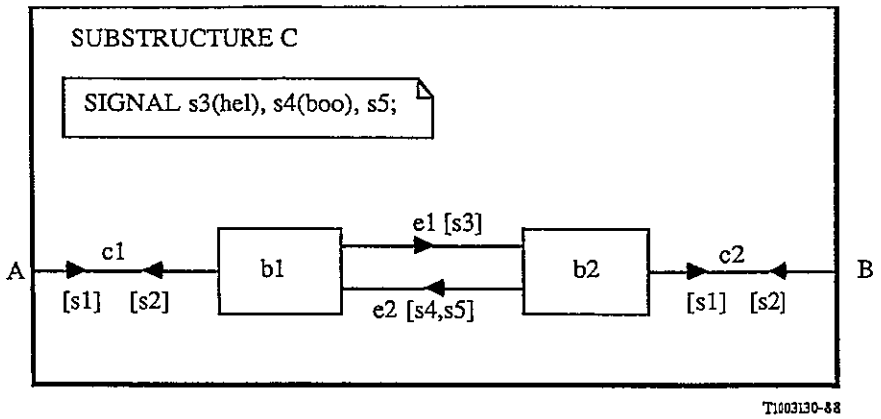


FIGURA § 3.2.3

Diagrama de subestructura de canal para un canal C

3.3 *Refinamiento*

El refinamiento se efectúa refinando una definición de señal para formar un conjunto de definiciones de subseñal. Una definición de subseñal es una definición de señal y, a su vez, puede ser refinada. Este refinamiento puede repetirse cualquier número de veces, y como resultado de ello se obtiene una estructura jerárquica de definiciones de señal y sus definiciones de subseñal. Obsérvese que una definición de subseñal de una definición de señal no se considera un componente de la definición de señal.

Gramática abstracta

Refinamiento-de-señal :: *Definición-de-subseñal*-**set**

Definición-de-subseñal :: [REVERSE] *Definición-de-señal*

Para cada *conexión-de-canal* se debe cumplir que para cada *identificador-de-señal* asociado al *identificador-de-canal*, bien el *identificador-de-señal* está asociado por lo menos a uno de los *identificadores-de-subcanal*, bien cada uno de sus identificadores de subseñal está asociado a por lo menos uno de los *identificadores-de-subcanal*. Este es un cambio de las reglas de partición correspondientes.

En el conjunto completo de señales de entrada válidas de una definición de proceso o de los *nodos-de-salida* de una definición de proceso no puede haber dos señales en dos niveles diferentes de refinamiento de la misma señal.

Gramática textual concreta

<refinamiento de señal> ::=
REFINEMENT
{<definición de subseñal>}+
ENDREFINEMENT

<definición de subseñal> ::=
[REVERSE] <definición de señal>

Semántica

Cuando se define que una señal sea transportada por un canal, el canal será automáticamente el portador de todas las subseñales de esa señal. Puede tener lugar un refinamiento cuando el canal sea fraccionado o dividido en subcanales. En tal caso los subcanales portarán las subseñales en lugar de la señal refinada. El sentido de flujo de una subseñal está determinado por el subcanal portador; una subseñal puede tener el sentido opuesto al de la señal refinada, lo que se indica por la palabra clave REVERSE. Las señales no pueden refinarse cuando un canal se divide en rutas de señales.

Cuando una definición de sistema contiene un refinamiento de señal, el concepto de subconjunto de partición consistente queda restringido. Se dice que tal definición de sistema contiene varios subconjuntos de refinamiento consistentes.

Un subconjunto de refinamiento consistente es un subconjunto de partición consistente restringido por la regla siguiente:

– Al seleccionar el subconjunto de partición consistente, el conjunto de señales por las rutas de señales conectadas a un punto extremo de un canal no debe contener señales progenitoras de subseñales contenidas y, a menos que el otro punto extremo sea el sistema ENVIRONMENT, el conjunto de señales para el primer punto extremo debe ser igual al conjunto de señales por las rutas de señales conectadas al otro punto extremo,

Ejemplo

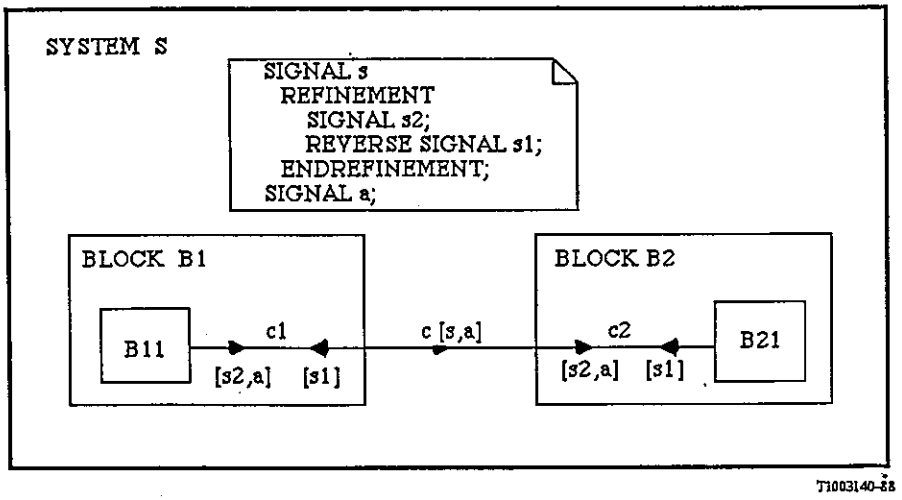


FIGURA § 3.3

Diagrama de sistema que contiene un refinamiento de señal

En el ejemplo anterior, la señal s es refinada en las definiciones de bloque B1 y B2, pero la señal a no es refinada. En el nivel de refinamiento más elevado, los procesos B1 y B2 comunican utilizando señales s y a. En el nivel inferior inmediato, los procesos B11 y B21 comunican utilizando las señales s1, s2 y a.

Obsérvese que no está autorizado el refinamiento en una sola de las definiciones de bloque B1 y B2, pues no hay una transformación dinámica entre una señal y sus subseñales, sino solamente una relación estática.

4 Conceptos adicionales en LED

4.1 *Introducción*

Este capítulo define cierto número de conceptos adicionales. Se trata de notaciones taquigráficas estándar, que se modelan en términos de los conceptos primitivos de LED, utilizando sintaxis concreta. Se presentan por razones de conveniencia para los usuarios de LED, además de las notaciones taquigráficas definidas en otros capítulos de la Recomendación.

Las propiedades de las notaciones taquigráficas se derivan de la forma en que se modelan en términos de (o se transforman en) los conceptos primitivos. A fin de asegurar una utilización cómoda e inequívoca de las notaciones taquigráficas, y de reducir los efectos secundarios cuando se combinan varias notaciones taquigráficas, estos conceptos se transforman en un orden especificado como sigue:

1 Macro, § 4.2
2 Sistemas genéricos, § 4.3
3 Estado asterisco, § 4.4

4 Lista de estados, § 2.6.3
5 Múltiple aparición de estado, § 4.5
6 Entrada asterisco, § 4.6
7 Conservación asterisco, § 4.7
8 Lista de estímulos, § 2.6.4
9 Lista de salida, § 2.7.4
10 Transición implícita, § 4.8
11 Estado siguiente indicado por guión, § 4.9
12 Servicio, § 4.10
13 Señal continua, § 4.11
14 Condición habilitante, § 4.12
15 Valor importado y exportado § 4.13

Este orden se sigue también cuando se definen los conceptos en esta sección. El orden especificado de transformación significa que en la transformación de una notación taquigráfica de orden *n* puede utilizarse otra notación taquigráfica de orden *m*, siempre que $m > n$.

Como no hay sintaxis abstracta para las notaciones taquigráficas, en sus definiciones se utilizan términos bien de sintaxis gráfica, bien de sintaxis textual. La elección entre términos de sintaxis gráfica y términos de sintaxis textual se basa en consideraciones prácticas, y no limita la utilización de las notaciones taquigráficas a una sintaxis concreta particular.

4.2 *Macro*

En el texto que sigue, los términos definición de macro y llamada a macro se utilizan en un sentido general, que abarca tanto el LED/GR como el LED/PR. Una definición de macro contiene una colección de símbolos gráficos y/o unidades léxicas, que pueden incluirse en uno o más lugares en la <definición de sistema concreto>. Cada uno de estos lugares se indica por una llamada a macro. Antes de que pueda analizarse una <definición de sistema concreto>, cada llamada a macro deberá remplazarse por la correspondiente definición de macro.

4.2.1 *Reglas léxicas*

```
<nombre formal> ::=
        [<nombre> %] <parámetro de macro>
        {% <nombre> %<parámetro de macro> | %<parámetro de macro> }* [%<nombre>]
```

4.2.2 *Definición de macro*

Gramática textual concreta

```
<definición de macro> ::=
        MACRODEFINITION <nombre de macro>
        [<parámetros formales de macro>] <fin>
        <cuerpo de macro>
        ENDMACRO [<nombre de macro>] <fin>

<parámetros formales de macro> ::=
        FPAR <parámetros formales de macro> {, <parámetro formal de macro>}*

<parámetro formal de macro> ::=
        <nombre>

<cuerpo de macro> ::=
        {<unidad léxica> | <nombre formal>}*

<parámetro de macro> ::=
        <parámetro formal de macro>
    |   MACROID
```

Los <parámetro formal de macro>s tienen que ser distintos. Los <parámetro efectivo de macro>s de una llamada a macro tienen que ser aplicados uno a uno con sus correspondientes <parámetro formal de macro>s.

El <cuerpo de macro> no podrá contener las palabras claves ENDMACRO y MACRODEFINITION.

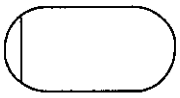
Gramática gráfica concreta

<diagrama de macro> ::=
<símbolo de casilla> **contains** {<encabezamiento de macro> <área de cuerpo de macro>}

<encabezamiento de macro> ::=
MACRODEFINITION <nombre de macro> [<parámetros formales de macro>]

<área de cuerpo de macro> ::=
{{<cualquier área>}*
<cualquier área> [**is connected to** <cuerpo de macro puerto1>]}**set**
|{ <cualquier área> **is connected to** <cuerpo de macro puerto2>
<cualquier área> **is connected to** <cuerpo de macro puerto2>
{ <cualquier área> [**is connected to** <cuerpo de macro puerto2>]}* }**set**

<símbolo de acceso de entrada de macro> ::=



<símbolo de acceso de salida de macro> ::=

<cuerpo de macro puerto1> ::=
<símbolo de acceso de salida> **is connected to** {<símbolo de casilla>
[is associated with <etiqueta de macro>]
| <símbolo de acceso de entrada de macro> [{**contains** | **is associated with** }
<etiqueta de macro>]
| <símbolo de acceso de salida de macro> [{**contains** | **is associated with** }
<etiqueta de macro>]}

<cuerpo de macro puerto 2> ::=
<símbolo de acceso de salida> **is connected to**{ <símbolo de casilla>
is associated with <etiqueta de macro>]
| <símbolo de acceso de entrada de macro> {**contains** | **is associated with** }
<etiqueta de macro>
| <símbolo de acceso de salida de macro> {**contains** | **is associated with** } <etiqueta
de macro>}

<etiqueta de macro> ::=
<nombre>

<símbolo de acceso de salida> ::=
<símbolo de acceso de salida ficticio>
| <símbolo de línea de flujo>
| <símbolo de canal>
| <símbolo de ruta de señales>
| <símbolo de asociación de trazo continuo>
| <símbolo de asociación de trazo discontinuo>
| <símbolo de línea de crear>

<símbolo de acceso de salida ficticio> ::=
<símbolo de asociación de trazo continuo>

<cualquier área> ::=
<área de texto de sistema>
| <área de interacción de bloques>
| <área de lista de señales>
| <área de bloque>
| <área de texto de bloque>
| <área de interacción de procesos>
| <referencia gráfica de procedimiento>
| <área de texto de proceso>
| <área de gráfico de proceso>
| <área de fusión>
| <área de cadena de transición>
| <área de estado>
| <área de entrada>
| <área de conservación>
| <área de inicialización>
| <área de reinicialización>
| <área de exportación>
| <área de ampliación de texto>
| <área de asociación de subestructura de canal>
| <área de subestructura de canal>
| <área de subestructura de bloque>
| <área de entrada prioritaria>
| <área de señal continua>
| <área de conector de entrada>
| <área de estado siguiente>
| <área de proceso>
| <área de definición de canal>
| <área de línea de crear>
| <área de definición de ruta de señales>
| <referencia gráfica de proceso>
| <diagrama de proceso>
| <área de arranque>
| <área de salida>
| <área de salida prioritaria>
| <área de tarea>
| <área de petición de crear>
| <área de llamada a procedimiento>
| <área de procedimiento>
| <área de decisión>
| <área de conector de salida>
| <área de texto de procedimiento>
| <área de gráfico de procedimiento>
| <área de arranque de procedimiento>
| <área de texto de subestructura de bloque>
| <área de interacción de bloques>
| <área de servicio>
| <área de definición de ruta de señales de servicio>
| <área de texto de servicio>
| <área de gráfico de servicio>
| <área de arranque de servicio>
| <área de comentario>
| <área de llamada a macro>
| <área de asociación de entrada>
| <área de asociación de conservación>
| <área de opción>
| <área de texto de subestructura de canal>
| <área de opción de transición>
| <área de interacción de servicios>
| <área de asociación de entrada prioritaria>
| <área de asociación de señal continua>
| <área de condición habilitante>

Un <símbolo de acceso de salida ficticio> lo único que puede tener asociado es una <etiqueta de macro>.

Para un <símbolo de acceso de salida> que no sea un <símbolo de acceso de salida ficticio>, el correspondiente <símbolo de acceso de entrada> en la llamada a macro tiene que ser un <símbolo de acceso de entrada ficticio>.

Un <cuerpo de macro> puede aparecer en cualquier texto a que se hace referencia en <cualquier área>.

Semántica

Una <definición de macro> contiene unidades léxicas, mientras que un <diagrama de macro> contiene unidades sintácticas. Así, la relación de correspondencia entre construcciones de macro en sintaxis textual y sintaxis gráfica generalmente no es posible. Por la misma razón se aplican reglas detalladas separadas para sintaxis textual y para sintaxis gráfica, aunque hay algunas reglas comunes.

El <nombre de macro> es visible en la totalidad de la definición de sistema, cualquiera que sea la definición de macro que aparezca. Una llamada a macro puede aparecer antes de la definición de macro correspondiente.

Una definición de macro puede contener llamadas a macro, pero no podrá llamarse a sí misma directamente, ni tampoco indirectamente a través de llamadas a macro en otras definiciones de macro.

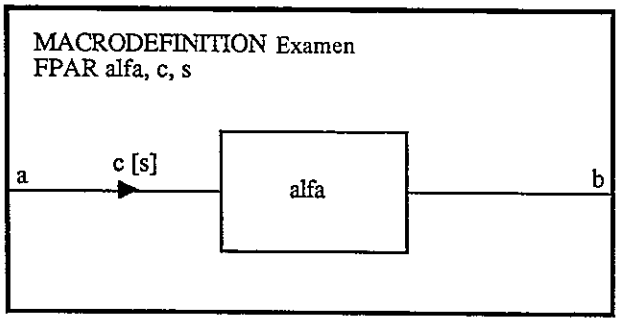
La palabra clave MACROID puede utilizarse como un pseudo parámetro formal de macro dentro de cada definición de macro. No se le podrá dar ningún <parámetro efectivo de macro> y se remplaza por un <nombre> único para cada expansión de una definición de macro (dentro de una expansión se utiliza el mismo <nombre> para cada ocurrencia de MACROID).

Ejemplo

A continuación se presenta un ejemplo de una <definición de macro>:

```
MACRODEFINITION Examen
FPAR alfa, c, s, x;
    BLOCK alfa REFERENCED;
    CHANNEL c FROM x TO alfa WITH s; ENDCHANNEL c;
ENDMACRO Examen;
```

El <diagrama de macro> para el mismo ejemplo se presenta a continuación. Sin embargo, el <parámetro formal de macro>, x, no se requiere en este caso.



T1003150-88

4.2.3 *Llamada a macro*

Gramática textual concreta

<llamada a macro> ::=
MACRO <nombre de macro> [<cuerpo de llamada a macro>] <fin>

<cuerpo de llamada a macro> ::=
(<parámetro efectivo de macro> {, <parámetro efectivo de macro>}*)

<parámetro efectivo de macro> ::=
{<unidad léxica>}*

La <unidad léxica> no puede ser una coma «,» ni un paréntesis derecho «)». Si uno cualquiera de estos caracteres debe ser utilizado en un <parámetro efectivo de macro>, el <parámetro efectivo de macro> tiene que ser una <cadena de caracteres>. Si el <parámetro efectivo de macro> es una <cadena de caracteres>, el valor de la <cadena de caracteres> se utiliza cuando el <parámetro efectivo de macro> remplaza un <parámetro formal de macro>.

Una <llamada a macro> puede aparecer en cualquier lugar en que esté autorizada una <unidad léxica>.

Gramática gráfica concreta

<área de llamada a macro> ::=
<símbolo de llamada a macro> **contains** {<nombre de macro> [<cuerpo de llamada a macro>]}
[**is connected to**
{<llamada a macro puerto1> | <llamada a macro puerto2> {<llamada a macro puerto2>} + }]

<símbolo de llamada a macro> ::=

<llamada a macro puerto 1> ::=
<símbolo de acceso de entrada> [**is associated with** <etiqueta de macro>]
is connected to <cualquier área>

<llamada a macro puerto 2> ::=
<símbolo de acceso de entrada> **is associated with** <etiqueta de macro>
is connected to <cualquier área>

<símbolo de acceso de entrada> ::=
símbolo de acceso de entrada ficticio>
| <símbolo de línea de flujo>
| <símbolo de canal>
| <símbolo de ruta de señales>
| <símbolo de asociación de trazo continuo>
| <símbolo de asociación de trazo discontinuo>
| <símbolo de línea de crear>

<símbolo de acceso de entrada ficticio> ::=
<símbolo de asociación de trazo continuo>

Un <símbolo de acceso de entrada ficticio> lo único que puede tener asociado es <etiqueta de macro>.

Para cada <símbolo de acceso de entrada> tiene que haber un <símbolo de acceso de salida> en el correspondiente <diagrama de macro>, asociado con la misma <etiqueta de macro>. Para un <símbolo de acceso de entrada> que no sea un <símbolo de acceso de entrada ficticio>, el <símbolo de acceso de salida> correspondiente tiene que ser un <símbolo de acceso de salida ficticio>.

Salvo en el caso de <símbolo de acceso de entrada ficticio>s y <símbolo de acceso de salida ficticio>s, es posible tener múltiples <unidad léxica>s (textuales) asociadas con un <símbolo de acceso de entrada> o <símbolo de acceso de salida>. En este caso la <unidad léxica> más próxima al <símbolo de llamada a macro> o el <símbolo de casilla> del <diagrama de macro> se toma por la <etiqueta de macro> asociada con el <símbolo de acceso de entrada> o <símbolo de acceso de salida>.

El <área de llamada a macro> puede aparecer en cualquier lugar en que esté autorizada un área. Sin embargo, se requiere cierto espacio entre el <símbolo de llamada a macro> y cualquier otro símbolo gráfico cerrado. Si de acuerdo con las reglas sintácticas tal espacio no puede estar vacío, el <símbolo de llamada a macro> se conecta al símbolo gráfico cerrado con un <símbolo de acceso de entrada ficticio>.

Semántica

Una definición de sistema puede contener definiciones de macro y llamadas a macro. Antes de que puedan analizarse estas definiciones de sistema, todas las llamadas a macro deben ser expandidas. La expansión de una llamada a macro significa que una copia de la definición de macro que tiene el mismo <nombre de macro> que el dado en la llamada a macro remplaza la llamada a macro.

Cuando se llama a una definición de macro, ésta es expandida. Esto significa que se crea una copia de la definición de macro y cada ocurrencia de los <parámetro formal de macro>s de la copia es remplazada por los correspondientes <parámetro efectivo de macro>s de la llamada a macro, después de lo cual las llamadas a macro en la copia, si existen, son expandidas. Cuando se remplazan los <parámetro formal de macro>s por los <parámetro efectivo de macro>s, se suprimen todos los caracteres «%» en los <nombre formal>s.

Debería haber una correspondencia biunívoca entre <parámetro formal de macro> y <parámetro efectivo de macro>.

– *Reglas para sintaxis gráfica*

Para remplazar el <área de llamada a macro> por una copia del <diagrama de macro> se procede de la siguiente manera. Todos los <símbolo de acceso de entrada de macro>s y <símbolo de acceso de salida de macro>s se suprimen. Un <símbolo de acceso de salida ficticio> se remplaza por el <símbolo de acceso de entrada> que tiene la misma <etiqueta de macro>. Un <símbolo de acceso de entrada ficticio> se remplaza por el <símbolo de acceso de salida> que tiene la misma <etiqueta de macro>. Después se suprimen las <etiqueta de macro>s asociadas a <símbolo de acceso de entrada>s y <símbolo de acceso de salida>s. También se suprimen los <cuerpo de macro puerto1> y <cuerpo de macro puerto2> que no tienen <llamada a macro puerto1> o <llamada a macro puerto2> correspondientes.

Ejemplo

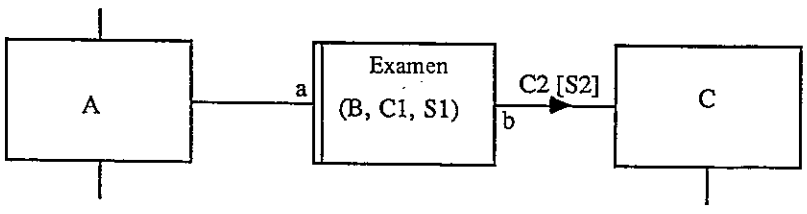
A continuación se presenta un ejemplo de una <llamada a macro>, dentro de un fragmento de una <definición de bloque>.

```
.....
BLOCK A REFERENCED;
MACRO Examen (B, C1, S1, A);
BLOCK C REFERENCED;
CHANNEL C2 FROM B TO C WITH S2; ENDCHANNEL C2;
.....
```

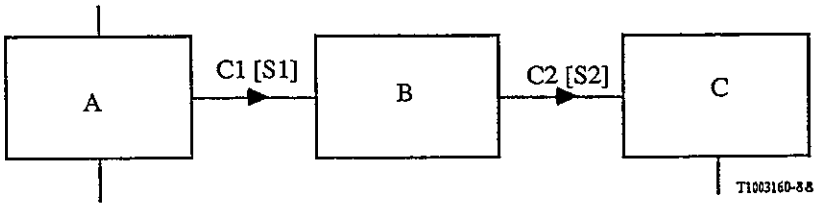
La expansión de esta llamada a macro, utilizando el ejemplo del § 4.2.2, da el siguiente resultado:

```
.....
BLOCK A REFERENCED;
BLOCK B REFERENCED;
CHANNEL C1 FROM A TO B WITH S1; ENDCHANNEL C1;
BLOCK C REFERENCED;
CHANNEL C2 FROM B TO C WITH S2; ENDCHANNEL C2;
.....
```

Seguidamente se presenta el <área de llamada a macro> para el mismo ejemplo, dentro de un fragmento de un <área de interacción de bloques>.



La expansión de esta llamada a macro da el siguiente resultado:



4.3 *Sistemas genéricos*

Una especificación de sistema debe tener partes opcionales y parámetros de sistema con valores indefinidos a fin de satisfacer diversas necesidades. Tal especificación de sistema se denomina genérica; su propiedad genérica se especifica por medio de sinónimos externos (que son análogos a parámetros formales en una definición de procedimiento). Una especificación de sistema genérico se concretiza seleccionando un subconjunto adecuado de la misma y proporcionando un valor para cada uno de los parámetros de sistema. La especificación de sistema resultante no contiene sinónimos externos, y se denomina especificación de sistema específico.

4.3.1 *Sinónimo externo*

Gramática textual concreta

<definición de sinónimo externo> ::=
SYNONYM <nombre de sinónimo externo> <género predefinido> = EXTERNAL

<sinónimo externo> ::=
<identificador de sinónimo externo>

Una <definición de sinónimo externo> puede aparecer en cualquier lugar en que se permita una <definición de sinónimo> (véase § 5.4.1.13). Un <sinónimo externo> puede utilizarse en cualquier lugar en que se permita un <sinónimo> (véase § 5.4.2.3). Los géneros predefinidos son: boolean (booleano), character (carácter), charstring (cadena-de-caracteres), integer (entero), natural, real, PId, duration (duración) y time (tiempo).

Semántica

Un <sinónimo externo> es un <sinónimo> cuyo valor no está especificado en la definición de sistema. Esto se indica por la palabra clave EXTERNAL que se utiliza en lugar de una <expresión simple>.

Una definición de sistema genérico es una definición de sistema que contiene <sinónimo externo>s o <texto informal> en una opción de transición (véase § 4.3.4). Una definición de sistema específico se crea a partir de una definición de sistema genérico proporcionando valores para los <sinónimo externo>s y transformando <texto informal> en construcciones formales. La manera de efectuar esto, y la relación con la gramática abstracta, no forma parte de la definición del lenguaje.

4.3.2 *Expresión simple*

Gramática textual concreta

<expresión simple> ::=
<expresión fundamental>

Una <expresión simple> contendrá solamente operadores, sinónimos y literales de los géneros predefinidos.

Semántica

Una expresión simple es una *expresión-fundamental.*

4.3.3 *Definición opcional*

Gramática textual concreta

```
<definición de seleccionar> ::=
        SELECT IF (<expresión simple booleana>) <fin>
        {<definición de bloque>
            | <referencia textual de bloque>
            | <definición de canal>
            | <definición de señal>
            | <definición de lista de señales>
            | <definición de datos>
            | <definición de proceso>
            | <referencia textual de proceso>
            | <definición de temporizador>
            | <definición de ruta de señales de servicio>
            | <conexión de canal>
            | <conexión de punto extremo de canal>
            | <definición de variable>
            | <definición de visión>
            | <definición de importación>
            | <definición de procedimiento>
            | <referencia textual de procedimiento>
            | <definición de servicio>
            | <referencia textual de servicio>
            | <definición de ruta de señales>
            | <conexión de canal a ruta>
            | <conexión de ruta de señales>
            | <definición de seleccionar>
            | <definición de macro>}+
        ENDSELECT <fin>
```

Una <expresión simple booleana> no dependerá de ninguna definición dentro de la <definición de seleccionar>. Una <definición de seleccionar> sólo debe contener las definiciones que estén sintácticamente permitidas en ese lugar.

Gramática gráfica concreta

```
<área de opción> ::=
        <símbolo de opción> contains
        {SELECT IF (<expresión simple booleana>)
        {<área de bloque>
            | <área de definición de canal>
            | <área de texto de sistema>
            | <área de texto de bloque>
            | <área de texto de proceso>
            | <área de texto de procedimiento>
            | <área de texto de subestructura de bloque>
            | <área de texto de subestructura de canal>
            | <área de texto de servicio>
            | <diagrama de macro>
            | <área de proceso>
            | <área de definición de ruta de señales>
            | <área de línea de crear>
            | <área de procedimiento>
            | <área de opción>
            | <área de servicio>
            | <área de definición de ruta de señales de servicio> }+ }
```

El <símbolo de opción> es un polígono de trazo discontinuo con esquinas de trazo continuo, por ejemplo:

Un <símbolo de opción> contiene lógicamente la totalidad de cualquier símbolo gráfico unidimensional cortado por su frontera (es decir, con un punto extremo en su exterior).

La <expresión simple booleana> no dependerá de ningún área o diagrama dentro del <área de opción>.

Un <área de opción> puede aparecer en cualquier lugar, salvo dentro de un <área de gráfico de proceso>, <área de gráfico de procedimiento> o <área de gráfico de servicio>. Un <área de opción> tiene que contener solamente las áreas y diagramas que están sintácticamente permitidos en ese lugar.

Semántica

Si el valor de la <expresión simple booleana> es Falso, las construcciones contenidas en la <definición de seleccionar> y en el <símbolo de opción> no se seleccionan. En el otro caso se seleccionan las construcciones.

Modelo

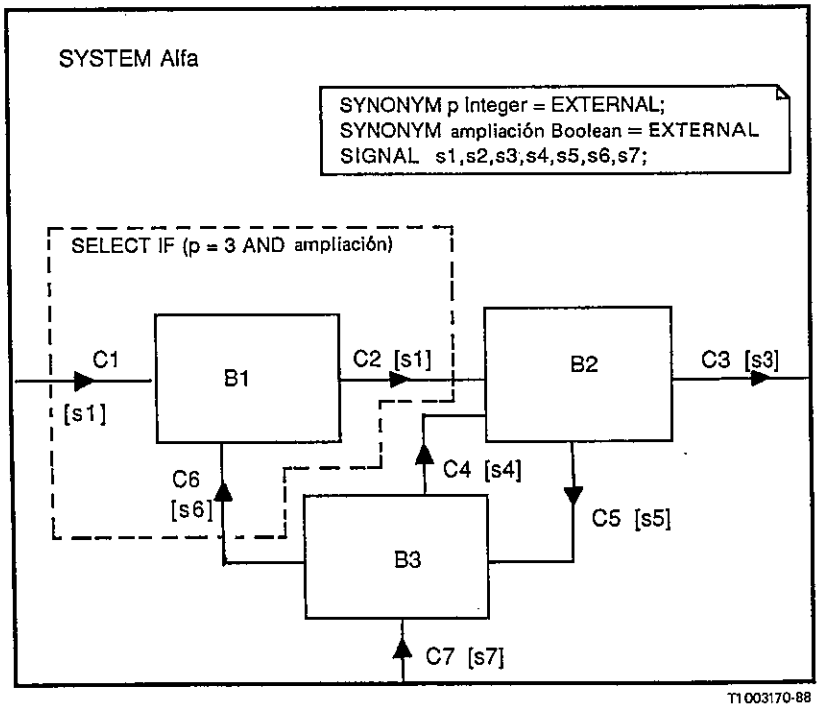
En una transformación la <definición de seleccionar> y el <área de opción> se remplazan por las construcciones seleccionadas contenidas, si existen.

Ejemplo

En el sistema Alfa hay tres bloques: B1, B2, y B3. El bloque B1 y los canales conectados a él son opcionales, y dependen de los valores de los sinónimos externos p y ampliación. En LED/PR, este ejemplo se representa como sigue.

```
SYSTEM Alfa;
    SYNONYM p Integer = EXTERNAL;
    SYNONYM ampliación Boolean = EXTERNAL;
    SIGNAL s1,s2,s3,s4,s5,s6,s7;
    SELECT IF (p = 3 AND ampliación);
     BLOCK B1 REFERENCED;
     CHANNEL C1 FROM ENV TO B1 WITH s1; ENDCHANNEL C1;
     CHANNEL C2 FROM B1 TO B2 WITH s2; ENDCHANNEL C2;
     CHANNEL C6 FROM B3 TO B1 WITH s6; ENDCHANNEL C6;
    ENDSELECT;
    CHANNEL C3 FROM B2 TO ENV WITH s3; ENDCHANNEL C3;
    CHANNEL C4 FROM B3 TO B2 WITH s4; ENDCHANNEL C4;
    CHANNEL C5 FROM B2 TO B3 WITH s5; ENDCHANNEL C5;
    CHANNEL C7 FROM ENV TO B3 WITH s7; ENDCHANNEL C7;
    BLOCK B2 REFERENCED;
    BLOCK B3 REFERENCED;
ENDSYSTEM Alfa;
```

El mismo ejemplo en sintaxis LED/GR se representa como sigue.



4.3.4 *Cadena de transición opcional*

Gramática textual concreta

<opción de transición> ::=
ALTERNATIVE <pregunta de alternativa> <fin>
{<parte respuesta> <parte otro caso>
| <parte respuesta> { <parte respuesta> } + [<parte otro caso>] }
ENDALTERNATIVE

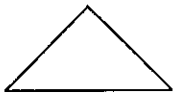
<pregunta de alternativa> ::=
<expresión simple>
| <texto informal>

Toda <expresión fundamental> en <respuesta> tiene que ser una <expresión simple>. Las <respuesta>s en una <opción de transición> tienen que ser mutuamente exclusivas. Si la <pregunta de alternativa> es una <expresión>, la *condición-de-intervalo* de las <respuesta>s tienen que ser del mismo género que el de la <pregunta de alternativa>.

Gramática gráfica concreta

<área de opción de transición> ::=
<símbolo de opción de transición> **contains** {<pregunta de alternativa>}
is followed by {<opción acceso de salida 1> {<opción acceso de salida 1>|<opción acceso de salida 2>}
{<opción de acceso de salida 1>}* }**set**

<símbolo de opción de transición> ::=



<opción acceso de salida 1> ::=
<símbolo de línea de flujo> **is associated with** <respuesta gráfica>
is followed by <área de transición>

<opción acceso de salida 2> ::=
<símbolo de línea de flujo> **is associated with** ELSE
is followed by <área de transición>

El <símbolo de línea de flujo> en <opción acceso de salida 1> y <opción acceso de salida 2> está conectado a la parte inferior del <símbolo de opción de transición>. Los <símbolo de línea de flujo>s que salen de un <símbolo de opción de transición> pueden tener un trayecto de origen común. La <respuesta gráfica> y ELSE pueden colocarse a lo largo del <símbolo de línea de flujo> asociado, o en el <símbolo de línea de flujo> interrumpido.

