INTERPRETACIÓN Y TRADUCCIÓN DE TEXTO Y MATEMÁTICAS EN BRAILLE ESCRITO A MÁQUINA

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
Gabriel González Cano
i dirigit per
Gemma Sánchez Albaladejo
Bellaterra, 26 de Juny de 2009

Universitat Autònoma de Barcelona

etse
Escola Tècnica Superior d'Enginyeria

El sotasignat, Gemma Sánchez Albaladejo

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en Gabriel González Cano

I per tal que consti firma la present.

Signat: .....

Bellaterra, 26 de juny de 2009

Agradecimientos

A la directora de mi proyecto Gemma Sánchez, por haberme guiado a lo largo de su realización.

A mi familia y amigos por haberme aconsejado y animado durante estos años.

A Maribel, porque sin ella nunca habría finalizado este proyecto.

Índice general

Capítulo 1

Introducción

1.1. Descripción del problema

La tecnología de la visión por computador puede ofrecer grandes posibilidades como herramienta de ayuda a personas invidentes. Aunque estas personas pueden leer y escribir perfectamente en Braille, la mayoría de la gente desconoce este sistema lo que supone una gran barrera comunicativa que se hace particularmente notable en la educación, donde conviven alumnos invidentes con profesores y demás alumnos que pueden no conocer el sistema Braille.

En este proyecto nos centramos en la aplicación de técnicas de análisis de documentos como un recurso educativo e integrador de niños invidentes en escuelas de educación primaria. Este trabajo se enmarca dentro de un proyecto colaborativo entre el CVC y el equipo de educadores del centro Joan Amades (ONCE) y forma parte de una de las herramientas que se proponen al centro: un intérprete de textos braille con contenido matemático y que permita a un usuario desconocedor del sistema Braille leer los documentos Braille escritos por un alumno.

(a) (b)

Figura 1.1: (a) Máquina PERKINS (b) Profesor de apoyo

La herramienta presentada permite la traducción a tinta de textos Braille creados mediante una máquina Perkins (Figura 1.1a) y previamente escaneados, facilitando el intercambio de material entre niños invidentes, profesores (Figura 1.1b) y resto de compañeros y resolviendo la

necesidad de interpretar, traducir y reproducir textos Braille. Actualmente un escrito en Braille que se quiera compartir con otras personas y que no esté en formato digital se debe reescribir, no se puede fotocopiar debido al relieve de los caracteres. Pero con el sistema presentado se podría escanear, interpretar y guardar en formato electrónico sin necesidad de reescribirlo.

1.2. Estado del arte

Durante las dos últimas décadas han aparecido diversos proyectos que han abordado el problema de la interpretación de documentos con contenido matemático en Braille. Aunque la notación matemática tradicional para personas sin deficiencias visuales es internacional, la notación matemática Braille difiere en cada idioma. En el caso de la notación Braille en castellano o catalán existen muy pocas referencias a proyectos que hayan tratado de traducir documentos con contenido matemático.

1.2.1. Traductores de Braille-tinta

Podemos clasificar algunos de los proyectos según la 'dirección' en que se realiza la traducción:

1. **De notación matemática a notación matemática Braille:** Bramanet [1] convierte fórmulas representadas en MathML a la notación matemática Braille utilizada en Francia. LaBraDoor (LAtex to BRAille DOOR) [2] traduce matemáticas escritas con LaTeX a notación Marburg Braille (usada en Alemania) y HRTEX (Human Readable TeX). Math2Braille [3] traduce notación MathML al Braille utilizado en los Países Bajos. Infty [4] traduce documentos matemáticos reconocidos a través de un OCR a Unified Braille Code y notación Braille en Japonés. Una vez realizada la traducción a Braille, estos proyectos permiten imprimir el resultado utilizando una impresora Braille.

2. **De notación matemática Braille a notación matemática:** En este apartado podemos distinguir entre el origen de la traducción realizada. Mientras que algunos de los proyectos utilizan un OBR (Optical Braille Recognition) para el reconocimiento de caracteres en un documento Braille, otros parten de expresiones codificadas mediante la notación ASCII-Braille (cada carácter Braille equivale a un carácter ASCII).

   - OBR: El proyecto Insight [5] reconoce en una imagen escaneada texto y contenido matemático en Nemeth Braille (usado en Estados Unidos) traduciéndolo a Latex. El proyecto Insight está basado en MAVIS (Mathematics Accessible To Visually Impaired Students) [6] que fue la primera solución en traducir Nemeth Braille a Latex.

     En el artículo 'Automatic Braille Code Translation System' [7] de la Universidad de Algarve (Portugal) se explica la traducción de documentos codificados en el Braille utilizado en Portugal. Se realizan tres Parsers, uno para texto, otro para matemáticas y un último para química. Esta aplicación parte de un documento de texto formado por 1's y 0's creado a partir de la utilización de un OBR (Optical Braille Recognizer).
   - ASCII-Braille: Existen dos proyectos que utilizan la notación matemática Braille en castellano. Blat(mat) [8] traduce matemáticas representadas en formato ASCII-Braille a MathML y SBT (A translator from Spanish Mathematical Braille to MathML)

[9] que incorpora un editor para modificar matemáticas escritas en ASCII-Braille y las traduce a MathML. Aunque estos dos proyectos permiten la traducción de texto Braille además de matemáticas, éste debe ser etiquetado como tal.

3. **En las dos direcciones:** MMBT (Multi-Language Mathematical Braille Translator) [10] proyecto opensource que permite la traducción en ambas direcciones desde notación Braille en Francés, Inglés y Alemán a notación LaTeX o MathML. UMCL (Universal Maths Conversión Library) [11] creada por IGroupUMA es una librería que encapsula varios traductores y permite la traducción desde Latex y MathML a notación Braille en formato Nemeth, Marburg, Francés, Inglés, italiano y vice versa.

1.2.2. Proyectos destinados a la integración de niños invidentes en centros educativos

Existen algunos proyectos que tratan de facilitar específicamente la colaboracíón entre alumnos invidentes con profesores y el resto de alumnos.

- El proyecto Vickie (Visually Impaired Children Kit for Inclusive Education) [12] incluye una interfaz unificada que utiliza 3 vistas de un mismo documento. Una para alumnos invidentes y que utiliza dispositivos Braille específicos, otra utilizando una representación gráfica útil para profesores y una última adaptada para los alumnos con visión parcial que incluye colores y fuentes ajustables. Este proyecto utiliza MathML para representar fórmulas matemáticas y MMBT (Multi-Language Mathematical Braille Translator) para realizar la conversión a Braille. El proyecto permite que un alumno introduzca una fórmula utilizando Braille y que el profesor pueda ver la traducción en MathML.

- El proyecto MAWEN (Mathematical Work Environment) [13] requiere la sincronización de dos representaciones, una dedicada a los alumnos invidentes que utiliza Braille y otra gráfica destinada a los profesores. MAWEN está pensado para trabajar con documentos que incluyen texto y fórmulas matemáticas utilizando cualquier codificación Braille soportada por la librería UMCL. Está diseñado para representar simultáneamente el contenido matemático en Braille y su representación visual en MathML, permitiendo editar este contenido mientras la representación persiste y ayudando al alumno en la representación de contenido matemático.

- El proyecto LAMBDA [14] (Linear Access to Mathemathics for Braille Device and Audiosynthesis) es un sistema de escritura y lectura diseñado para estudiantes invidentes. Consta de un editor que puede ser utilizado por el estudiante permitiéndole introducir y editar expresiones matemáticas de forma sencilla. Para evitar las distintas notaciones Braille propias de cada país define el LAMBDA code, un sistema de etiquetas para representar expresiones matemáticas de forma lineal. La salida se produce utilizando síntesis de voz, codificación Braille especial de 8 puntos o representación visual en MathML.

1.3. Descripción del método utilizado

El problema de la traducción de textos escritos en Braille puede dividirse en dos fases. La primera de ellas parte de una imagen previamente escaneada con contenido Braille matemático

y genera un documento de texto con los caracteres reconocidos en la imagen. Por tanto estamos hablando de un problema de visión por computador.

La segunda fase, se encarga de interpretar los caracteres previamente reconocidos utilizando para ello un analizador sintáctico. Esta fase debe contemplar la posible existencia de errores en el documento, que pueden ser provocados por el reconocimiento incorrecto de los caracteres en la etapa anterior o bien por un fallo del alumno a la hora de escribir el documento con la máquina Perkins.

1.3.1. Primera fase: OBR

El OCR (Optical Character Recognition) consiste en el reconocimiento de los caracteres de la imagen que se crea al escanear un documento de texto. Un sistema OCR permite a un usuario modificar un texto escaneado, ya que reconoce cada uno de los caracteres del documento original en papel. De la misma idea parten los OBR (Optical Braille recognition), que son un tipo de OCR pero que permiten identificar caracteres del alfabeto Braille. Existen ciertas diferencias entre un OBR y un OCR debido a la naturaleza de los documentos con los que tratan cada uno de estos sistemas. Por ejemplo, un OBR no debe preocuparse por el tamaño de la fuente utilizada en la creación de un documento, ya que la apariencia de los caracteres Braille es siempre la misma, además, los caracteres Braille siempre están formados por seis puntos distribuidos en una matriz de tres filas y dos columnas. Otra diferencia es que un OBR no parte de un texto impreso como un OCR, sinó de la imagen resultante de escanear un documento con el relieve de los caracteres Braille, así que la imagen a tratar consiste en las sombras provocadas por los puntos perforados en la matriz Braille.

1.3.2. Segunda fase: analizador sintáctico

Debido al contenido matemático de los documentos Braille a traducir, podemos utilizar un Parser (analizador sintáctico) que defina los operadores matemáticos más comunes y que facilite la interpretación de las matemáticas escritas en Braille. El analizador sintáctico, comprueba si una secuencia de símbolos pertenecen al lenguaje generado por una gramática. Un mismo carácter Braille puede ser interpretado de distinta manera dependiendo del contexto en el que se encuentre. Por ejemplo: se utilizan los mismos símbolos Braille para representar dígitos y letras. La utilización del Parser nos permite decidir en cada momento el carácter correcto según el contexto en el que se encuentre éste. Un problema a considerar es la existencia de errores en documentos escritos incorrectamente (los alumnos pueden haber cometido fallos de escritura) el intérprete debe ser capaz de encontrar dichos errores, señalarlos y continuar con el proceso de traducción normalmente.