Las <respuesta gráfica>s en un <área de opción de transición> tienen que ser mutuamente exclusivas.

Semántica

Se seleccionan construcciones en una <opción acceso de salida 1> si la <respuesta> contiene el valor de la <pregunta de alternativa>. Si ninguna de las <respuesta>s contiene el valor de la <pregunta de alternativa>, se seleccionan las construcciones en la <opción acceso de salida 2>.

Si no existe una <opción acceso de salida 2> y no se selecciona ninguno de los trayectos salientes, la selección es inválida.

Modelo

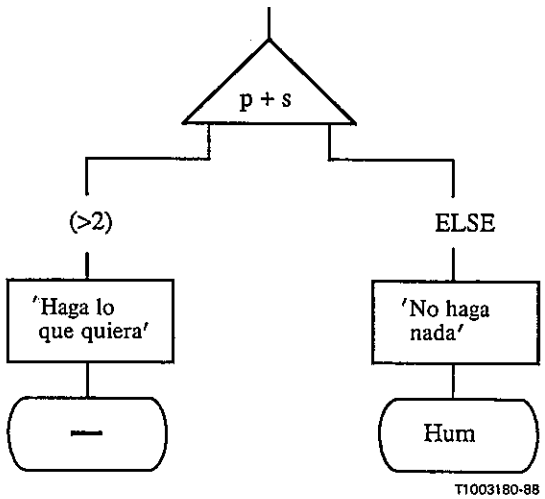
En una transformación, la <opción de transición> y el <área de opción de transición> se remplazan por las construcciones seleccionadas contenidas.

Ejemplo

A continuación se presenta un fragmento de una <definición de proceso> que contiene una <opción de transición>. *p* y *s* son sinónimos.

```
.....
ALTERNATIVE p + s;
      (>2) : TASK 'Haga lo que quiera';
          NEXTSTATE -;
      ELSE: TASK 'No haga nada';
          NEXTSTATE Hum;
ENDALTERNATIVE;
.....
```

Seguidamente se presenta el mismo ejemplo en sintaxis gráfica concreta.



4.4 *Estado asterisco*

Gramática textual concreta

<lista de estados asterisco> ::=
<asterisco> [(<nombre de estado> {,<nombre de estado>}*)]

<asterisco> ::=

*

En un <cuerpo de proceso>, <cuerpo de procedimiento> o <cuerpo de servicio>, por lo menos una <lista de estados> tiene que ser diferente de <lista de estados asterisco>. Los <nombre de estado>s en una <lista de estados asterisco> tienen que ser distintos y tienen que estar contenidos en otras <lista de estados>s del <cuerpo de proceso>, <cuerpo de procedimiento> o <cuerpo de servicio> circundante.

Los <nombre de estado>s en <la lista de estados asterisco> no podrán incluir todos los <nombre de estado> <cuerpo de proceso>, <cuerpo de procedimiento> o <cuerpo de servicio> circundante.

Gramática gráfica concreta

Un <área de estado> que contiene una <lista de estados asterisco> no podrá coincidir con un <área de estado siguiente>.

Modelo

Una <lista de estados asterisco> se transforma en una <lista de estados> que contiene todos los <nombre de estado>s del <cuerpo de proceso>, <cuerpo de servicio> o <cuerpo de procedimiento> en cuestión, salvo en lo que concierne a los <nombre de estado>s contenidos en la <lista de estados asterisco>.

4.5 *Múltiple aparición de estado*

Gramática textual concreta

Un <nombre de estado> puede aparecer en más de un <estado> de un <cuerpo de proceso>, <cuerpo de servicio> o <cuerpo de procedimiento>.

Modelo

Cuando varios <estado>s contienen el mismo <nombre de estado>, estos <estado>s se concatenan para formar un <estado> que tiene ese <nombre de estado>.

4.6 *Entrada asterisco*

Gramática textual concreta

<lista de entrada asterisco> ::=
<asterisco>

Un <estado> puede contener cuando más una <lista de entradas asterisco>. Un <estado> no contendrá a la vez <lista de entradas asterisco> y <lista de conservaciones asterisco>.

Modelo

Una <lista de entradas asterisco> se transforma en una lista de <estímulo>s que contiene el conjunto completo de señales de entrada válidas de la <definición de proceso> o <definición de servicio> circundante, salvo el caso de <identificador de señal>s de señales implícitas y de <identificador de señal>s contenidos en las otras <lista de entradas>s y <lista de conservaciones>s del <estado> y en todas las <entrada prioritaria>s de la <definición de servicio> (véase el § 4.10).

4.7 *Conservación asterisco*

Gramática textual concreta

<lista de conservaciones asterisco> ::=
<asterisco>

Un estado puede contener cuando más una <lista de conservaciones asterisco>. Un <estado> no contendrá a la vez <lista de entradas asterisco> y <lista de conservaciones asterisco>.

Modelo

Una <lista de entradas asterisco> se transforma en una lista de <estímulo>s que contiene el conjunto completo de señales de entrada válidas de la <definición de proceso> o <definición de servicio> circundante, salvo el caso de <identificador de señal>s de señales implícitas y de <identificador de señal>s contenidos en las otras <lista de entradas>s y <lista de conservaciones>s del <estado> y en todas las <entrada prioritaria>s de la <definición de servicio> (véase el § 4.10).

4.8 *Transición implícita*

Gramática textual concreta

Un <identificador de señal> contenido en el conjunto completo de señales de entrada válidas de una <definición de proceso> o <definición de servicio> puede omitirse en el conjunto de <identificador de señal>s contenido en las <lista de entradas>s, <lista de entradas prioritarias>s y la <lista de conservaciones> de un <estado>.

Modelo

Para cada <estado> hay una <parte entrada> implícita que contiene una <transición> que sólo contiene un <estado siguiente> que hace retornar al mismo <estado>.

4.9 *Estado siguiente indicado por guión*

Gramática textual concreta

<estado siguiente indicado por guión> ::=
<guión>

<guión> ::=
-

La <transición> contenida en un <arranque> no podrá conducir, directa ni indirectamente, a un <estado siguiente indicado por guión>.

Modelo

En cada <estado siguiente> de un <estado> el <estado siguiente indicado por guión> se remplaza por el <nombre de estado> del <estado>.

4.10 *Servicio*

El comportamiento de un proceso en LED básico se define por un gráfico de proceso. El concepto de servicio ofrece una alternativa al gráfico de proceso mediante un conjunto de definiciones de servicio. En muchas situaciones, las definiciones de servicio pueden reducir la complejidad global y mejorar la legibilidad de una definición de proceso. Además, cada definición de servicio puede definir un comportamiento parcial del proceso, lo que puede ser útil en algunas aplicaciones.

4.10.1 *Descomposición de servicio*

Gramática textual concreta

<descomposición de servicio> ::=
{<definición de ruta de señales de servicio>
| <conexión de ruta de señales>
| <definición de servicio>
| <definición de seleccionar>
| <referencia textual de servicio>}+

<definición de ruta de señales de servicio> ::=
SIGNALROUTE <nombre de ruta de señales de servicio>
<trayecto de ruta de señales de servicio>
[<trayecto de ruta de señales de servicio>]

<trayecto de ruta de señales de servicio> ::=
{FROM <identificador de servicio> TO <identificador de servicio>
| FROM <identificador de servicio> TO ENV
| FROM ENV TO <identificador de servicio>}
WITH <lista de señales> <fin>

<conexión de ruta de señales> ::=
CONNECT <identificador de ruta de señales>
AND <identificador de ruta de señales de servicio> {,<identificador de ruta de señales de servicio>}* <fin>

<referencia textual de servicio> ::=
SERVICE <nombre de servicio> REFERENCED <fin>

Cuando una <definición de proceso> contiene una <descomposición de servicio>, no podrá contener <definición de temporizador>s fuera de la <descomposición de servicio>.

Una <descomposición de servicio> debe contener al menos una <definición de servicio>.

Para <ruta de señales de servicio> se aplican reglas de formación correcta similares a las de <ruta de señales>.

Gramática gráfica concreta

<área de interacción de servicios> ::=
{ <área de servicio> | <área de definición de ruta de señales de servicio> }+

<área de servicio> ::=
<referencia gráfica de servicio>
| <diagrama de servicio>

<referencia gráfica de servicio> ::=
<símbolo de servicio> **contains** <nombre de servicio>

<símbolo de servicio> ::=

<área de definición de ruta de señales de servicio> ::=
<símbolo de ruta de señales>
is associated with {<nombre de ruta de señales de servicio>
[<identificador de ruta de señales>]
<área de lista de señales>
[<área de lista de señales>] }**set**
is connected to {<área de servicio>
{<área de servicio> | <símbolo de casilla>} }**set**

Cuando el <símbolo de ruta de señales> se conecta al <símbolo de casilla>, el <identificador de ruta de señales> identifica una ruta de señales externa a la cual está conectada la ruta de señales.

Semántica

La <descomposición de servicio> es una alternativa al <cuerpo de proceso>, y expresa el mismo comportamiento.

Modelo

El concepto de servicio se modela transformando la <descomposición de servicio> en conceptos primitivos. La transformación de <definición de ruta de señales de servicio>s y <conexión de ruta de señales>s no produce ningún resultado.

4.10.2 *Definición de servicio*

Gramática textual concreta

```
<definición de servicio> ::=
        SERVICE {<nombre de servicio> | <identificador de servicio>} <fin>
        [<conjunto de señales de entrada válidas>]
        {<definición de variable>
            | <definición de datos>
            | <definición de temporizador>
            | <definición de visión>
            | <definición de importación>
            | <definición de seleccionar>
            | <definición de macro>
            | <definición de procedimiento>
            | <referencia textual de procedimiento>}*
        <cuerpo de servicio>
        ENDSERVICE [{<nombre de servicio> | <identificador de servicio>}] <fin>

<cuerpo de servicio> ::=
        <cuerpo de proceso>

<entrada prioritaria> ::=
        PRIORITY INPUT <lista de entradas prioritarias> <fin> <transición>

<lista de entradas prioritarias> ::=
        <estímulo prioritario> {, <estímulo prioritario>}*

<estímulo prioritario> ::=
        <identificador de señal prioritaria [ ( [<identificador de variable>]
        {, [<identificador de variable>] }* ) ]

<salida prioritaria> ::=
        PRIORITY OUTPUT <cuerpo de salida prioritaria>

<cuerpo de salida prioritaria> ::=
        <identificador de señal prioritaria> [<parámetros efectivos>]
        {, <identificador de señal prioritaria> [<parámetros efectivos>]}*
```

Una señal es una señal de alta prioridad en un proceso únicamente si está mencionada en una <entrada prioritaria> de una <definición de servicio> en ese proceso.

Una <definición de variable> en una <definición de servicio> no podrá contener las palabras clave EXPORTED ni REVEALED.

Un <identificador de señal prioritaria> en una <salida prioritaria> no podrá estar contenido en una <parte entrada> ni en una <parte conservación>. Un <identificador de señal prioritaria> en una <entrada prioritaria> no podrá estar contenido en una <salida>.

Se aplica la misma regla sobre conjunto de señales de entrada válidas y ruta de señales de servicio enunciada en § 2.5.2 en relación con los procesos.

La <descomposición de servicio> puede contener <definición de ruta de señales de servicio>s sólo si la <definición de bloque> envolvente contiene <definición de ruta de señales>s.

Sólo una de las <definición de servicio>s en una <descomposición de servicio> está autorizada para tener un <arranque> que contenga una <cadena de transición>. Todos los demás <arranque>s deben contener únicamente <estado siguiente>.

Los conjuntos completos de señales de entrada válidas (siendo cada uno de dichos conjuntos la unión del <conjunto de señales de entrada válidas> y del conjunto de señales transportadas en <ruta de señal de servicio>s entrantes de una <definición de servicio>) de las <definición de servicio>s dentro de una <definición de proceso> deben ser disjuntos.

Una <definición de procedimiento> no debe tener <estado>s cuando la <definición de proceso> que la encierra contiene una <definición de servicio>. Las <definición de procedimiento>s visibles para más de un servicio no deben contener una construcción VIA.

El conjunto de prioridades asociadas a <señal continua>s dentro de las diversas <definición de servicio>s de una <descomposición de servicio> no deben superponerse.

Reglas similares de formación correcta son aplicables a <conexión de ruta de señales> y a <conexión de canal a ruta>.

Si la <descomposición de servicio> envolvente contiene algunas <definición de ruta de señales de servicio>s, para cada <identificador de ruta de señales> de una <salida>, debe existir una ruta de señales de servicio procedente del servicio envolvente y conectada a la ruta de señales, que pueda transportar las señales denotadas mediante los <identificador de señal>s contenidos en la <salida>.

Si una <salida> no incluye una construcción VIA, debe existir al menos un trayecto de comunicación [implícito en el propio servicio o vía rutas de señales de servicio (posiblemente implícitas), y posiblemente, rutas de señales y canales] procedente del servicio, que pueda transportar las señales denotadas mediante los <identificador de señal>s contenidas en la <salida>.

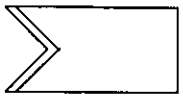
Para cada <salida prioritaria> debe existir al menos un trayecto de comunicación [implícito en el propio servicio o vía [rutas de señales de servicio (posiblemente implícitas)] procedente del servicio, que pueda transportar las señales denotadas mediante los <identificador de señal de prioridad>s contenidos en la <salida prioritaria>.

<entrada de prioritaria> sólo está permitida en un <cuerpo de servicio>. <salida prioritaria> sólo está permitida en un <cuerpo de servicio> y en <cuerpo de procedimiento>.

Gramática gráfica concreta

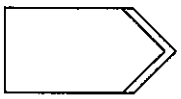
```
<diagrama de servicio> ::=
        <símbolo de casilla> contains
        {<encabezamiento de servicio>
        {    {<área de texto de servicio> }*
             {<referencia gráfica de procedimiento>}*
             {<diagrama de procedimiento>}*
             {<diagrama de macro>}*
             <área de gráfico de servicio> }set  }

<encabezamiento de servicio> ::=
        SERVICE {<nombre de servicio> | <identificador de servicio>}

<área de texto de servicio> ::=
        <símbolo de texto> contains
        { <definición de variable>
             | <definición de datos>
             | <definición de temporizador>
             | <definición de visión>
             | <definición de importación>
             | <definición de seleccionar>
             | <definición de macro> }*

<área de gráfico de servicio> ::=
        <área de gráfico de proceso>

<área de asociación de entrada prioritaria> ::=
        <símbolo de asociación de trazo continuo> is connected to <área de entrada prioritaria>

<área de entrada prioritaria> ::=
        <símbolo de entrada prioritaria> contains <lista de entradas prioritarias>
        is followed by <área de transición>
```

<símbolo de entrada prioritaria> ::=



<área de salida prioritaria> ::=
<símbolo de salida prioritaria> **contains** <cuerpo de salida prioritaria>

<símbolo de salida prioritaria> ::=



Semántica

Las propiedades de un servicio se derivan de la exigencia de que la <descomposición de servicio> que remplaza un <cuerpo de proceso> exprese el mismo comportamiento que el <cuerpo de proceso>.

Dentro de una instancia de proceso hay una instancia de servicio para cada <definición de servicio> en la <definición de proceso>. Las instancias de servicio son componentes de la instancia de proceso y no pueden ser manipuladas (creadas, direccionadas, ni abortadas) como objetos separados. Estas instancias de servicio comparten el puerto de entrada y las expresiones SELF, PARENT, OFFSPRING y SENDER de la instancia de proceso.

Una instancia de servicio es una máquina de estados finitos, pero no puede funcionar en paralelo con otras instancias de servicio de la instancia de proceso; es decir, dentro de una instancia de proceso sólo una instancia de servicio puede efectuar una transición en un instante dado.

En <cuerpo de salida prioritaria>, la construcción TO SELF está implícita. Las señales prioritarias son una clase especial de señales que tienen una prioridad superior a la de las ordinarias. Estas señales sólo pueden enviarse entre instancias de servicio dentro de la misma instancia de proceso.

A la instancia de servicio que es capaz de recibir esa señal se le da una señal de entrada desde el puerto de entrada.

Modelo

a) *Transformación de definiciones*

Las definiciones locales dentro de una <definición de servicio> se transforman al nivel de proceso remplazando cada ocurrencia de un nombre en el servicio por el mismo nombre nuevo distinto. Todas las referencias a servicios en calificadores desaparecen.
Las definiciones de visión o las definiciones de importación que contienen la misma variable de visión o importación se fusionan en una sola definición de visión o importación.

b) *Transformación de <cuerpo de servicio>s*

Un conjunto de <cuerpo de servicio>s se transforma en un <cuerpo de proceso>. Esto puede hacerse de varias maneras. Aquí se ha elegido una transformación simple, pues el objetivo principal es definir el concepto de servicio por una sintaxis concreta estricta. Por razones prácticas un <cuerpo de servicio> y un <cuerpo de proceso> se consideran como un gráfico compuesto de estados, cadenas de transición entre estados, cadenas de transición de parada y una cadena de transición de arranque. Una cadena de transición se define unívocamente por un estado de arranque, una entrada y un estado de fin.

1) *Estados*

Un estado en el gráfico de proceso resultante se identifica por una tupla de nombres. La dimensión de la tupla es el número de gráficos de servicio. Cada componente de una tupla se refiere unívocamente a uno de los gráficos de servicio originales, y el valor del componente de la tupla es uno de los nombres de estado del gráfico de servicio referido. Los nombres de estado del gráfico de proceso serán entonces el conjunto de tuplas que es posible construir utilizando estas reglas. Ejemplo:

Dado dos gráficos de servicio y sus estados

f1: <a> <b>
f2: <A> <B> <C>

el gráfico de proceso tiene los siguientes estados

<a.A> <a.B> <a.C> <b.A> <b.B> <b.C>

Normalmente, esta explosión de estados puede reducirse sustancialmente, pero esto no se trata aquí.

2) *Cadenas de transición*

Cada cadena de transición en un gráfico de servicio se copia en el gráfico de proceso en uno o más lugares. Esto se efectúa para conectar cada par de tuplas de estado que satisface las siguientes condiciones:

- Un componente de la tupla de estado de arranque se refiere al estado de arranque de la cadena de transición
- Un componente de la tupla de estado de fin se refiere al estado de fin de la cadena de transición
- Los otros valores componentes deben ser iguales para las dos tuplas de estado

Ejemplo:

En el ejemplo precedente tenemos una cadena de transición en f2 entre <B> y <C>. En el gráfico de proceso resultante, esta cadena de transición conectará <a.B> con <a.C> y <b.B> con <b.C>. Esto puede expresarse de una manera concisa (utilizando la notación taquigráfica de la sintaxis concreta):

<*.B> se transforma en <-.C>

3) *Cadenas de transición de arranque*

Si uno de los gráficos de servicio contiene una cadena de transición de arranque, esta cadena de transición se transforma en la cadena de transición de arranque del gráfico de proceso. La cadena de transición de arranque del gráfico de proceso conduce a la tupla de estado que tiene como componentes todos los nombres de estado inicial del gráfico de servicio.

4) *Cadenas de transición de parada*

Cada transición que conduce a una <parada> es copiada en el gráfico de proceso y conectada a cada tupla de estado que tiene un componente que se refiere al estado de arranque de la transición.

5) *Señales prioritarias*

Las señales prioritarias se transforman como sigue.

Cada estado del gráfico de proceso resultante se divide en dos estados. Las entradas prioritarias en el estado original se conectan al primer estado, todas las otras entradas son conectadas al segundo estado y conservadas en el primer estado. La cadena de transición conducente al estado original conduce ahora al primer estado. A esta cadena de transición se añade la siguiente cadena de acción:

- se genera un valor de testigo único y se asigna a la variable implícita SAME_TOKEN
- se envía a SELF la señal implícita X_CONT, que transporta el valor de testigo.

Una entrada para la señal implícita X_CONT se añade al primer estado, seguida de la siguiente cadena de transición:

- Una decisión compara el valor de testigo recibido con el valor de SAME_TOKEN. Si los valores son iguales, se elige un trayecto que conduce al segundo estado; en el caso contrario, una parte que retorna al primer estado.

Ejemplo

A continuación se presenta un ejemplo de una <definición de proceso> que contiene una <descomposición de servicio>, así como las <definición de servicio>s correspondientes. Este proceso tiene el mismo comportamiento que el indicado en la figura 2.9.9, en el § 2.9.

```
PROCESS Game;
    FPAR Player pid;
    SIGNAL Proberers (integer);
    DCL A integer;

    SIGNALROUTE IR1 FROM Game_handler TO ENV WITH Score,Gameid;
    SIGNALROUTE IR2 FROM Game_handler TO ENV WITH Subscr,Endsubscr;
    SIGNALROUTE IR3 FROM ENV TO Game_handler WITH Result,Endgame;
    SIGNALROUTE IR4 FROM ENV TO Bump_handler WITH Probe;
    SIGNALROUTE IR5 FROM ENV TO Bump_handler WITH Bump;
    SIGNALROUTE IR6 FROM Bump_handler TO ENV WITH Lose,Win;
    SIGNALROUTE IR7 FROM Bump_handler TO Game_handler WITH proberers;

    CONNECT R5 AND IR5;
    CONNECT R2 AND IR3,IR4;
    CONNECT R3 AND IR1,IR6;
    CONNECT R4 AND IR2;

    SERVICE Game_handler REFERENCED;
    SERVICE Bump_handler REFERENCED;

ENDPROCESS Game;
```

```
SERVICE Game_handler;

    /*El servicio controla el desarrollo de un juego con acciones para comenzar un juego, terminar un juego,
    llevar la cuenta (score) y darla*/

    DCL Count integer;
    /*Contador para llevar el score*/

    START;
        OUTPUT Subscr;
        OUTPUT Gameid TO Player;
        TASK Count:=0;
        NEXTSTATE STARTED;
    STATE STARTED;
        PRIORITY INPUT Proberers(A);
            TASK Count:=Count+A;
            NEXTSTATE_;
        INPUT Result;
            OUTPUT Score(Count) TO Player;
            NEXTSTATE_;
        INPUT Endgame;
            OUTPUT Endsubscr;
            STOP;
    ENDSTATE STARTED;
ENDSERVICE Game_handler;

SERVICE Bump_handler;

    /*El servicio tiene acciones para registrar los "bumps" y para tratar sondas del jugador (player).
    El resultado de la sonda se comunica al jugador, y también al servicio 'Game_handler' (controlador del
    juego)*/

    START;
        NEXTSTATE EVEN;
    STATE EVEN;
        INPUT Probe;
            OUTPUT Lose TO Player;
            PRIORITY OUTPUT Proberers(-1);
            NEXTSTATE_;
        INPUT Bump;
            NEXTSTATE ODD;
    ENDSTATE EVEN;
    STATE ODD;
        INPUT Bump;
            NEXTSTATE EVEN;
        INPUT Probe;
            OUTPUT Win TO Player;
            PRIORITY OUTPUT Proberers(+1);
            NEXTSTATE_;
    ENDSTATE ODD;
ENDSERVICE Bump_handler;
```

En los diagramas siguientes se presenta el mismo ejemplo en LED/GR:

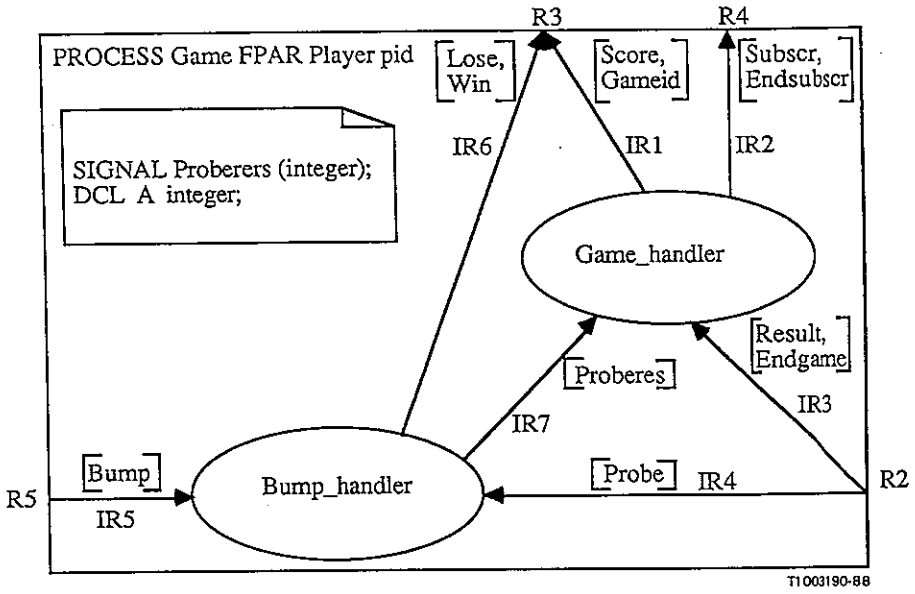


FIGURA 4.10.1

Ejemplo de un diagrama de proceso con descomposición de servicio

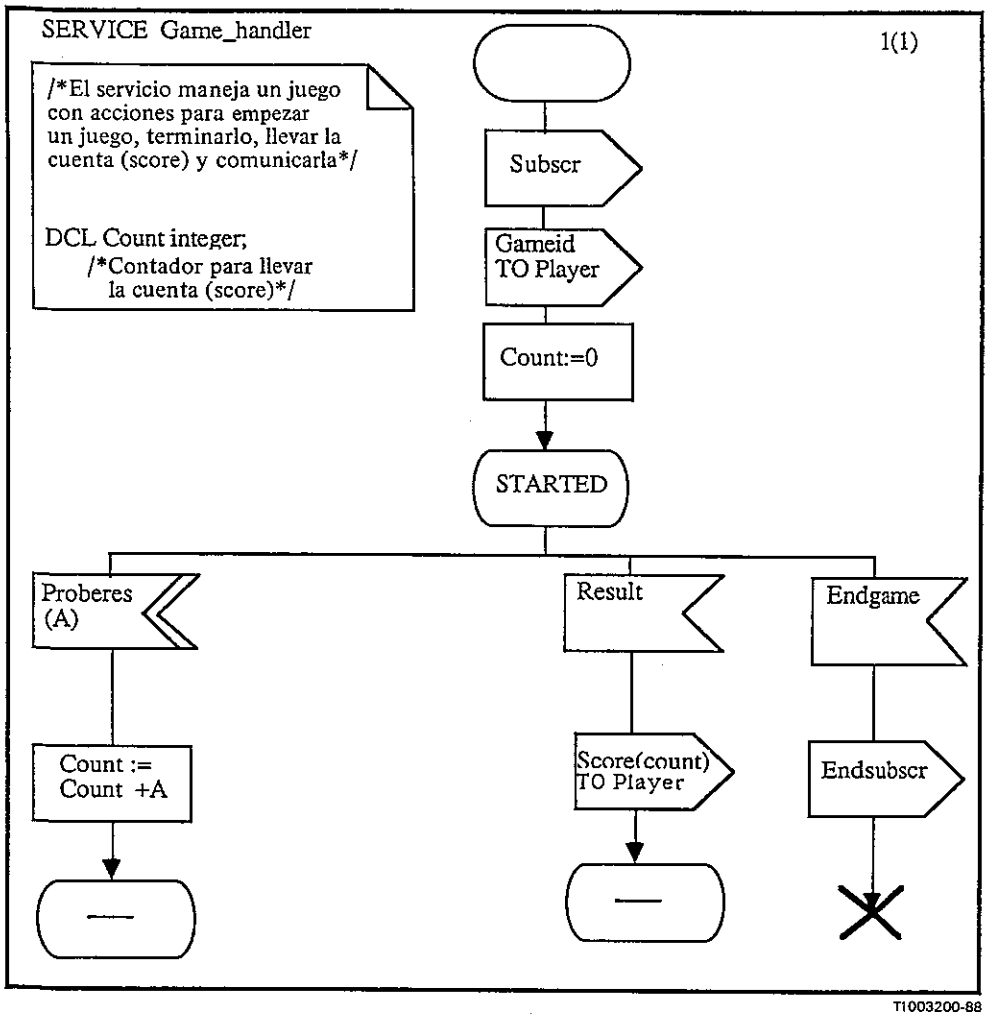


FIGURA 4.10.2

Ejemplo de un diagrama de servicio

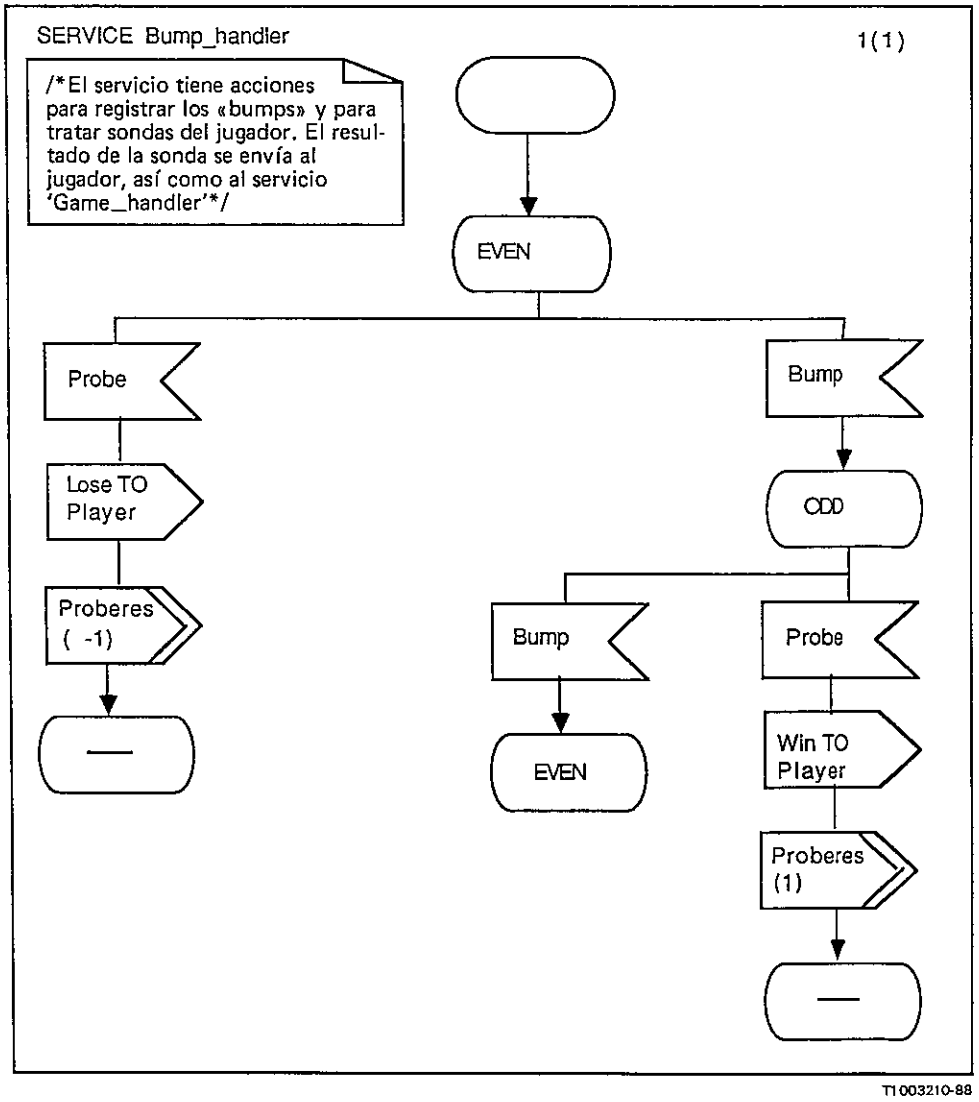


FIGURA 4.10.3

Ejemplo de un diagrama de servicio

Aplicando las reglas 1 a 4 de la transformación se obtiene el gráfico de proceso de la figura 4.10.4; este gráfico contiene señales prioritarias aún no transformadas. Simplificando de una manera obvia las transiciones que contienen señales prioritarias y usando el concepto de estado asterisco se puede obtener el mismo proceso de la figura 2.9.10 (§ 2.9). (Obsérvese que los estados EVEN y ODD corresponden respectivamente a los estados STARTED.EVEN y STARTED.ODD.)