Esta memoria se centra exclusivamente en la explicación de la resolución de este punto: cómo interpretar un documento Braille de contenido matemático. El analizador sintáctico parte de un documento reconocido por un OBR y devuelve su traducción escrita en tinta.

1.3.3. Estructura de la memoria

Esta memoria se divide en cuatro capítulos estructurados de la siguiente manera:
El capítulo 1 introduce el problema que tratamos de resolver y explica los beneficios y la necesidad de un intérprete de contenido matemático en Braille.

El capítulo 2 expone el origen y funcionamiento del sistema Braille. Explica la codificación de texto y matemáticas en Braille, comentando cada uno de los operadores que más tarde serán incorporados a la gramática que permite la interpretación de documentos.
El capítulo 3 describe el proceso completo utilizado en la interpretación de un documento Braille y explica cada una de las fases que intervienen en la traducción de un documento: el reconocimiento de caracteres y su interpretación. En este capítulo se describe con detalle el proceso de creación de un analizador sintáctico mediante una gramática que permite interpretar tanto texto como matemáticas Braille.
El capítulo 4 muestra los resultados obtenidos al traducir imágenes de ejemplo utilizando el analizador sintáctico desarrollado.
Por último, el capítulo 5 expone las conclusiones finales y el trabajo futuro.

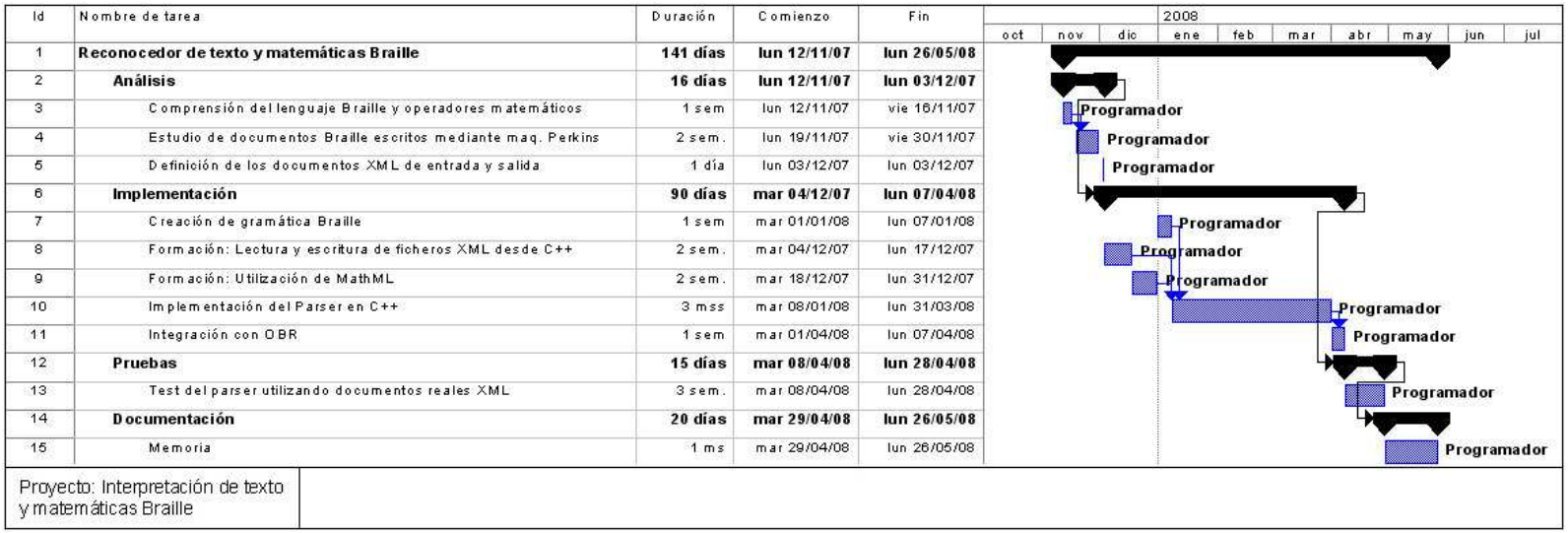
1.3.4. Objetivos y planificación

En esta sección se explican los objetivos del proyecto y la planificación establecida para alcanzarlos. Veremos como la planificación inicial, que estaba pensada para llevarse a cabo en un año académico, se debe ampliar debido a una serie de nuevas especificaciones necesarias sobre el proyecto inicial que se detectan durante la implementación del mismo. Por tanto, la realización del intérprete de Braille se ha dividido en dos grandes fases delimitadas por la ampliación de los objetivos iniciales y de la funcionalidad del intérprete desarrollado.

En la primera fase, mostrada en la planificación de la figura 1.2a, el objetivo del proyecto consistía en construir un parser que permitiera la interpretación de matemáticas y texto en documentos escritos en Braille. El parser a desarrollar tendría como entrada un fichero XML con la definición de una imagen Braille y como salida la interpretación de los caracteres codificados. Para poder definir las especificaciones iniciales del problema a solucionar, se contaba con una serie de documentos Braille escritos por niños utilizando una máquina Perkins. La traducción de estos documentos nos proporcionó una relación del conjunto de operadores matemáticos utilizados por los alumnos y del tipo de expresiones matemáticas que el parser debería ser capaz de traducir. Una vez realizado el análisis inicial, la implementación requería definir una gramática para la interpretación de los operadores Braille utilizados en los documentos y la creación de un proyecto en Visual C++ que codificara dicha gramática utilizando el lenguaje C++. La planificación mostrada en la figura 1.2a además de especificar el tiempo requerido para cada de una de esas tareas, muestra el tiempo planificado para la realización de esta memoria y para la fase de test de la aplicación.

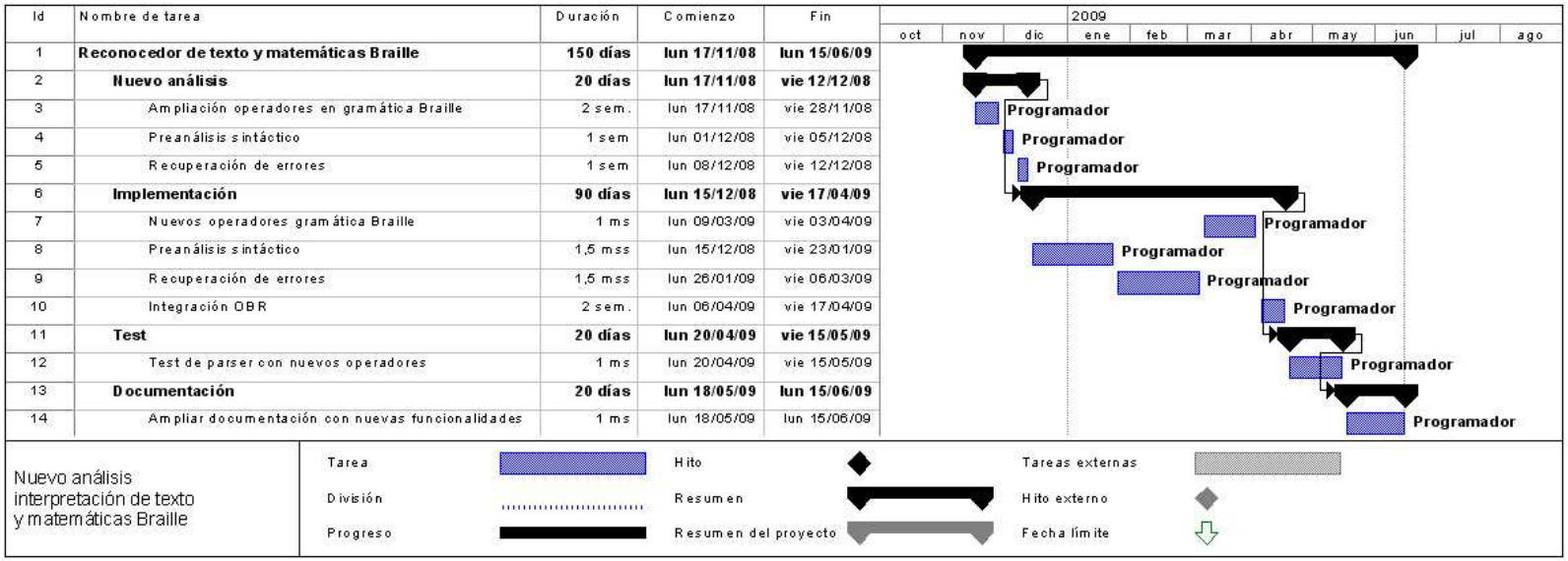
En esta fase de test, se utilizó un número mayor de imágenes al usado en el análisis inicial y se detectó que la gramática planteada inicialmente era limitada y que existía un número importante de documentos que no podían ser reconocidos mediante el parser implementado. Además, la presencia de errores sintácticos en los documentos provocaba que el análisis del mismo no finalizara correctamente. Con el tiempo disponible no era viable la realización de un nuevo análisis e implementación, por lo que se acaba definiendo una segunda fase del proyecto y una nueva planificación (figura 1.2b).