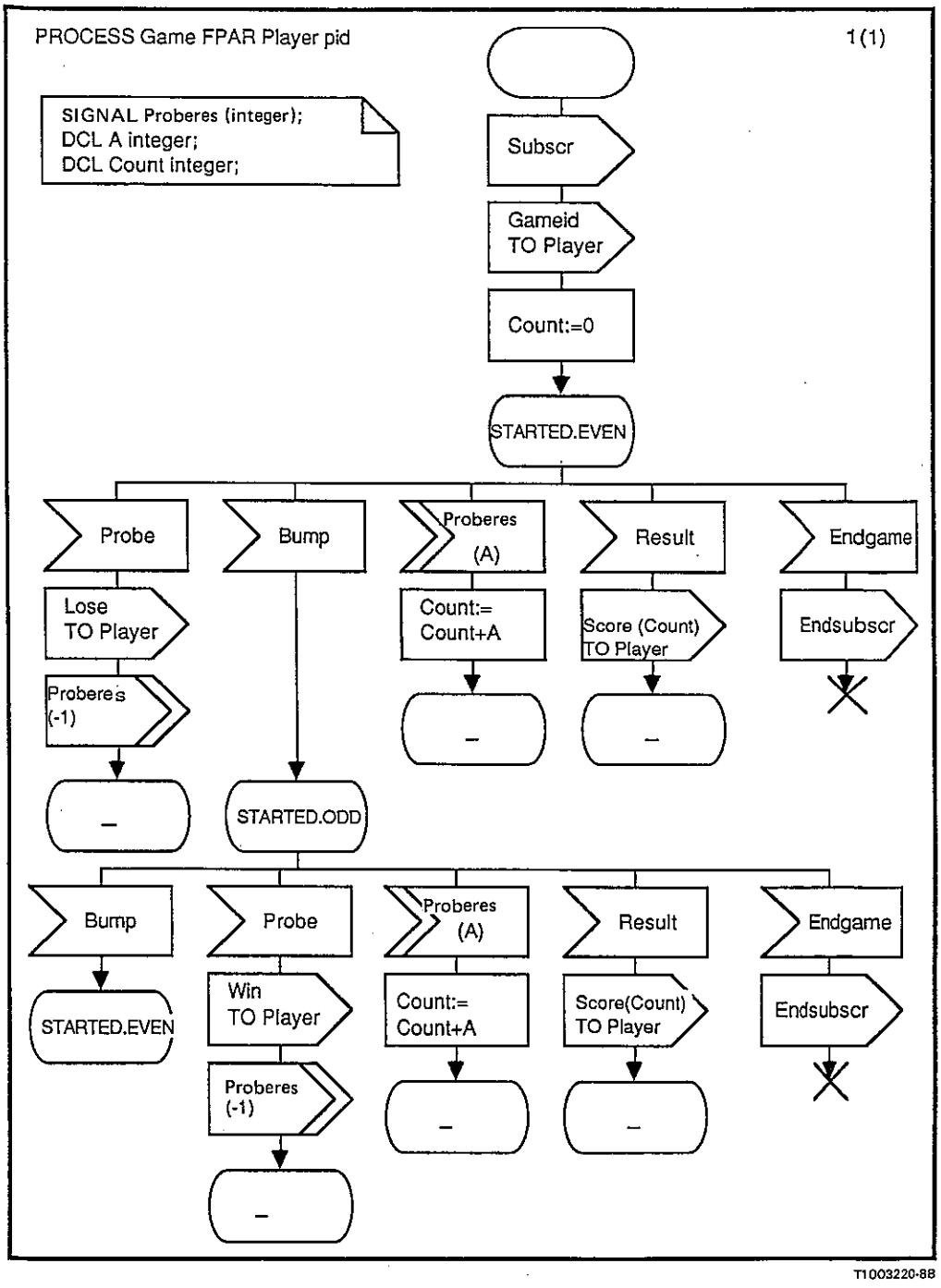


FIGURA 4.10.4

Ejemplo de transformación parcial

4.11 *Señal continua*

Al describir sistemas con LED puede surgir una situación en que un usuario desearía mostrar que una transición es causada directamente por un valor Verdadero de una expresión booleana. El modelo para conseguir esto consiste en evaluar la expresión durante el estado e iniciar la transición si la expresión da Verdadero. Una notación taquigráfica para esto es la denominada señal continua, que permite iniciar directamente una transición cuando se cumple cierta condición.

Gramática textual concreta

<señal continua> ::=
PROVIDED <expresión booleana> <fin>
[PRIORITY <nombre de literal entero> <fin>] <transición>

Los valores de los <nombre de literal entero>s en <señal continua>s de un <estado> tienen que ser distintos. La construcción PRIORITY sólo puede omitirse si el <estado> contiene exactamente una <señal continua>.

Gramática gráfica concreta

<área de asociación de señal continua> ::=
<símbolo de asociación de trazo continuo> **is connected to** <área de señal continua>

<área de señal continua> ::=
<símbolo de condición habilitante>
contains {<expresión booleana> [[<fin>] PRIORITY <nombre de literal entero>]}
is followed by <área de transición>

Semántica

La <expresión booleana> de la <señal continua> se evalúa al entrar en el estado al cual está asociada, y durante la espera en ese estado, siempre que no haya <estímulos> de una <lista de entradas> asociada aplicados en el puerto de entrada. Si el valor de la <expresión booleana> es Verdadero, se inicia la transición. Cuando el valor de la <expresión booleana> es Verdadero en más de una <señal continua>, la transición a iniciar es determinada por la <señal continua> que tiene la prioridad más elevada, es decir, el valor más bajo para <nombre de literal entero>.

Modelo

El estado con el nombre nombre_de_estado que contiene <señal continua>s se transforma en lo siguiente. Esta transformación requiere dos variables implícitas n y nuevon. La variable n se inicializa a 0. Se requiere además una señal implícita emptyQ que transporta un valor entero.

1) Todos los <estado siguiente>s que mencionan el nombre_de_estado se reemplazan por JOIN 1;

2) Se inserta la siguiente transmisión:

```
1:  TASK n:=n+1;
    OUTPUT emptyQ (n) TO SELF;
    NEXTSTATE nombre_de_estado;
```

3) Se añade la siguiente <parte entrada> al <estado> nombre_de_estado:

```
    INPUT emptyQ (nuevon);
```

y una <decisión> que contiene la <pregunta>

```
    (nuevon=n)
```

4a) La <parte respuesta> falsa contiene

```
    NEXTSTATE nombre_de_estado;
```

4b) La <parte respuesta> verdadera contiene una secuencia de <decisión>s que corresponde a las <señal continua>s en orden de prioridad (una prioridad más elevada se indica por un valor menor del <nombre de literal entero>).
La <parte respuesta> falsa contiene la <decisión> siguiente, salvo la última <decisión> para la cual esta <parte respuesta> contiene: JOIN 1;
Cada <parte respuesta> verdadera de estas <decisión>s conduce a la transición de la <señal continua> correspondiente.

Ejemplo

Véase § 4.12

4.12 *Condición habilitante*

En LED, la recepción de una señal en un estado inicia inmediatamente una transmisión. El concepto de condición habilitante hace posible imponer una condición adicional para el inicio de una transición.

Gramática textual concreta

<condición habilitante> ::=
PROVIDED <expresión booleana> <fin>

Gramática gráfica concreta

<área de condición habilitante> ::=
<símbolo de condición habilitante> **contains** <expresión booleana>

<símbolo de condición habilitante> ::=

Semántica

La <expresión booleana> en la <condición habilitante> se evalúa antes de pasar al estado en cuestión, y en cualquier momento en que se reentra en el mismo estado por la llegada de un <estímulo>. En caso de más de una condición habilitante, éstas se evalúan secuencialmente siguiendo un orden no determinista, antes de acceder al estado. El modelo de transformación garantiza una reevaluación repetida de la expresión enviando <estímulo>s adicionales a través del puerto de entrada. Una señal denotada en la <lista de entradas> que precede a la <condición habilitante> puede comenzar una transición solamente si el valor de la <expresión booleana> correspondiente es Verdadero. En cambio, si este valor es Falso, la señal se conserva.

Modelo

El estado con el nombre nombre_de_estado que contiene <condición habilitante>s se transforma en lo siguiente. Esta transformación requiere dos variables implícitas n y nuevon. La variable n se inicializa a cero. Se requiere también una señal implícita emptyQ que transporta un valor entero.

1) Todos los <estado siguiente>s que mencionan el nombre_de_estado se reemplazan por JOIN 1;

2) Se inserta la siguiente transición:

   1: TASK n:=n+1;
   OUTPUT emptyQ (n) TO SELF;

   Se añaden jerárquicamente, siguiendo un orden no determinista, cierto número de decisiones, cada una de las cuales contiene sólo una <expresión booleana> correspondiente a alguna <condición habilitante> vinculada al estado, de modo que sea posible evaluar todas las combinaciones de valores verdaderos, para todas las condiciones habilitantes vinculadas al estado.

   Cada una de esas combinaciones conduce a un estado nuevo distinto.

3) Cada uno de estos nuevos estados tiene un conjunto de <parte entrada>s que consisten en una copia de estas <parte entrada>s del estado sin condiciones habilitantes más las <parte entrada>s para las cuales las <expresión booleana>s de la <condición habilitante> dieron un valor Verdadero para este estado.

   Los <estímulo>s para las <parte entrada>s restantes constituyen la <lista de conservaciones> para una nueva <parte conservación> asociada a este estado. Las <parte conservación>s del estado original se copian también para este nuevo estado.

4) A cada uno de los nuevos estados se añade:

   INPUT emptyQ (nuevon);

   Una <decisión> que contiene la <pregunta> (nuevon=n);
   La <parte respuesta> falsa contiene un <estado siguiente> que retorna a este mismo nuevo estado.

5) La <parte respuesta> verdadera contiene un JOIN 1;

6) Si se utilizan <señal continua>s y <condición habilitante>s en el mismo <estado>, las evaluaciones de las <expresión booleana>s a partir de las <señal continua>s se efectúan reemplazando el paso 5 del modelo para <condición habilitante> por el paso 4b del modelo para <señal continua>.

Ejemplo

A continuación se presenta un ejemplo que ilustra la transformación de señal continua y condición habilitante que aparecen en un estado.

Obsérvese en el ejemplo que el conector ec se ha introducido por razones de conveniencia y no forma parte del modelo de transformación.

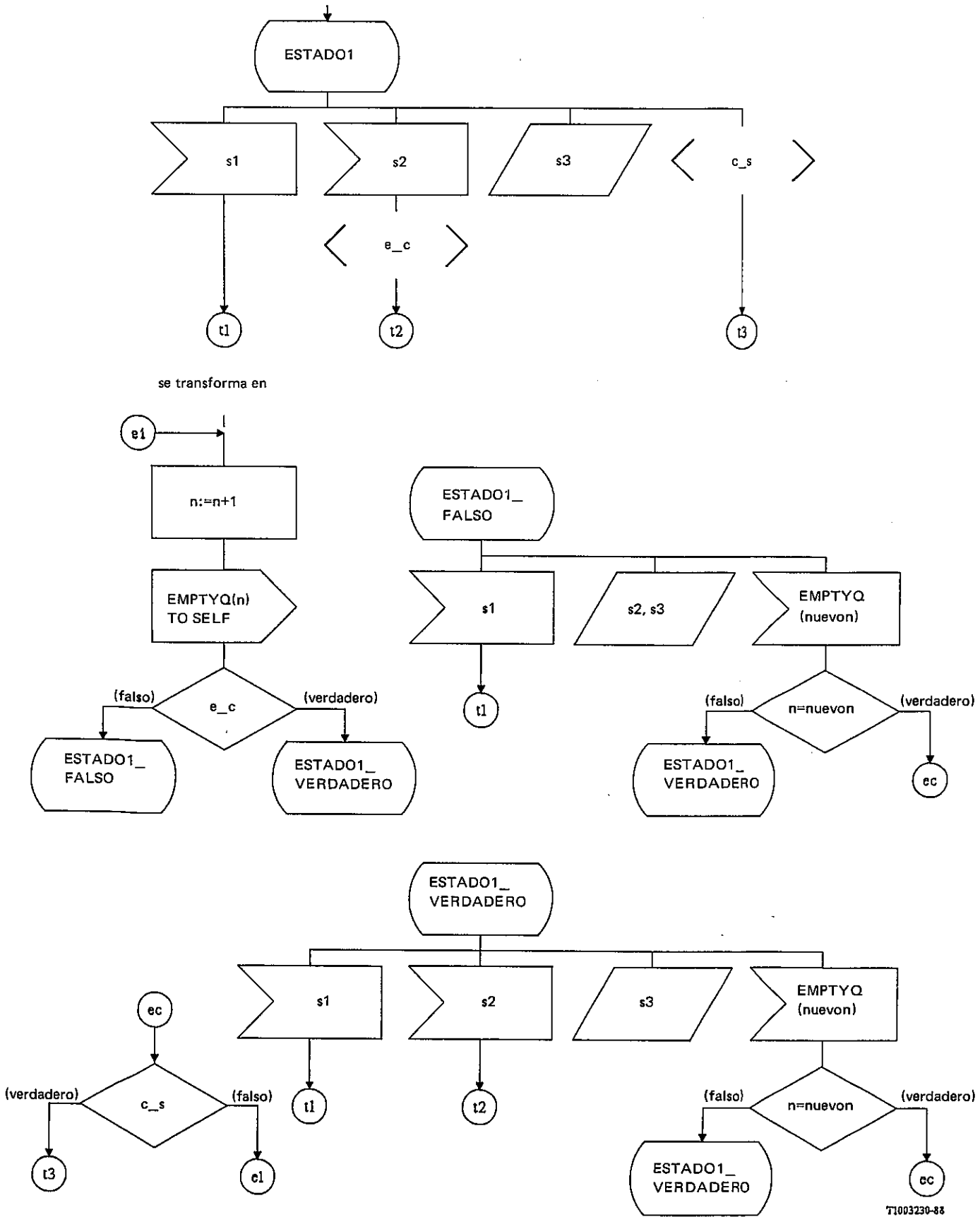


FIGURA 4.12.1

Transformación de señal continua y condición habilitante en el mismo estado

4.13 *Valor importado y exportado*

En LED una variable siempre pertenece a una instancia de proceso, para la cual es local. Normalmente la variable sólo es visible para la instancia de proceso que la posee, aunque puede ser declarada como un valor compartido (véase § 2), lo que permite a otras instancias de proceso en el mismo bloque tener acceso al valor de la variable. Si una instancia de proceso en otro bloque necesita acceder al valor de una variable, es necesario un intercambio de señales con la instancia de proceso que posee la variable.

Esto puede conseguirse mediante la siguiente notación taquigráfica denominada valor importado y exportado. La notación taquigráfica puede utilizarse también para exportar valores a otras instancias de proceso dentro del mismo bloque, en cuyo caso proporciona una alternativa al uso de valores compartidos.

Gramática textual concreta

<definición de importación> ::=
IMPORTED <nombre de importación> {, <nombre de importación> }* <género>
{, <nombre de importación> {, <nombre de importación> }* <género>}* <fin>

<expresión de importación> ::=
IMPORT (<identificador de importación> [, <expresión de pid>])

<exportación> ::=
EXPORT (<identificador de variable> {, <identificador de variable> }*)

Gramática gráfica concreta

<área de exportación> ::=
<símbolo de tarea> **contains** < exportación>

Semántica

La instancia de proceso que posee una variable cuyos valores son exportados a otras instancias de proceso se denomina el exportador de la variable. Otras instancias de proceso que utilizan estos valores se denominan importadores de la variable. La variable se denomina variable exportada.

Una instancia de proceso puede ser importador y exportador a la vez, pero no puede importar desde, ni exportar hacia, el entorno.

a) *Operación de exportación*

Las variables exportadas tienen la palabra clave EXPORTED en sus <definición de variable>s, y tienen una copia implícita para ser utilizada en operaciones de importación.

Una operación de exportación es la ejecución de una <exportación> por la cual un exportador revela el valor vigente de una variable exportada. Una operación de exportación causa el almacenamiento del valor vigente de la variable exportada en su copia implícita.

b) *Operación de importación*

Para cada <definición de importación> en un importador hay un conjunto de variables implícitas, todas las cuales tienen el nombre y el género dado en la <definición de importación>. Estas variables implícitas se utilizan para el almacenamiento de valores importados.

Una operación de importación es la ejecución de una <expresión de importación> por la cual un importador accede al valor de una variable exportada. El valor se almacena en una variable implícita denotada por el <identificador de importación> en la <expresión de importación>. El exportador que contiene la variable exportada se especifica por la <expresión de pid> en la <expresión de importación>. Si no se especifica ninguna <expresión de PId>, debe haber sólo una instancia que exporte esa variable. La asociación entre la variable exportada en el exportador y la variable implícita en el importador se especifica haciendo que el <identificador> sea el mismo en la <exportación> y en la <expresión de importación>. Además, la variable exportada y la variable implícita tienen que tener el mismo género.

Modelo

Una operación de importación se modela mediante intercambio de señales. Estas señales son implícitas y se transportan por canales y rutas de señales implícitos. El importador envía una señal al exportador, y espera la contestación. En respuesta a esta señal el exportador devuelve al importador una señal con el valor contenido en la copia implícita de la variable exportada.

Si a la variable de exportación se vincula una asignación por omisión, o si la variable de exportación es iniciada cuando es definida, se inicia también la copia implícita, con el mismo valor que la variable de exportación.

Hay dos <definición de señal>s implícitas para cada combinación de <nombre de importación> y <género> contenida en todas las <definición de importación>s en una definición de sistema. Los <nombre de señal>s en estas <definición de señal>s se denotan por *xtQUERY* respectivamente *xtREPLY*, donde *x* denota un <nombre de importación> y *t* denota un <género>. La copia implícita de la variable exportada se denota por *imcx*.

a) *Importador*

La <expresión de importación> 'IMPORT (x, pidexp)' se transforma en lo siguiente:

```
OUTPUT xtQUERY TO pidexp;
Esperar en el estado xtWAIT, conservando todas las otras señales;
INPUT xtREPLY (x);
Reemplazar la <expresión de importación> por x, (el <nombre> de la variable implícita);
```

Si una <expresión de importación> ocurre más de una vez en una <expresión>, se utiliza una variable implícita distinta, con el mismo <nombre>, para cada ocurrencia.

b) *Exportador*

A todos los <estado>s del exportador, incluyendo los estados implícitos, se añade la <parte entrada> siguiente:

```
INPUT xtQUERY;
OUTPUT xtREPLY (imcx) TO SENDER;
/* estado siguiente es el mismo*/
```

La <exportación> 'EXPORT (x)' se transforma en lo siguiente:

```
TASK imcx := x;
```

5 Datos en LED

5.1 *Introducción*

Esta introducción presenta un bosquejo del modelo formal utilizado para definir tipos de datos e información sobre la forma en que está estructurado el resto de § 5.

En un lenguaje de especificación es esencial permitir que los tipos de datos sean descritos formalmente en términos de su comportamiento, más bien que componiéndolos a partir de primitivas proporcionadas, como en algunos lenguajes de programación. Este último método implica invariablemente una realización particular del tipo de datos, y por tanto restringe la libertad de que dispone el realizador para elegir representaciones apropiadas del tipo de datos. El método del tipo abstracto de datos permite cualquier realización a condición de que sea factible y correcta con respecto a la especificación.

5.1.1 *Abstracción en tipos de datos*

Todos los datos utilizados en LED se basan en tipos abstractos de datos que se definen en términos de sus propiedades abstractas más bien que en términos de alguna realización concreta. Ejemplos de definición de tipos abstractos de datos se presentan en § 5.6, que define las facilidades predefinidas de datos del lenguaje.

Aunque todos estos tipos de datos son abstractos, y las facilidades predefinidas de datos pueden ser contraordenadas por el usuario, LED intenta proporcionar un conjunto de facilidades predefinidas de datos que son familiares tanto en su comportamiento como en su sintaxis. Están predefinidos:

a) Booleano (Boolean)

b) Carácter (Character)

c) Cadena (String)

d) Cadena-de-caracteres (Charstring)

e) Entero (Integer)

f) Natural

g) Real

h) Matriz (Array)

i) Conjuntista (Powerset)

j) PId

k) Duración (Duration)

l) Tiempo (Time).

El concepto de género estructurado (STRUCT) puede utilizarse para formar objetos compuestos.

5.1.2 *Bosquejo de formalismos utilizados para modelar datos*

Los datos se modelan por un álgebra inicial. El álgebra tiene géneros designados y un conjunto de operadores que establecen relaciones de correspondencia entre los géneros. Cada género es la colección de todos los valores posibles que pueden ser generados por el conjunto conexo de operadores. Cada valor puede denotarse por al menos una expresión en el lenguaje que contiene solamente literales y operadores (salvo el caso especial de los valores de PId). Los literales constituyen un caso especial de operadores sin argumentos.

Los géneros y operadores, junto con el comportamiento (especificado por reglas algebraicas) del tipo de datos, forman las propiedades del tipo de datos. Un tipo de datos es introducido en cierto número de definiciones parciales de tipo, cada una de las cuales define un género y operadores, así como reglas algebraicas asociadas con ese género.

La palabra clave NEWTYPE introduce una definición parcial de tipo que define un género nuevo distinto. Se puede crear un género con propiedades heredadas de otro género, pero con identificadores diferentes para el género y los operadores.

La introducción de un sintipo nomina un subconjunto de los valores de un género ya existente.

Un generador es una descripción NEWTYPE incompleta: antes de que asuma la condición de género, debe ser instanciado proporcionando la información que falta.

Algunos operadores tienen una relación de correspondencia con el género, produciendo así valores (posiblemente nuevos) del género. Otros operadores dan significado al género mediante una relación de correspondencia con otros géneros definidos. Muchos operadores tienen una relación de correspondencia con el género booleano a partir de otros géneros, pero está estrictamente prohibido que estos operadores amplíen el género booleano.

En LED, una función es conocida como un operador pasivo y no puede producir efecto sobre los valores asociados con variables dadas como parámetros. LED define también la asignación, que puede cambiar los valores asociados con variables.

5.1.3 *Terminología*

La terminología utilizada en § 5 o el modelo de datos se eligen de modo que estén en armonía con los trabajos publicados sobre álgebras iniciales. En particular, «tipo de datos», se utiliza para hacer referencia a una colección de géneros más una colección de operadores asociados con estos géneros y la definición de las propiedades de estos géneros y operadores por ecuaciones algebraicas. Un «género» es un conjunto de valores con características comunes. Un «operador» es una relación entre géneros. Una «ecuación» es una definición de equivalencia entre términos de un género. Un valor es un conjunto de términos equivalentes. Un «axioma» es una ecuación que define un valor booleano como equivalente a Verdadero. Sin embargo, «axiomas» se utiliza como un término que designa «axioma»(s) o «ecuación»(es), y una «ecuación» puede ser un «axioma».

5.1.4 *División de texto en datos*

El modelo de álgebra inicial utilizado para datos en LED se describe en una forma que permite definir la mayor parte de los conceptos de datos en términos de un núcleo (kernel) de datos del lenguaje abstracto de datos LED.

El texto de § 5 se divide en esta introducción (§ 5.1), el lenguaje núcleo de datos (§ 5.2), el modelo de álgebra inicial (§ 5.3), el uso pasivo de datos (§ 5.4), el uso activo de datos (§ 5.5) y los datos predefinidos (§ 5.6).

El lenguaje núcleo de datos define la parte de datos en LED que corresponde directamente con el método de álgebra inicial subyacente.

El texto sobre álgebra inicial da una introducción más detallada a la base matemática de este método. Una formulación matemática más precisa se presenta en apéndice I.

El uso pasivo de LED incluye las características implícitas y taquigráficas de datos LED que permiten su uso para la definición de tipos abstractos de datos. Incluye también la interpretación de expresiones que no contienen valores asignados a variables. Estas expresiones «pasivas» corresponden a un uso funcional del lenguaje.

El uso activo de datos amplía el lenguaje de modo que incluye asignación. Esto a su vez incluye asignación a, uso de, e inicialización de variables. Cuando LED se utiliza para asignar valores a variables o acceder a valores de variables, se dice que se utiliza activamente. La diferencia entre expresiones activas y pasivas es que el valor de una expresión pasiva es independiente del momento en que se efectúa su interpretación, en tanto que la interpretación de una expresión activa puede dar valores diferentes según los valores asociados con las variables o el estado del sistema en cada instante.

La sección final trata de los datos predefinidos.

5.2 *El lenguaje núcleo de datos*

El núcleo de datos puede utilizarse para definir tipos abstractos de datos.

Construcciones más convenientes para definir tipos de datos pueden definirse en términos de construcciones definidas para el núcleo de datos, salvo donde se necesiten los conceptos de asignación a una variable. (Los conceptos de errores y sintipos podrían definirse en términos del núcleo, pero en § 5.4.1.7 y § 5.4.1.9 se utilizan definiciones alternativas, más concisas.)

5.2.1 *Definiciones de tipos de datos*

En un punto cualquiera de una especificación LED hay una definición aplicable de tipo de datos. La definición de tipo de datos define la validez de expresiones y la relación entre expresiones. La definición introduce operadores y conjuntos de valores (géneros).

No hay una correspondencia simple entre la sintaxis concreta y la abstracta para definiciones de tipos de datos pues la sintaxis concreta introduce incrementalmente la definición de tipo de datos con énfasis en los géneros (véase también § 5.3).

Las definiciones en la sintaxis concreta son a menudo independientes y no pueden dividirse en unidades de ámbito diferentes. Por ejemplo:

```
NEWTYPE even LITERALS 0;
    OPERATORS  plusee       : even, even   -> even;
               plosoo       : odd, odd     -> even;
    AXIOMS     plusee(a,0))       ==a;
               plusee(a,b)        ==plusee(b,a);
               plusoo(a,b)        ==plusoo(b,a);
    ENDNEWTYPE even COMMENT '"números" pares con plus-depende de impar';
NEWTYPE odd LITERALS 1;
    OPERATORS  plusoe       : odd, even    -> odd;
               pluseo       : even, odd    -> odd;
    AXIOMS     plusoe(a,0)        ==a;
               pluseo(a,b)        ==plusoe(b,a);
    ENDNEWTYPE odd; /*«números» impares con plus-depende de par*/
```

Cada definición de tipo de datos es una definición completa. No hay referencias a géneros u operadores que no estén incluidos en la definición de tipo de datos que se aplica en un punto dado. Asimismo, una definición de tipo de datos no podrá invalidar la semántica de una definición de tipo de datos en la unidad de ámbito inmediatamente circundante. Un tipo de datos en una unidad de ámbito encerrada sólo enriquece operadores de géneros definidos en la unidad de ámbito exterior. Un valor de un género definido en una unidad de ámbito puede utilizarse libremente y pasarse entre unidades de ámbito jerárquicamente inferiores o a partir de estas unidades. Puesto que los datos predefinidos están definidos a nivel de sistema, los géneros predefinidos (por ejemplo booleano y entero) pueden utilizarse libremente en todo el sistema.

Gramática abstracta

Definición-de-tipo-de-datos	::	*Nombre-de-tipo* *Unión-de-tipo* *Géneros* *Signatura*-**set** *Ecuaciones*
Unión-de-tipo	=	*Identificador-de-tipo*-**set**
Identificador-de-tipo	=	*Identificador*
Géneros	=	*Nombre-de-género*-**set**
Nombre-de-tipo	=	*Nombre*
Nombre-de-género	=	*Nombre*
Ecuaciones	=	*Ecuación*-**set**

Dentro de una *definición-de-tipo-de-datos*, para cada *género* tiene que haber al menos una *signatura* con un *resultado* (véase § 5.2.2) que es el mismo que el *género*.

Una *definición-de-tipo-de-datos* no agregará nuevos valores a ningún *género* del tipo de datos identificado por la *unión-de-tipo*.

Si un *término* (véase § 5.2.3) es inequivalente a otro *término* de acuerdo con el tipo de datos identificado por la *unión-de-tipo* de una *definición-de-tipo-de-datos*, estos *términos* no serán definidos como equivalentes por la *definición-de-tipo-de-datos*.

Además, los dos *términos* booleanos Verdadero y Falso no pueden estar definidos (directa ni indirectamente) como equivalentes (véase § 5.4.3.1). Tampoco está permitido reducir el número de valores para el género predefinido PId.

Nota – La sintaxis abstracta permite más de una identidad de tipo para una *unión-de-tipo*, a fin de asegurar la armonía con la clase más general de álgebras utilizadas para el modelo subyacente – en LED sólo se hace referencia a un tipo, porque en la sintaxis concreta el tipo de datos visible etá implícitamente definido por la <clase de unidad de ámbito> circundante; por tanto <unión de tipo> sólo está referenciada en la sintaxis abstracta y es, bien el *identificador-de-tipo* de la unidad de ámbito circundante, bien, en el caso de una <definición de sistema>, un conjunto vacío.

Gramática textual concreta

<definición parcial de tipo> ::=
NEWTYPE <nombre de género> [<propiedades ampliadas>] <expresión de propiedades>
ENDNEWTYPE [<nombre de género>]

<expresión de propiedades> ::=
<operadores> [AXIOMS <axiomas>] [<correspondencia de literales>] [<asignación por defecto>]

Las <propiedades ampliadas>, <correspondencia de literales> y <asignación por defecto> facultativas no forman parte del núcleo de datos y se definen en § 5.4.1, § 5.4.1.15 y § 5.5.3.3 respectivamente.

La *definición-de-tipo-de-datos* se representa por la colección de todas las <definición parcial de tipo>s en la <clase de unidad de ámbito> vigente combinada con la *definición-de-tipo-de-datos* identificada por la *unión-de-tipo* de la <clase de unidad de ámbito> circundante. El nombre de tipo de una <definición de tipo de datos> es implícito y no tiene una representación en sintaxis concreta. El *identificador-de-tipo* de una *unión-de-tipo* es, implícitamente, la identidad de la *definición-de-tipo-de-datos* de la unidad de ámbito circundante.

Cada una de las siguientes <clase de unidad de ámbito>s (véase § 2.2.2) representa un ítem en la sintaxis abstracta que contiene una *definición-de-tipo-de-datos*: <definición de sistema>, <definición de bloque>, <definición de proceso>, <definición de procedimiento>, <definición de subestructura de canal> o <definición de subestructura de bloque> o los diagramas correspondientes en sintaxis gráfica. La <definición parcial de tipo> en una <definición de servicio> representa parte de la *definición-de-tipo-de-datos* en la <definición de proceso> circundante de la <definición de servicio> (véase § 4.10).

Para una <clase de unidad de ámbito>, los *géneros* se representan por el conjunto de <nombre de género>s introducido por el conjunto de <definición parcial de tipo>s de la <clase de unidad de ámbito>.

Para una <clase de unidad de ámbito>, el conjunto *signatura* y las *ecuaciones* se representan por las <expresión de propiedades>s de las <definición parcial de tipo>s de la <clase de unidad de ámbito>.

Los <operadores> de una <expresión de propiedades> representan parte del conjunto *signatura* en la sintaxis abstracta. El conjunto *signatura* completo es la unión de los conjuntos *signatura* definidos por las <definición parcial de tipo>s en la <clase de unidad de ámbito>.

Los <axiomas> de una <expresión de propiedades> representan parte del conjunto *ecuación* en la sintaxis abstracta. Las *ecuaciones* son la unión de los conjuntos *ecuación* definidos por las <definición parcial de tipo>s en la <clase de unidad de ámbito>.

Los géneros de datos predefinidos tienen sus <definición parcial de tipo>s implícitas en el nivel de sistema.

Si un <nombre de género> se da después de la palabra clave ENDNEWTYPE, debe ser el mismo que el <nombre de género> dado después de la palabra clave NEWTYPE.

Semántica

La definición de tipo de datos define un tipo de datos. Un tipo de datos tiene un conjunto de propiedades de tipo, es decir: un conjunto de géneros, un conjunto de operadores y un conjunto de ecuaciones.

Las propiedades de tipos de datos se definen en la sintaxis concreta mediante definiciones parciales de tipo. Una definición parcial de tipo no introduce todas las propiedades de un tipo de datos sino, solamente, define parcialmente algunas de las propiedades relacionadas con el género introducido en la definición parcial de tipo. Las propiedades completas de un tipo de datos se hallan examinando la combinación de todas las definiciones parciales de tipo que se aplican dentro de la unidad de ámbito que contiene la definición de tipo de datos.

Un género es un conjunto de valores de datos. Dos géneros diferentes no tienen valores en común.

La definición de tipo de datos se forma a partir de la definición de tipo de datos de la unidad de ámbito que define la unidad de ámbito vigente tomada conjuntamente con los géneros, operadores y ecuaciones definidos en la unidad de ámbito vigente. La definición de sistema contiene la definición de los géneros de datos predefinidos.

Salvo dentro de una <definición parcial de tipo>, un <refinamiento de señal> o una <definición de servicio>, la definición de tipo de datos que se aplica en cualquier punto es el tipo de datos definido para la unidad de ámbito que circunda inmediatamente ese punto. Dentro de una <definición parcial de tipo> o un <refinamiento de señal>, la definición de tipo de datos que se aplica es la definición de tipo de datos de la unidad de ámbito que circunda a la <definición parcial de tipo> o <refinamiento de señal> respectivamente. Dentro de una <definición de servicio> es la *definición-de-tipo-de-datos* de la <definición de proceso> circundante de la <definición de servicio> la que se aplica (véase § 4.10).

El conjunto de géneros de un tipo de datos es el conjunto de géneros introducido en la unidad de ámbito vigente más el conjunto de géneros del tipo de datos identificado por la unión de tipo. El conjunto de operadores de un tipo de datos es el conjunto de operadores introducidos en la unidad de ámbito vigente más el conjunto de operadores del tipo de datos identificado por la unión de tipo. El conjunto de ecuaciones de un tipo de datos es el conjunto de ecuaciones introducido en la unidad de ámbito vigente más el conjunto de ecuaciones del tipo de datos identificado por la unión de tipo.