En esta segunda fase se realiza un nuevo análisis para mejorar la aplicación desarrollada inicialmente. El objetivo es incrementar el número de documentos de ejemplo que pueden ser



Id	Nombre de tarea	Duración	Comienzo	Fin
1	**Reconocedor de texto y matemáticas Braille**	**141 días**	**lun 12/11/07**	**lun 26/05/08**
2	**Análisis**	**16 días**	**lun 12/11/07**	**lun 03/12/07**
3	Comprensión del lenguaje Braille y operadores matemáticos	1 sem	lun 12/11/07	vie 16/11/07
4	Estudio de documentos Braille escritos mediante maq. Perkins	2 sem.	lun 19/11/07	vie 30/11/07
5	Definición de los documentos XML de entrada y salida	1 día	lun 03/12/07	lun 03/12/07
6	**Implementación**	**90 días**	**mar 04/12/07**	**lun 07/04/08**
7	Creación de gramática Braille	1 sem	mar 01/01/08	lun 07/01/08
8	Formación: Lectura y escritura de ficheros XML desde C++	2 sem.	mar 04/12/07	lun 17/12/07
9	Formación: Utilización de MathML	2 sem.	mar 18/12/07	lun 31/12/07
10	Implementación del Parser en C++	3 mss	mar 08/01/08	lun 31/03/08
11	Integración con OBR	1 sem	mar 01/04/08	lun 07/04/08
12	**Pruebas**	**15 días**	**mar 08/04/08**	**lun 28/04/08**
13	Test del parser utilizando documentos reales XML	3 sem.	mar 08/04/08	lun 28/04/08
14	**Documentación**	**20 días**	**mar 29/04/08**	**lun 26/05/08**
15	Memoria	1 ms	mar 29/04/08	lun 26/05/08

(a) Planificación inicial



Id	Nombre de tarea	Duración	Comienzo	Fin
1	**Reconocedor de texto y matemáticas Braille**	**150 días**	**lun 17/11/08**	**lun 15/06/09**
2	**Nuevo análisis**	**20 días**	**lun 17/11/08**	**vie 12/12/08**
3	Ampliación operadores en gramática Braille	2 sem.	lun 17/11/08	vie 28/11/08
4	Preanálisis sintáctico	1 sem	lun 01/12/08	vie 05/12/08
5	Recuperación de errores	1 sem	lun 08/12/08	vie 12/12/08
6	**Implementación**	**90 días**	**lun 15/12/08**	**vie 17/04/09**
7	Nuevos operadores gramática Braille	1 ms	lun 09/03/09	vie 03/04/09
8	Preanálisis sintáctico	1,5 mss	lun 15/12/08	vie 23/01/09
9	Recuperación de errores	1,5 mss	lun 26/01/09	vie 06/03/09
10	Integración OBR	2 sem.	lun 06/04/09	vie 17/04/09
11	**Test**	**20 días**	**lun 20/04/09**	**vie 15/05/09**
12	Test de parser con nuevos operadores	1 ms	lun 20/04/09	vie 15/05/09
13	**Documentación**	**20 días**	**lun 18/05/09**	**lun 15/06/09**
14	Ampliar documentación con nuevas funcionalidades	1 ms	lun 18/05/09	lun 15/06/09

(b) Planificación revisada

Figura 1.2: Planificación del intérprete de texto y matemáticas Braille.

reconocidos correctamente y permitir la traducción completa de documentos aunque éstos contengan errores de tipo sintáctico. Al analizar sintácticamente documentos Braille escritos con una máquina Perkins es muy común encontrar símbolos incorrectos, estos errores pueden ser provocados por un reconocimiento incorrecto de los caracteres o por un error cometido por el alumno al escribir el documento. Sin el control de errores del primer parser, la traducción de un documento se limitaba al número de líneas hasta encontrar el primer error sintáctico. En el nuevo análisis se tiene en cuenta la creación de un sistema de recuperación de errores que permita finalizar el análisis de documentos aunque éstos sean incorrectos sintácticamente. La nueva planificación muetra las nuevas tareas a realizar en la implementación, la mejora de la gramática inicial añadiendo nuevos operadores matemáticos y la implementación de un nuevo proceso llamado preanálisis sintactico. Por otra parte, muestra también el tiempo empleado en la codificación de las subrutinas que implementan la recuperación de errores. Durante esta memoria se explicará como estas nuevas especificaciones implican un cambio profundo en la aplicación, que incrementará notablemente el número total de horas final en la realización del proyecto. Todas estas modificaciones sobre el trabajo inicial hacen necesarios unos nuevos tests para la aplicación que se ven reflejados en la planificación al igual que la actualización de la documentación del proyecto.

Para el cómputo del número total de horas de realización del proyecto se tiene en cuenta una dedicación semanal aproximada de 8 horas en la primera fase del proyecto que suman un total de 226 horas. En la segunda fase el tiempo dedicado semanalmente es inferior, aproximadamente 5 horas semanales que se traducen en 150 horas destinadas al diseño de los nuevos requerimientos.

Capítulo 2

El lenguaje Braille

El origen de la escritura en relieve se remonta al siglo XIII cuando Al-Imam Al-Amadi empezó a utilizar caracteres táctiles para la lectura. Al-Amadi vendía libros en Arabia y al ser invidente necesitó desarrollar un sistema propio de caracteres en relieve para reconocer los libros y sus precios. Un siglo más tarde, un erudito árabe, Zain-Din Al Amidi, profesor de la madraza (escuela coránica) Moustanzarieh de Bagdad, empleaba un procedimiento parecido para identificar las obras de su valiosa biblioteca y marcar cierta información, sobretodo el precio de los libros. Ajustaba trozos de fino papel para formar las letras o los números, que pegaba en el interior de la tapa.

En 1808, Charles Barbier (1767-1841), un capitán de artillería de Luis XVIII, creó lo que denominó escritura nocturna. La idea del sistema era permitir que en la oscuridad los soldados pudiesen comunicarse en las trincheras sin tener que hablar o emplear luz delatando su posición. En su versión clásica, el código del Barbier comprende 36 sonidos repartidos en una tabla de 6 columnas, cada una de las cuales se divide en 6 casillas. Cada sonido puede representarse a través de 2 cifras, la primera indica el número de la línea, la segunda el número de la columna. En versión nocturna, esta «sonografía» es representada no por cifras, sino por puntos en relieve dispuestos en 2 columnas de 6 puntos cada una (figura 2.1). El número de puntos de la izquierda indica el número de la línea y el número de puntos de derecha nos informa del de la columna. Sin embargo, la sonografía de Barbier tenía algunos problemas importantes: representaba sonidos y no letras, se trataba de un código basado en el sonido del lenguaje, no tenía en cuenta para nada la ortografía, ni tampoco preveía la realización de operaciones matemáticas o la escritura de partituras musicales. Aunque sus dos principales fallos eran el número demasiado elevado de puntos y la forma de acceso a los diferentes sonidos, que impidían una lectura táctil rápida.

Louis Braille nació el 4 de enero de 1809 (1809-1852), en Coupvray, cerca de París. Quedó ciego por un accidente a los 3 años, en 1818 fue enviado a la Escuela para Ciegos de París, creada por Valentin Hauy, hizo rápidos progresos en todos sus estudios, especialmente en la ciencia y la música. En 1828, se hizo profesor del Instituto en el que se educó y empezó a desarrollar una idea, hallar un sistema que permitiera a un invidente leer y escribir en relieve, modificando la idea inicial de Charles Barbier. La virtud del sistema braille fue su sencillez y manejabilidad: 6 puntos, diseñados para ser reconocidos a través del tacto, y que permiten 63 combinaciones diferentes, que incluyen tanto las letras del alfabeto como los números, los signos de puntuación, etc.

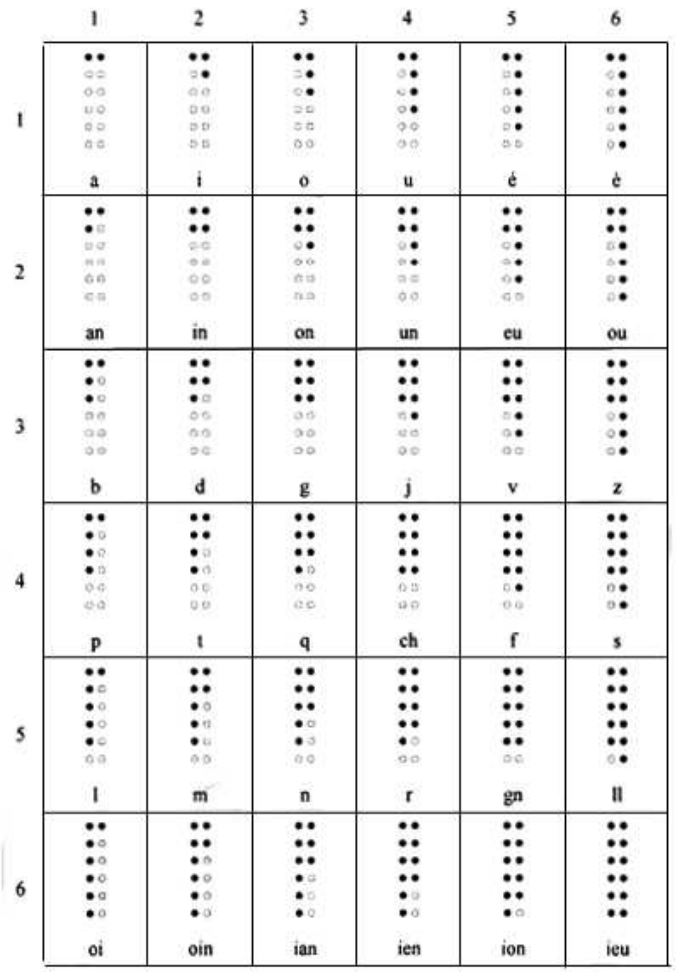


Figura 2.1: Tabla de la sonografía Barbier. Los puntos negros representan las partes en relieve de la matriz de doce puntos. Para cada sonido, el número de puntos de la izquierda corresponde al número de la línea, el número de puntos de derecha al número de la columna.

2.1. Codificación Braille

El sistema Braille se basa en la distribución de 6 puntos en una celda o cajetín Braille (Figura 2.2a), cuya forma y tamaño son estables y universales. El tamaño de cada punto Braille oscila entre 0,381 y 0,508 milímetros, con una distancia entre sí de 2,28 milímetros si pertenecen a la misma celdilla. La distancia horizontal entre celdillas es de 6,35 milímetros y la vertical entre líneas es de 10,16 milímetros. Louis Braille numeró las posiciones de los puntos dentro del cajetín: de arriba hacia abajo 1-2-3 al lado izquierdo y 4-5-6 al lado derecho. La combinación de estos puntos, es decir, aquellos que aparecen en relieve en cada cajetín, formarán cada letra o signo específico. En cada cajetín se puede formar una sola letra.

Lo más frecuente en la lectura Braille es que se realice con el dedo índice, el cual se desliza

(a) (b)

Figura 2.2: (a) Cajetín Braille, (b) Libro Braille

sobre los signos Braille, con la presión necesaria para una óptima percepción (Figura 2.2b). Durante la lectura, los dedos realizan tres tipos de movimientos: horizontales, verticales y de presión. Del mismo modo, las manos deben desplazarse en una línea de izquierda a derecha. Generalmente la mano derecha practicará la lectura y la izquierda debe seguirla y apoyarla al finalizar la línea de lectura y bajar al próximo renglón.
La escritura Braille puede realizarse con un instrumento llamado 'regleta' (Figura 2.3a). Consta de tres partes (tabla, regleta y punzón) o de dos partes (regleta y punzón). La regleta puede ser metálica o de plástico. Posee un extremo abierto y el otro con bisagra que permite abrir y cerrar. La hoja superior tiene unas oberturas llamadas cajetines y la hoja inferior contiene puntos en bajo relieve que guiarán la escritura Braille.