Cada género introducido en una definición de tipo de datos tiene un identificador que es el nombre introducido por una definición parcial de tipo en la unidad de ámbito calificada por el identificador de la unidad de ámbito.

Un tipo de datos tiene un identificador que es el nombre de tipo único en la sintaxis abstracta calificado por la identidad de la unidad de ámbito. En la sintaxis concreta no hay nombre para un tipo de datos.

Ejemplo

```
NEWTYPE teléfono
        /*los operadores y la construcción de valores se definen en otro lugar*/
ENDNEWTYPE teléfono;
```

5.2.2 *Literales y operadores parametrizados*

Gramática abstracta

Signatura	=	*Signatura-de-literal* | *Signatura-de-operador*
Signatura-de-literal	::	*Nombre-de-operador-literal* *Resultado*
Signatura-de-operador	::	*Nombre-de-operador* *Lista-de-argumentos* *Resultado*
Lista-de-argumentos	=	*Identificador-de-referencia-de-género* $^+$
Resultado	=	*Identificador-de-referencia-de-género*
Identificador-de-referencia-de-género	=	*Identificador-de-género* | *Identificador-de-sintipo*

Nombre-de-operador-literal = *Nombre*

Nombre-de-operador = *Nombre*

Identificador-de-género = *Identificador*

Los sintipos e *identificadores de sintipo* no forman parte del núcleo (véase § 5.4.1.9).

Gramática textual concreta

<operadores> ::=
[<lista de literales>] [<lista de operadores>]

<lista de literales> ::=
LITERALS <signatura de literal> {, <signatura de literal> }* [<fin>]

<signatura de literal> ::=
<nombre de operador literal>
| <nombre literal ampliado>

<lista de operadores> ::=
OPERATORS
<signatura de operador> { <fin> <signatura de operador> }* [<fin>]

<signatura de operador> ::=
<nombre de operador> : <lista de argumentos> -> <resultado>
| <ordenación>

<nombre de operador> ::=
<nombre de operador>
| <nombre de operador ampliado>

<lista de argumentos> ::=
<género de argumento> {, <género de argumento> }*

<género de argumento> ::=
<género ampliado>

<resultado> ::=
<género ampliado>

<género ampliado> ::=
<género>
| <género de generador>

<género> ::=
<identificador de género>
| <sintipo>

Las alternativas <nombre de operador ampliado>, <nombre de literal ampliado>, <ordenación>, <género de generador> y <sintipo> no forman parte del núcleo de datos y se definen en § 5.4.1, § 5.4.1, § 5.4.1.8, § 5.4.1.12.1, § 5.4.1.12.1 y § 5.4.1.9 respectivamente.

Los literales se introducen mediante <signatura de literal>s enumeradas después de la palabra clave LITERALS. El *resultado* de una *signatura-de-literal* es el género introducido por la <definición parcial de tipo> que define el literal.

Cada <signatura de operador> en la lista de <signatura de operador>s después de la palabra clave OPERATORS representa una *signatura-de-operador* con un *nombre-de-operador*, una *lista-de-argumentos* y un *resultado*.

El <nombre de operador> corresponde a un *nombre-de-operador* en la sintaxis abstracta que es único dentro de la unidad de ámbito definidora, aunque el nombre puede no ser único en la sintaxis concreta.

El *nombre-de-operador* o *nombre-de-operador-literal* único en la sintaxis abstracta se deriva de:

a) el <nombre de operador> (o <nombre de operador literal>), más

b) la lista de identificadores de género de argumento, más

c) el identificador de género de resultado, más

d) el identificador de género de la definición parcial de tipo en que está definido el <nombre de operador> (o <nombre de operador literal>).

Siempre que se especifica un <identificador de operador>, el *nombre-de-operador* único en el *identificador-de-operador* se deriva de la misma manera con la lista de géneros de argumento y el género de resultado derivado del contexto. Dos operadores que tienen el mismo <nombre> y que difieren en uno o más de los géneros de argumento o resultado, tienen *nombres* diferentes.

Cada <género de argumento> en una <lista de argumentos> representa un *identificador-de-referencia-de-género* en una *lista-de-argumentos.* Un <resultado> representa el *identificador-de-referencia-de-género* de un *resultado.*

Dondequiera que un <calificador> de un <identificador de operador> (o <identificador de operador literal>) contiene un <ítem de trayecto> con la palabra clave TYPE, el <nombre de género> después de la palabra clave no forma parte del *calificador* del *identificador-de-operador* (o *identificador-de-operador-literal*) pero se utiliza para derivar el *nombre* único del *identificador.* En este caso el *calificador* se forma a partir de la lista de <ítem de trayecto>s que precede a la palabra clave TYPE.

Semántica

Un operador es «total», lo que significa que la aplicación del operador a cualquier lista de valores de los géneros de argumento denota un valor del género de resultado.

Una signatura de operador define cómo puede utilizarse el operador en expresiones. La signatura de operador es la identidad de operador más la lista de géneros de los argumentos y el género del resultado. La signatura de operador es la que determina si una expresión es una expresión válida en el lenguaje de acuerdo con las reglas requeridas para la comparación de los géneros de expresiones de argumento.

Un operador sin argumento se denomina un literal.

Un literal representa un valor fijo, perteneciente al género resultado del operador.

Un operador tiene un género resultado que es el género identificado por el resultado.

Nota – A título de directriz: una <signatura de operador> debe mencionar el género introducido por la <definición parcial de tipo> circundante bien como un <argumento>, o como un <resultado>.

Ejemplo 1

```
LITERALS    libre, ocupado;
```

Ejemplo 2

```
OPERATORS
    estado buscar:  Teléfono -> Disponibilidad;
```

Ejemplo 3

```
LITERALS    lista_vacía
OPERATORS   añadir_a_la_lista  : lista_de_teléfonos, teléfono -> lista_de_teléfonos;
            sublista           : lista_de_teléfonos, teléfono -> lista_de_teléfonos
```

5.2.3 *Axiomas*

Los axiomas determinan qué términos representan el mismo valor. A partir de los axiomas de una definición de tipo de datos se determina la relación entre valores de argumento y valores de resultado de operadores, dando así significado a los operadores. Los axiomas se dan como axiomas booleanos o en forma de ecuaciones de equivalencia algebraica.

Gramática abstracta

Ecuación	=	*Ecuación-incuantificada* | *Ecuaciones-cuantificadas* | *Ecuación-condicional* | *Texto-informal*
Ecuación-incuantificada	::	*Término* *Término*
Ecuaciones-cuantificadas	::	*Nombre-de-valor*-**set** *Identificador-de-género* *Ecuaciones*
Nombre-de-valor	=	*Nombre*
Término	=	*Término-fundamental* | *Término-compuesto* | *Término-de-error*
Término-compuesto	::	*Identificador-de-valor* | *Término-de-identificador-de-operador*$^{+}$ | *Término-compuesto-condicional*
Identificador-de-valor	=	*Identificador*
Identificador-de-operador	=	*Identificador*

Término-fundamental :: *Identificador-de-operador-literal* |
Término-fundamental-de-identificador-de-operador^{+} |
Término-fundamental-condicional

Identificador-de-operador-literal = *Identificador*

Las alternativas *término-compuesto-condicional* y *término-fundamental-condicional* en las reglas *término-compuesto* y *término-fundamental* respectivamente no forman parte del núcleo de datos, aunque las ecuaciones que contienen estos términos pueden remplazarse por ecuaciones semánticamente equivalentes escritas en el lenguaje núcleo (véase § 5.4.1.6). La alternativa *término de error* en la regla *término* no forma parte del núcleo de datos y se define en § 5.4.1.7.

Las definiciones de *texto-informal* y *ecuación-condicional* se dan en § 2.2.3 y § 5.2.4 respectivamente.

Cada *término* (o *término-fundamental*) en la lista de términos después de un *identificador-de-operador* debe tener el mismo género que el género correspondiente (por posición) en la *lista-de-argumentos* de la *signatura-de-operador.*

Los dos *términos* de una *ecuación-incuantificada* tienen que ser del mismo género.

Gramática textual concreta

<axiomas> ::=
<ecuación> { <fin> <ecuación>}* [<fin>]

<ecuación> ::=
<ecuación incuantificada>
| <ecuaciones cuantificadas>
| <ecuación condicional>
| <texto informal>

<ecuaciones cuantificadas> ::=
<cuantificación> (<axiomas>)

<cuantificación> ::=
FOR ALL <nombre de valor> {, <nombre de valor> }* IN <género ampliado>

<ecuación incuantificada> ::=
<término> == <término>
| <axioma booleano>

<término> ::=
<término fundamental>
| <término compuesto>
| <término de error>
| <término de ortografía>

<término compuesto> ::=
<identificador de valor>
| <identificador de operador> (<lista de términos compuestos>)
| (<término compuesto>)
| <término compuesto ampliado>

<lista de términos compuestos> ::=
<término compuesto> {, <término> }*
| <término>, <lista de términos compuestos>

<término fundamental> ::=
<identificador de literal>
| <identificador de operador> (<término fundamental> {, <término fundamental> }*)
| (<término fundamental>)
| <término fundamental ampliado>

<identificador de literal> ::=
<identificador de operador literal>
| <identificador de literal ampliado>

Las alternativas <axioma booleano> de la regla <ecuación incuantificada>, <término de error> y <término de ortografía> de la regla <término>, <término compuesto ampliado> de la regla <término compuesto>, <término fundamental ampliado> de la regla <término fundamental> e <identificador de literal ampliado> de la regla <identificador de literal> no forman parte del núcleo de datos y se definen en § 5.4.1.5, § 5.4.1.7, § 5.4.1.15, § 5.4.1, § 5.4.1 y § 5.4.1 respectivamente.

El <género> en una <cuantificación> representa el *identificador-de-género* en *ecuaciones-cuantificadas.* Los <nombre de valor>s en una <cuantificación> representan el conjunto *nombre-de-valor* en *ecuaciones-cuantificadas.*

Una <lista de términos compuestos> representa una lista de *términos.* Un *identificador-de-operador* seguido por una lista de *términos* es un *término-compuesto* solamente si la lista de *términos* contiene al menos un *identificador-de-valor.*

Un <identificador> que es un nombre incalificado que aparece en un <término> representa:

a) un *identificador-de-operador* si precede a un paréntesis redondo de apertura (o es un <nombre de operador> que es un <nombre de operador ampliado> – véase § 5.4.1); de no ser así,

b) un *identificador-de-valor* si hay una definición de ese nombre en una <cuantificación> de <ecuaciones cuantificadas> que encierra el <término> de un género adecuado para el contexto; de no ser así,

c) un *identificador-de-operador-literal* si hay un literal visible con ese nombre de un género adecuado para el contexto; de no ser así,

d) un *identificador-de-valor* que tiene una *ecuación-cuantificada* implícita en la sintaxis abstracta para la <ecuación incuantificada>.

Dos o más ocurrencias del mismo <identificador de valor> no vinculado en una <ecuación> implican una sola <cuantificación>.

Un *identificador-de-operador* se deriva del contexto, de modo que si el <nombre de operador> está sobrecargado (es decir, si se utiliza el mismo <nombre> para más de un operador), será el *nombre-de-operador* que identifica un operador visible con el mismo nombre y con géneros de argumento y género de resultado consistentes con la aplicación del operador. Si el <nombre de operador> está sobrecargado, puede que sea necesario derivar los géneros de argumento a partir de los argumentos y el género de resultado a partir del contexto para determinar el *nombre-de-operador.*

Dentro de una <ecuación incuantificada> tiene que haber exactamente un género para cada identificador de valor implícitamente cuantificado, que sea consistente con todos los usos.

Tiene que ser posible vincular cada <identificador de operador> o <identificador de operador literal> incalificado a exactamente un *identificador-de-operador* o *identificador-de-operador-literal* definido que satisfaga las condiciones en la construcción en la cual se utiliza el <identificador>. Esto es: la vinculación tiene que ser única.

Nota – A título de directriz: un axioma debe ser relevante para el género de la definición parcial de tipo circundante al mencionar un operador o literal con un resultado de este género o un operador que tenga un argumento de este género; un axioma debe definirse sólo una vez.

Semántica

Cada ecuación es una sentencia sobre la equivalencia algebraica de términos. El término del lado izquierdo y el término del lado derecho se enuncian como equivalentes, de modo que cuando aparece un término puede ser sustituido por el otro. Cuando un identificador de valor aparece en una ecuación, puede ser sustituido simultáneamente en esa ecuación por el mismo término para cada ocurrencia del identificador de valor. Para esta sustitución, el término puede ser cualquier término fundamental del mismo género que el identificador de valor.

Se introducen identificadores de valor por los nombres de valor en ecuaciones cuantificadas. Un identificador de valor se utiliza para representar cualquier valor de datos que pertenezca al género de la cuantificación. Una ecuación queda satisfecha si el mismo valor remplaza simultáneamente cada ocurrencia del identificador de valor en la ecuación, independientemente del valor elegido para la sustitución.

Un término fundamental es un término que no contiene ningún identificador de valor. Un término fundamental representa un valor particular, conocido. Para cada valor en un género existe al menos un término fundamental que representa ese valor.

Si uno o más axiomas cualesquiera contienen texto informal, la interpretación de expresiones no está formalmente definida por LED pero puede ser determinada a partir del texto informal por el interpretador. Se parte del supuesto de que si se especifica texto informal, se sabe que el conjunto de ecuaciones está incompleto; por lo tanto, no se ha dado una especificación formal completa en LED.

Un nombre de valor es siempre introducido por ecuaciones cuantificadas en la sintaxis abstracta, y el valor correspondiente tiene un identificador de valor que es el nombre de valor calificado por el identificador de género de las ecuaciones cuantificadas circundantes. Por ejemplo:

```
FOR ALL z,z IN X ( FOR ALL z IN X ...))
```

introduce sólo un identificador denominado z de género X.

En la sintaxis concreta no está permitido especificar un calificador para identificadores de valor.

Cada identificador de valor introducido por ecuaciones cuantificadas tiene un género que es el identificado en las ecuaciones cuantificadas por el *identificador-de-referencia-de-género.* El género de las cuantificaciones implícitas es el género requerido por el contexto (o los contextos) de la ocurrencia del identificador no vinculado. Si los contextos de un identificador de valor que tiene una cuantificación implícita permiten géneros diferentes, el identificador está vinculado a un género que es consistente con todos sus usos en la ecuación.

Un término tiene un género que es el género del identificador de valor o el género resultado del operador (literal).

A menos que pueda deducirse de las ecuaciones que dos literales denotan el mismo valor, cada literal denotará un valor diferente.

Ejemplo 1

FOR ALL b IN logical (eq(b,b)= =T)

Ejemplo 2

neq(T,F)= =T; neq(T,T)= =F;

neq(F,T)= =; neq(F,F)= =F;

Ejemplo 3

eq(b, b) = =T;

eq(F, eq(T,F)) = =T;

eq(eq(b,a),eq(a,b)) = =T;

5.2.4 *Ecuaciones condicionales*

Una ecuación condicional permite la especificación de ecuaciones que sólo son aplicables cuando se cumplen ciertas restricciones. Las restricciones se escriben en forma de ecuaciones simples.

Gramática abstracta

Ecuación-condicional	::	*Restricción*-**set** *Ecuación-restringida*
Restricción	=	*Ecuación-incuantificada*
Ecuación-restringida	=	*Ecuación-incuantificada*

Gramática textual concreta

<ecuación condicional> ::=
<restricción> {, <restricción> }* = = > <ecuación restringida>

<ecuación restringida> ::=
<ecuación incuantificada>

<restricción> ::=
<ecuación incuantificada>

Semántica

Una ecuación restringida establece que unos términos denotan el mismo valor solamente cuando cualquier identificador de valor en las ecuaciones restringidas denota un valor con relación al cual se puede demostrar, mediante otras ecuaciones, que satisface la restricción. Un valor satisfará una restricción solamente si la restricción puede deducirse de otras ecuaciones para este valor.

La semántica de un conjunto de ecuaciones para un tipo de datos que incluye ecuaciones condicionales se deriva de la manera siguiente:

a) La cuantificación se suprime generando todas las ecuaciones de términos fundamentales que puedan derivarse de las ecuaciones cuantificadas. Como esto se aplica a la cuantificación explícita e implícita, se genera un conjunto de ecuaciones incuantificadas en términos fundamentales.

b) Se entiende por ecuación condicional comprobable una ecuación condicional para la que puede demostrarse que todas las restricciones (únicamente en términos fundamentales) se cumplen a partir de ecuaciones incuantificadas que no son ecuaciones restringidas. Si existe una ecuación condicional comprobable, se sustituye por la ecuación restringida de la ecuación condicional comprobable.

c) Si quedan ecuaciones condicionales en el conjunto de ecuaciones, y ninguna de esas ecuaciones condicionales es una ecuación condicional comprobable, se suprimen esas ecuaciones condicionales, y si no, se vuelve al paso (b).

d) El conjunto de ecuaciones incuantificadas restantes define la semántica del tipo de datos.

Ejemplo

z/=0= =Verdadero= = > (x/z)*z= =x

5.3 *Modelo de álgebra inicial (descripción informal)*

La definición de datos en LED se basa en el núcleo de datos definido en § 5.2. Es necesario dar a los operadores y valores algún significado suplementario además de la definición anterior, de modo que pueda darse una interpretación a las expresiones. Por ejemplo, las expresiones utilizadas en señales continuas, condiciones habilitantes, llamadas a procedimiento, acciones de salida, peticiones de crear, sentencias de asignación, sentencias de inicializar y de reinicializar, sentencias de exportación, sentencias de importación, decisiones y visión.

Para dar a las expresiones el significado adicional necesario se utiliza el formalismo de álgebra inicial explicado en § 5.3.1 a § 5.3.6.[1)]

En cualquier punto de una especificación LED se aplicará el último tipo de datos jerárquicamente definido, pero habrá un conjunto de géneros visibles. El conjunto de géneros será la unión de todos los géneros a niveles jerárquicamente superiores al lugar en cuestión, como se explica en § 5.2.

(En esta sección se utiliza el símbolo = como un símbolo de equivalencia de ecuación, en tanto que en LED se utiliza el símbolo = = para equivalencia de ecuación, de modo que el símbolo = pueda utilizarse para el operador de igualdad. El símbolo = se utiliza en esta sección por ser el símbolo convencional utilizado en las obras publicadas sobre álgebras iniciales.)

5.3.1 *Introducción*

El significado y la interpretación de datos basados en álgebra inicial se explica en tres etapas:

a) Signaturas

b) Términos

c) Valores

5.3.1.1 *Representaciones*

Es bien sabido que notaciones diferentes pueden representar el mismo concepto. Por ejemplo, se acepta generalmente que los números arábigos positivos (1,2,3,4,...) y los numerales romanos (I,II,III,IV,...) representan el mismo conjunto de números con las mismas propiedades. Como otro ejemplo, se acepta corrientemente que la notación funcional prefija (más(1,1)), la notación infija (1+1) y la notación polaca inversa (1 1 +) pueden representar, todas ellas, el mismo operador. Además, diferentes usuarios pueden utilizar diferentes nombres (quizás por el hecho de que utilizan idiomas diferentes) para los mismos conceptos, de modo que los pares {true, false}, {T, F}, {0,1}, {vrai, faux}, {verdadero, falso} podrían ser diferentes representaciones del género booleano.

Lo esencial es la relación abstracta entre identidades y no la representación concreta. Así, en el caso de los numerales, lo que interesa es que la relación entre 1 y 2 sea la misma que la relación entre I y II. De la misma manera, en cuanto a los operadores, lo que interesa es la relación entre la identidad de operador y otras identidades de operador y la lista de argumentos. Construcciones concretas tales como los paréntesis, que permiten distinguir entre (a+b)*c y a+(b*c) sólo ofrecen interés por el hecho de que permiten determinar los conceptos abstractos subyacentes.

Estos conceptos abstractos están incorporados en una sintaxis abstracta del concepto que puede realizarse mediante más de una sintaxis concreta. Por ejemplo, los dos ejemplos concretos siguientes describen las mismas propiedades de tipos de datos, pero en sintaxis concretas diferentes.

[1)] El texto de § 5.3.1 a § 5.3.6 ha sido convenido entre la ISO y el CCITT como una descripción informal común del modelo de álgebra inicial para tipos abstractos de datos. Este texto, además de figurar en la presente Recomendación (con modificaciones apropiadas de tipografía y numeración), constituye también un anexo a ISO IS8807.

```
NEWTYPE     bool LITERALS true, false;
OPERATORS   "not" :bool->bool;
AXIOMS
        not(true)           ==false;
        not(not(a))         ==a;
ENDNEWTYPE bool;

NEWTYPE int LITERALS zero, one;
OPERATORS   plus        :int,int->int;
            minus       :int,int->int;
AXIOMS
        plus(zero,a)                ==a;
        plus(a,b)                   ==plus(b,a);
        plus(a,plus(b,c))           ==plus(plus(a,b),c);
        minus(a,a)                  ==zero;
        minus(a,zero)               ==a;
        minus(a,minus(b,c)          ==minus(plus(a,c),b);
        minus(minus(a,b),c)         ==minus(a,plus(b,c));
        plus(minus(a,b),c)          ==minus(plus(a,c),b);
ENDNEWTYPE int;

NEWTYPE tree LITERALS nil;
OPERATORS
        tip     :int        ->tree;
        isnil   :tree       ->bool;
        istip   :tree       ->bool;
        node    :tree,tree  ->tree;
        sum     :tree       ->int;
AXIOMS
        istip(nil)              ==false;
        istip(tip(i))           ==true;
        istip(node(t1,t2))      ==false;
        isnil(nil)              ==true;
        isnil(tip(i))           ==false;
        isnil(node(t1,t2))      ==false;
        sum(node(t1,t2))        ==plus(sum(t1),sum(t2));
        sum(tip(i))             ==i;
        sum(nil)                ==zero;
ENDNEWTYPE tree;
```

EJEMPLO 1

```
TYPE      bool        IS
SORTS     bool
OPNS      true  :     ->bool
          false :     ->bool
          not   : bool ->bool
EQNS OFSORT           bool FORALL a:bool
      not(true)       =false;
      not(not(a))     =a
ENDTYPE

TYPE      int   IS    bool WITH
SORTS     int
OPNS      zero  :                 ->int
          one   :                 ->int
          plus  :     int,int     ->int
          minus:      int,int     ->int
EQNS OFSORT           int FORALL a,b,c:int
      plus(zero,a)                =a;
      plus(a,b)                   =plus(b,a);
      plus(a,plus(b,c))           =plus(plus(a,b),c);
      minus(a,a)                  =zero;
      minus(a,zero)               =a;
      minus(a,minus(b,c))         =minus(plus(a,c),b);
      minus(minus(a,b),c)         =minus(a,plus(b,c));
      plus(minus(a,b),c)          =minus(plus(a,c),b)
ENDTYPE

TYPE      tree  IS    int   WITH
SORTS     tree
OPNS      nil   :               ->tree
          tip   :int            ->tree
          isnil :tree           ->bool
          istip :tree           ->bool
          node:tree,tree        ->tree
          sum   :tree           ->int
EQNS OFSORT           bool FORALL i:int, t1,t2:tree
          istip(nil)            =false;
          istip(tip(i))         =true;
          istip(node(t1,t2))    =false
          isnil(nil)            =true
          isnil(tip(i))         =false
          isnil(node(t1,t2))    =false
      OFSORT          int FORALL i:int, t1,t2:tree
          sum (node(t1,t2))     =plus(sum(t1),sum(t2));
          sum(tip(i))           =i;
          sum(nil)              =zero
ENDTYPE
```

EJEMPLO 2

Este ejemplo se utilizará para ilustración. Inicialmente se considerará la definición de géneros y literales.

Debe observarse que los literales se consideran un caso especial de operadores, es decir, operadores sin parámetros.

Se puede introducir algunos géneros y literales, empleando la primera forma, como sigue:

```
NEWTYPE int LITERALS zero, one; ...
NEWTYPE bool LITERALS true, false; ...
NEWTYPE tree LITERALS nil; ...
```

o, empleando la segunda forma, como sigue:

```
...
SORTS   bool
OPNS    true  :    ->bool
        false :    ->bool
...
SORTS   int
OPNS    zero  :    ->int
        one   :    ->int
...
SORTS   tree
OPNS    nil   :    ->tree
...
```

En lo sucesivo se empleará la segunda forma solamente pues está más próxima a la formulación utilizada en muchas publicaciones de álgebra inicial. Debe señalarse que la forma de los términos es la misma en ambos casos y que la diferencia más importante es la manera en que se introducen los literales. Debe recordarse que es necesario adoptar una notación concreta para comunicar los conceptos, pero el significado de las álgebras es independiente de la notación, de modo que una redenominación sistemática de los nombres (reteniendo la misma unicidad) y un cambio de la notación prefija a la notación polaca no cambiará el significado definido por las definiciones de tipo.

5.3.2 *Signaturas*

Asociados con cada género habrá uno o más operadores. Cada operador tiene una funcionalidad de operador; es decir, se define para relacionar uno o más géneros entrada con un género resultado.

Por ejemplo, se pueden añadir los siguientes operadores a los géneros definidos anteriormente:

```
...
SORTS   bool
OPNS    true  :      ->bool
        false :      ->bool
        not   : bool->bool
...
SORTS   int
OPNS    zero  :                ->int
        one   :                ->int
        plus  :    int,int     ->int
        minus:     int,int     ->int
...
SORTS   tree
OPNS    nil   :                ->tree
        tip   :    int         ->tree
        isnil :    tree        ->bool
        istip :    tree        ->bool
        node:      tree,tree   ->tree
        sum   :    tree        ->int
...
```

La signatura del tipo que se aplica es el conjunto de géneros y el conjunto de operadores (tanto literales como operadores con parámetros) que son visibles.

Se dice que una signatura de un tipo está completa (cerrada) si, para cada operador de la signatura, los géneros de la funcionalidad del operador están incluidos en el conjunto de géneros del tipo.

5.3.3 *Términos y expresiones*

El lenguaje de interés es el que admite expresiones que son variables, literales u operadores aplicados a expresiones. Una variable es un objeto datos que está asociado con una expresión. Una interpretación de la variable puede remplazarse por una interpretación de la expresión asociada a la variable. De esta forma pueden eliminarse variables, de modo que una interpretación de una expresión puede reducirse a la aplicación de diversos operadores a literales.

Así, al ser interpretada, una expresión abierta (una expresión que comprende variables) pasa a ser una expresión cerrada (una expresión sin variables), proporcionando a la expresión abierta argumentos efectivos (es decir, expresiones cerradas).

Una expresión cerrada corresponde a un término fundamental.

El conjunto de todos los términos fundamentales posibles de un género se denomina conjunto de términos fundamentales del género. Por ejemplo, para el género booleano, definido más arriba, el conjunto de términos fundamentales contendrá

{true, false, not(true), not(false), not(not(true)), . . .}

Puede verse que incluso para este género muy sencillo, el conjunto de términos fundamentales es infinito.

5.3.3.1 *Generación de términos*

Dada una signatura de un tipo, es posible generar el conjunto de términos fundamentales para ese tipo.

Se considera que el conjunto de literales del tipo es el conjunto básico de términos fundamentales. Cada literal tiene un género; por tanto, cada término fundamental tiene un género. Para el tipo definido anteriormente, el conjunto básico de términos fundamentales será

{zero, one, true, false, nil}

Para cada operador del conjunto de operadores del tipo se generan términos fundamentales sustituyendo cada argumento por todos los términos fundamentales, del género correcto, generados antes para ese argumento. El género resultado de cada operador es el género del término fundamental generado por ese operador. El conjunto resultante de términos fundamentales se agrega al conjunto existente de términos fundamentales para generar un nuevo conjunto de términos fundamentales. Para el tipo antes mencionado, este es

{zero,	one,	true,	false,	nil,
plus(zero,zero),	plus(one,one),	plus(zero,one),	plus(one,zero),	
minus(zero,zero),	minus(one,one),	minus(zero,one),	minus(one,zero),	
not(true),	not(false),	tip(zero),	tip(one),	
isnil(nil),	istip(nil),	node(nil,nil),	sum(nil),	}

Este nuevo conjunto de términos fundamentales se toma entonces como el conjunto precedente de téminos fundamentales para una ulterior aplicación del último algoritmo, para generar un conjunto ulterior de términos fundamentales. Este conjunto de términos fundamentales incluirá

{zero,	one,	true,	false,	nil,
plus(zero,zero)),	plus(one,one),	plus(zero,one),	plus(one,zero),	. . .
plus(zero,plus(zero,zero)),		plus(zero,plus(one,one)),		. . .
plus(zero,sum(nil)),		. . .		
isnil(node(nil,nil)),		istip(node(nil,nil)),	node(nil,node(nil,nil)),	
. . . ,			sum(node(nil,nil)) }	

Este algoritmo se aplica repetidamente para generar todos los términos fundamentales posibles del tipo, lo cual es el conjunto de términos fundamentales de tipo. El conjunto de términos fundamentales de un género es el conjunto de términos fundamentales del tipo que tienen ese género.

Normalmente la generación continuará dando, indefinidamente, un número infinito de términos.

5.3.4 *Valores y álgebras*

Cada término de un género representa un valor de ese género. De lo anterior puede verse que incluso un género simple como el booleano tiene un número infinito de términos y en consecuencia un número infinito de valores, a menos que se defina de algún modo la manera en que los términos son equivalentes (es decir, representan el mismo valor). Esta definición se da mediante ecuaciones definidas sobre términos. En ausencia de istip e isnil, el género booleano puede limitarse a dos valores mediante las ecuaciones

not(true) = false;
not(false) = true

Estas ecuaciones definen términos como equivalentes y es entonces posible obtener las dos clases equivalentes de términos

{ true,	not(false),	not(not(true)),	not(not(not(false))), . . .}
{ false,	not(true),	not(not(false)),	not(not(not(true))), . . .}

Cada clase de equivalencia representa un valor y los miembros de la clase son representaciones diferentes del mismo valor.

Obsérvese que, a menos que se definan como equivalentes por ecuaciones, los términos son inequivalentes (es decir, no representan el mismo valor).

Un álgebra define el conjunto de términos que satisface la signatura del álgebra. Las ecuaciones del álgebra relacionan términos entre sí.

En general habrá más de una representación para cada valor de un género en un álgebra.

Un álgebra para una signatura determinada es un álgebra inicial únicamente si cualquier otra álgebra que dé las mismas propiedades para la signatura puede ser transformada sistemáticamente en el álgebra inicial. (Formalmente, tal transformación se conoce como homomorfismo.)

Si not, istip e isnil producen siempre valores en las clases de equivalencia de true (verdadero) y false (falso), un álgebra inicial para el género booleano es el par de literales

{true, false}

sin ecuaciones.

5.3.4.1 *Ecuaciones y cuantificación*

Para un género como el booleano, donde hay solamente un número limitado de valores, todas las ecuaciones pueden escribirse utilizando términos fundamentales solamente, es decir, términos que sólo contienen literales y operadores.

Cuando un género contiene muchos valores, no es práctico escribir todas las ecuaciones utilizando términos fundamentales y, para géneros que tienen un número infinito de valores (por ejemplo, los enteros), tal enumeración explícita resulta imposible. La técnica de escribir ecuaciones cuantificadas se utiliza para representar un conjunto posiblemente infinito de ecuaciones por medio de una ecuación cuantificada.

Una ecuación cuantificada contiene identificadores de valores en forma de términos. Estos términos se denominan términos compuestos. El conjunto de ecuaciones escritas solamente con términos fundamentales puede derivarse de la ecuación cuantificada, generando sistemáticamente ecuaciones con cada identificador de valor sustituido en la ecuación por uno de los términos fundamentales del género del identificador de valor. Por ejemplo:

FOR ALL b : bool not(not(b)) = b

representa

not(not(true)) = true;
not(not(false)) = false

Un conjunto alternativo de ecuaciones para bool puede tomarse ahora como

FOR ALL b : bool
not(not(b)) = b;
not(true) = false

Cuando el género del identificador de valor cuantificado es obvio por el contexto, es corriente omitir la cláusula que define el identificador de valor, y el ejemplo pasa a ser:

not(not(b)) = b;
not(true) = false

5.3.5 *Especificación algebraica y semántica (significado)*

Una especificación algebraica consiste en una signatura y conjuntos de ecuaciones para cada género de la signatura. Estos conjuntos de ecuaciones inducen relaciones de equivalencia que definen el significado de la especificación.

El símbolo = denota una relación de equivalencia que satisface las propiedades reflexiva, simétrica y transitiva, así como la propiedad de sustitución.

Las ecuaciones dadas con un tipo permiten colocar términos en clases de equivalencia. Dos términos cualesquiera que pertenezcan a la misma clase de equivalencia se interpretan como que tienen el mismo valor. Este mecanismo puede utilizarse para identificar términos sintácticamente diferentes que tienen el mismo valor deseado.