(a) (b) (c)

Figura 2.3: (a) Regleta (b) Máquina Perkins (c) Pulsaciones letra 'g' en máq. Perkins

El punzón suele ser de madera o plástico con punta de metal. Para escribir, se coloca la regleta con la apertura hacia el lado derecho, se abren las hojas metálicas de la regleta y se pone el papel encima de la hoja inferior. Se cierra la hoja superior y se presiona para fijar el papel. Cada cajetín de la regleta corresponde a un cajetín Braille. Con el punzón se presionan los puntos adecuados para formar la letra deseada de derecha a izquierda, presionando los puntos en posición inversa que en la lectura. La máquina Perkins (Figura 2.2b y Figura 2.2c) reproduce las letras sobre el papel tal como se leen. La persona puede leer y corregir inmediatamente lo escrito. Para formar cada letra se deben presionar de forma simultanea los puntos que la constituyen. En la figura 2.4 vemos la codificación Braille utilizada en castellano.

Tinta	Braille	Tinta	Braille	Tinta	Braille	Tinta	Braille
a		b		1		2	
c		d		3		4	
e		f		5		6	
g		h		7		8	
i		j		9		0	
k		l					
m		n					
ñ		o					
p		q					
r		s					
t		u					
v		w					
x		y					
z							

Figura 2.4: Alfabeto Braille

2.2. Codificación matemática Braille

Cada una de las codificaciones Braille utilizadas en los distintos idiomas tiene su propia notación matemática, en esta sección haremos una introducción a la notación matemática utilizada en la codificación Braille en castellano y catalán. Los operadores matemáticos que serán explicados a continuación son una pequeña muestra de las matemáticas que pueden llegar a ser representadas utilizando Braille y que permiten representar todo tipo de expresiones matemáticas como por ejemplo cálculo integral, límites, sucesiones, álgebra, teoría de conjuntos, lógica simbólica etc. El subconjunto de operadores que se explican a continuación son los utilizados en los documentos Braille analizados y escritos utilizando una máquina Perkins por alumnos del Centro de Recursos Educativos Joan Amades. Para una explicación en mayor detalle de las matemáticas en Braille consultar la referencia [15].

2.2.1. Representación de números enteros y decimales

Braille intenta reducir el número de caracteres de su alfabeto para facilitar tanto la lectura como la escritura. En la representación de números se utilizan los mismos símbolos que en la representación de letras. La presencia del símbolo clave () mantiene el valor de número mientras no se advierta de su desactivación, como ocurría por ejemplo al encontrar un operador matemático. En general, el valor de conversión de letras a números sólo se transmite a través de:

- Las diez primeras letras del abecedario (a-j) que corresponderán a los valores numéricos

(1-0)

- El punto de separación en grupos de tres cifras.
- La coma decimal. También puede representar el inicio del periodo en un número decimal.

2.2.2. Operadores matemáticos

2.2.2.1. Operaciones Aritméticas

Entendemos por operaciones aritméticas la suma, resta, multiplicación y división. En la figura 2.5 vemos la representación en Braille para cada uno de estos operadores. Mientras que la suma y la resta son representadas con un único símbolo, la multiplicación y división admiten distintas representaciones aunque las más utilizadas son las que aparecen en primer lugar en la figura. Tanto la multiplicación utilizada en aritmética como la utilizada en álgebra (en forma de punto) se representan en Braille utilizando un mismo símbolo.

Operación	Notación tinta	Notación Braille
Suma	+	
Resta	-	
Multiplicación	×	
	.	
División	÷	
	:	
	/	

Figura 2.5: Operaciones aritméticas

2.2.2.2. Representación de fracciones en forma simplificada

Además de los tres símbolos Braille que codifican la división y que no tienen ninguna restricción, existe una cuarta representación en la representación de fracciones que es la más utilizada al ser simplificada y que sólo se puede utilizar con "fracciones numéricas enteras de denominador positivo". En el caso de expresiones numéricas, las representaciones que utilizan los signos (, ) son didacticamente desaconsejables.

La división en su forma simplificada sigue las siguientes normas:

- Sólo aparece el signo de numero () al comienzo de la expresión, excepto si el numerador viene precedido de signo (+ , -) en cuyo caso precederán al símbolo de número.

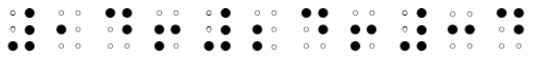
$\frac{1}{4} + \frac{2}{4} + \frac{3}{4}$ 

Figura 2.6: División simplificada

- El numerador (dividendo) se escribe a continuación del signo de número y desplazado a la parte inferior de la celda braille también denominada parte baja. Esto es sólo posible con fracciones numéricas ya que los dígitos (0..9) ocupan la parte alta de la celda braille.
- El denominador (divisor) se escribe a continuación SIN SIGNO DE NUMERO y ocupando la parte alta de la celda Braille.

Dígitos	0	1	2	3	4	5	6	7	8	9
Parte alta										
Parte baja										

Figura 2.7: Dígitos representados en la parte alta y baja de la celda Braille

De esta manera, desplazar los símbolos correspondientes al numerador a su parte baja permite diferenciar numerador de denominador y prescindir de signo (Figura 2.6).

2.2.2.3. Paréntesis auxiliar en Braille

Cuando en una expresión matemática un operando se encuentra sometido a distintas operaciones, existe un orden específico en el que realizar estas operaciones, esto normalmente se consigue mediante:

- La propia prioridad de las operaciones: potencia y raíz, multiplicación y división, suma y resta.
- El uso de paréntesis que dan prioridad a las expresiones que encierran.

Además de la prioridad de los operadores y del uso de paréntesis, en tinta podemos unificar expresiones mediante la situación espacial de las mismas. Es decir, la situación o colocación de una expresión respecto al resto agrupa todos los operandos de esa expresión. Sin embargo, la linealidad propia de Braille obliga a la utilización de un paréntesis específico de Braille o paréntesis auxiliar ( ) que permitirá representar este tipo de unificación.

El paréntesis específico o auxiliar de Braille ayuda a distinguir:

- Numerador o denominador muy complejos, unificados por la barra de fracción. (figura 2.9).
- La expresión radicando, unificada por la cobertura que le ofrece el trazo horizontal del signo raíz (figura 2.8).

- Exponentes o superíndices y subíndices; unificados/diferenciados por su desplazamiento respecto del nivel ordinario de escritura (figura 2.10).
- El índice de una raíz; diferenciado/unificado por su emplazamiento sobre el ángulo del signo de raíz (figura 2.8).

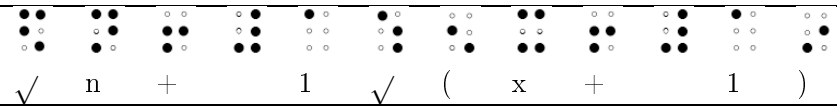
$\sqrt[n+1]{x+1}$	 √ n + 1 √ (x + 1)

Figura 2.8: Uso de paréntesis invisibles para distinguir el radicando en raíces de cualquier índice.

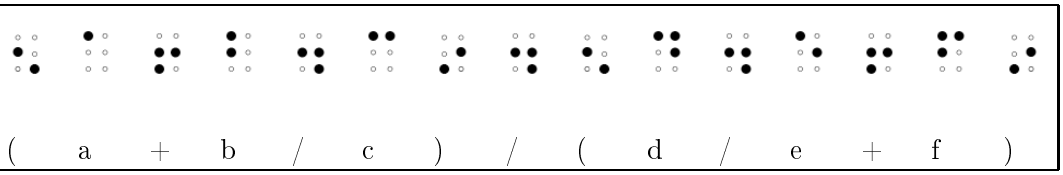
$\dfrac{a+\dfrac{b}{c}}{\dfrac{d}{e}+f}$	 (a + b / c) / (d / e + f)

Figura 2.9: Uso de paréntesis invisibles en fracciones

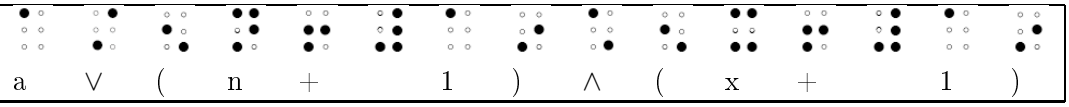
${a_{n+1}}^{x+1}$	 a ∨ (n + 1) ∧ (x + 1)

Figura 2.10: Uso de paréntesis invisibles con superíndices y subíndices.

2.2.2.4. Raíces

La representación de raíces implica por una parte la representación del radicando y por otra el orden de la raíz. Para indicar una raíz en Braille se utilizan los símbolos (⠜ y ⠻). Entre el primero y el segundo de estos símbolos se especifica el orden de la raíz tal y como vemos en la figura 2.11. En tinta, el signo radical 'encierra bidimensionalmente al radicando', bien por su trazo horizontal o por su parte inicial vertical. En Braille, únicamente al tener como radicando una expresión muy compleja se utilizarán los paréntesis invisibles (⠣ y ⠜) que permiten agrupar a los símbolos que la forman.

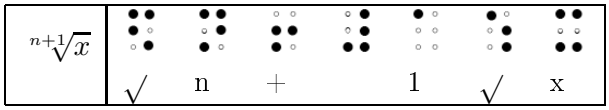


Figura 2.11: Ejemplo de raíz.