Dos términos del mismo género, TERM1 y TERM2, pertenecen a la misma clase de equivalencia si

a) hay una ecuación
TERM1 = TERM2,
o

b) una de las ecuaciones derivadas del conjunto dado de ecuaciones cuantificadas es
TERM1 = TERM2,
o

c) i) TERM1 pertenece a una clase de equivalencia que contiene TERMA, y

ii) TERM2 pertenece a una clase de equivalencia que contiene TERMB, y

iii) hay una ecuación o una ecuación derivada del conjunto dado de ecuaciones cuantificadas, tal que
TERMA = TERMB,
o

d) sustituyendo un subtérmino de TERM1 por un término de la misma clase que el subtérmino que produce un término TERM1A es posible mostrar que TERM1A pertenece a la misma clase que TERM2.

Aplicando todas las ecuaciones, los términos de cada género son «particionados» en una o más clases de equivalencia. Hay tantos valores para el género como clases de equivalencia. Cada clase de equivalencia representa un valor y cada miembro de una clase representa el mismo valor.

5.3.6 *Representación de valores*

Interpretar una expresión significa entonces en primer lugar derivar el término fundamental determinando el valor efectivo de las variables utilizadas en una expresión en el punto de interpretación, y después hallar la clase de equivalencia de este término fundamental. La clase de equivalencia de este término determina el valor de la expresión.

Así, se da significado a los operadores utilizados en expresiones, determinando el valor resultante dado a un conjunto de argumentos.

Es usual elegir un literal perteneciente a la clase de equivalencia para representar el valor de la clase. Por ejemplo, booleano (bool) sería representado por verdadero y falso y los números naturales por 0,1,2,3, etc. Cuando no hay literal se utiliza generalmente el término de menor complejidad posible (menor número de operadores). Por ejemplo, para enteros negativos la notación usual es -1, -2, -3, etc.

5.4 *Uso pasivo de datos LED*

En § 5.4.1 se definen ampliaciones de las construcciones de definición de datos indicadas en § 5.2. La manera de interpretar el uso de tipos abstractos de datos en expresiones se define en § 5.4.2 para la expresión «pasiva» (es decir, cuando no depende de variables ni del estado del sistema). La manera de interpretar expresiones que no son pasivas (es decir, expresiones «activas») se define en § 5.5.

5.4.1 *Construcciones ampliadas de definición de datos*

Las construcciones definidas en § 5.2 son la base de formas más concisas explicadas más abajo.

Gramática abstracta

No hay una sintaxis abstracta adicional para la mayor parte de estas construcciones. En § 5.4.1 y en todos los apartados de § 5.4.1 la sintaxis abstracta pertinente es generalmente la descrita en § 5.2.

Gramática textual concreta

```
<propiedades ampliadas> ::=
        <regla de herencia>
    |   <instanciaciones de generador>
    |   <definición de estructura>

<término compuesto ampliado> ::=
        <identificador de operador ampliado> ( <lista de términos compuestos> )
    |   <término compuesto> <operador infijo> <término>
    |   <término> <operador infijo> <término compuesto>
    |   <operador monádico> <término compuesto>
    |   <término compuesto condicional>

<término fundamental ampliado> ::=
        <identificador de operador ampliado>
            (<término fundamental> {, <término fundamental> }* )
    |   <término fundamental> <operador infijo> <término fundamental>
    |   <operador monádico> <término fundamental>
    |   <término fundamental condicional>

<identificador de operador ampliado> ::=
        <identificador de operador> <admiración>
    |   <nombre formal de generador>
    |   [ <calificador> ] <operador entre comillas>

<nombre de operador ampliado> ::=
        <nombre de operador> <admiración>
    |   <nombre formal de generador>
    |   <operador entre comillas>

<admiración> ::=
        !

<nombre de literal ampliado> ::=
        <literal cadena de caracteres>
    |   <nombre formal de generador>
    |   <literal clase de nombre>

<identificador de literal ampliado> ::=
        <identificador de literal cadena de caracteres>
    |   <nombre formal de generador>
```

Las reglas <propiedades ampliadas>, <término compuesto ampliado>, <término fundamental ampliado>, <nombre de operador ampliado>, <nombre de literal ampliado> e <identificador de literal ampliado> amplían las reglas para <definición parcial de tipo> (§ 5.2.1), <término compuesto> (§ 5.2.3), <término fundamental> (§ 5.2.3), <nombre de operador> (§ 5.2.2), <literal> (§ 5.2.2) e <identificador de literal> (§ 5.2.3) respectivamente en el núcleo de datos. Estas reglas son ampliadas aún más por las reglas <regla de herencia> (§ 5.4.1.11), <instanciaciones de generador> (§ 5.4.1.12.2), <nombre formal de generador> (§ 5.4.1.12.1), <término compuesto condicional> (§ 5.4.1.6), <término fundamental condicional> (§ 5.4.1.6), <literal cadena de caracteres> e <identificador de literal cadena de caracteres> (§ 5.4.1.2) y <literal clase de nombre> (§ 5.4.1.14). Las reglas <operador infijo>, <operador monádico>, <operador infijo entre comillas> y <operador monádico entre comillas> se definen en § 5.4.1.1.

Alternativas con <nombre formal de generador>s sólo son válidas en una <expresión de propiedades> en un <texto de generador> (véase § 5.4.1.12) que tenga ese nombre definido como un parámetro formal.

Las alternativas de <término compuesto ampliado> y <término fundamental ampliado> con un <nombre formal de generador> que precede un «(» sólo son válidas si el <nombre formal de generador> se define de modo que sea de la clase OPERATOR (véase § 5.4.1.12).

La alternativa de <nombre de literal ampliado> con un <nombre formal de generador> sólo es válida si el <nombre formal de generador> está definido de modo que sea de la clase LITERAL (véase § 5.4.1.12).

La alternativa de <identificador de literal ampliado> con un <nombre formal de generador> sólo es válida si el <nombre formal de generador> está definido de modo que sea de la clase LITERAL o de la clase CONSTANT (véase § 5.4.1.12).

Si un nombre de operador está definido con una <admiración>, la <admiración> forma semánticamente parte del *nombre.*

Las formas <nombre de operador> <admiración> o <identificador de operador> <admiración> representan *nombre de operador* (§ 5.2.2) e *identificador de operador* (§ 5.2.3) respectivamente.

Semántica

Un nombre de operador definido con una <admiración> tiene la semántica normal de un operador, pero el nombre de operador sólo es visible en axiomas.

5.4.1.1 *Operadores especiales*

Estos son nombres de operador que tienen formas sintácticas especiales. La sintaxis especial se introduce de modo que los operadores aritméticos y operadores booleanos puedan tener su forma sintáctica usual. Es decir, el usuario puede escribir «(1 + 1) = 2» y no está forzado a utilizar por ejemplo equal(add(1,1),2). Los géneros que serán válidos para cada operador dependerán de la definición de tipo de datos.

Gramática textual concreta

```
<operador entre comillas> ::=
        <comillas> <operador infijo> <comillas>
    |   <comillas> <operador monádico> <comillas>

<comillas> ::=
        "

<operador infijo> ::=
        =>
    |   OR
    |   XOR
    |   AND
    |   IN
    |   /=
    |   =
    |   >
    |   <
    |   <=
    |   >=
    |   +
    |   /
    |   *
    |   //
    |   MOD
    |   REM
    |   -

<operador monádico> ::=
        -
    |   NOT
```

Semántica

Un operador infijo es un término que tiene la semántica normal de un operador pero con sintaxis de infijo o de prefijo entre comillas, como se ha indicado anteriormente.

Un operador monádico es un término que tiene la semántica normal de un operador pero con la sintaxis de prefijo o prefijo entre comillas, como se ha indicado anteriormente.

Las formas entre comillas de operadores infijo o monádico son nombres válidos para operadores.

Los operadores infijos tienen un orden de precedencia que determina la vinculación de operadores. La vinculación es la misma que la vinculación en <expresión>s, como se especifica en § 5.4.2.1.

Cuando la vinculación es ambigua, como lo es en

a OR b XOR c ;

entonces va de izquierda a derecha, de modo que este término es equivalente a

(a OR b) XOR c ;

Obsérvese que los <operador entre comillas>s MOD y REM no tienen semántica predefinida dado que no se definen en los géneros de datos predefinidos.

Modelo

Un término de la forma

<término 1> <operador infijo> <término 2>

es sintaxis derivada para

"<operador infijo>" (<término 1>, <término 2>)

con "<operador infijo>" como un nombre lícito. "<operador infijo>" representa un *nombre de operador.*

De manera similar

<operador monádico> <término>

es sintaxis derivada para

"<operador monádico>" (<término>)

con "<operador monádico>" como un nombre lícito que representa un *nombre de operador.*

[Obsérvese que el operador de igualdad LED (=) no debe confundirse con el símbolo de equivalencia de términos LED (==).]

5.4.1.2 *Literales cadena de caracteres*

Gramática textual concreta

<identificador de literal cadena de caracteres> ::=
[<calificador>] <literal cadena de caracteres>

<literal cadena de caracteres> ::=
<cadena de caracteres>

Una <cadena de caracteres> es una unidad léxica definida en § 2.2.1.

Un <identificador de literal cadena de caracteres> representa un *identificador-de-operador-literal* en la sintaxis abstracta.

Un <literal cadena de caracteres> representa un *nombre-de-operador-literal* (§ 5.2.2) en la sintaxis abstracta derivada de la <cadena de caracteres>.

Semántica

Identificadores de literal cadena de caracteres son los identificadores formados a partir de literales cadena de caracteres en términos y expresiones.

Los literales cadena de caracteres se utilizan para los géneros de datos predefinidos cadena-de-caracteres (Charstring) y carácter (Character) (véase § 5.6). Tienen también una relación especial con los literales clase de nombre (véase § 5.4.1.14) y correspondencias de literales (véase § 5.4.1.15). Estos literales pueden también definirse de modo que tengan otros usos.

Un <literal cadena de caracteres> tiene una longitud que es el número de <alfanumérico>s más <otro carácter>s más <especial>s más <punto>s más <subrayado>s más <espacio>s más pares <apóstrofo> <apóstrofo> en la <cadena de caracteres> (véase § 2.2.1).

Si un <literal cadena de caracteres>

a) tiene una longitud superior a 1, y

b) tiene una subcadena formada por la supresión del último carácter (<alfanumérico> u <otro carácter> o <especial> o <punto> o <subrayado> o <espacio> o pares <apóstrofo> <apóstrofo>) de la <cadena de caracteres>, y

c) esa subcadena está definida como un literal tal que
subcadena // carácter_suprimido_entre_comillas
es un término válido con el mismo género que el <literal cadena de caracteres>,

entonces hay una ecuación implícita dada por la sintaxis concreta según la cual el <literal cadena de caracteres> es equivalente a la subcadena seguida por el operador infijo "//" seguido por el carácter suprimido con apóstrofos para formar una <cadena de caracteres>.

Por ejemplo los literales 'ABC', 'AB''', y 'AB' en

```
NEWTYPE s
LITERALS 'ABC', 'AB''' 'AB', 'A', 'B', '''';
OPERATORS "//": s, s -> s;
```

tienen las ecuaciones implícitas

```
'ABC'   == 'AB' // 'C';
'AB'''  == 'AB' // '''';
'AB'    == 'A' // 'B';
```

5.4.1.3 *Datos predefinidos*

Los datos predefinidos, incluyendo el género booleano que define propiedades para dos literales, True (Verdadero) y False (Falso), se definen en § 5.6. La semántica de igualdad (§ 5.4.1.4), los axiomas booleanos (§ 5.4.1.5), los términos condicionales (§ 5.4.1.6), la ordenación (§ 5.4.1.8), y los sintipos (§ 5.4.1.9) se basan en la definición del género booleano (§ 5.6.1). La semántica de los literales clase de nombre (si se utilizan <intervalo regular>s – § 5.4.1.14) y la correspondencia de literales (§ 5.4.1.15) se basan también en la definición de carácter (Character) (§ 5.6.2) y cadena-de-caracteres (Charstring) (§ 5.6.4) respectivamente.

Se considera que los datos predefinidos están definidos a nivel de sistema.

5.4.1.4 *Igualdad*

Gramática textual concreta

Cada nombre de género introducido en una <definición parcial de tipo> tiene una *signatura de operador* implícita tanto para = como para /=, y un conjunto de *ecuaciones* implícito para estos operadores.

Una <definición parcial de tipo> que introduce un género denominado S tiene implícito un par de *signaturas de operador* equivalente a

```
"=" : S, S -> Boolean;
"/=": S, S -> Boolean;
```

donde Boolean es el género booleano predefinido.

Una <definición parcial de tipo> que introduce un género denominado S tiene un conjunto implícito de *ecuaciones*

```
FOR ALL a, b, c IN S (
    a = a                                  ==True;
    a = b                                  ==b = a;
    (( a=b ) AND ( b=c )) => a=c           ==True;
    a /= b                                 ==NOT ( a=b );
    a = b == True ==>  a                   == b )
```

La última ecuación expresa la propiedad de sustitución para igualdad.

Si es posible derivar de las ecuaciones (explícitas, implícitas y derivadas) que

```
True == False,
```

lo que está en contradicción con las propiedades supuestas del tipo de datos booleano, la definición tiene que ser inválida. No debe poderse derivar

```
True == False;
```

Cada expresión fundamental booleana que se utiliza fuera de definiciones de tipo de datos tiene que interpretarse bien como Verdadero o como Falso. Si no es posible reducir tal expresión a Verdadero o Falso, la especificación es incompleta y permite más de una interpretación del tipo de datos.

Semántica

Para cada género introducido por una definición parcial de tipo hay una definición implícita de operadores y ecuaciones para igualdad.

Los símbolos = y /= en la sintaxis concreta representan los nombres de los operadores que son denominados los operadores igual y no igual.

5.4.1.5 *Axiomas booleanos*

Gramática textual concreta

```
<axioma booleano> ::=
        <término booleano>
```

Semántica

Un axioma booleano es una sentencia de verdad que se cumple para todas las condiciones del tipo de datos que se está definiendo, y por tanto puede utilizarse para especificar el comportamiento del tipo de datos.

Modelo

Un axioma de la forma

<término booleano>;

es sintaxis derivada para la ecuación de sintaxis concreta

<término booleano> = = Verdadero;

que tiene la relación normal de una ecuación con la sintaxis abstracta.

5.4.1.6 *Términos condicionales*

En lo sucesivo, la ecuación que contiene el término condicional se denomina ecuación de término condicional.

Gramática abstracta

Término-compuesto-condicional	=	*Término-condicional*
Término-fundamental-condicional	=	*Término-condicional*
Término-condicional	::	*Condición* *Consecuencia* *Alternativa*
Condición	=	*Término*
Consecuencia	=	*Término*
Alternativa	=	*Término*

El género de la *condición* tiene que ser el género booleano predefinido, y la *condición* no debe ser el *término-de-error*. La *consecuencia* y la *alternativa* tienen que ser del mismo género.

Un *término condicional* es un *término compuesto condicional* si, y sólo si, uno o más de los *términos* en la *condición*, la *consecuencia* o la *alternativa* son *términos compuestos*.

Un *término condicional* es un *término fundamental condicional* si, y sólo si, todos los *términos* en la *condición*, la *consecuencia* o la *alternativa* son *términos fundamentales*.

Gramática textual concreta

<término compuesto condicional> ::=
<término condicional>

<término fundamental condicional> ::=
<término condicional>

<término condicional> ::=
IF <condición> THEN <consecuencia> ELSE <alternativa> FI

<condición> ::=
<término booleano>

<consecuencia> ::=
<término>

<alternativa> ::=
<término>

Semántica

Un término condicional utilizado en una ecuación es semánticamente equivalente a dos conjuntos de ecuaciones en los cuales se han eliminado todos los identificadores de valor cuantificado en el término booleano.

El conjunto de ecuaciones puede formarse sustituyendo simultáneamente en toda la ecuación de término condicional cada *identificador-de-valor* en la *condición* por cada *término-fundamental* del género apropiado. En este conjunto de ecuaciones, la *condición* siempre habrá sido remplazada por un *término-fundamental* booleano. En lo sucesivo, este conjunto de ecuaciones será el conjunto fundamental ampliado.

Una ecuación de término condicional es equivalente al conjunto de *ecuaciones* que contiene:

a) para cada *ecuación* del conjunto fundamental ampliado en la cual la *condición* es equivalente a Verdadero, esa *ecuación* del conjunto fundamental ampliado con el *término-condicional* remplazado por la *consecuencia* (fundamental) y

b) para cada *ecuación* del conjunto fundamental ampliado en la cual la *condición* es equivalente a Falso, esa *ecuación* del conjunto fundamental ampliado con el *término-condicional* remplazado por la *alternativa* (fundamental).

Obsérvese que el caso especial de una ecuación de la forma

ex1 = = IF a THEN b ELSE c FI;

es equivalente al par de ecuaciones condicionales

a = = True = = > ex1 = = b;
a = = False = = > ex1 = = c;

Ejemplo

IF i = j * j THEN posroot(i) ELSE abs(j) FI = = IF positive(j) THEN j ELSE -j FI;

Nota – Hay formas mejores de especificar estas propiedades; éste es sólo un ejemplo.

5.4.1.7 *Errores*

Se utilizan errores para hacer posible que las propiedades de un tipo de datos queden completamente definidas incluso en casos en que no se puede dar un significado específico al resultado de un operador.

Gramática abstracta

Término-de-error :: ()

Un *término-de-error* no debe utilizarse como un *término* de argumento para un *identificador-operador* en un término *compuesto.*

Un *término-de-error* no podrá utilizarse como parte de una *restrición.*

No será posible derivar de *ecuaciones* que un *identificador-de-operador-literal* es igual a un *término-de-error.*

Gramática textual concreta

<término de error> ::=
ERROR <admiración>

Semántica

Un término puede ser un error, por lo que es posible especificar las circunstancias en las cuales un operador produce un error. Si estas circunstancias surgen durante la interpretación, el comportamiento ulterior del sistema está indefinido.

5.4.1.8 *Ordenación*

Gramática textual concreta

<ordenación> ::=
ORDERING

(se hace referencia a <ordenación> en § 5.2.2)

Semántica

La palabra clave para ordenación (ORDERING) es una notación taquigráfica para especificar explícitamente operadores de ordenación y un conjunto de ecuaciones de ordenación para una definición parcial de tipo.

Modelo

Una <definición parcial de tipo> que introduce un género denominado S con la palabra clave ORDERING implica un conjunto de *signaturas-de-operador* equivalente a las definiciones explícitas:

```
"<"   : S,S -> Boolean;
">"   : S,S -> Boolean;
"<="  : S,S -> Boolean;
">="  : S,S -> Boolean;
```

donde Boolean es el género booleano predefinido, e implica también los *axiomas* booleanos:

```
FOR ALL a,b IN S
(
    "<"(a,a) = = False;
    "<"(a,b) = = ">"(b,a);
    "<="(a,b) = = "OR"("<"(a,b),"="(a,b));
    ">="(a,b) = = "OR"(">"(a,b),"="(a,b));
    "<"(a,b) = > NOT("<"(b,a)) ;
    "<"(a,b) AND "<"(b,c) = > "<"(a,c) ;
);
```

Cuando una <definición parcial de tipo> incluye una lista de literales y la palabra clave ORDERING, las <signatura de literal>s se designan en orden ascendente, es decir

```
LITERALS A,B,C;
OPERATORS ORDERING;
```

implica A < B, B < C.

5.4.1.9 *Sintipos*

Un sintipo especifica un conjunto de valores de un género. Un sintipo utilizado como un género tiene la misma semántica que el género referenciado por el sintipo, salvo verificaciones de que unos valores pertenecen al conjunto de valores del género.

Gramática abstracta

Identificador-de-sintipo	=	*Identificador*
Definición-de-sintipo	::	*Nombre-de-sintipo* *Identificador-de-género-de-progenitor* *Condición-de-intervalo*
Nombre-de-sintipo	=	*Nombre*
Identificador-de-género-de-progenitor	=	*Identificador-de-género*

Gramática textual concreta

```
<sintipo> ::=
        <identificador de sintipo>

<definición de sintipo> ::=
        SYNTYPE
            <nombre de sintipo> = <identificador de género de progenitor>
            [ <asignación por defecto> ] [ CONSTANTS <condición de intervalo> ]
            ENDSYNTYPE [ <nombre de sintipo> ]
      | NEWTYPE <nombre de sintipo> [ <propiedades ampliadas> ]
            <expresión de propiedades> CONSTANTS <condición de intervalo>
            ENDNEWTYPE [ <nombre de sintipo> ]

<identificador de género de progenitor> ::=
        <género>
```

Un <sintipo> es una alternativa para un <género> (véase § 5.2.2).

Una <definición de sintipo> con la palabra clave SYNTYPE y "= <identificador de sintipo>" es una sintaxis derivada definida más adelante.

Una <definición de sintipo> con la palabra clave SYNTYPE en la sintaxis concreta corresponde a *definición-de-sintipo* en la sintaxis abstracta.

Una <definición de sintipo> con la palabra clave NEWTYPE puede distinguirse de una <definición parcial de tipo> por la inclusión de CONSTANTS <condición de intervalo>. Tal <definición de sintipo> es una notación taquigráfica para introducir una <definición parcial de tipo> con un nombre anónimo seguido de una <definición de sintipo> con la palabra clave SYNTYPE basada en este género anónimamente denominado. Esto es

```
        NEWTYPE X /* detalles */
            CONSTANTS /* lista de constantes */
        ENDNEWTYPE X;
```

es equivalente a

```
        NEWTYPE anon /* detalles */
        ENDNEWTYPE anon;
```

seguido de

```
        SYNTYPE X = anon
            CONSTANTS /* lista de constantes */
        ENDSYNTYPE X;
```

Cuando un <identificador de sintipo> se utiliza como un <argumento> en una <lista de argumentos> que define un operador, el género del argumento en una *lista-de-argumentos* es el *identificador-de-género-de-progenitor* del sintipo.

Cuando un <identificador de sintipo> se utiliza como un resultado de un operador, el género del *resultado* es el *identificador-de-género-de-progenitor* del sintipo.

Cuando un <identificador de sintipo> se utiliza como un calificador para un nombre, el *calificador* es el *identificador-de-género-de-progenitor* del sintipo.

El <nombre de sintipo> opcional dado al final de una <definición de sintipo> después de la palabra clave ENDSYNTYPE o ENDNEWTYPE debe ser el mismo que el <nombre de sintipo> especificado después de SYNTYPE o NEWTYPE respectivamente.

Si se usa la palabra clave SYNTYPE y se omite la <condición de intervalo>, todos los valores del género están en la condición de intervalo de modo que el <identificador de sintipo> tiene exactamente la misma semántica que el identificador de género, y la condición de intervalo es siempre verdadera.

Semántica

Una definición de sintipo define un sintipo que hace referencia a un identificador de género y una condición de intervalo. Especificar un especificador de sintipo es lo mismo que especificar el identificador de género de progenitor del sintipo, con excepción de los casos siguientes:

a) asignación a una variable declarada con un sintipo (véase § 5.5.3),

b) una salida de una señal si uno de los géneros especificados para la señal es un sintipo (véase § 2.7.4),

c) llamada a un procedimiento cuando uno de los géneros especificados para las variables de parámetro IN del procedimiento es un sintipo (véase § 2.4.5),

d) creación de un proceso cuando uno de los géneros especificados para los parámetros del proceso es un sintipo (véase § 2.7.2 y § 2.4.4),

e) entrada de una señal y una de las variables que está asociada a la entrada tiene un género que es un sintipo (véase § 2.6.4),

f) utilización, en una expresión, de un operador que tiene un sintipo definido bien como un género de argumento o como un género de resultado (véase § 5.4.2.2 y § 5.5.2.4),

g) una sentencia de inicializar o reinicializar un temporizador y uno de los géneros en la definición de temporizador es un sintipo (véase § 2.8),

h) una definición de importación (véase § 4.13),

Por ejemplo una <definición de sintipo> con la palabra clave SYNTYPE y "= <identificador de sintipo>" es equivalente a sustituir el <identificador de género de progenitor> por el <identificador de género de progenitor> de la <definición de sintipo> del <identificador de sintipo>. Es decir

```
SYNTYPE s2 = n1 CONSTANTS a1:a3; ENDSYNTYPE s2;
SYNTYPE s3 = s2 CONSTANTS a1:a2; ENDSYNTYPE s3;
```

es equivalente a

```
SYNTYPE s2 = n1 CONSTANTS a1:a3; ENDSYNTYPE s2;
SYNTYPE s3 = n1 CONSTANTS a1:a2; ENDSYNTYPE s3;
```

Cuando un sintipo se especifica en términos de <identificador de sintipo>, los dos sintipos no pueden estar mutuamente definidos.

Un sintipo definido por una definición de sintipo tiene una identidad que es el nombre introducido por el nombre de sintipo calificado por la identidad de la unidad de ámbito circundante.

Un sintipo tiene un género que es el género identificado por el identificador de género de progenitor dado en la definición de sintipo.

Un sintipo tiene un intervalo que es el conjunto de valores especificados por las constantes de la definición de sintipo.

5.4.1.9.1 *Condición de intervalo*

Gramática abstracta

Condición-de-intervalo	::	*Identificador-de-operador-O* *Ítem-de-condición*-**set**
Ítem-de-condición	=	*Intervalo-abierto* | *Intervalo-cerrado*
Intervalo-abierto	::	*Identificador-de-operador* *Expresión-fundamental*
Intervalo-cerrado	::	*Identificador-de-operador-Y* *Intervalo-abierto* *Intervalo-abierto*
Identificador-de-operador-O	=	*Identificador*
Identificador-de-operador-Y	=	*Identificador*

Gramática textual concreta

```
<condición de intervalo> ::=
        { <intervalo cerrado> | <intervalo abierto> }{ , { <intervalo cerrado> | <intervalo abierto> }}*
<intervalo cerrado> ::=
        <constante> : <constante>
<intervalo abierto> ::=
        <constante>
    |   { = | /= | < | > | <= | >= } <constante>
<constante> ::=
        <expresión fundamental>
```

El símbolo "<" ("<=", ">", ">=" respectivamente) sólo debe utilizarse en la sintaxis concreta de la <condición de intervalo> si dicho símbolo ha sido definido con una <signatura de operador>

P, P -> Boolean;

donde P es el género del sintipo. Estos símbolos representan *identificador-de-operador.*

Un <intervalo cerrado> sólo debe utilizarse si el símbolo "<=" está definido con una <signatura de operador>

P, P -> Boolean;

donde P es el género del sintipo.

Una <constante> en una <condición de intervalo> tiene que tener el mismo género que el del sintipo.

Semántica

Una condición de intervalo define una verificación de intervalo. Una verificación de intervalo se utiliza cuando un sintipo tiene semántica adicional al género del sintipo (véase § 5.4.1.9 y los casos de sintipos con semántica diferente – véase § 5.5.3, § 2.6.4, § 2.7.2, § 2.5.4, § 5.4.2.2 y § 5.5.4). Una verificación de intervalo se utiliza también para determinar la interpretación de una decisión (véase § 2.7.5).

La verificación de intervalo es la aplicación del operador formado a partir de la condición de intervalo. La aplicación de este operador tiene que ser equivalente a verdadero; de no ser así, el comportamiento ulterior del sistema está indefinido. La verificación de intervalo se deriva como sigue:

a) Cada elemento (<intervalo abierto> o <intervalo cerrado>) en la <condición de intervalo> tiene un *intervalo abierto* o *intervalo cerrado* correspondiente en el *ítem de condición.*

b) Un <intervalo abierto> de la forma <constante> es equivalente a un <intervalo abierto> de la forma = <constante>.

c) Para un término dado, A, entonces

   i) un <intervalo abierto> de la forma = <constante>, /= <constante>, <<constante>, <= <constante>, > <constante>, y >= <constante>, tiene subtérminos en la verificación de intervalo de la forma A = <constante>, A /= <constante>, A < <constante>, A <= <constante>, A > <constante>, y A >= <constante> respectivamente.

   ii) un <intervalo cerrado> de la forma <primera constante> : <segunda constante> tiene un subtérmino en la verificación de intervalo de la forma <primera constante> <= A AND A <= <segunda constante> donde AND corresponde al operador AND booleano y corresponde al *identificador-de-operador-Y* en la sintaxis abstracta.

d) Hay un *identificador-de-operador-O* para el operador distribuido entre todos los elementos del *ítem-de-condición*-**set** que es una unión booleana (OR) de todos los elementos. La verificación de intervalo es el término formado a partir de la unión booleana (OR) de todos los subtérminos derivados de la <condición de intervalo>.

Si un sintipo se especifica sin una <condición de intervalo>, la verificación de intervalo es Verdadero.

5.4.1.10 *Géneros estructura*

Gramática textual concreta

<definición de estructura> ::=
STRUCT <lista de campos> [<fin>] [ADDING]

<lista de campos> ::=
<campos> { <fin> <campos> }*

<campos> ::=
<nombre de campo> {, <nombre de campo> }* <género de campo>

<género de campo> ::=
<género>

Cada <nombre de campo> de un género estructura tiene que ser diferente de cualquier otro <nombre de campo> de la misma <definición de estructura>.

Semántica

Una definición de estructura define un género estructura cuyos valores están compuestos a partir de una lista de valores de campo de géneros.

La longitud de la lista de valores está determinada por la definición de estructura, y el género de un valor está determinado por su posición en la lista de valores.

Modelo

Una definición de estructura es sintaxis derivada para la definición de:

a) un operador, Make! (¡hacer!), para crear valores de estructura, y

b) operadores para modificar valores de estructura y para extraer valores de campo a partir de valores de estructura.

El nombre del operador implicado para modificar un campo es el nombre de campo concatenado con «Modify!» (¡modificar!).

El nombre del operador implicado para extraer un campo es el nombre de campo concatenado con «Extract!» (¡extraer!).

La <lista de argumentos> para el operador Make! es la lista de <género de campo>s que aparecen en la lista de campos, y en el orden de aparición de los mismos.

El <resultado> para el operador Make! es el identificador de género de la estructura.

La <lista de argumentos> para el operador de modificación de campo es el identificador de género de la estructura seguido por el <género de campo> de ese campo. El <resultado> para un operador de modificación de campo es el identificador de género de la estructura.

La <lista de argumentos> para un operador de extracción de campo es el identificador de género de la estructura. El <resultado> para un operador de extracción de campo es el <género de campo> de ese campo.

Hay una ecuación implícita para cada campo que define que modificar un campo de una estructura para darle un valor es lo mismo que construir un valor de estructura con ese valor para el campo.

Hay una ecuación implícita para cada campo que define que extraer un campo de un valor de estructura da el valor asociado con ese campo cuando la estructura fue construida.

Por ejemplo

```
NEWTYPE s STRUCT
    b  Boolean;
    i  Integer;
    c  Character;
ENDNEWTYPE    s;
```

implica

```
NEWTYPE s
OPERATORS
    Make!        : Boolean, Integer, Character    -> s;
    bModify!     : s, Boolean                     -> s;
    iModify!     : s, Integer                     -> s;
    cModify!     : s, Character                   -> s;
    bExtract!    : s                              -> Boolean;
    iExtract!    : s                              -> Integer;
    cExtract!    : s                              -> Character;
AXIOMS
    bModify!     (Make!(x,y,z),b)                 == Make!(b,y,z);
    iModify!     (Make!(x,y,z),i)                 == Make!(x,i,z);
    cModify!     (Make!(x,y,z),c)                 == Make!(x,y,c);
    bExtract!    (Make!(x,y,z))                   == x;
    iExtract!    (Make!(x,y,z))                   == y;
    cExtract!    (Make!(x,y,z))                   == z;
ENDNEWTYPE s;
```

5.4.1.11 *Herencia*

Gramática textual concreta

<regla de herencia> ::=
    INHERITS <género progenitor> [<redenominación de literal>]
        [[OPERATORS] { ALL | (<lista de herencias>) } [<fin>]] [ADDING]

<género progenitor> ::=
    <género>

<lista de herencias> ::=
    <operador heredado> {, <operador heredado> }*

<operador heredado> ::=
    [<nombre de operador> =] <nombre de operador heredado>

<nombre de operador heredado> ::=
    <nombre de operador de género progenitor>

<redenominación de literal> ::=
LITERALS <lista de redenominaciones de literal> <fin>

<lista de redenominaciones de literal> ::=
<par de redenominación de literal> {, <par de redenominación de literal> }*

<par de redenominación de literal> ::=
<signatura de redenominación de literal> = <signatura de redenominación de literal progenitor>

<signatura de redenominación de literal> ::=
<nombre de operador literal>
| <literal cadena de caracteres>

Un género no debe basarse circularmente en sí mismo por herencia.

Todas las <signatura de redenominación de literal>s en una <lista de redenominaciones de literal> tienen que ser distintas. Todas las <signatura de redenominación de literal progenitor>s en una <lista de redenominaciones de literal> tienen que ser diferentes.

Todos los <nombre de operador heredado>s en una <lista de herencias> tienen que ser distintos. Todos los <nombre de operador>s en una <lista de herencias> tienen que ser distintos.

Un <nombre de operador heredado> especificado en una <lista de herencias> tiene que ser un operador visible del <género progenitor> definido en la <definición parcial de tipo> que define el <género progenitor>. Un nombre de operador no es visible en este punto si está definido con una <admiración>.

Cuando varios operadores del <género progenitor> tienen el mismo nombre que el <nombre de operador heredado>, todos estos operadores son heredados.