2.2.2.5. Índices

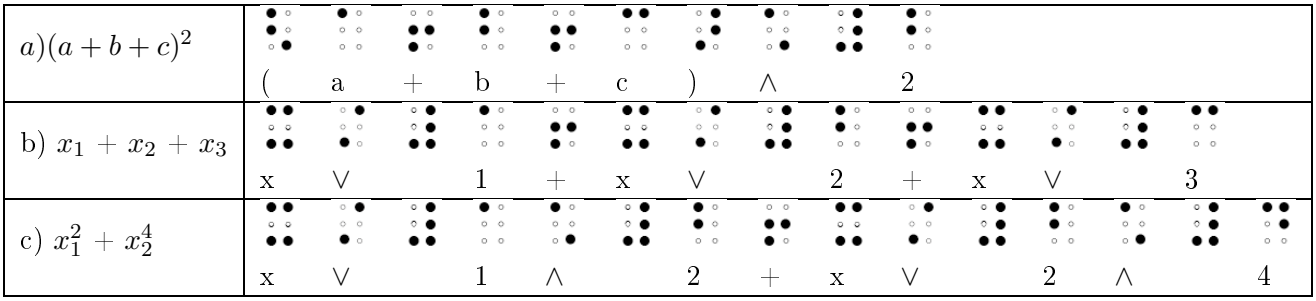


Figura 2.12: (a) Ejemplo de expresión con superíndices. (b) Ejemplo de expresión con subíndices. (c) Uso de superíndices y subíndices en una misma expresión.

Los índices son grupos de símbolos situados en algún lugar alrededor de una expresión base. En el caso de índices que ocupan la posición superior a la derecha de la expresión base hablamos de superíndices mientras que si ocupan la parte inferior derecha hablamos de subíndices (figura 2.12). Se utiliza el símbolo (⠘) para indicar que la siguiente expresión actúa como exponente o superíndice (figura 2.12a). De la misma manera, se utiliza el signo (⠸) para representar la presencia de un subíndice (figura 2.12b).

En el caso de una expresión base que contenga subíndice y superíndice, primero se representarán por convenio los subíndices seguidos de los superíndices. En la figura 2.12c definimos primero la expresión base, seguida del subíndice y superíndice para cada uno de los términos de la expresión.

2.2.3. Operaciones con enteros, escritura posicional

Los algoritmos tradicionales para la resolución de operaciones aritméticas se basan de forma esencial en la descomposición polinómica de un número en base 10, es decir la escritura posicional (figura 2.13).

Mediante la escritura posicional facilitamos:
a) Simplificar cálculos complejos en cálculos elementales.
b) Permitir la comprobación y el repaso de operaciones detectando rápidamente errores.
c) Transformar la mayoría de las reglas del algoritmo en esquemas y órdenes espaciales simples, fáciles de asimilar y fijar.

Aunque mediante la utilización de la máquina Perkins se hacen posibles estos objetivos, en Braille existen algunas limitaciones (escasez, homogeneidad de símbolos y limitación de la exploración mediante el tácto) que hacen que se simplifique la notación Braille cuando se utilizan

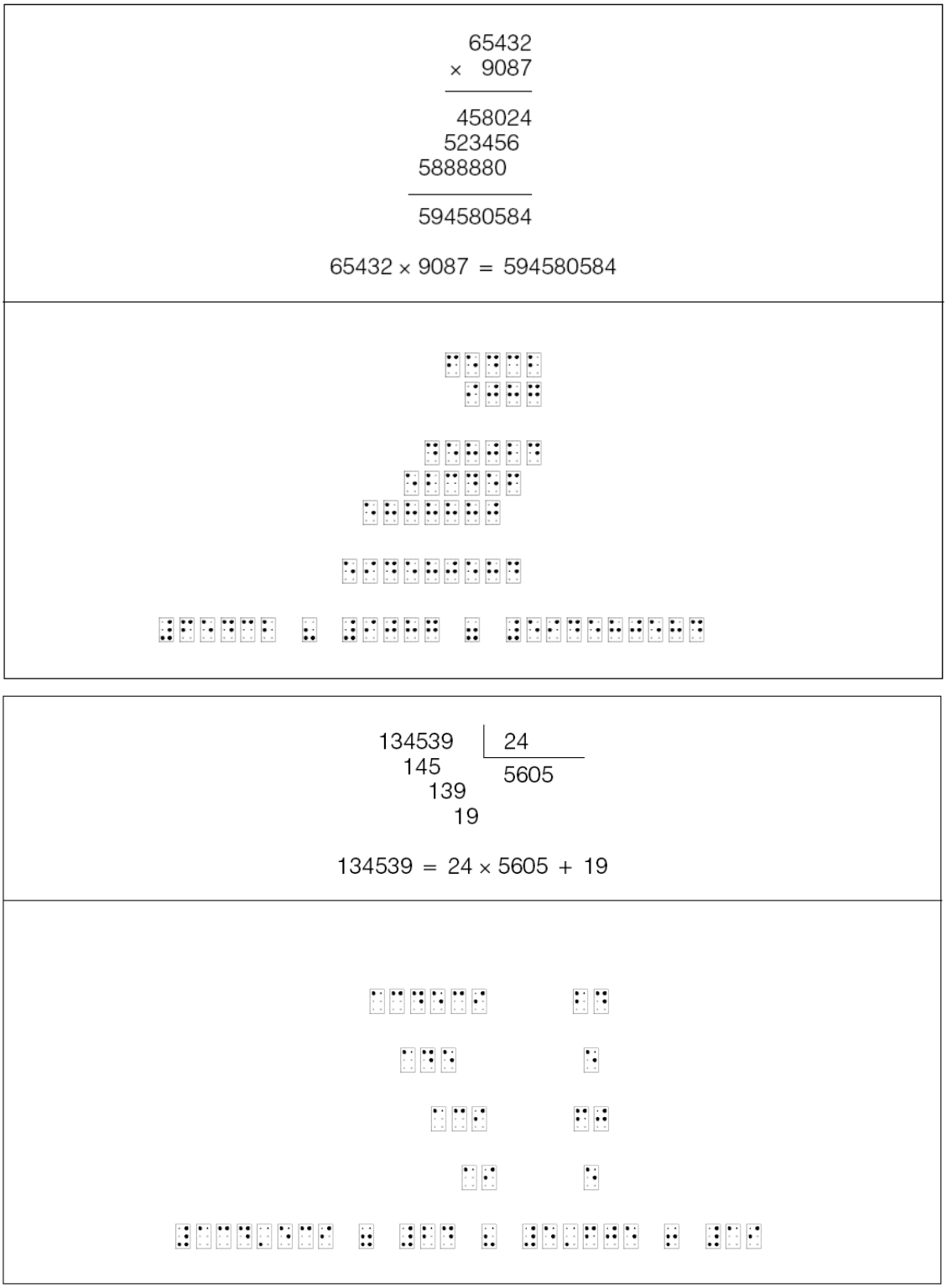


Figura 2.13: Ejemplos de multiplicación y división mediante escritura posicional tanto en tinta como en Braille.

este tipo de algoritmos.

Éstas son las medidas que se aplican para simplificar la escritura posicional en Braille:

1. Se omite el signo predecesor de número (⠼) ya que se presupone la condición de cantidad por el contexto aritmético en el que ya nos encontramos.
2. No se especifica el signo de operación. Nos apoyamos de nuevo en razones contextuales, para evitar confusiones entre signo de operación (posición baja) y cifra (posición alta), aunque estuvieran separados por espacio en blanco.
3. Las habituales líneas de separación en tinta se sustituyen por líneas en blanco, eliminando elementos superfluos y ahorrando tiempo en la confección de los resultados.
4. Se conserva esencialmente la configuración espacial de la expresión en tinta, excepto en la división, en la que el cociente se escribe progresivamente en columna, ver figura (2.13).

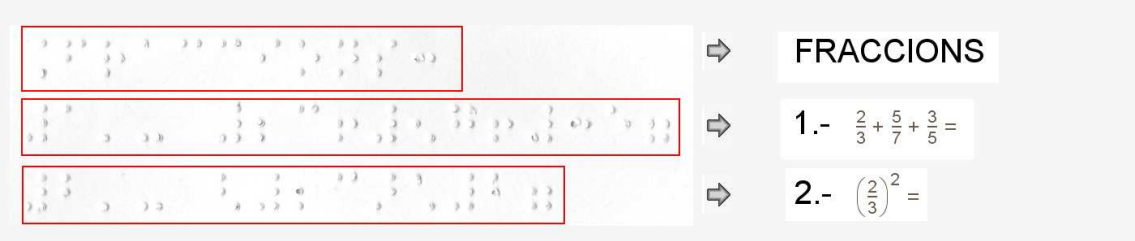
Capítulo 3

Sistema general

El proyecto se puede dividir en dos grandes módulos. La primera parte (figura 3.1a) se encarga de reconocer los caracteres existentes en la imagen del documento Braille escaneado y generar un documento XML con la información de los caracteres reconocidos. Por tanto estamos hablando de un problema de visión por computador, de reconocimiento de imágenes.

El segundo bloque (figura 3.1b), parte del documento XML generado en el módulo anterior, interpretándolo y generando un nuevo documento en formato MathML con la interpretación definitiva. Necesita por tanto un analizador sintáctico o Parser que se encarga de interpretar el documento XML inicial. Esta memoria explica el desarrollo de ese segundo módulo, la creación de un analizador sintáctico que reconoce contenido matemático y texto en un documento Braille.

(a)



(b)

Figura 3.1: (a) Proceso de reconocimiento de caracteres Braille por el OBR (Optical Braille recognition) (b) Proceso de interpretación y traducción de los caracteres Braille reconocidos

3.1. Reconocedor de escritos Braille

En este capítulo veremos paso a paso como funciona el reconocedor de escritos Braille. Veremos cada uno de los dos principales bloques que lo forman, el analizador léxico en primer lugar y después el analizador sintáctico. El analizador léxico se encarga de resolver el problema del reconocimiento de caracteres en la imagen original escaneada mientras que el analizador sintáctico interpreta el significado de esos caracteres en conjunto.

3.2. Analizador léxico: OBR (Optical Braille Recognition)

3.2.1. OBR (Optical Braille Recognition)

El análisis de documentos es un campo de la visión por computador que combina técnicas de análisis de imágenes y de reconocimiento de textos para el procesado e interpretación de documentos en papel (digitalizados mediante un escáner). El reconocimiento óptico de caracteres (OCR) consiste en la conversión automática de imágenes digitales de documentos de texto (adquiridos mediante escáner o cámara digital) a formato electrónico interpretando el contenido, es decir, reconociendo los caracteres. Existen muchas soluciones OCR, algunas incluídas con los propios escaners, que permiten escanear documentos en papel y reconocerlos como texto. Un sistema OCR permite por tanto evitar a un usuario tener que reescribir un documento ya impreso. Un sistema OCR consta de tres etapas: procesamiento de bajo nivel, segmentación/detección de la estructura del documento y reconocimiento de caracteres o clasificación. El procesamiento de bajo nivel consiste en aplicar técnicas de procesamiento de imágenes para la mejora de la imagen digital obtenida a partir del papel (aumento de contraste, filtrado para la eliminación de ruido que dificulta el reconocimiento, binarización, etc.) La segmentación es el proceso de separación de los componentes que forman la estructura física del documento y que se estructurarán de manera jerárquica (párrafos, líneas, palabras, caracteres). El proceso de reconocimiento o clasificación, por último, consiste en identificar para cada zona de la imagen que contiene un elemento que es un carácter, a qué clase pertenece, es decir, de qué carácter se trata. Este proceso consiste en comparar un conjunto de características extraídas de la imagen con unos patrones o modelos previamente aprendidos en una base de datos de consulta. Aunque el reconocimiento óptico de caracteres es un problema ya resuelto aun hay puntos que mejorar como el reconocimiento omnifuente, es decir, OCRs que sean capaces de reconocer textos imprimidos con cualquier tipografía, o bien el reconocimiento de documentos escritos a mano.

El reconocimiento de escritura Braille OBR (Optical Braille recognition), no es mas que un tipo de OCR. Presenta dos diferencias principalmente respecto los OCR clásicos. En primer lugar, el texto a reconocer no está impreso sino que está formado por el relieve que producen en el papel las perforaciones hechas por la máquina que imprime Braille o bien por el punzón utilizado en la escritura manual. La segunda diferencia es la tipografía. No tenemos una tipografía formada por un conjunto de símbolos (letras, números o símbolos especiales) sino que los caracteres se forman a partir de una matriz de seis puntos distribuidos en una matriz de dos columnas y tres filas. La codificación de cada carácter se establece en base a las perforaciones de estos puntos. Según esto, la segmentación de los caracteres en la imagen digital se hará en base a la detección de puntos que visualmente se aprecian como pequeñas sombras provocadas por el relieve de la perforación. En cuanto al reconocimiento, una vez identificada la posición

de cada carácter, consistirá en clasificar las combinaciones de seis puntos según si están activos (perforados) o no (no perforados) respecto a los patrones de referencia. Existen diversos trabajos relacionados con la lectura de Braille mediante técnicas de procesamiento de imágenes.

3.2.2. Analizador léxico

En el proceso de compilación del código fuente de cualquier programa, una de las fases fundamentales es el análisis léxico. La función principal del análisis léxico consiste en leer los caracteres de entrada del código fuente y elaborar una salida formada por símbolos o tokens. Por tanto, estos símbolos estarán formados por uno o más caracteres que juntos tienen un significado propio dentro del lenguaje de programación. En la mayoría de lenguajes de programación, se consideran símbolos o tokens a los identificadores, palabras reservadas, operadores, constantes y signos de puntuación como los paréntesis, la coma y el punto y coma. En el ejemplo de la figura 3.2 vemos el reconocimiento de tokens realizado por un analizador léxico para una función llamada par(). En la figura se puede observar como el analizador léxico reconoce algunas palabras reservadas como (int, if, else, return, paréntesis, llaves), Identificadores como el propio nombre de la función o los parámetros que recibe.

Durante el análisis léxico, el reconocimiento de algunos de estos tokens puede resultar sencillo, como pasaría con las palabras reservadas (int, if, else, return, paréntesis, llaves) ya que sabemos que los caracteres que deben ser encontrados durante el análisis, son los que deletrean cada una de estas palabras reservadas. Sin embargo, el reconocimiento de tokens más complejos, necesita la utilización de expresiones regulares para su reconocimiento.
Por ejemplo, si suponemos que los identificadores de la figura 3.2 se definen como una letra seguida de 0 o más letras y dígitos, necesitaríamos definir una expresión regular como la siguiente para el reconocimiento de estos identificadores:

$$\text{letra (letra} \mid \text{dígito)}^*$$

En esta expresión regular, la barra vertical significa "o", los paréntesis se usan para agrupar subexpresiones, el asterisco significa "cero o más casos" de la expresión entre paréntesis, y la unión entre letra y el resto de la expresión entre paréntesis indica concatenación.

Normalmente, la implementación final del analizador léxico acaban siendo una serie de subrutinas que son llamadas desde el analizador sintáctico para obtener los tokens siguientes tal y como se muestra en la figura 3.3. De esta manera, la comunicación entre el analizador léxico y el sintáctico es constante. El analizador sintáctico le pide al analizador léxico el siguiente token y el analizador sintáctico utiliza este token para aplicar la correspondiente regla de su gramática. Este procedimiento se verá en detalle en la siguiente sección en la que se explicará en el análisis sintáctico.

Hasta aquí hemos visto como funciona un analizador léxico que particiona el código fuente de un programa reconociendo los símbolos o tokens.

```
int par(int a)
{
        if (a%2 = = 0)
                return 1
        else  return 0
}
```

(a)

P_reservada:**int** Identificador:**par Paréntesis_abierto** P_reservada:**int** Identificador:**a Paréntesis_cerrado**
Llave_abierta
P_reservada:**if Paréntesis_abierto** Identificador:**a Módulo** Entero:**2 Asignación Asignación** Entero:**0 Paréntesis_cerrado**
P_reservada: **return** Entero:**1**
P_reservada:**else** P_reservada:**return** Entero:**0**
Llave_cerrada

(b)

Figura 3.2: (a) Secuencia de caracteres de entrada (b) Reconocimiento de símbolos o tokens a partir de la secuencia de entrada

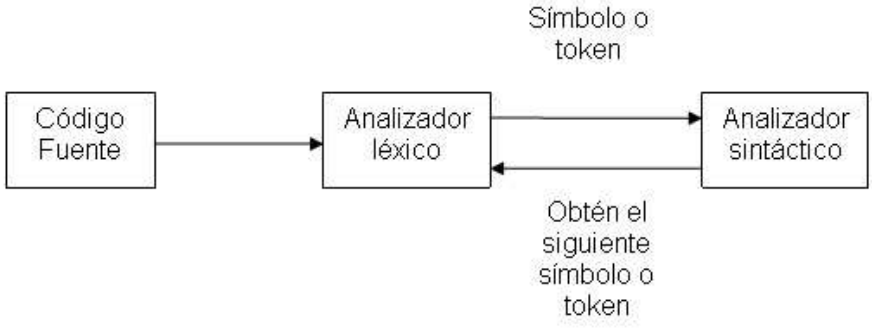


Figura 3.3: Interacción del analizador léxico con el analizador sintáctico.

El caso del reconocedor de matemáticas Braille es diferente ya que partimos de una imagen escaneada y no del código fuente escrito en un lenguaje de programación. Aquí, el problema principal a nivel léxico consiste en reconocer los símbolos Braille dentro de la imagen previamente escaneada. Por tanto, en nuestro caso en particular, la función de analizador léxico la está realizando el propio OBR generando cada uno de los tokens o símbolos Braille. Será el analizador sintáctico el encargado de realizar los procesos de interpretación y de traducción de cada uno de estos tokens en fases posteriores. Este analizador sintáctico, que será explicado en detalle en la sección 3.3, recibe como entrada un fichero XML (The Extensible Markup Language) que genera el propio OBR con la descripción de cada uno de los caracteres Braille reconocidos.

A continuación describiremos el fichero XML que permite la comunicación entre el OBR y el analizador sintáctico. El fichero XML contiene la descripción de cada una de las páginas escaneadas, líneas y caracteres. Su estructura nos especifica qué caracteres contiene cada una de las líneas, posición de estos caracteres dentro de la línea e información sobre el autor del documento. A continuación tenemos una lista de los diferentes tags o etiquetas que permiten definir una página braille completamente en XML.

Etiquetas XML utilizadas en la descripción de un documento Braille:

- <PAGINA>: El fichero XML puede contener la descripción de más de una imagen escaneada, se utiliza una etiqueta página para agrupar todos los caracteres que aparecen en cada una de las imágenes. Una etiqueta <pagina>contiene múltiples etiquetas de tipo <fila>. Una etiqueta página cuenta con una serie de atributos que describen la página:
  - DATA: Fecha del documento escaneado.
  - ALUMNE: Nombre del alumno que realizó el documento.
  - NUMLINIES: Número total de filas del documento.
  - NUMPIXELS_X,NUMPIXELS_Y: Tamaño de la imagen en píxeles.
  - DH, DV: Distancia horizontal y vertical en píxeles entre los puntos de una misma celda Braille. Un documento Braille puede haberse escaneado con distinta resolución. Por eso la distancia entre los puntos de una celda Braille puede cambiar de un documento a otro.
  - DISTINT: Este atributo indica la distancia en píxeles entre dos caracteres Braille.
- <FILA>: Una etiqueta <fila> del fichero XML se utiliza para describir cada una de las líneas del documento Braille. Una etiqueta <fila> contiene múltiples etiquetas <caracter> para cada uno de los caracteres que forman la fila. Una <fila> tiene una serie de atributos NL, NC, XS, YS, XI, YI que describen el tamaño y el contenido de la fila. El atributo NL indica el número de fila que estamos codificando. NC el número de caracteres que tiene la fila. Los atributos (XS,YS) y (XI,YI) son las coordenadas en pixeles de la esquina superior izquierda y de la esquina inferior derecha del rectángulo que contiene todos los caracteres de la fila.
- <CARACTER>: Una etiqueta de tipo carácter describe en formato binario cada uno de los caracteres Braille reconocidos por el analizador léxico. La codificación binaria utiliza 6 bits (0,1) para representar cada uno de los puntos de una celda Braille. Además cada carácter tiene un atributo 'POSX' con el valor del desplazamiento horizontal (en píxeles) respecto al margen izquierdo del documento.

  La descripción de los bits de estos caracteres se realiza en el mismo orden utilizado para numerar los puntos en el cajetín braille (Figura 3.4).

  Por ejemplo, el carácter braille (⠖) utilizado para representar la suma (+) se representaría dentro de un tag carácter de la siguiente forma:
  <CARACTER POSX='105'>011010</CARACTER>

En la figura 3.7 tenemos un ejemplo del fichero XML generado por el analizador léxico y que corresponde a la imagen Braille de la figura 3.6 creada utilizando una máquina Perkins . El recuadro en esta imagen representa la expresion de la figura 3.5.