Semántica

Un género puede basarse en otro género utilizando NEWTYPE en combinación con una regla de herencia. El género definido utilizando la regla de herencia es disjunto del género progenitor.

Si el género progenitor tiene literales definidos, los nombres de literal se heredan como nombres para literales del género, exceptuando los literales a los que se haya aplicado una redenominación de literal. Una redenominación de literal ha tenido lugar para un literal si el nombre de literal progenitor aparece como segundo nombre en un par de redenominación de literal, en cuyo caso el literal es redenominado con el primer nombre de ese par.

Hay un operador heredado para todo operador del género progenitor excepto "=" y "/=". Un operador del género progenitor es cualquier operador que, a la vez:

a) está definido por cualquier definición parcial de tipo o definición de sintipo (exceptuada la que se está definiendo) que define un género visible en el punto de herencia, y

b) tiene el género progenitor bien como un argumento o como un resultado.

Los nombres de operadores se heredan como se especifica por ALL o la lista de herencias. El nombre de un operador heredado es:

a) el mismo que el nombre del operador de género progenitor si se especifica ALL y el nombre está explícita o implícitamente definido como un nombre de operador en la definición parcial de tipo o definición de sintipo que define el género progenitor; de no ser así,

b) si el identificador de operador progenitor está dado en la lista de herencias y un nombre de operador seguido por "=" está dado para el operador heredado, entonces es redenominado con ese nombre; de no ser así,

c) si el identificador de operador progenitor está dado en la lista de herencias y un nombre de operador seguido por "=" no está dado para el operador heredado, entonces el nombre es el mismo nombre que el del operador de género progenitor; de no ser así,

d) si no se especifica ALL y el identificador de operador progenitor no se menciona en la lista de herencias, entonces se efectúa la redenominación con un nombre invisible pero único. Tales nombres no pueden utilizarse explícitamente en axiomas ni en expresiones.

Los géneros de argumento y resultado de un operador heredado son los mismos que los del operador correspondiente del género progenitor, salvo si el género de argumento o resultado es el género progenitor, en cuyo caso se modifica de modo que sea el género que se está definiendo. Esto es, toda ocurrencia del género progenitor en los operadores heredados se cambia al nuevo género.

De cada ecuación del género progenitor se deriva por herencia una ecuación. Las ecuaciones del género progenitor son:

a) cualquier ecuación que contenga un operador (o literal) del género progenitor, y también

b) cualquier ecuación definida por cualquier definición parcial de tipo o definición de sintipo (excepto el que se está definiendo) que defina un género visible en el punto de herencia.

Una ecuación heredada es la misma que la ecuación correspondiente del género progenitor, con las siguientes salvedades:

a) toda ocurrencia del género progenitor se cambia al nuevo género, y

b) los operadores (o literales) del género progenitor que tienen operadores (o literales) heredados redenominados sufren la misma redenominación en la ecuación heredada.

Como consecuencia de los cambios de géneros mencionados en a), las identidades de los literales heredados y las identidades de los operadores heredados se cambian de modo que sean calificadas por la identidad de género del nuevo género.

Modelo

La sintaxis concreta de una <regla de herencia> está relacionada con la sintaxis concreta de la <expresión de propiedades> en la <definición parcial de tipo> o <definición de sintipo> que contiene la <regla de herencia>.

El conjunto de <literal>s del nuevo género en la sintaxis abstracta corresponde al conjunto de <signatura de literal>s en la <expresión de propiedades> más el conjunto de literales heredados.

El conjunto de <operador>s del nuevo género en la sintaxis abstracta corresponde al conjunto de <signatura de operador>s en la <expresión de propiedades> más el conjunto de operadores heredados.

El conjunto de <ecuaciones> del nuevo género en la sintaxis abstracta corresponde a los <axiomas> de la <expresión de propiedades> más el conjunto de ecuaciones heredadas.

Ejemplo

```
NEWTYPE     bit
    INHERITS  Boolean
        LITERALS      1 = True, 0 = False;
        OPERATORS ("NOT", "AND", "OR")
    ADDING
    OPERATORS
        EXOR: bit,bit -> bit;
    AXIOMS/* nota – se utilizan aquí 2 formas diferentes de escribir NOT */
        EXOR(a,b) = = (a AND "NOT"(b)) OR (NOT a AND b);
ENDNEWTYPE bit;
```

5.4.1.12 *Generadores*

Un generador permite definir una plantilla de texto parametrizada que se expande por instanciación antes de considerar la semántica de los tipos de datos.

5.4.1.12.1 *Definición de generador*

Gramática textual concreta

<definición de generador> ::=
GENERATOR <nombre de generador> (<lista de parámetros de generador>) <texto de generador>
ENDGENERATOR [<nombre de generador>]

<texto de generador> ::=
[<instanciaciones de generador>] <expresión de propiedades>

<lista de parámetros de generador> ::=
<parámetro de generador> {, <parámetro de generador> }*

<parámetro de generador> ::=
{ TYPE | LITERAL | OPERATOR | CONSTANT }
<nombre formal de generador> {, <nombre formal de generador> }*

<nombre formal de generador> ::=
<nombre formal de generador>

<género de generador> ::=
<nombre formal de generador>
| <nombre de generador>

Un <nombre de generador> o <nombre formal de generador> sólo se utilizará en una <expresión de propiedades> si la <expresión de propiedades> está en un <texto de generador>.

En una <definición de generador> todos los <nombre formal de generador>s de la misma clase (TYPE, LITERAL, OPERATOR o CONSTANT) tienen que ser distintos. Un nombre de la clase de LITERAL tiene que ser distinto de cada uno de los nombres de la clase CONSTANT en la misma <definición de generador>. El <nombre de generador> después de la palabra clave GENERATOR tiene que ser distinto de todos los nombres de género en la <definición de generador> y también distinto de todos los <parámetro de generador>s TYPE de esa <definición de generador>.

Un <género de generador> sólo es válido si aparece como un <género ampliado> (véase § 5.2.2) en un <texto de generador> y el nombre es, bien el <nombre de generador> de esa <definición de generador>, bien un <nombre formal de generador> definido por esa definición.

Si un <género de generador> es un <nombre formal de generador>, tiene que estar definido de modo que sea de la clase TYPE.

El <nombre de generador> opcional después de ENDGENERATOR tiene que ser el mismo que el <nombre de generador> dado después de GENERATOR.

Un <nombre formal de generador> no debe utilizarse en un <calificador>. Un <nombre de generador> o <nombre formal de generador> no debe:

a) ser calificado, ni
b) ir seguido de una <admiración>, ni
c) ser utilizado en una <asignación por defecto>.

Semántica

Un generador denomina una pieza de texto que puede utilizarse en instanciaciones de generador.

Se considera que los textos de instanciaciones de generador dentro de un texto de generador se expanden en el punto de definición del texto de generador.

Cada parámetro de generador tiene una clase (TYPE, LITERAL, OPERATOR, o CONSTANT) especificada por las palabras clave TYPE, LITERAL, OPERATOR o CONSTANT respectivamente.

Modelo

El texto definido por una definición de generador sólo se relaciona con la sintaxis abstracta si el generador está instanciado. No hay sintaxis abstracta correspondiente para la definición de generador en el punto de definición.

Ejemplo

```
GENERATOR bag(TYPE item)
LITERALS empty;
OPERATORS
        put     : item, bag -> bag;
        count   : item, bag -> Integer;
        take    : item, bag -> bag;
AXIOMS
        take(i,put(i,b))        == b;
        take(i,empty)           == ERROR!;
        count(i,empty)          == 0;
        count(i,put(j,b))       == count(i,b) + IF i=j THEN 1 ELSE 0 FI;
        put(i,put(j,b))         == put(j,put(i,b));
ENDGENERATOR bag;
```

Nota – La definición formal (anexo F.2) no admite el uso de <nombre formal de generador> en calificadores. La Recomendación fue corregida sobre el particular después de la impresión del anexo F.2, de suerte que dicho anexo es inválido al respecto.

5.4.1.12.2 *Instanciación de generador*

Gramática textual concreta

```
<instanciaciones de generador> ::=
        { <instanciación de generador> [ <fin> ] [ADDING] }+

<instanciación de generador> ::=
        <identificador de generador> (<lista de generadores efectivos>)

<lista de generadores efectivos> ::=
        <generador efectivo> {, <generador efectivo> }*

<generador efectivo> ::=
        <género ampliado>
    |   <signatura de literal>
    |   <nombre de operador>
    |   <término fundamental>
```

Si la clase de un <parámetro de generador> es TYPE, el <generador efectivo> correspondiente tiene que ser un <género ampliado>.

Si la clase de un <parámetro de generador> es LITERAL, el <generador efectivo> correspondiente tiene que ser una <signatura de literal>.

Una <signatura de literal> que es un <literal clase de nombre> puede utilizarse como un <generador efectivo> únicamente si el correspondiente <nombre formal de generador> no aparece en los <axiomas>, o <correspondencia de literales> de la <expresión de propiedades> en el <texto de generador>.

Si la clase de un <parámetro de generador> es OPERATOR, el correspondiente <generador efectivo> tiene que ser un <nombre de operador>.

Si la clase de un <parámetro de generador> es CONSTANT, el correspondiente <generador efectivo> tiene que ser un <término fundamental>.

Si el <generador efectivo> es un <nombre formal de generador>, la clase del <nombre formal de generador> tiene que ser la misma que la clase del <generador efectivo>.

Semántica

El uso de una instanciación de generador en propiedades ampliadas o en un texto de generador denota instanciación del texto identificado por el identificador de generador. Un texto instanciado para literales, operadores y axiomas se forma a partir del texto de generador con

a) los parámetros del generador sustituidos por los parámetros de generador efectivo y

b) con el nombre del generador sustituido,

   i) si la instanciación de generador está en una definición parcial de tipo o definición de sintipo, por la identidad del género que está siendo definida por la definición parcial de tipo o definición de sintipo; de no ser así,

   ii) en el caso de instanciación de generador dentro de un generador, por el nombre de ese generador.

El texto instanciado para literales es el texto instanciado a partir de los literales en la expresión de propiedades del texto de generador, omitiendo la palabra clave LITERALS.

El texto instanciado para operadores es el texto instanciado a partir de la lista de operadores en la expresión de propiedades del texto de generador, omitiendo la palabra clave OPERATORS.

El texto instanciado para axiomas es el texto instanciado a partir de los axiomas en la expresión de propiedades del texto de generador, omitiendo la palabra clave AXIOMS.

Cuando hay más de una instanciación de generador en la lista de instanciaciones de generador, los textos instanciados para literales (operadores y axiomas) se forman concatenando el texto instanciado para los literales (operadores, axiomas, respectivamente) de todos los generadores en el orden en que aparecen en la lista.

El texto instanciado para literales es una lista de literales para la expresión de propiedades de la definición parcial de tipo, definición de sintipo o definición de generador circundante que ocurre antes de cualquier lista de literales explícitamente mencionada en la expresión de propiedades. Es decir, si se ha especificado una ordenación, los literales definidos por instanciaciones de generador aparecerán en el orden en que son instanciados y antes de cualesquiera otros literales.

Los textos instanciados para operadores y axiomas se agregan a la lista de operadores y axiomas respectivamente de la definición parcial de tipo, definición de sintipo o definición de generador circundante.

Cuando se añade texto instanciado a una expresión de propiedades, se considera que las palabras clave LITERALS, OPERATORS y AXIOMS se añaden, si es necesario, para crear una sintaxis concreta correcta.

Modelo

La sintaxis abstracta correspondiente a una instanciación de generador se determina después de la instanciación. La relación se determina a partir del texto instanciado, en el punto de instanciación.

Ejemplo

```
NEWTYPE boolbag bag(Boolean)
        ADDING
        OPERATORS
            yesvote : boolbag -> Boolean;
        AXIOMS
            yesvote(b) = = count(True,b) > count(False,b);
ENDNEWTYPE boolbag;
```

5.4.1.13 *Sinónimos*

Un sinónimo da un nombre a una expresión fundamental que representa uno de los valores de un género.

Gramática textual concreta

<definición de sinónimo> ::=
SYNONYM <nombre de sinónimo> [<género>] = <expresión fundamental>
| <definición de sinónimo externo>

La <definición de sinónimo externo> alternativa se describe en § 4.3.1.

Si el género de la <expresión fundamental> no puede determinarse de una manera inequívoca, hay que especificar un género en la <definición de sinónimo>.

El género identificado por el <género> tiene que ser uno de los géneros a los cuales puede estar vinculada la <expresión fundamental>.

La <expresión fundamental> no referirá al sinónimo definido por la <definición de sinónimo>, directa ni indirectamente (vía otro sinónimo).

Semántica

El valor que el sinónimo representa está determinado por el contexto en el cual aparece la definición de sinónimo.

Si el género de la expresión fundamental no puede determinarse de manera inequívoca en el contexto del sinónimo, entonces el género viene dado por el <género>.

Un sinónimo tiene un valor que es el del término fundamental en la definición de sinónimo.

Un sinónimo tiene un género que es el del término fundamental en la definición de sinónimo.

Modelo

La <expresión fundamental> en la sintaxis concreta denota un *término fundamental* en la sintaxis abstracta, como se define en § 5.4.2.2.

Si se especifica un <género>, el resultado de la <expresión fundamental> está vinculado a ese *género*. La <expresión fundamental> representa un *término fundamental* en la sintaxis abstracta que tiene un *identificador de operador* con el mismo nombre y los mismos géneros argumento indicados por la sintaxis concreta y el género resultado igual al género especificado en la sintaxis concreta.

5.4.1.14 *Literales clase de nombre*

Un literal clase de nombre es una notación taquigráfica para escribir un conjunto (que podría ser infinito) de nombres de literal definidos por una expresión regular.

Gramática textual concreta

```
<literal clase de nombre> ::=
        NAMECLASS <expresión regular>

<expresión regular> ::=
        <expresión regular parcial>
            { [OR] <expresión regular parcial> }*

<expresión regular parcial> ::=
        <elemento regular> [ <nombre de literal natural> | + |*]

<elemento regular> ::=
        (<expresión regular>)
    |   <literal cadena de caracteres>
    |   <intervalo regular>

<intervalo regular> ::=
        <literal cadena de caracteres> : <literal cadena de caracteres>
```

Los nombres formados por el <literal clase de nombre> tienen que satisfacer las condiciones estáticas normales para literales (véase § 5.2.2) y, bien las reglas sintácticas para nombres (véase § 2.21), bien la sintaxis concreta para <literal cadena de caracteres> (véase § 5.4.1.2).

Los <literal cadena de caracteres>s en un <intervalo regular> tienen que ser ambos, al mismo tiempo, literales de longitud 1, y literales definidos por el género carácter (véase § 5.6.2).

Semántica

Un literal clase de nombre es una forma alternativa de especificar signaturas de literal.

Modelo

El conjunto de nombres al cual es equivalente un literal clase de nombre se define como el conjunto de nombres que satisface la sintaxis especificada por la <expresión regular>. El literal clase de nombre es equivalente a este conjunto de nombres en la sintaxis abstracta.

Una <expresión regular> que es una lista de <expresión regular parcial>s sin un OR especifica que los nombres pueden formarse a partir de los caracteres definidos por la primera <expresión regular parcial> seguidos de los caracteres definidos por la segunda <expresión regular parcial>.

Cuando se especifica un OR entre dos <expresión regular parcial>s, los nombres se forman a partir de la primera o la segunda de estas <expresión regular parcial>s. Obsérvese que OR vincula más estrechamente que una simple secuenciación, de modo que

NAMECLASS 'A' '0' OR '1' '2';

es equivalente a

NAMECLASS 'A' ('0' OR '1') '2';

y define los literales A02, A12.

Si un <elemento regular> va seguido de <nombre de literal natural> la <expresión regular parcial> es equivalente al <elemento regular> repetido el número de veces especificado por el <nombre de literal natural>.

Por ejemplo

NAMECLASS 'A' ('A' OR 'B')2

define nombres AAA, AAB, ABA, y ABB.

Si un <elemento regular> va seguido de un '*' la <expresión regular parcial> es equivalente al <elemento regular> repetido cero o más veces.

Por ejemplo:

NAMECLASS 'A' ('A' OR 'B')*

define nombres A, AA, AB, AAA, AAB, ABA, ABB, AAAA, ... etc.

Si un <elemento regular> va seguido de '+' la <expresión regular parcial> es equivalente al <elemento regular> repetido una o más veces.

Por ejemplo:

NAMECLASS 'A' ('A' OR 'B')+

define nombres AA, AB, AAA, AAB, ABA, ABB, AAAA, ... etc.

Un <elemento regular> que es una <expresión regular> encerrada entre paréntesis define las secuencias de caracteres definidas por la <expresión regular>.

Un <elemento regular> que es un <literal cadena de caracteres> define la secuencia de caracteres dada en el literal cadena de caracteres (omitiendo las comillas).

Un <elemento regular> que es un <intervalo regular> define todos los caracteres especificados por el <intervalo regular> como secuencias de caracteres alternativas. Los caracteres definidos por el <intervalo regular> son, todos ellos, mayores que o iguales al primer carácter y menores que o iguales al segundo carácter de acuerdo con la definición del género carácter (véase § 5.6.2). Por ejemplo

'a':'f'

define las alternativas 'a' o 'b' o 'c' o 'd' o 'e' o 'f'.

Si la secuencia de definición de los nombres es importante (por ejemplo si se especifica ORDERING), se considera que los nombres están definidos en orden, de modo que están clasificados alfabéticamente de acuerdo con la ordenación del género cadena de caracteres. Si dos nombres comienzan por los mismos caracteres pero tienen diferente longitudes, se considera que el nombre más corto está definido primero.

5.4.1.15 *Correspondencia de literales*

Las correspondencias de literales son notaciones taquigráficas utilizadas para definir la correspondencia de literales con valores.

Gramática textual concreta

<correspondencia de literales> ::=
    MAP <ecuación de literales> { <fin> <ecuación de literales> }* [<fin>]

<ecuación de literales> ::=
    <cuantificación de literales>
        (<axiomas de literales> { <fin> <axiomas de literales> }* [<fin>])

<axiomas de literales> ::=
    <ecuación>
  | <ecuación de literales>

<cuantificación de literales> ::=
    FOR ALL <nombre de valor> { , <nombre de valor> }* IN <género ampliado> LITERALS

<término de ortografía> ::=
    SPELLING (<identificador de valor>)

Las reglas <correspondencia de literales> y <término de ortografía> no forman parte del núcleo de datos pero ocurren en las reglas <expresión de propiedades> y <término fundamental>, en § 5.2.1 y § 5.2.3 respectivamente.

Semántica

La correspondencia de literales es una notación taquigráfica para definir un número grande (posiblemente infinito) de axiomas que pueden comprender todos los literales de un género. La correspondencia de literales permite establecer una relación de correspondencia entre los literales de un género y los valores del género. Cuando el género contiene un número grande (o infinito) de valores, una correspondencia de literales es el único modo práctico de definir el valor correspondiente a cada literal.

El mecanismo de término de ortografía se utiliza en correspondencia de literales para referirse a la cadena de caracteres que contiene la ortografía del literal. Este mecanismo permite utilizar operadores cadena-de-caracteres (Charstring) para definir correspondencias de literales.

Modelo

Una <correspondencia de literales> es una notación taquigráfica para un conjunto de <axiomas>. Este conjunto de <axiomas> se deriva de las <ecuación de literales>s en la <correspondencia de literales>. Las <ecuación>s que se utilizan para esta derivación son todas las <ecuación>s contenidas en <axiomas> de las reglas <axiomas de literales>. En cada una de estas <ecuación>s, los <identificador de valor>s definidos por el <nombre de valor> en la <cuantificación de literal> se remplazan. En cada <ecuación> derivada, cada ocurrencia del mismo <identificador de valor> se remplaza por el mismo <identificador de operador literal> del <género> de la <cuantificación de literales>. El conjunto derivado de <axiomas> contiene todas las posibles <ecuación>s que pueden derivarse de esta forma. Los <axiomas> derivados para <ecuación de literales>s se agregan a los <axiomas> (si existen) definidos después de la palabra clave AXIOMS y antes de la palabra clave MAP en la misma <definición parcial de tipo>.

Por ejemplo:

```
NEWTYPE abc LITERALS 'A',b,'c';
    OPERATORS
        "<" : abc,abc -> Boolean;
        "+" : abc,abc -> Boolean;
    MAP FOR ALL x,y IN abc LITERALS
        (x < y => y + x);
    ENDNEWTYPE abc;
```

es sintaxis concreta derivada para

```
NETWTYPE abc LITERALS 'A',b,'c';
    OPERATORS
        "<" : abc,abc -> Boolean;
        "+" : abc,abc -> Boolean;
    AXIOMS
        'A'     < 'A'     => 'A'     + 'A';
        'A'     < b       => b       + 'A';
        'A'     < 'c'     => 'c'     + 'A';
        b       < 'A'     => 'A'     + b;
        b       < b       => b       + b;
        b       < 'c'     => 'c'     + b;
        'c'     < 'A'     => 'A'     + 'c';
        'c'     < b       => b       + 'c';
        'c'     < 'c'     => 'c'     + 'c';
    ENDNEWTYPE abc;
```

Si una <cuantificación de literales> contiene uno o más <término de ortografía>s, existe una sustitución de los <término de ortografía>s por literales cadena-de-caracteres (véase § 5.6.3).

Si la <signatura de literal> del <identificador de operador literal> de un <término fundamental> es un <nombre de operador literal> (véase § 5.2.2), el <término de ortografía> es una notación taquigráfica de una cadena-de-caracteres en mayúsculas derivada del <identificador de operador literal>. La cadena-de-caracteres contiene la ortografía en mayúsculas del <nombre de operador literal> del <identificador de operador literal>.

Si la <signatura de literal> del <identificador de operador literal> de un <término de ortografía> es un <literal cadena de caracteres> (véase § 5.2.2 y § 5.4.1.2), el <término de ortografía> es una notación taquigráfica de una cadena-de-caracteres derivada del <literal cadena de caracteres>. La cadena-de-caracteres contiene la ortografía del <literal cadena de caracteres>.

La cadena-de-caracteres se utiliza para remplazar el <identificador de valor> después de que la <ecuación de literal> que contiene el <término de ortografía> es expandida como se ha indicado anteriormente.

Por ejemplo

```
NEWTYPE abc LITERALS 'A',Bb,'c';
    OPERATORS
        "<" : abc,abc -> Boolean;
    MAP FOR ALL x,y IN abc LITERALS
        SPELLING(x) < SPELLING(y) = > x < y;
    ENDNEWTYPE abc;
```

es sintaxis concreta derivada para

```
NETWTYPE abc LITERALS 'A',Bb,'c';
    OPERATORS
        "<" : abc,abc -> Boolean;
    AXIOMS
        /*obsérvese que 'A', Bb, 'c' están vinculados al género local abc */
        /* '''A''', 'BB' y '''c''' deben ser calificados por el identificador de cadena-de-caracteres si
        estos literales son ambiguos – por razones de brevedad, esto se ha omitido en lo que sigue*/
        '''A'''     < '''A''' = > 'A'     < 'A';
        '''A'''     < 'BB' = > 'A'        < Bb;
        '''A'''     < '''c''' = > 'A'     < 'c';
        'BB'        < '''A''' = > Bb      < 'A';
        'BB'        < 'BB' = > Bb         < Bb;
        'BB'        < '''c''' = > Bb      < 'c';
        '''c'''     < '''A''' = > 'c'     < 'A';
        '''c'''     < 'BB' = > 'c'        < Bb;
        '''c'''     < '''c''' = > 'c'     < 'c';
    ENDNEWTYPE abc;
```

Toda <ecuación incuantificada> en <axiomas de literales> tiene que contener un <término de ortografía> o un <identificador de operador literal>. Un <término de ortografía> tiene que estar en una <correspondencia de literales>.

El <identificador de valor> en un <término de ortografía> tiene que ser un <identificador de valor> definido por una <cuantificación de literales>.

5.4.2 *Uso de datos*

A continuación se define la manera de interpretar tipos de datos, géneros, literales, operadores y sinónimos en expresiones.

5.4.2.1 *Expresiones*

Expresiones son literales, operadores, accesos variables, expresiones condicionales y operadores imperativos.

Gramática abstracta

Expresión = *Expresión-fundamental* | *Expresión-activa*

Una *expresión* es una *expresión activa* si contiene un *primario activo* (véase § 5.5).

Una *expresión* que no contiene un *primario activo* es una *expresión fundamental.*

Gramática textual concreta

Para simplificar la descripción no se distingue entre la sintaxis concreta de *expresión fundamental* y *expresión activa.* La sintaxis concreta para <expresión> se da en § 5.4.2.2.

Semántica

Una expresión se interpreta como el valor de la expresión fundamental o expresión activa. Si el valor es un error, el comportamiento ulterior del sistema está indefinido.

La expresión tiene el género de la expresión fundamental o expresión activa.

5.4.2.2 *Expresiones fundamentales*

Gramática abstracta

Expresión-fundamental :: *Término-fundamental*

Las condiciones estáticas para el *término fundamental* son también aplicables a la *expresión fundamental.*

```
<expresión fundamental> ::=
        <expresión fundamental>

<expresión> ::=
        <operando0>
    |   <subexpresión> => <operando0>

<subexpresión> ::=
        <expresión>

<operando0> ::=
        <operando1>
    |   <suboperando0> { OR | XOR } <operando1>

<suboperando0> ::=
        <operando0>

<operando1> ::=
        <operando2>
    |   <suboperando1> AND <operando2>

<suboperando1> ::=
        <operando1>

<operando2> ::=
        <operando3>
    |   <suboperando2> { = | /= | > | >= | < | <= | IN } <operando3>

<suboperando2> ::=
        <operando2>

<operando3> ::=
        <operando4>
    |   suboperando3> { + | - | // } <operando4>

<suboperando3> ::=
        <operando3>

<operando4> ::=
        <operando5>
    |   <suboperando4> { * | / | MOD | REM } <operando5>

<suboperando4> ::=
        <operando4>

<operando5> ::=
        [ - | NOT ] <primario>

<primario> ::=
        <primario fundamental>
    |   <primario activo>
    |   <primario ampliado>

<primario fundamental> ::=
        <identificador de literal>
    |   <identificador de operador> ( <lista de expresiones fundamentales> )
    |   ( <expresiones fundamentales> )
    |   <expresión fundamental condicional>

<primario ampliado> ::=
        <sinónimo>
    |   <primario indizado>
    |   <primario de campo>
    |   <primario de estructura>

<lista de expresiones fundamentales> ::=
        <expresión fundamental> {, <expresión fundamental> }*

<identificador de operador> ::=
        <identificador de operador>
        |[<calificador>] <operador entre comillas>
```

Una <expresión> que no contiene ningún <primario activo> representa una *expresión fundamental* en la sintaxis abstracta. Una <expresión fundamental> no contendrá un <primario activo>.

Si una <expresión> es un <primario fundamental> con un <identificador de operador> y un <género de argumento> de la <signatura de operador> es un <sintipo>, la verificación de intervalo para ese sintipo, definida en § 5.4.1.9.1, se aplica al correspondiente valor de argumento. El valor de la verificación de intervalo tiene que ser Verdadero.

Si una <expresión> es un <primario fundamental> con un <identificador de operador> y el <género resultado> de la <signatura de operador> es un <sintipo>, la verificación de intervalo para ese sintipo, definida en § 5.4.1.9.1, se aplica al valor de resultado. El valor de la verificación de intervalo tiene que ser Verdadero.

Si una <expresión> contiene un <primario ampliado> (que es un <sinónimo>, <primario indizado>, <primario de campo> o <primario de estructura>), se remplaza en el nivel de sintaxis concreta como se define en § 5.4.2.3, § 5.4.2.4, § 5.4.2.5 y § 5.4.2.6 respectivamente antes de considerar la relación con la sintaxis abstracta.

El <calificador> opcional antes de un <operador entre comillas> tiene la misma relación con la sintaxis abstracta que un <calificador> de un <identificador de operador> (véase § 5.2.2).

Semántica

Una expresión fundamental se interpreta como el valor denotado por el término fundamental sintácticamente equivalente a la expresión fundamental.

En general, no hay necesidad ni motivo para distinguir entre el término fundamental y el valor del término fundamental. Por ejemplo, el término fundamental para el valor de entero unidad puede escribirse "1". De ordinario hay varios términos fundamentales que denotan el mismo valor, por ejemplo los términos fundamentales enteros "0+1", "3-2" y "(7+5)/12", y es usual considerar que el valor es denotado por una forma simple del término fundamental (en este caso "1").

El género de una expresión fundamental es el género del término fundamental equivalente.

El valor de una expresión fundamental es el valor del término fundamental equivalente.

5.4.2.3 *Sinónimo*

Gramática textual concreta

<sinónimo> ::=
    <identificador de sinónimo>
| <sinónimo externo>

La alternativa <sinónimo externo> se describe en § 4.3.1.

Semántica

Un sinónimo es una notación taquigráfica para denotar una expresión definida en otro lugar.

Modelo

Un <sinónimo> representa la <expresión fundamental> definida en la <definición de sinónimo> identificada por el <identificador de sinónimo>. Un <identificador> utilizado por la <expresión fundamental> representa un *identificador* en la sintaxis abstracta de acuerdo con el contexto de la <definición de sinónimo>.

5.4.2.4 *Primario indizado*

Un primario indizado es una notación sintáctica taquigráfica que puede utilizarse para denotar «indización» de un valor de «matriz». Sin embargo, aparte de la forma sintáctica especial, un primario indizado no tiene propiedades especiales y denota un operador que tiene como parámetro el primario.

Gramática textual concreta

<primario indizado> ::=
    <primario> (<lista de expresiones>)

Semántica

Una expresión indizada representa la aplicación de un operador Extract! (¡extraer!).

Modelo

Un <primario> seguido de una <lista de expresiones> entre paréntesis es sintaxis concreta derivada para la sintaxis concreta

Extract!(<primario>, <lista de expresiones>)

y se considera entonces una expresión lícita aunque Extract! no esté autorizado como un nombre de operador en la sintaxis concreta para expresiones. La sintaxis abstracta se determina a partir de esta expresión concreta de acuerdo con § 5.4.2.2.

5.4.2.5 *Primario de campo*

Un primario de campo es una notación sintáctica taquigráfica que puede utilizarse para denotar «selección de campo» de «estructuras». Sin embargo, aparte de la forma sintáctica especial, un primario de campo no tiene propiedades especiales y denota un operador que tiene como parámetro el primario.

Gramática textual concreta

<primario de campo> ::=
<primario> <selección de campo>

<selección de campo> ::=
!<nombre de campo>
| (<nombre de campo> { , <nombre de campo>}*)

El nombre de campo tiene que ser un nombre de campo definido para el género del primario.

Semántica

Un primario de campo representa la aplicación de uno de los operadores extraer campo de un género estructurado.

Modelo

La forma
<primario> (<nombre de campo>)
es sintaxis derivada para
<primario> ! <nombre de campo>

La forma
<primario> (<nombre de primer campo> { , <nombre de campo>}*)
es sintaxis derivada para
<primario> ! <nombre de primer campo> { ! <nombre de campo> }*
donde se preserva el orden de los nombres de campo.

La forma
<primario> ! <nombre de campo>
es sintaxis derivada para
<nombre de operador extraer campo> (<primario>)

donde el nombre del operador extraer campo se ha formado por la concatenación del nombre de campo y «Extract!» en ese orden. Por ejemplo
s ! f1
es sintaxis derivada para
f1Extract!(s)

y se considera entonces que ésta es una expresión lícita aunque f1Extract! no esté autorizado como un nombre de operador en la sintaxis concreta para expresiones. La sintaxis abstracta se determina a partir de esta expresión concreta de acuerdo con § 5.4.2.2.

En el caso en que hay un operador definido para un género de modo que
Extract!(s,nombre)

es un término válido cuando «nombre» es el mismo que un nombre de campo válido del género de s, entonces un primario
s(nombre)

es sintaxis concreta derivada para
Extract!(s,nombre)

y la selección de campo debe escribirse
s ! nombre

5.4.2.6 *Primario de estructura*

Gramática textual concreta

<primario de estructura> ::=
[<calificador>] (. <lista de expresiones> .)

Semántica

Un primario de estructura representa un valor de un género estructurado que está construido a partir de expresiones para cada campo de la estructura.

La forma
(. <lista de expresiones> .)

es sintaxis concreta derivada para

Make!(<lista de expresiones>)

y se considera que ésta es una expresión fundamental lícita aunque Make! no esté autorizado como un nombre de operador en la sintaxis concreta para expresiones fundamentales. La sintaxis abstracta se determina a partir de esta expresión fundamental concreta de acuerdo con § 5.4.2.2.

5.4.2.7 *Expresión fundamental condicional*

Gramática textual concreta

<expresión fundamental condicional> ::=
IF <expresión fundamental booleana>
THEN <expresión fundamental de consecuencia>
ELSE <expresión fundamental de alternativa>
FI

<expresión fundamental de consecuencia> ::=
<expresión fundamental>

<expresión fundamental de alternativa> ::=
<expresión fundamental>

La <expresión fundamental condicional> representa una *expresión fundamental* en la sintaxis abstracta. Si la <expresión fundamental booleana> representa Verdadero, entonces la *expresión fundamental* está representada por una <expresión fundamental de consecuencia>; de no ser así, está representada por la <expresión fundamental de alternativa>.