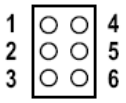


Figura 3.4: Cajetín o celda Braille

$$\frac{(5^4 \cdot 5^2 \cdot 5^3)}{5^8} =$$

Figura 3.5: Expresión en tinta

3.3. Analizador sintáctico

Existen muchas diferencias entre la interpretación de documentos con texto Braille únicamente y la interpretación de documentos con contenido matemático en Braille. A continuación explicaremos las diferencias que hacen necesaria la utilización de un analizador sintáctico o parser en la interpretación de documentos con contenido matemático.

Cuando tratamos de reconocer texto Braille, una vez reconocidos los caracteres mediante el OBR o analizador léxico, la traducción de texto Braille a tinta es un problema fácil de solucionar. Simplemente basta con recorrer cada uno de los caracteres del documento Braille y hacer una traducción directa de estos caracteres a nuestro alfabeto, utilizando para ello una tabla de traducción con las posibles traducciones de un carácter Braille.

Por ejemplo, en la tabla de traducción de la figura 3.8 vemos la diferente interpretación de un mismo carácter Braille en catalán o castellano. Al traducir un documento en castellano que contenga el carácter (⠿) nos dirigiremos a la primera y segunda columna de la tabla para recuperar los valores (Á y á) mientras que si traducimos un documento en catalán nos dirigiremos a la tercera y a la cuarta columna para recuperar (À o à). En esta tabla vemos que la representación de un carácter es igual en mayúsculas y minúsculas. En Braille, la presencia del símbolo (⠠) delante de una letra implica la representación del carácter en mayúsculas, por eso tendríamos que guardar en la tabla de traducción los caracteres en su representación en mayúsculas y en minúsculas. En la figura 3.9 podemos ver un ejemplo de traducción directa de un texto Braille a castellano utilizando una tabla simple de traducción como la mostrada en la figura 2.4.

Sin embargo, la interpretación de matemáticas en Braille no es directa. A continuación enumeramos una serie de problemas que requieren la utilización de un analizador sintáctico para ser solucionados. En la Figura 3.10 se interpreta la expresión $\sqrt[b+1]{a+2}$, una de las razones que requieren la utilización de un analizador sintáctico para su correcta interpretación es la notación utilizada para representar la raíz. En tinta, al representar una raíz utilizamos un símbolo característico que mediante su trazo horizontal enmarca los operandos y expresiones a los que afecta. Sin embargo en Braille, debido a la representación matricial de los caracteres en filas y columnas, no podemos representar este trazo horizontal y necesitamos una serie de símbolos que nos ayuden a representarlo. Por una parte el índice de la raíz estará situado entre dos símbolos (⠿) y (⠳). Por otra, para representar la base de la raíz necesitamos los dos paréntesis invisibles (⠐) y (⠂) propios de Braille y que fueron explicados en la sección 2.2.2.3. Estos paréntesis se

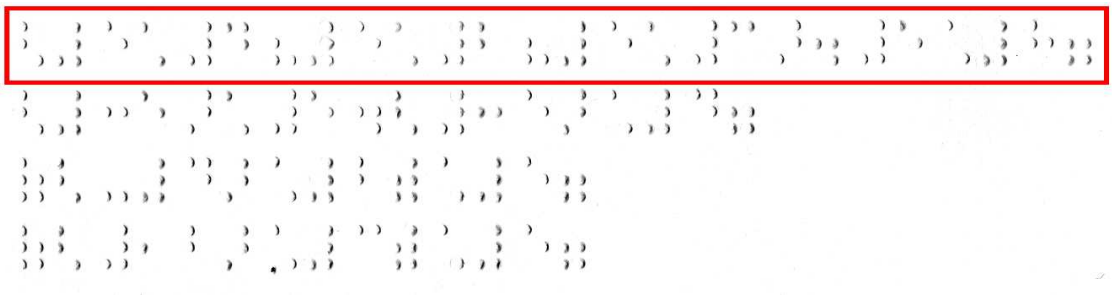


Figura 3.6: Representación de la expresión $\frac{(5^4 \cdot 5^2 \cdot 5^3)}{5^8} =$ escrita en Braille utilizando una maquina PERKINS.

utilizan para agrupar elementos en Braille que forman expresiones complejas, en tinta no son necesarios y no deben traducirse como los paréntesis visibles utilizados comúnmente en matemáticas. Además de la representación de la raíz, en la traducción de la figura 3.10 deberíamos tener en cuenta la diferente interpretación de un carácter Braille. En esta traducción se observa cómo el mismo símbolo (⠼) se utiliza tanto para expresar el número '2' en la base como el carácter 'b' en el índice. Sin embargo, la sola presencia del símbolo (⠰) indicador de precedencia de numero antes del símbolo (⠼) cambia la interpretación del carácter 'b' por la de '2'. Este mismo caso ocurre con el carácter (⠁) utilizado para representar tanto el carácter 'a' en la base como el número '1' en el índice.

En la figura 3.11 se muestra otro ejemplo de la necesidad de utilizar un analizador sintáctico para la interpretación de matemáticas Braille. El ejemplo muestra el uso de la división Braille simplificada explicada en la sección 2.2.2.2. Recordemos que este tipo de representación de fracciones no utiliza el operador división en ningún momento. Representa los dígitos del numerador de la fracción desplazando los puntos de la celda Braille a su parte baja. El denominador le sigue a continuación con los puntos de cada uno de los dígitos en su parte alta. Este tipo de división sería imposible de interpretar utilizando únicamente tablas de traducción ya que estamos imprimiendo un signo de división que no está en el texto a traducir. Otro motivo para utilizar un analizador sintáctico son las distintas traducciones que puede tener un mismo símbolo Braille. Observamos en la figura como el valor (⠾) se utiliza tanto para representar el valor numérico '7' en el denominador de la fracción cómo a la variable 'g'. La presencia del carácter (⠰) delante de los caracteres Braille que representan a las letras (a-j) modifica su valor por el de (1-0). Si desplazamos los puntos del carácter braille (⠾) a su parte baja, obtenemos el carácter (⠷) que a su vez puede tener dos traducciones distintas dependiendo del contexto. Este símbolo se traduce como el número '7' cuando forma parte del numerador de una fracción, ya que los puntos de la celda braille que representan el número '7' se desplazan a la parte baja. El mismo símbolo tal y como se ve en la figura representa también la igualdad matemática.

La utilización de un analizador sintáctico resuelve otro problema que no es propio de la interpretación Braille sino de la correcta interpretación de la precedencia de los operadores matemáticos. Si consideramos la expresión 2+3/5 podríamos tener dos posibles interpretaciones: 2+(3/5) o (2+3)/5. Al traducir esta expresión de Braille a tinta debemos saber si representar la

Contenido XML	Símbolo	Notación Braille
<XML version='1.0'>		
<PAGINA ALUMNE='GABRIEL' DATA='06-06-09'>		
<FILA NL='1' NC='26' XS='40' YS='11' XI='1080' YI='44'>		
<CARACTER POSX='40'>110001</CARACTER>	(	
<CARACTER POSX='80'>001111</CARACTER>	NUM	
<CARACTER POSX='120'>100010</CARACTER>	(5 | e)	
<CARACTER POSX='160'>100001</CARACTER>	Exponencial	
<CARACTER POSX='200'>001111</CARACTER>	NUM	
<CARACTER POSX='240'>100110</CARACTER>	(4 | d)	
<CARACTER POSX='280'>011001</CARACTER>	*	
<CARACTER POSX='320'>001111</CARACTER>	NUM	
<CARACTER POSX='360'>100010</CARACTER>	(5 | e)	
<CARACTER POSX='400'>100001</CARACTER>	Exponencial	
<CARACTER POSX='440'>001111</CARACTER>	NUM	
<CARACTER POSX='480'>110000</CARACTER>	(2 | b)	
<CARACTER POSX='520'>011001</CARACTER>	*	
<CARACTER POSX='560'>001111</CARACTER>	NUM	
<CARACTER POSX='600'>100010</CARACTER>	(5 | e)	
<CARACTER POSX='640'>100001</CARACTER>	Exponencial	
<CARACTER POSX='680'>001111</CARACTER>	NUM	
<CARACTER POSX='720'>100100</CARACTER>	(3 | c)	
<CARACTER POSX='760'>001110</CARACTER>	)	
<CARACTER POSX='800'>010011</CARACTER>	/	
<CARACTER POSX='840'>001111</CARACTER>	NUM	
<CARACTER POSX='880'>100010</CARACTER>	(5 | e)	
<CARACTER POSX='920'>100001</CARACTER>	Exponencial	
<CARACTER POSX='960'>001111</CARACTER>	NUM	
<CARACTER POSX='1000'>110010</CARACTER>	(8 | h)	
<CARACTER POSX='1040'>011011</CARACTER>	=	
</FILA>		
</PAGINA>		

Figura 3.7: Representación de la expresión matemática en el fichero XML

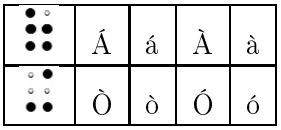


Figura 3.8: Tabla de traducción de texto Braille con caracteres en castellano y catalán.

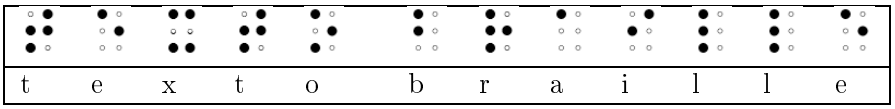


Figura 3.9: Traducción de texto Braille a tinta.

primera interpretación $2+\frac{3}{5}$ o la segunda $\frac{2+3}{5}$. Necesitamos conocer la prioridad de cada uno de los operadores para saber el orden en el que se deben agrupar los operandos. En aritmética, la multiplicación y la división tienen mayor prioridad que la suma y la resta, es por eso que en este caso la interpretación correcta sería $2+\frac{3}{5}$. Mediante la utilización de un analizador sintáctico y de la correspondiente gramática, resolveremos los problemas de precedencia de operadores.