El género de la <expresión fundamental de consecuencia> tiene que ser el mismo que el género de la <expresión fundamental de alternativa>.

Semántica

Una expresión fundamental condicional es un primario fundamental que se interpreta, bien como la expresión fundamental de consecuencia, bien como la expresión fundamental de alternativa.

Si la <expresión fundamental booleana> tiene el valor Verdadero, la <expresión fundamental de alternativa> no se interpreta. Si la <expresión fundamental booleana> tiene el valor Falso, la <expresión fundamental de consecuencia> no se interpreta. Si la <expresión fundamental> que se interpreta tiene el valor de un error, el comportamiento ulterior del sistema está indefinido.

El género de una expresión fundamental condicional es el género de la expresión fundamental de consecuencia (y también el género de la expresión fundamental de alternativa).

5.5 *Uso de datos con variables*

Esta sección define el uso de datos y variables declaradas en procesos y procedimientos, así como los operadores imperativos que obtienen valores a partir del sistema subyacente.

Una variable tiene un género y un valor asociado de ese género. El valor asociado a una variable puede cambiarse asignando un nuevo valor a la variable. El valor asociado a la variable puede utilizarse en una expresión accediendo a la variable.

Una expresión que contiene una variable se considera «activa» pues el valor obtenido al interpretar la expresión puede variar de conformidad con el último valor asignado a la variable.

5.5.1 *Definiciones de variables y de datos*

Gramática textual concreta

<definición de datos> ::=
{ <definición parcial de tipo>
| <definición de sintipo>
| <definición de generador>
| <definición de sinónimo> } <fin>

Una definición de datos forma parte de una *definición de tipo de datos* si es una <definición parcial de tipo> o <definición de sintipo>, definidas en § 5.2.1 y § 5.4.1.9 respectivamente. Las reglas <definición de generador> y <definición de sinónimo> se definen en § 5.4.1.12 y § 5.4.1.13 respectivamente.

La sintaxis para introducir variables de proceso y para variables de parámetros de procedimiento se indica en § 2.5.1.1 y § 2.3.4 respectivamente. Una variable definida en un procedimiento no será revelada.

Semántica

Una definición de datos se utiliza sea para la definición de una parte de un tipo de datos, sea para la definición de un sinónimo para una expresión, tal como se define más detalladamente en § 5.2.1, § 5.4.1.9 o § 5.4.1.13.

Una variable, cuando es creada, contiene un valor especial denominado indefinido, que es distinto de todo otro valor del género de esa variable.

5.5.2 *Acceder a variables*

A continuación se define la manera de interpretar una expresión que comprende variables.

5.5.2.1 *Expresiones activas*

Gramática abstracta

Expresión activa = *Acceso-a-variable* |
Expresión-condicional |
Aplicación-de-operador |
Operador-imperativo

Gramática textual concreta

<expresión activa> ::=
<expresión activa>

<primario activo> ::=
<acceso a variable>
| <aplicación de operador>
| <expresión condicional>
| <operador imperativo>
| (<expresión activa>)
| <primario ampliado activo>

<primario ampliado activo> ::=
<primario ampliado activo>

<lista de expresiones> ::=
<expresión> { , <expresión> }*

Por razones de brevedad, la sintaxis concreta para <expresión activa> se da como <expresión> en § 5.4.2.2. Una <expresión> es una <expresión activa> si contiene un <primario activo>.

También por razones de brevedad, la sintaxis concreta para <primario ampliado activo> se da como <primario ampliado> en § 5.4.2.2. Un <primario ampliado> es un <primario ampliado activo> si contiene un <primario activo>. Para un <primario ampliado> se produce una sustitución en el nivel de sintaxis concreta como se define en § 5.4.2.3, § 5.4.2.4, § 5.4.2.5 y § 5.4.2.6 antes de considerar la relación con la sintaxis abstracta.

Semántica

Una expresión activa es una expresión cuyo valor dependerá del estado en que se encuentre el sistema.

El género de una expresión activa es el género del término fundamental equivalente.

El valor de una expresión activa es el término fundamental equivalente a la expresión activa en el momento de la interpretación

Modelo

Cada vez que se interpreta la expresión activa, su valor se determina hallando el término fundamental equivalente a la expresión activa. Este término fundamental se determina a partir de una expresión fundamental formada remplazando cada primario activo en la expresión activa por el término fundamental equivalente al valor de dicho primario activo. El valor de una expresión activa es el mismo que el de la expresión fundamental.

Dentro de una expresión activa, cada operador se interpreta en el orden determinado por la sintaxis concreta dada en § 5.4.2.2 o, en caso de ambigüedad, de izquierda a derecha. En una lista de expresiones activas o lista de expresiones, cada elemento de la lista se interpreta en el orden de izquierda a derecha.

5.5.2.2 *Acceso a variable*

Gramática abstracta

Acceso-a-variable = *Identificador-de-variable*

Gramática textual concreta

<acceso a variable> ::=
<identificador de variable>

Semántica

La interpretación de un acceso a variable da el valor asociado a la variable identificada.

Un acceso a variable tiene el género de la variable que él identifica.

El valor de un acceso a variable es el último valor asociado a la variable, o, si éste es el valor especial «indefinido», entonces es un error. Si el valor de un acceso a variable es un error, el comportamiento ulterior del sistema está indefinido.

5.5.2.3 *Expresión condicional*

Una expresión condicional es una expresión que se interpreta bien como la consecuencia, bien como la alternativa.

Gramática abstracta

Expresión-condicional	::	*Expresión-booleana* *Expresión-de-consecuencia* *Expresión-de-alternativa*
Expresión-booleana	=	*Expresión*
Expresión-de-consecuencia	=	*Expresión*
Expresión-de-alternativa	=	*Expresión*

El género de la *expresión de consecuencia* tiene que ser igual al de la *expresión de alternativa.*

Gramática textual concreta

```
<expresión condicional>  ::=
        IF <expresión activa booleana>
            THEN <expresión de consecuencia>
            ELSE <expresión de alternativa>
            FI
    |   IF <expresión booleana>
            THEN <expresión de consecuencia activa>
            ELSE <expresión de alternativa>
            FI
    |   IF <expresión booleana>
            THEN <expresión de consecuencia>
            ELSE <expresión de alternativa activa>
            FI

<expresión de consecuencia> ::=
        <expresión>

<expresión de alternativa>  ::=
        <expresión>
```

Una <expresión condicional> se distingue de una <expresión fundamental condicional> por la ocurrencia de una <expresión activa> en la expresión condicional.

Semántica

Una expresión condicional se interpreta como la interpretación de la condición seguida, bien por la interpretación de la expresión de consecuencia, bien por la interpretación de la expresión de alternativa. La consecuencia sólo se interpreta si la condición tiene el valor Verdadero, de modo que, si la condición tiene el valor Falso, el comportamiento ulterior del sistema está indefinido solamente si la expresión de alternativa es un error. De manera similar, la alternativa sólo se interpreta si la condición tiene el valor Falso, de modo que si la condición tiene el valor Verdadero el comportamiento del sistema está indefinido solamente si la expresión de consecuencia es un error.

El género de la expresión condicional es el género de la consecuencia y de la alternativa.

El valor de la expresión condicional es el valor de la consecuencia si la condición es Verdadero, o el valor de la alternativa si la condición es Falso.

5.5.2.4 *Aplicación de operador*

Una aplicación de operador es la aplicación de un operador, siendo uno o más de los argumentos efectivos una expresión activa.

Gramática abstracta

Aplicación-de-operador	::	*Identificador-de-operador* *Expresión*$^+$

Si un *género* argumento de la *signatura de operador* es un *sintipo* y la *expresión* correspondiente en la lista de *expresiones* es una *expresión fundamental,* la verificación de intervalo definida en § 5.4.1.9.1 aplicada al valor de la *expresión* tiene que ser Verdadero.

Gramática textual concreta

```
<aplicación de operador> ::=
        <identificador de operador> ( <lista de expresiones activas> )

<lista de expresiones activas>  ::=
        <expresión activa> [ , <lista de expresiones> ]
    |   <expresión fundamental> , <lista de expresiones activas>
```

Una <aplicación de operador> se distingue de la <expresión fundamental> sintácticamente similar porque una de las <expresión>s en la lista de <expresión>s entre paréntesis es una <expresión activa>. Si todas las <expresión>s entre paréntesis son <expresión fundamental>s, la construcción representa una *expresión fundamental*, definida en § 5.4.2.2.

Semántica

Una aplicación de operador es una expresión activa que tiene el valor del término fundamental equivalente a la aplicación de operador. El término fundamental equivalente se determina como se indica en § 5.5.2.1.

La lista de expresiones para la aplicación de operador se interpreta en el orden dado, antes de interpretar el operador.

Si un género argumento de la signatura de operador es un sintipo y la expresión correspondiente en la lista de expresiones activas es una expresión activa, la verificación de intervalo definida en § 5.4.1.9.1 se aplica al valor de la expresión. Si la verificación de intervalo es Falso en el momento de la interpretación, el sistema está en error y su comportamiento ulterior está indefinido.

Si el género resultado de la signatura de operador es un sintipo, la verificación de intervalo definida en § 5.4.1.9.1 se aplica al valor de la aplicación de operador. Si la verificación de intervalo es Falso en el momento de la interpretación, el sistema está en error y su comportamiento ulterior está indefinido.

5.5.3 *Sentencia de asignación*

Gramática abstracta

Sentencia-de-asignación :: *Identificador-de-variable*
Expresión

El género del *identificador de variable* y el género de la *expresión* deben ser iguales.

Si la *variable* está declarada con un *sintipo* y la *expresión* es una *expresión fundamental*, la verificación de intervalo definida en § 5.4.1.9.1 aplicada a la expresión tiene que ser Verdadero.

Gramática textual concreta

<sentencia de asignación> ::=
<variable> := <expresión>

<variable> ::=
<identificador de variable>
| <variable indizada>
| <variable de campo>

Si la <variable> es un <identificador de variable> la <expresión> en la sintaxis concreta representa la <expresión> en la sintaxis abstracta. Las otras formas de <variable>, <variable indizada> y <variable de campo> son sintaxis derivada, y la <expresión> en la sintaxis abstracta se halla a partir de la sintaxis concreta equivalente definida en § 5.5.3.1 y § 5.5.3.2.

Semántica

La interpretación de una sentencia de asignación crea una asociación de la variable identificada en la sentencia de asignación con el valor de la expresión en la sentencia de asignación. La asociación anterior de la variable desaparece.

Si la variable está declarada con un sintipo y la expresión es una expresión activa, la verificación de intervalo definida en § 5.4.1.9.1 se aplica a la expresión. Si esta verificación de intervalo es equivalente a Falso, la asignación es errónea y el comportamiento ulterior del sistema está indefinido.

5.5.3.1 *Variable indizada*

Una variable indizada es una notación sintáctica taquigráfica que puede utilizarse para denotar «indización» de «matrices». Sin embargo, aparte de la forma sintáctica especial, un primario activo indizado no tiene propiedades especiales y denota un operador con el primario activo como parámetro.

Gramática textual concreta

<variable indizada> ::=
<variable> (<lista de expresiones>)

Debe haber una definición apropiada de un operador denominado Modify!.

Semántica

Una variable indizada representa la asignación de un valor formado por la aplicación del operador Modify! a un acceso de la variable y la expresión dada en la variable indizada.

Modelo

La forma de sintaxis concreta

<variable> (<lista de expresiones>) := <expresión>

es sintaxis concreta derivada para

<variable> := Modify!(<variable>,<lista de expresiones>,<expresión>)

donde se repite la misma <variable> y se considera que el texto es una asignación lícita aunque Modify! no esté autorizado como nombre de operador en la sintaxis concreta para expresiones. La sintaxis abstracta se determina para esta <sentencia de asignación> de acuerdo con § 5.5.3.

El modelo para variables indizadas debe aplicarse antes del modelo para importación (véase § 4.13).

5.5.3.2 *Variable de campo*

Una variable de campo es una notación taquigráfica para asignar un valor a una variable de modo que sólo cambie el valor de un campo de esa variable.

Gramática textual concreta

<variable de campo> ::=

<variable> <selección de campo>

Debe haber una definición apropiada de un operador denominado Modify!. Normalmente esta definición estará implicada por una definición de género estructurada.

Semántica

Una variable de campo representa la asignación de un valor formado por la aplicación de un operador de modificar campo.

Modelo

Una selección de campo entre paréntesis es sintaxis derivada para ! <nombre de campo> en la selección de campo definida en § 5.4.2.5

La forma de sintaxis concreta

<variable> ! <nombre de campo> := <expresión>

es sintaxis concreta derivada para

<variable> := <nombre de operador de modificación de campo> (<variable>, <expresión>)

donde

a) se repite la misma <variable>, y

b) el <nombre de operador de modificación de campo> se forma a partir de la concatenación del nombre de campo y «Modify!», después de lo cual,

c) se considera que el texto es una asignación lícita aunque el <nombre de operador de modificación de campo> no esté autorizado como nombre de operador en la sintaxis concreta para expresiones.

Si hay más de un <nombre de campo> en la selección de campo, se les modela como se ha indicado antes expandiendo cada ! <nombre de campo>, uno por uno, de derecha a izquierda y considerando la parte restante de la <variable de campo> como una <variable>. Por ejemplo

var ! campoa ! campob := expresión;

es modelado primeramente por

var ! campoa := campobModify!(var ! campoa, expresión);

y después por

var := campoaModify!(var, campobModify!(var ! campoa, expresión));

La sintaxis abstracta se determina para la <sentencia de asignación> formada mediante el modelado conforme a § 5.5.3.

5.5.3.3 *Asignación por defecto*

Una asignación por defecto es una notación taquigráfica para asignar el mismo valor a todas las variables de un género especificado inmediatamente después de haberse creado aquéllas.

Gramática textual concreta

<asignación por defecto> ::=

DEFAULT <expresión fundamental> [<fin>]

Una <definición parcial de tipo> o <definición de sintipo> contendrá no más de una <asignación por defecto>. (Así se evita que surjan múltiples asignaciones a partir de instanciaciones de generador.)

Semántica

Una asignación por defecto se añade opcionalmente a una expresión de propiedades de un género. Una asignación por defecto especifica que a cualquier variable declarada con el género introducido por la definición parcial de tipo o definición de sintipo se le asigna inmediatamente el valor de la expresión fundamental.

Si no hay asignación por defecto cuando se declara una variable, se asociará a ésta el valor indefinido.

Cuando se declara una variable puede asignársele un valor alternativo incluyendo una asignación explícita en la declaración.

Las asignaciones por defecto no se heredan.

Modelo

La forma de sintaxis concreta

DEFAULT <expresión fundamental>

utilizada en una expresión de propiedades en que se introduce el género s implica una asignación de la <expresión fundamental> a una variable. Esta asignación se interpreta inmediatamente después de la declaración de la variable y antes de interpretar cualquier acción especificada explícitamente en el mismo proceso o procedimiento.

Por ejemplo, si

DEFAULT 2*dnumber

se da para género s y hay una declaración en la sintaxis concreta

DCL v s;

hay entonces una asignación implícita

v := 2*dnumber;

Si la declaración tiene también un <valor inicial>, el <valor inicial> se asigna a la variable después de la <expresión fundamental> en la <asignación por defecto>.

La sentencia de asignación implicada tiene la relación normal de una <sentencia de asignación> con la sintaxis abstracta (véase § 5.5.3).

Si se especifica una <asignación por defecto> para una <definición de datos>, el <género> (que representa un sintipo o género) tiene un valor de asignación por defecto que es el valor de la <expresión fundamental> de la <asignación por defecto>. Si no hay <asignación por defecto> en una <definición de sintipo>, el sintipo tiene un valor de asignación por defecto si el *identificador-de-género-progenitor* (que identifica un sintipo o género) dado en la *definición-de-sintipo* tiene un valor de asignación por defecto.

Para una <definición de sintipo>, las asignaciones se interpretan únicamente si la verificación de intervalo definida en § 5.4.1.9.1 da Verdadero cuando se aplica al valor de asignación por defecto. Esto es, para cada variable del sintipo hay una decisión implícita de la forma

DECISION <verificación de intervalo>;
(Verdadero) : <asignación por defecto>
ELSE: ENDDECISION.

5.5.4 *Operadores imperativos*

Los operadores imperativos obtienen valores del estado del sistema subyacente.

Gramática abstracta

Operador-imperativo	=	*Expresión-ahora* |
		Expresión-de-PId |
		Expresión-de-visión |
		Expresión-activa-de-temporizador

Gramática textual concreta

<operador imperativo> ::=
<expresión ahora>
| <expresión de importación>
| <expresión de PId>
| <expresión de visión>
| <expresión activa de temporizador>

La alternativa <expresión de importación> se define en § 4.13.

Los operadores imperativos son expresiones para verificar si los temporizadores están activos, o para acceder al reloj de sistema, los valores PId asociados con un proceso, o variables importadas.

5.5.4.1 *NOW*

Gramática abstracta

Expresión-ahora :: ()

Gramática textual concreta

<expresión ahora> ::=
NOW

Semántica

La expresión ahora es una expresión que accede a una variable de reloj de sistema para determinar el tiempo absoluto de sistema.

La expresión ahora representa una expresión que solicita el valor corriente del reloj de sistema que da la hora. El origen y la unidad de tiempo dependen del sistema. El hecho de que dos ocurrencias de NOW en la misma transición den el mismo valor depende del sistema.

Una expresión ahora es del género tiempo.

5.5.4.2 *Expresión IMPORT*

Gramática textual concreta

La sintaxis concreta para una expresión de importación se define en § 4.3.

Semántica

Además de la semántica definida en § 4.13, una expresión de importación se interpreta como un acceso a variable (véase § 5.5.2.2) a la variable implícita para la expresión de importación.

Modelo

La expresión de importación tiene una sintaxis implícita para la importación del valor, definida en § 4.13, y también un *acceso a variable* implícito de la variable implícita para la importación en el contexto en que aparece <expresión de importación>.

5.5.4.3 *Expresión de PId*

Gramática abstracta

Expresión-de-PId = *Expresión-de-mismo* |
Expresión-de-progenitor |
Expresión-de-vástago |
Expresión-de-emisor

Expresión-de-mismo :: ()

Expresión-de-progenitor :: ()

Expresión-de-vástago :: ()

Expresión-de-emisor :: ()

Gramática textual concreta

<expresión de PId> ::=
SELF
| PARENT
| OFFSPRING
| SENDER

Semántica

Una expresión de PId accede a una de las variables implícitas de proceso definidas en § 2.4.4. La expresión de variable de proceso se interpreta como el último valor asociado a la variable implícita correspondiente.

Una expresión de PId tiene un género, que es PId.

Una expresión de PId tiene un valor, que es el último valor asociado a la variable correspondiente, como se define en § 2.4.4.

5.5.4.4 *Expresión de visión*

Una expresión de visión permite a un proceso obtener el valor de una variable de otro proceso en el mismo bloque, como si la variable estuviese definida localmente. El proceso que observa no puede modificar el valor asociado a la variable.

Gramática abstracta

Expresión-de-visión :: *Identificador-de-variable*
Expresión

La *expresión* tiene que ser una expresión de PId.

El *identificador-de-variable* tiene que ser uno de los identificadores de una de las variables en el proceso identificado por la *expresión.*

Gramática textual concreta

<expresión de visión> ::=
VIEW (<identificador de variable>, <expresión de PId>)

El <identificador de variable> tiene que definirse para ser visto en una <definición de visión> en el proceso que contiene la <expresión de visión>. El <calificador> en <identificador de variable> puede omitirse únicamente si ninguna otra variable con la misma parte <nombre> está contenida en una <definición de visión> para la <definición de proceso> circundante.

Semántica

Una expresión de visión se interpreta de la misma manera que un acceso a variable (véase § 5.5.2.2). La variable a la que se accede es la variable en el proceso identificado por la expresión de PId que corresponde a la expresión de PId (véase § 5.5.4.3).

Una expresión de visión tiene un valor y un género que son el valor y el género del acceso a variable.

La expresión de PId tiene que identificar un proceso existente en el mismo bloque que el proceso en el que se interpreta la expresión de PId; de no ser así, la expresión de visión es errónea y el comportamiento ulterior del sistema está indefinido. La expresión de PId tiene que identificar el mismo tipo de proceso que el *identificador-de-proceso* en la correspondiente definición de visión.

5.5.4.5 *Expresión activa de temporizador*

Gramática abstracta

Expresión-activa-de-temporizador :: *Identificador-de-temporizador*
*Expresión**

Los géneros de la *expresión** en la *expresión-activa-de-temporizador* tienen que corresponder en posición con el *identificador-de-referencia-de-género** que sigue inmediatamente al *nombre-de-temporizador* (§ 2.8) identificado por el *identificador-de-temporizador.*

Gramática textual concreta

<expresión activa de temporizador> ::=
ACTIVE (<identificador de temporizador> [(<lista de expresiones>)])

<lista de expresiones> se define en § 5.5.2.1.

Semántica

Una expresión activa de temporizador es una expresión del género booleano que tiene el valor Verdadero si el temporizador identificado por el identificador de temporizador, e inicializado con los mismos valores que los denotados por la lista de expresiones (si existe), está activo (véase § 2.8.2). De no ser así, la expresión activa de temporizador tiene el valor Falso. Las expresiones se interpretan en el orden dado.

5.6 *Datos predefinidos*

Esta sección define géneros de datos y generadores de datos definidos implícitamente a nivel de sistema.

Obsérvese que en el § 5.4.1.1 se define la sintaxis y la precedencia de los operadores especiales (infijos y monádicos), pero la semántica de estos operadores (excepto REM y MOD) se define mediante las definiciones de datos de esta sección.

5.6.1 *Género booleano (boolean)*

5.6.1.1 *Definición*

```
NEWTYPE     Boolean

    LITERALS  True, False;
    OPERATORS

        "NOT"      : Boolean                -> Boolean;
                                                    /*
        "="        : Boolean, Boolean       -> Boolean;    Los operadores "=" y "/"
                                                           están implícitos.
        "/="       : Boolean, Boolean       -> Boolean;    Véase § 5.4.1.4
                                                    */
        "AND"      : Boolean, Boolean       -> Boolean;
        "OR"       : Boolean, Boolean       -> Boolean;
        "XOR"      : Boolean, Boolean       -> Boolean;
        "=>"       : Boolean, Boolean       -> Boolean;
    AXIOMS
        "NOT" (True)              == False;
        "NOT" (False)             == True;

        "=" (True, True)          == True;
        "=" (True, False)         == False;
        "=" (False, True)         == False;
        "=" (False, False)        == True;

        "/=" (True, True)         == False;
        "/=" (True, False)        == True;
        "/=" (False, True)        == True;
        "/=" (False, False)       == False;

        "AND" (True, True)        == True;
        "AND" (True, False)       == False;
        "AND" (False, True)       == False;
        "AND" (False, False)      == False;

        "OR" (True, True)         == True;
        "OR" (True, False)        == True;
        "OR" (False, True)        == True;
        "OR" (False, False)       == False;

        "XOR" (True, True)        == False;
        "XOR" (True, False)       == True;
        "XOR" (False, True)       == True;
        "XOR" (False, False)      == False;

        "=>" (True, True)         == True;
        "=>" (True, False)        == False;
        "=>" (False, True)        == True;
        "=>" (False, False)       == True;

ENDNEWTYPE Boolean;
```

5.6.1.2 *Uso*

El género booleano se utiliza para representar los valores Verdadero y Falso. Se utiliza a menudo como resultado de una comparación.

El género booleano es utilizado por muchas de las formas taquigráficas de datos en LED tales como axiomas sin el símbolo "= =" y los operadores de igualdad implícitos "=" y "/=".

5.6.2 *Género carácter (character)*

5.6.2.1 *Definición*

```
NEWTYPE Character

    LITERALS
        NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL,
        BS,  HT,  LF,  VT,  FF,  CR,  SO,  SI,
        DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB,
        CAN, EM,  SUB, ESC, IS4, IS3, IS2, IS1,
        ' ', '!', '"', '#', '¤', '%', '&', '''',
        '(', ')', '*', '+', ',', '-', '.', '/',
        '0', '1', '2', '3', '4', '5', '6', '7',
        '8', '9', ':', ';', '<', '=', '>', '?',
        '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
        'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
        'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
        'X', 'Y', 'Z', '[', '\', ']', '^', '_',
        '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g',
        'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
        'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
        'x', 'y', 'z', '{', '|', '}', '~',  DEL;
/*''''es un apóstrofo,' ' es un espacio, '~' es una tilde */


OPERATORS
                                                    /*
    "="  : Character, Character -> Boolean;           Las signaturas de operador "=" y "/="
    "/=" : Character, Character -> Boolean;           están implícitas. Véase § 5.4.1.4.
                                                    */
    "<"  : Character, Character -> Boolean;
    "<=": Character, Character -> Boolean;
    ">"  : Character, Character -> Boolean;
    ">=": Character, Character -> Boolean;
AXIOMS
/*lo siguiente especifica "menor que" entre literales carácter adyacentes*/
    NUL < SOH == True;      SOH < STX == True;
    STX < ETX == True;      ETX < EOT == True;
    EOT < ENQ == True;      ENQ < ACK == True;
    ACK < BEL == True;      BEL < BS  == True;
    BS  < HT  == True;      HT  < LF  == True;
    LF  < VT  == True;      VT  < FF  == True;
    FF  < CR  == True;      CR  < SO  == True;
    SO  < SI  == True;      SI  < DLE == True;
    DLE < DC1 == True;      DC1 < DC2 == True;
    DC2 < DC3 == True;      DC3 < DC4 == True;
    DC4 < NAK == True;      NAK < SYN == True;
```

```
SYN  < ETB  == True;    ETB  < CAN == True;
CAN  < EM   == True;    EM   < SUB == True;
SUB  < ESC  == True;    ESC  < IS4 == True;
IS4  < IS3  == True;    IS3  < IS2 == True;
IS2  < IS1  == True;    IS1  < ' ' == True;
' '  < '!'  == True;    '!'  < '"' == True;
'"'  < '#'  == True;    '#'  < '¤' == True;
'¤'  < '%'  == True;    '%'  < '&' == True;
'&'  < '''  == True;    '''  < '(' == True;
'('  < ')'  == True;    ')'  < '*' == True;
'*'  < '+'  == True;    '+'  < ',' == True;
','  < '-'  == True;    '-'  < '.' == True;
'.'  < '/'  == True;    '/'  < '0' == True;
'0'  < '1'  == True;    '1'  < '2' == True;
'2'  < '3'  == True;    '3'  < '4' == True;
'4'  < '5'  == True;    '5'  < '6' == True;
'6'  < '7'  == True;    '7'  < '8' == True;
'8'  < '9'  == True;    '9'  < ':' == True;
':'  < ';'  == True;    ';'  < '<' == True;
'<'  < '='  == True;    '='  < '>' == True;
'>'  < '?'  == True;    '?'  < '@' == True;
'@'  < 'A'  == True;    'A'  < 'B' == True;
'B'  < 'C'  == True;    'C'  < 'D' == True;
'D'  < 'E'  == True;    'E'  < 'F' == True;
'F'  < 'G'  == True;    'G'  < 'H' == True;
'H'  < 'I'  == True;    'I'  < 'J' == True;
'J'  < 'K'  == True;    'K'  < 'L' == True;
'L'  < 'M'  == True;    'M'  < 'N' == True;
'N'  < 'O'  == True;    'O'  < 'P' == True;
'P'  < 'Q'  == True;    'Q'  < 'R' == True;
'R'  < 'S'  == True;    'S'  < 'T' == True;
'T'  < 'U'  == True;    'U'  < 'V' == True;
'V'  < 'W'  == True;    'W'  < 'X' == True;
'X'  < 'Y'  == True;    'Y'  < 'Z' == True;
'Z'  < '['  == True;    '['  < '\' == True;
'\'  < ']'  == True;    ']'  < '^' == True;
'^'  < '_'  == True;    '_'  < '`' == True;
'`'  < 'a'  == True;    'a'  < 'b' == True;
'b'  < 'c'  == True;    'c'  < 'd' == True;
'd'  < 'e'  == True;    'e'  < 'f' == True;
'f'  < 'g'  == True;    'g'  < 'h' == True;
'h'  < 'i'  == True;    'i'  < 'j' == True;
'j'  < 'k'  == True;    'k'  < 'l' == True;
'l'  < 'm'  == True;    'm'  < 'n' == True;
'n'  < 'o'  == True;    'o'  < 'p' == True;
'p'  < 'q'  == True;    'q'  < 'r' == True;
'r'  < 's'  == True;    's'  < 't' == True;
't'  < 'u'  == True;    'u'  < 'v' == True;
'v'  < 'w'  == True;    'w'  < 'x' == True;
'x'  < 'y'  == True;    'y'  < 'z' == True;
'z'  < '{'  == True;    '{'  < '|' == True;
'|'  < '}'  == True;    '}'  < '~' == True;
'~'  < DEL  == True;    .

FOR ALL a, b, c IN Character (
    a < a                          == False
    a < b AND b < c => a < c       == True
    a < b                          == b > a;
    a < b OR a > b                 ==a/= b;
    a < b => NOT (b < a);
    NOT ( a/= b )                  ==a = b;
    a < b OR a = b                 ==a <= b;
    a > b OR a = b                 ==a >= b;)

ENDNEWTYPE Character;
```

5.6.2.2 *Uso*

El género carácter define cadenas de caracteres de longitud 1, donde los caracteres son los del Alfabeto Internacional N.º 5. Estos caracteres se definen como cadenas o como abreviaturas de conformidad con la Versión Internacional de Referencia del alfabeto. La representación impresa puede variar según el uso nacional del alfabeto.

Hay 128 literales y valores diferentes definidos para carácter. Se define la ordenación de los valores, la igualdad y la desigualdad.

5.6.3 *Generador cadena (string)*

5.6.3.1 *Definición*

```
GENERATOR String(TYPE Itemsort, LITERAL Emptystring) /*Las cadenas son «indizadas» a partir de uno*/
    LITERALS Emptystring;
    OPERATORS
        MkString: Itemsort      ->;                          /* hacer una cadena a partir de un ítem*/
        Length  : String        -> Integer;                  /* longitud de cadena */
        First   : String        -> Itemsort;                 /* primer ítem en cadena*/
        Last    : String        -> Itemsort;                 /* último ítem en cadena */
        "//"      String, String        -> String;           /* concatenación */
        Extract! : String, Integer      -> Itemsort;         /* tomar ítem de cadena */
        Modify! : String, Integer, Itemsort          -> String;  /* modificar valor de cadena */
        SubString: String, Integer,Integer           -> String;  /* tomar subcadena de cadena */
            /*subcadena (s,i,j) da una cadena de longitud j que comienza por el i-ésimo elemento */
    AXIOMS
        FOR ALL item, itemi, itemj, item 1, item 2 IN Itemsort (
        FOR ALL s, s1, s2, s3 IN String (
        FOR ALL i, j, IN Integer (
        type String Length (Emptystring)                     == 0;
        type String Length (MkString(item))                  == 1;
        type String Extract! (MkString(item),1)              == item;
        First(s)                                             == Extract!(s,1);
        Last(s)                                              == Extract!(s, Length(s));
        Length(s1//s2)                                       == Length(s1) + Length(s2);
        Length(Modify!(s,i,item))                            == Length(s));
        (s1 // s2) // s3                                     == s1 // (s2 // s3);
        Emptystring // 3                                     == s;
        s // Emptystring                                     == s;
        Emptystring = (MkString (item) // s2)                == False;
        (MkString (item1) // s1) = (MkString (item2) // s2) == (item1 = item2) AND (s1 = s2);

        i > 0 AND i <=Length(s) == True ==>
                        Extract!(Modify!(s,i,item),i) == item;
        i /= j AND i > 0 AND i <= Length(s) AND j > 0 AND j <= Length(s) == True ==>
                        Extract!(Modify!(s,i,item),j) == Extract!(s,j);
        i <= 0 OR i > Length(s) == True ==> Extract!(s,i) == ERROR!;

        i/= j == True ==>
            Modify!(Modify!(s,i,itemi),j,itemj) == Modify!(Modify!(s,j,itemj),i,itemi);
        Modify!(Modify!(s,i,item1),i,item2) == Modify!(s,i,item2);
        i <= 0 OR i > Length(s) == True ==> Modify!(s,i,item) == ERROR!;

        i <= Length(s1) == True ==>
                        Extract!(s1 // s2, i)       == Extract!(s1,i);
        i > Length(s1) == True ==>
                        Extract!(s1 // s2, i)       == Extract!(s2,i - Length(s1));

        i > 0 AND i <= Length(s) == True ==> SubString(s,i,0)== Emptystring;
        i > 0 AND i <= Length(s) == True ==> SubString(s,i,1)== MkString(Extra ct!(s,i));
        i > 0 AND i <= Length(s) AND i-1+j <= Length(s) AND j > 1 == True ==>
                        SubString(s,i,j) == SubString(s,i,1) // SubString(s,i+1,j-1);
        i < 0 OR i > Length(s) OR j <= 0 OR i+j > Length(s) == True ==>
                        SubString(s,i,j) == ERROR!;


        i > 0 AND i <= Length(s) == True ==>
                Modify!(s,i,item)    ==
                        Substring(s,1,i-1) // MkString(item) // Substring(s,i+1,Length(s)-i );)));
ENDGENERATOR String;
```

5.6.3.2 *Uso*

Un generador cadena puede utilizarse para definir un género que permite construir cadenas de ítems de cualquier género. El uso más común será para la cadena-de-caracteres definida más abajo.

Los operadores Extract! y Modify! se utilizarán típicamente con las notaciones taquigráficas definidas en § 5.4.2.4 y § 5.5.3.1 para acceder a los valores de cadenas y asignar valores a cadenas.

5.6.4 *Género cadena-de-caracteres (charstring)*

5.6.4.1 *Definición*

```
NEWTYPE Charstring (Character,'')
    ADDING LITERALS NAMECLASS "" ( ( ' ':'&') OR """" OR ( '(': '~' ) )+ "";
    /* cadenas de caracteres de cualquier longitud y formadas por cualesquiera caracteres desde un espacio ' '
    a una tilde '~' */
    /* ecuaciones de la forma
        'ABC' = = 'AB' // 'C';
        están implícitas; véase § 5.4.1.2 */
    MAP     FOR ALL c IN Character LITERALS (
                FOR ALL charstr IN Charstring LITERALS (
                        Spelling( charstr) = = Spelling(c) = = > charstr = = Mkstring(c);
            )   ); /* la cadena 'A' se forma a partir del carácter 'A' etc. */
ENDNEWTYPE Charstring;
```

5.6.4.2 *Uso*

El género cadena-de-caracteres define cadenas de caracteres. Un literal cadena-de-caracteres puede contener caracteres imprimibles y espacios. Un carácter no imprimible puede utilizarse como una cadena, empleando Mkstring, por ejemplo Mkstring(DEL).

```
/*Ejemplo*/ SYNONYM newline_prompt Charstring = Mkstring(CR) // Mkstring(LF) // '$>';
```

5.6.5 *Género entero (integer)*

5.6.5.1 *Definición*

```
NEWTYPE Integer
    LITERALS NAMECLASS ('0':'9')* ('0':'9') ;
        /*secuencia de números opcional antes de uno de los números 0 a 9 */
    OPERATORS
        "-"   : Integer              -> Integer;

        "+"  : Integer,Integer       -> Integer;
        "-"   : Integer,Integer      -> Integer;
        "*"   : Integer,Integer      -> Integer;
        "/"   : Integer,Integer      -> Integer;
                                        /*
        "="  : Integer,Integer       -> Boolean;      Las signaturas de operador "=" y "/=" están
        "/=": Integer,Integer        -> Boolean;      implícitas; véase § 5.4.1.4
                                        */
        "<"  : Integer,Integer       -> Boolean;
        ">"  : Integer,Integer       -> Boolean;
        "<=": Integer,Integer        -> Boolean;
        ">=": Integer,Integer        -> Boolean;

        Float: Integer               -> Real;    /* axiomas en la definición NEWTYPE Real */
        Fix   : Real                 -> Integer; /* axiomas en la definición NEWTYPE Real */
```

```
AXIOMS
    FOR ALL a, b, c IN Integer (
    /*negación*/
    0 - a                          == -a;
    /* adición:*/
    0 + a                          == a;
    a + b                          == b + a;
    a + (b + c)                    == (a + b) + c;
    /*substracción*/
    a - a                          == 0;
    (a - b) - c                    == a - (b + c);
    (a - b) + c                    == (a + c) - b;
    a - (b - c)                    == (a + c) - b;
    /*multiplicación*/
    a * 0                          == 0;
    a * 1                          == a;
    a * b                          == b * a;
    a * (b * c)                    == (a * b) *c;
    a * (b + c)                    == a * b + a * c;
    a * (b - c)                    == a * b - a * c;
    /*ordenación*/
    a + 1 > a                      == True;
    a - 1 < a                      == True;
    /* igualdad */
    (a > b) OR (b > a) == NOT (a =b);
    /* axiomas normales de ordenación */
    "<" (a,a)                      == False;
    "<" (a,b)                      == ">" (b,a);
    "<="(a,b)                      == "OR"("<"(a,b),"="(a,b));
    ">="(a,b)                      == "OR"(">"(a,b),"="(a,b));
    "<"(a,b) == True                    ==>    NOT("<"(b,a)) == True;
    "<"(a,b) AND "<"(b,c)== True   ==>    "<"(a,c)      == True;
    /*división*/
    a/0                            ==ERROR!;
    a>=0 AND b > a == True                          ==> a/b == 0;
    a>=0 AND b <=a AND b > 0 == True                ==> a/b == 1 + (a-b)/b;
    a>=0 AND b < 0 == True                          ==> a/b == -(a/(-b));
    a< 0 AND b < 0 == True                          ==> a/b == (-a)/(-b);
    a< 0 AND b > 0 == True                          ==> a/b ==-( (-a)/b );
    /* Literales 2 a 9*/
    TYPE Integer 2 == 1 + 1; TYPE Integer 3 == 2 + 1;
    TYPE Integer 4 == 3 + 1; TYPE Integer 5 == 4 + 1;
    TYPE Integer 6 == 5 + 1; TYPE Integer 7 == 6 + 1;
    TYPE Integer 8 == 7 + 1; TYPE Integer 9 == 8 + 1;
MAP /* Literales distintos de 0 a 9*/
    FOR ALL a,b,c IN Integer LITERALS
    ( Spelling(a) == Spelling(b) // Spelling(c), Length(Spelling(c)) == 1 ==>
            a  == b * (9 + 1) + c;


    );
ENDNEWTYPE Integer;
```

5.6.5.2 *Uso*

El género entero se utiliza para enteros matemáticos con notación decimal.

5.6.6 *Sintipo natural*

5.6.6.1 *Definición*

```
SYNTYPE Natural = Integer CONSTANTS >= 0 ENDSYNTYPE Natural;
```

5.6.6.2 *Uso*

El sintipo natural se utiliza cuando sólo se requieren enteros positivos. Todos los operadores serán los operadores de enteros; sin embargo, cuando un valor se utiliza como parámetro, o es asignado, deberá ser verificado. Un valor negativo será un error.

5.6.7 *Género real*

5.6.7.1 *Definición*

```
NEWTYPE Real
    LITERALS NAMECLASS ( ('0':'9')* ('0':'9') ) OR ( ('0':'9')* '.' ('0':'9')+ );
    OPERATORS
        "-"     : Real              -> Real;

        "+"     : Real, Real        -> Real;
        "-"     : Real, Real        -> Real;
        "*"     : Real, Real        -> Real;
        "/"     : Real, Real        -> Real;
                                        /*
        "="     : Real, Real        -> Boolean;     Las signaturas de operador "=" y "/="
        "/="    : Real, Real        -> Boolean;     están implícitas; véase § 5.4.1.4
                                        */
        "<"     : Real, Real        -> Boolean;
        ">"     : Real, Real        -> Boolean;
        "<="    : Real, Real        -> Boolean;
        ">="    : Real, Real        -> Boolean;
    AXIOMS
        FOR ALL a, b, c IN Real (
        /*negación*/
        0 - a                       ==  -a;
        /* adición*/
        0 + a                       ==  a;
        a + b                       ==  b + a;
        a + (b + c)                 ==  (a + b) + c;
        /*substracción*/
        a - a                       ==  0;
        (a - b) -c                  ==  a - (b + c);
        (a - b) + c                 ==  (a + c) - b;
        a - (b - c)                 ==  (a + c) - b;
        /*multiplicación*/
        a * 0                       ==  0;
        a * 1                       ==  a;
        a * b                       ==  b * a;
        a * (b * c)                 ==  (a * b) *c;
        a * (b + c)                 ==  a * b + a * c;
        a * (b - c)                 ==  a * b - a * c);
        /*ordenación*/
        FOR ALL i, j, IN Integer (
        Float(i) > Float(j)         ==TYPE Integer ">"(i,j);
        Float(j) = 0 == False       ==>Float(i) / Float(j) > 0 == Float(i) > 0 AND Float(j) > 0
                                            OR Float(i) < 0 AND Float(j) < 0;
        Float(i) > 0 AND Float(j) > 0 AND Float(i) > Float(j)
                                    ==>Float(i) / Float(j) > 1 == True;);
        FOR ALL a,r,b IN Real (a + r < b + r == a < b;
        r > 0 ==> a * r < b * r == a < b;
        r < 0 ==> a * r < b * r == b < a;);
```

```
        /* axiomas normales de ordenación */
        FOR ALL a, b, c, d IN Real
        /* igualdad y ordenación */
        (a > b) OR (b > a) = NOT (a = b);
        "<"(a,a)                    = = False;
        "<"(a,b)                    = = ">" (b,a);
        "< ="(a,b)                  = = "OR" ("<"(a,b),"="(a,b));
        "> ="(a,b)                  = = "OR" (">"(a,b),"="(a,b));
        "<"(a,b) = = True                 = = >  NOT("<"(b,a))= = True;
        "<"(a,b) AND "<"(b,c) = =True= = >  "<"(a,c)      = = True;
        /*división*/
        a/0                         = =ERROR!;
        a = 0 = = False = = > a / a= = 1;
        a = 0 = = False = = > 0 / a = = 0;
        b = 0 = = False = = > (a / b) * b = = a;
        b = 0 OR c = 0 = = False = = > (a * b)/(c * b) = = a/c;
        b = 0 OR d = 0 = = False = = > a/b + c/d = = (a * d + b * c)/(b * d);
        b = 0 OR d = 0 = = False = = > a/b - c/d = = (a * d - b * c)/(b * d);
        b = 0 OR d = 0 = = False = = > (a/b) * (c/d) = = (a * c)/(b * d);
        b = 0 OR d = 0 = = False = = > (a/b) / (c/d) = = (a * d)/(b * c););
        /*conversiones entre entero y real*/
        FOR ALL a, i, j IN Integer (
        FOR ALL r IN Real (
        Fix(Float(a))               = = a;
        r - 1.0 < Float(Fix(r))     = = True; /* Note Fix(1.5) = = 1, Fix(-0.5)= = -1 */
        Float(Fix(r)) < = r         = = True;
        Float(TYPE integer "+"(i,j)) = =      Float(i) + Float(j);));
    MAP
        FOR ALL r,s IN Real LITERALS (
            FOR ALL i,j IN Integer LITERALS (
                Spelling(r) = = Spelling(i) = = > r = = Float(i);
                Spelling(r) = = Spelling(i) = = > i = = Fix(r);
                Spelling(r) = = Spelling(i) // Spelling(s), Spelling(s)= ='.'// Spelling(j)
                                            = = > r = = Float(i) + s;
                Spelling(r) = = '.' //Spelling(i), Lenght(Spelling(i)) = = 1,
                                            = = > r = = Float(i) / 10;
                Spelling(r) = = '.' //Spelling(i) //Spelling(j), Lenght(Spelling(i)) = = 1,
                        Spelling(s) = = '.' // Spelling(j)
                                            = = > r = = (Float(i) + s) / 10;
        )    );
ENDNEWTYPE Real;
```

5.6.7.2 *Uso*

El género real se utiliza para representar números reales. El género real puede representar todos los números que pueden ser representados por un entero dividido por otro. Los números que no pueden representarse de esta manera (números irracionales – por ejemplo $\sqrt{2}$) no forman parte del género real. Sin embargo, para fines prácticos de ingeniería, puede utilizarse generalmente una aproximación de una exactitud suficiente. La definición de un conjunto de números que incluya todos los irracionales no es posible sin el empleo de técnicas adicionales.

5.6.8 *Generador matriz (array)*

5.6.8.1 *Definición*

```
GENERATORS Array (TYPE Index, TYPE Itemsort)
    OPERATORS
        Make!   : Itemsort                  -> Array;
        Modify! : Array,Index,Itemsort      -> Array;
        Extract! : Array,Index              -> Itemsort;
    AXIOMS
        FOR ALL item, item1, item2, itemi, itemj IN Itemsort (
        FOR ALL i, j, ipos IN Index (
        FOR ALL a, s IN Array (
        type Array Extract!(Make!(item,i)) = =item;
        Modify!(Modify!(s,i,item1),i,item2) = = Modify!(s,i,item2);
        Extract!(Modify!(a,ipos,item),ipos) = = item;
        i = j = = False          = = > Extract!(Modify!(a,j,item),i) = = Extract!(a,i);
        i = j = = False          = = >
            Modify!(Modify!(s,i,itemi),j,itemj) = = Modify!(Modify!(s,j,itemj),i,itemi);)));

        /* igualdad */
        type Array Make! (item1) = Make! (item2) = = item1 = item2;
        Modify! (a, i, item) = s                = = (Extract! (s, i) = item AND (a = s);
ENDGENERATOR Array;
```

5.6.8.2 *Uso*

El generador matriz puede utilizarse para definir un género que está indizado por otro. Por ejemplo

```
NEWTYPE indexbychar Array(Character,Integer)
ENDNEWTYPE indexbychar;
```

define una matriz que contiene enteros y está indizada por caracteres.

Las matrices se utilizan usualmente en combinación con las formas taquigráficas de Modify! y Extract! definidas en § 5.5.3.1 y § 5.4.2.4 para indización. Por ejemplo

```
DLC charvalue indexbychar;
.....
TASK charvalue('A') := charvalue('B')-1;
```

5.6.9 *Generador conjuntista (powerset)*

5.6.9.1 *Definición*

```
GENERATOR Powerset (TYPE Itemsort)
    LITERALS Empty;
    OPERATORS
        "IN"    : Itemsort,    Powerset-> Boolean;   /* es miembro de operador                */
        Incl    : Itemsort,    Powerset-> Powerset;  /* incluir ítem en conjunto              */
        Del     : Itemsort,    Powerset-> Powerset;  /* suprimir ítem de conjunto             */
        "<"     : Powerset,    Powerset-> Boolean;   /* es subconjunto apropiado de operador  */
        ">"     : Powerset,    Powerset-> Boolean;   /* es superconjunto apropiado de operador*/
        "<="    : Powerset,    Powerset-> Boolean;   /* es subconjunto de operador            */
        ">="    : Powerset,    Powerset-> Boolean;   /* es superconjunto de operador          */
        "AND"   : Powerset,    Powerset-> Powerset;  /* intersección de conjuntos             */
        "OR"    : Powerset,    Powerset-> Powerset;  /*unión de conjuntos                     */
```

```
AXIOMS
    FOR ALL i, j IN Itemsort (
    FOR ALL p, ps, a, b, c IN Powerset (
    i IN type Powerset Empty                    == False;
    i IN Incl(i,ps)                             == True;
    i IN ps                                     == i IN Incl(j,ps);
    type Powerset Del(i,Empty)                  == Empty;
    NOT(i IN ps)                                == Del(i,ps) = ps;
    DEL(i,Incl(i,ps))                           == ps;
    i = j == False ==> Del(i,Incl(j,ps))        == Incl(j,Del(i,ps));
    Incl(i,Incl(j,p))                           == Incl(j,Incl(i,p));
    Incl(i,Incl(i,p))                           == Incl(i,p);
    a < b ==> (i IN a ==> i IN b)               == True;
    i IN ( a AND b )                            == TYPE Boolean "AND"( i IN a, i IN b);
    i IN ( a OR b )                             == TYPE Boolean "OR"( i IN a, i IN b);

    /* igualdad*/
    Empty = Incl (i, ps) == False;
    Incl (i, a) = b        ==(i IN b) AND (a=Del (i,b));

    /* axiomas normales de ordenación*/
    "<"(a,a)==      False;
    "<"(a,b)==      ">"(b,a);
    "<="(a,b)==     "OR"("<"(a,b),"="(a,b));
    ">="(a,b)==     "OR"(">"(a,b),"="(a,b));
    "<"(a,b) == True ==> NOT("<"(b,a)) == True;
    TYPE Boolean "AND"("<"(a,b),"<"(b,c)) == True ==> "<"(a,c) == True;))

ENDGENERATOR Powerset;
```

5.6.9.2 *Uso*

Los conjuntistas se utilizan para representar conjuntos matemáticos. Por ejemplo

NEWTYPE Boolset Powerset(Boolean) ENDNEWTYPE Boolset; puede utilizarse para una variable que puede estar vacía o contener (Verdadero), (Falso) o (Verdadero,Falso).

5.6.10 *Género PId*

5.6.10.1 *Definición*

```
NEWTYPE PId
    LITERALS Null;
    OPERATORS       unique!     : PId -> PId;

                    /*
                    "="         : Pid,Pid -> Boolean;     Las signaturas de operador "=" y "/="
                    "/="        : Pid,Pid -> Boolean;     están implícitas – véase § 5.4.1.4
                    */
    AXIOMS
        FOR ALL p, p1, p2 IN PId (
        unique! (p) = Null                  == False;
        unique! (p1) = unique! (p2)         == p1 = p2);

    DEFAULT Null;
    ENDNEWTYPE PId;
```

5.6.10.2 *Uso*

El género PId se utiliza para identidades de proceso. Obsérvese que el único literal es el valor Null. Cuando se crea un proceso, el sistema subyacente utiliza el operador unique! para generar un nuevo valor único.

5.6.11 *Género duración (duration)*

5.6.11.1 *Definición*

```
NEWTYPE Duration INHERITS Real ( "+","-",">")
ADDING
        OPERATORS
            "*"  : Duration, Real -> Duration;
            "/"  : Duration, Real -> Duration;
        AXIOMS /* en lo sucesivo toda d tiene que ser un valor de duración, del contexto */
        FOR ALL d,z IN Duration (
        FOR ALL r IN Real (
            /* igualdad */
            (d > z) OR (z > d) == NOT (d = z);
            /* Duración multiplicada por Real */
            d * 0                    == 0;
            0 * r                    == 0;
            d * TYPE Real "+"(1,r)   == d + (d * r);
            d * TYPE Real "-"(1,r)   == d - (d * r);
            d * TYPE Real "-"(r,1)   == (d * r) - d;
            d * TYPE Real "-"(r)     == 0 - (d * r);
            /* Duración dividida por Real */
            d / 0 == ERROR!;
            r = 0 == False ==> d / r == d * TYPE Real "/" (1,r);
            /* o sea, dividir es lo mismo que multiplicar por la recíproca (real) */
            r = 0 == False ==> z * r = d        == ( d / r) = z; ));
        MAP
            FOR ALL d IN Duration LITERALS (
            FOR ALL r IN Real LITERALS ( Spelling(d) == Spelling(r) ==> d == 1 * r ));
ENDNEWTYPE Duration;
```

5.6.11.2 *Uso*

El género duración se utiliza para el valor que ha de añadirse al tiempo corriente para inicializar temporizadores. Los literales del género duración son los mismos que los literales para el género real. El significado de una unidad de duración dependerá del sistema que se define.

Las duraciones pueden multiplicarse y dividirse por reales.

5.6.12 *Género tiempo (time)*

5.6.12.1 *Definición*

```
NEWTYPE Time INHERITS Real OPERATORS ("<","<=",">",">=") ADDING
        OPERATORS
            "+" : Time, Duration      -> Time;
            "-" : Time, Duration      -> Time;
            "-" : Time, Time          -> Duration;
        AXIOMS
            FOR ALL t, t1, t2 IN Time (
            FOR ALL d,d1,d2, IN Duration (
            (t1 > t2) OR (t2 > t1) == NOT (t1 = t2);
            t + 0                  == t;
            t - d                  == t + TYPE Duration "-"( 0 , d );
            (t + d1) + d2          == t + TYPE Duration "+"(d1,d2);
            (t + d1) - (t + d2 )   == TYPE Duration "-"(d1,d2);));
        MAP
            FOR ALL d IN Duration LITERALS (
            FOR ALL t IN Time LITERALS ( Spelling(d) == Spelling(t) ==> t == 0 + d ));
ENDNEWTYPE Time;
```

5.6.12.2 *Uso*

La expresión NOW retorna un valor del género tiempo. A un valor de tiempo se le puede añadir o substraer una duración, para obtener otro valor de tiempo. Un valor de tiempo substraído de otro valor de tiempo da una duración. Los valores de tiempo se utilizan para fijar el tiempo de expiración de los temporizadores.

El origen de tiempo depende del sistema. Una unidad de tiempo es la cantidad de tiempo representada por la adición de una unidad de duración a un tiempo.

APÉNDICE I

(a la Recomendación Z.100)

Modelo formal de tipos de datos no parametrizados [1)]

I.1 *Álgebras multigénero*

Un **álgebra multigénero** A es una dupla <D,O> donde

a) D es un conjunto de conjuntos, y los elementos de D se conocen por **portadores de datos** (de A); los elementos de un portador de datos **dc** (del inglés data carrier) se conocen por **valores-de-datos**; y

b) O es un conjunto de funciones totales, donde el dominio de cada función es un producto cartesiano de portadores de datos de A y el intervalo de uno de los portadores de datos.

I.2 *Semántica de definiciones de tipos de datos*

I.2.1 *Conceptos generales*

I.2.1.1 *Signatura*

Una **signatura** SIG es una dupla <S,OP> donde

a) S es un conjunto de **identificadores-de-género** (a los cuales se denomina también géneros); y

b) OP es un conjunto de **operadores**.

Un operador consiste en un **identificador-de-operación** op, una lista de géneros (argumento) w con elementos en S y un género (intervalo) s S. Esto se escribe usualmente op:w→ s. Si w es igual a la lista vacía, el op:w→ s se denomina un operador **nul-ario** o **símbolo constante** de género s.

I.2.1.2 *Morfismo de signatura*

Sean $SIG_1 = <S_1,OP_1>$ y $<SIG_2 = <S_2,OP_2>$ signaturas. Un **morfismo de signatura** $g:SIG_1 \rightarrow SIG_2$ es un par de relaciones de correspondencia

$$g = <gs:S_1 \rightarrow S_2, gop:OP_1 \rightarrow OP_2>$$

tal que para todos los $e\text{-}opid_1 = <opidf_1, <gs(e\text{-}sidf_1), ...,gs(e\text{-}sidf_k)>,gs(e\text{-}res),pos> \in OP_1$
$gop(e\text{-}opid_1) = <opidf_2, <(e\text{-}sidf_1),...,(e\text{-}sidf_k)>,(e\text{-}res),pos>$

para algún identificador-de-operación $opidf_2$.

[1)] El texto de este apéndice ha sido acordado entre el CCITT y la ISO como una descripción formal común del modelo de álgebra inicial para tipos abstractos de datos. Además de aparecer en esta Recomendación, este texto (con los cambios apropiados de terminología, tipografía y numeración) aparece también en ISO IS8807. Los § I.1, I.2.1.1, I.2.1.2, I.2.1.3, I.2.1.4, I.2.1.5, I.2.1.6, I.3, I.4.1, I.4.2, I.4.3, I.4.4, I.4.5, e I.4.6 de este apéndice figuran en § 5.2, 7.2.2.1, 7.3.2.8, 7.2.2.2, 7.2.2.3, 7.2.2.4, 7.2.2.5, 4.7, 7.4.2.1, 7.4.2.2, 7.4.3, 7.4.3 y 7.4.4, de ISO IS8807, respectivamente. Las terminologías **identificador-de-género, operador, identificador-de-variable, variable, especificación algebraica SPEC** y **operaciones** de este apéndice se sustituyen por **variable-de-género, variable-de-operador, variable-de-valor, variable-de-valor, presentación de datos pres** y **funciones**, respectivamente en ISO IS8807.

I.2.1.3 *Términos*

Sea V cualquier conjunto de variables y sea $<S,OP>$ una signatura. Los conjuntos TERM(OP,V,s) de **términos** de género $s \in S$ con operadores en OP y variables en V, se definen inductivamente por los pasos siguientes:

a) cada variable en $x:s \in V$ está en TERM(OP,V,s);

b) cada operador nul-ario con operador $op \in OP$ con $res(op) = s$ está en TERM(OP,V,s);

c) si los términos t_i de género s_i están en TERM(OP,V,s_i) para $i = 1,...,n$ para cada $op \in OP$ con $arg(op) = <s_1,...,s_n>$ y $res(op) = s$, $op(t_1,...,t_n)$ está en TERM(OP,V,s).

Si el término t es un elemento de TERM(OP,V,s), s se llama el género de t, denotado como género (t).

El conjunto TERM(OP,s) de **términos fundamentales** de género $s \in S$ se define como el conjunto TERM(OP,{},s).

I.2.1.4 *Ecuaciones*

Una **ecuación** de género s con respecto a una signatura $<S,OP>$ es una tripla $<V,L,R>$ donde

a) V es un conjunto de identificadores-de-variable;

b) $L,R \in T(OP,V,s)$; y

c) $s \in S$.

Una ecuación $e' = <\{\},L',R'>$ es una **instancia fundamental** de una ecuación $e = <V,L,R>$ si L',R' puede obtenerse a partir de L,R para cada variable v:s en V sustituyendo todas las ocurrencias de esa variable en *L,R* por el mismo término fundamental con género s.

La notación $L = R$ se utiliza para la instancia fundamental $<\{\},L,R>$ de una ecuación.

Nota – Asimismo, una ecuación $<V,L,R>$ puede escribirse $L = R$ si con ello no se introducen complicaciones semánticas.

I.2.1.5 *Ecuaciones condicionales*

Una **ecuación condicional** de género s con respecto a la signatura $<S,OP>$ es una tripla $<V,Eq,e>$, donde

a) V es un conjunto de identificadores-de-variable; y

b) Eq es un conjunto de ecuaciones con respecto a $<S,OP>$, con variables en V; y

c) e es una ecuación de género s con respecto a $<S,OP>$, con variables en V.

I.2.1.6 *Especificaciones algebraicas*

Una **especificación algebraica** SPEC es una tripla $<S,OP,E>$ donde

a) $<S,OP>$ es una signatura; y

b) E es un conjunto de ecuaciones condicionales con respecto a $<S,OP>$.

I.3 *Sistemas de derivación*

Un **sistema de derivación** es una tripla $D = <A,Ax,I>$ con:

a) A un conjunto cuyos elementos se denominan **aserciones**,

b) $A \supseteq Ax$ el conjunto de **axiomas**,

c) I un conjunto de **reglas de inferencia**.

Cada regla de inferencia $R \in I$ tiene el siguiente formato

$$R: \frac{P_1,...,P_n}{Q}$$

donde $P_1,...,P_n,Q \in A$.

Una **derivación** de una aserción P en un sistema de derivación D es una secuencia finitas de aserciones que satisfacen las siguientes condiciones:

a) el último elemento de s es P,

b) si Q es un elemento de s, bien $Q \in Ax$, bien existe una regla $R \in I$

$$R: \frac{P_1,...,P_n}{Q}$$

con $P_1,...,P_n$ elementos de s que preceden a Q.

Si en un sistema de derivación D existe una derivación de P, esto se escribe $D \mid - P$. Si D está determinado unívocamente por el contexto, esto puede escribirse abreviadamente $\mid - P$.

I.4 *Semántica de especificaciones algebraicas*

Todas las ocurrencias de un conjunto de géneros S, un conjunto de operaciones OP y un conjunto de ecuaciones E en § I.4 se refieren a una especificación algebraica dada SPEC = <S,OP,E> como se define en § I.2.1.6.

Para definir la semántica de una especificación algebraica SPEC se utiliza un sistema de derivación asociado con SPEC. Este sistema de derivación se define en § I.4.1-I.4.3. Utilizando este sistema de derivación, en § I.4.4 e I.4.5 se define una relación sobre el conjunto de términos fundamentales con respecto a <S,OP,E>, y a las clases de congruencias. Esta relación se utiliza en § I.4.6 para definir un álgebra (véase § I.1) que representa el tipo de datos especificado por <S,OP,E>.

I.4.1 *Axiomas generados por ecuaciones*

Sea ceq una ecuación condicional. El conjunto de axiomas generado por ceq, notación Ax(ceq), se define como sigue:

a) si ceq = <V,Eq,e> con Eq ≠ {}, entonces Ax(ceq) = {}; y

b) si ceq = <V,{},e> entonces Ax(ceq) es el conjunto de todas las instancias fundamentales de e (véase § I.2.1.3).

I.4.2 *Reglas de inferencia generadas por ecuaciones*

Sea ceq una ecuación condicional. El conjunto de reglas de inferencia generado por ceq, notación Inf(ceq), se define como sigue:

a) si ceq = <V,{},e>, entonces Inf(ceq) = {}, y

b) si ceq = <V,{e_1,...,e_n},e> con n > 0, entonces Inf(ceq) contiene todas las reglas de la forma

$$\frac{e_1',\ldots,e_n'}{e'}$$

donde

e_1',...,e_n',e' son instancias fundamentales de e_1...,e_n,e respectivamente, que se obtienen remplazando, para cada variable x que ocurre en V, todas las ocurrencias de esa variable en e_1,...,e_n,e, por el mismo término fundamental con género **género (x)**.

I.4.3 *Sistema de derivación generado*

El **sistema de derivación** D = <A,Ax,I> (véase § I.3) generado por una especificación algebraica SPEC = <S,OP,E> se define como sigue:

a) A es el conjunto de todas las instancias fundamentales de ecuaciones con respecto a <S,OP>; y

b) Ax = ∪{Ax(ceq) | ceq ∈ E} ∪ ID,
con ID = {t = t | t es un término fundamental}; y

c) $I = \cup\{Inf(ceq) \mid ceq \quad E\} \cup SI$,
donde SI viene dado por el esquema siguiente

i) $\dfrac{t_1 = t_2}{t_2 = t_1}$ para todos los términos fundamentales t_1, t_2; y

ii) $\dfrac{t_1 = t_2,\ t_2 = t_3}{t_1 = t_3}$ para todos los términos fundamentales t_1, t_2, t_3; y

iii) $$\frac{t_1 = t_1', \ldots, t_n = t_n'}{op(t_1,\ldots,t_n) = op(t_1',\ldots,t_n')}$$

para todos los operadores $op: s_1,\ldots,s_n \rightarrow s \in OP$ con $n > 0$ y todos los términos fundamentales de t_i, t_i' de género s_i para $i = 1,\ldots,n$.

I.4.4 *Relación de congruencia generada por una especificación algebraica*

Sea D el sistema de derivación generado por una especificación algebraica SPEC = <S,OP,E>. Se dice que dos términos fundamentales t_1 y t_2 son **congruentes** con respecto a SPEC, notación $t_1 \equiv_{SPEC} t_2$, si

$$D \vdash t_1 = t_2.$$

I.4.5 *Clases de congruencia*

La **clase de congruencia-SPEC** [t] de un término fundamental t es el conjunto de todos los términos congruentes con t con respecto a SPEC, esto es:

$$[t] = \{ t' \mid t \equiv_{SPEC} t' \}.$$

I.4.6 *Álgebra de término cociente*

La interpretación semántica de una especificación algebraica SPEC = <S,OP,E> es la siguiente álgebra multigénero $Q = <Dq, O_q>$, denominada **álgebra de término cociente**, donde

a) Dq es el conjunto $\{ Q(s) \mid s \in S \}$ donde
$Q(s) = \{[t] \mid t$ es término fundamental de género $s\}$ para cada $s \in S$; y

b) Oq es el conjunto de operaciones $\{ op' \mid op \in OP \}$ donde las op' están definidas por
$op'([t_1],\ldots,[t_n]) = [op(t_1,\ldots,t_n)]$.

SERIES DE RECOMENDACIONES DEL UIT-T

Serie A	Organización del trabajo del UIT-T
Serie B	Medios de expresión: definiciones, símbolos, clasificación
Serie C	Estadísticas generales de telecomunicaciones
Serie D	Principios generales de tarificación
Serie E	Explotación general de la red, servicio telefónico, explotación del servicio y factores humanos
Serie F	Servicios de telecomunicación no telefónicos
Serie G	Sistemas y medios de transmisión, sistemas y redes digitales
Serie H	Sistemas audiovisuales y multimedios
Serie I	Red digital de servicios integrados
Serie J	Transmisiones de señales radiofónicas, de televisión y de otras señales multimedios
Serie K	Protección contra las interferencias
Serie L	Construcción, instalación y protección de los cables y otros elementos de planta exterior
Serie M	RGT y mantenimiento de redes: sistemas de transmisión, circuitos telefónicos, telegrafía, facsímil y circuitos arrendados internacionales
Serie N	Mantenimiento: circuitos internacionales para transmisiones radiofónicas y de televisión
Serie O	Especificaciones de los aparatos de medida
Serie P	Calidad de transmisión telefónica, instalaciones telefónicas y redes locales
Serie Q	Conmutación y señalización
Serie R	Transmisión telegráfica
Serie S	Equipos terminales para servicios de telegrafía
Serie T	Terminales para servicios de telemática
Serie U	Conmutación telegráfica
Serie V	Comunicación de datos por la red telefónica
Serie X	Redes de datos y comunicación entre sistemas abiertos
Serie Y	Infraestructura mundial de la información y aspectos del protocolo Internet
Serie Z	**Lenguajes y aspectos generales de soporte lógico para sistemas de telecomunicación**

Impreso en Suiza
Ginebra, 2010