Además de las razones hasta aquí explicadas, se debe tener en cuenta la naturaleza de los documentos o imágenes escaneadas. Recordemos que los documentos interpretados, previamente han pasado un proceso de reconocimiento mediante un OBR. Este OBR puede haber cometido errores, reconociendo caracteres incorrectamente y por tanto podrían aparecer caracteres fuera de contexto y que no tienen significado en el lugar del documento en el que aparecen. Además, los niños pueden haber cometido fallos en la realización del documento utilizando la máquina PERKINS, por lo que puede ser corriente encontrar errores sintácticos, olvido de algunos símbolos necesarios etc. Tanto los errores de un tipo como los de otro deben ser detectados y reportados para poder seguir con el análisis normal del documento.

Todos los puntos explicados anteriormente hacen que el proceso de interpretación de las matemáticas Braille sea similar a las técnicas empleadas por los compiladores para validar el código fuente escrito en un lenguaje de programación. Por tanto recurriremos a éstas técnicas, principalmente a la definición de un analizador sintáctico para tratar de interpretar las expresiones matemáticas Braille.

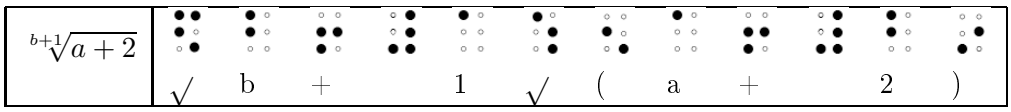


Figura 3.10: Traducción de una expresión de matemáticas Braille a tinta.

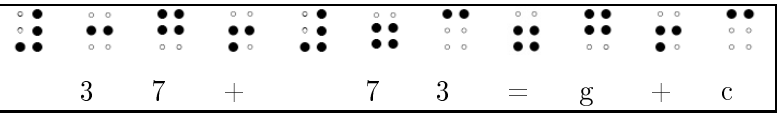
$\frac{3}{7} + \frac{7}{3} = g + c$	 3 7 + 7 3 = g + c

Figura 3.11: Ejemplo de expresión compleja en matemáticas Braille.

3.3.1. Definición de gramática y BNF

3.3.1.1. Gramática

Un analizador sintáctico o Parser se encarga de comprobar si una secuencia de símbolos o tokens pertenecen al lenguaje generado por una gramática. Recibe como entrada los tokens que le entrega el analizador léxico y comprueba si esos tokens van llegando en el orden correcto (orden definido por la gramática). Una gramática define una especificación sintáctica precisa de un lenguaje de programación. A partir de la definición de una gramática se puede construir automáticamente un analizador sintáctico eficiente que determine si un programa es correcto desde el punto de vista sintáctico.

Una gramática G es una tupla (T,N,I,P) donde:

- T: Representa al conjunto de símbolos terminales. Los terminales son los símbolos básicos que forman nuestras secuencias de símbolos.
- N: Representa al conjunto de símbolos no terminales. Los no terminales son variables sintácticas que denotan conjuntos de secuencias.
- I $\in$ N: símbolo no-terminal inicial.
- P: Definen el conjunto de producciones. Las producciones especifican cómo se pueden combinar los elementos terminales y no terminales para formar secuencias. Una producción se representa como un elemento no terminal, seguido por una flecha y continuación una secuencia de elementos terminales y no terminales.
- $\Rightarrow {}^{*}_{G}$ denotamos cero o más pasos de derivación.
- $\alpha\beta\varphi \Rightarrow \alpha\delta\varphi$ es un paso de derivación si $\beta \Rightarrow \delta \in$ P
- El **lenguaje L** generado por la gramática G es: L=$\{x \in T^* \mid I \Rightarrow^* x\}$

El lenguaje (L) está formado por todas las posibles secuencias de símbolos terminales (T) generados a partir del símbolo no terminal inicial (I). Se utilizan las producciones para derivar del símbolo inicial a los elementos terminales.

En el caso de nuestro analizador sintáctico, utilizamos una gramática que define un subconjunto de matemáticas en Braille. El analizador sintáctico Braille comprueba si la secuencia de tokens de entrada pertenecen al conjunto de las expresiones generables por nuestra gramática. Esta gramática implementa los operadores y la notación explicada en la sección 2.2, codificación matemática Braille.

3.3.1.2. Notación BNF

Utilizamos la notación BNF (Backus normal form) para definir la gramática que define las matemáticas Braille. La notación BNF permite describir de una manera formal un lenguaje de programación.

- **Regla BNF:**
  no terminal::= expresión BNF

- **Terminal:**
  - Forma única: secuencia de caracteres que la forma. (ejem: +,-.*,/)
  - Forma variable: Nombre que la identifica (ejem: Identificador, numero, String, literal)

- **No Terminal:** Identificador entre <>
  - Ejem: <expresión>, <instrucción>.

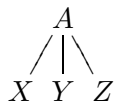
- **Secuencia vacía:** λ

- **Operadores:**
  -Unión: $\alpha \mid \beta$
  -secuencia: $\alpha\beta$
  -opcional: $[\alpha] = \lambda \mid \alpha$
  -repetición: $\{\alpha\} = \lambda \mid \alpha \mid \alpha\alpha \mid \alpha\alpha\alpha\alpha$

3.3.1.3. Árbol de análisis sintáctico

Un árbol de análisis sintáctico reproduce la estructura de una expresión para una gramática definida. En el árbol sintáctico vemos el proceso de derivación desde el símbolo inicial a una secuencia que pertenece al lenguaje generable por la gramática.

Si suponemos una producción de tipo A→XYZ, esta producción se representa mediante una árbol sintáctico en el cuál el nodo central aparece etiquetado con el valor terminal A y con tres nodos hijos X,Y,Z

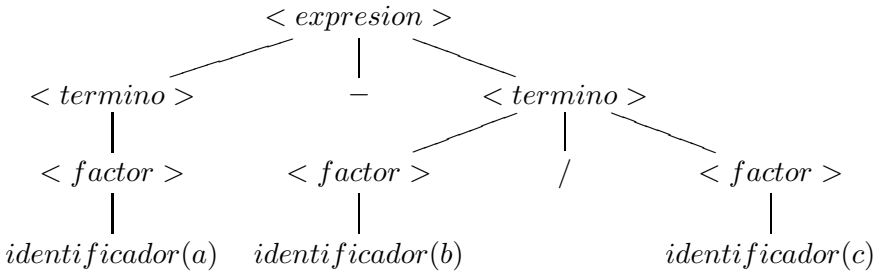


Dada una gramática, un árbol sintáctico es un árbol con las siguientes propiedades:
1. La raíz está etiquetada con el símbolo inicial.
2. Cada hoja está etiquetada con un símbolo terminal.
3. Cada nodo interior está etiquetado con un no terminal.
4. Una producción por ejemplo: A → X_1,X_2....X_n sería representada como el nodo no terminal A y X_1,X_2....X_n las etiquetas de los hijos de ese nodo. Los símbolos X_1,X_2....X_n podrían ser tanto elementos terminales como no terminales.

A continuación veremos un ejemplo del árbol sintáctico que representa la secuencia de caracteres: a-b/c para una gramática dada. Si definimos una gramática que permite interpretar las operaciones de suma, resta, multiplicación y division como:

<expresión>::= <término>('+' | '-') <término>
<término>::= <factor>('*'|'/')<factor
<factor>::= (identificador | numero)

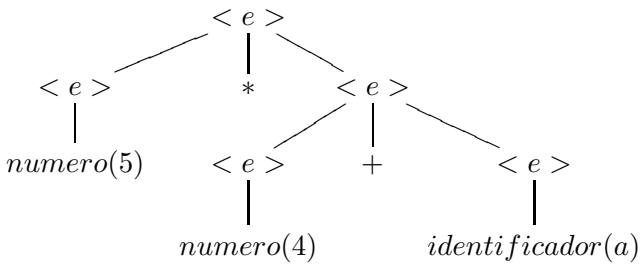
El árbol generado para la expresión a-b/c será el siguiente:

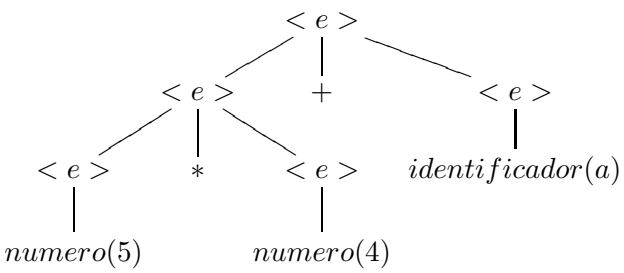


Definimos una gramática como **ambigua** cuando tenemos una gramática que puede representar dos o más árboles sintácticos distintos para una misma expresión. Este tipo de gramáticas deberían evitar utilizarse en la creación del analizador sintáctico ya que una misma expresión podría acabar teniendo múltiples significados. A continuación tenemos un ejemplo de gramática ambigua:

<expresion>::= <expresion>(+|-|*|/) <expresion>| numero | identificador

Si representamos 5*4+a utilizando la gramática anterior tenemos estas dos posibles interpretaciones:





3.3.1.4. Parser descendente y ascendente

Existen diversos algoritmos para crear analizadores sintácticos de tiempo lineal. Principalmente este conjunto de algoritmos se pueden dividir en dos clases, descendentes y ascendentes. En esta sección haremos una pequeña introducción al funcionamiento de este tipo de parsers y veremos cómo la eficiencia de un parser descendente hace que implementemos nuestro reconocedor de matemáticas Braille mediante este tipo de parser. Para una referencia completa sobre el diseño de parsers consultar [16].

Los parsers descendentes parten del símbolo inicial de la gramática el cuál acabará derivando en la secuencia de entrada. A medida que se lee cada uno de los símbolos de la entrada se aplica una de las reglas de la gramática formando de esta manera el árbol sintáctico de arriba a abajo. Este tipo de algoritmos descendentes son llamados también LL (Left - Left) ya que la secuencia de entrada se lee de izquierda a derecha (Left to right) y la derivación se hace a partir de la parte izquierda de las producciones (Leftmost).

Veamos con un ejemplo cómo funciona un parser descendente que reconoce una secuencia de caracteres a partir del símbolo inicial S.

Si consideramos la siguiente gramática:

$$
\begin{aligned}
&S \rightarrow aS \\
&S \rightarrow bAC \\
&A \rightarrow cB \\
&B \rightarrow d \\
&C \rightarrow e
\end{aligned}
$$

Podemos reconocer la secuencia de caracteres abcde mediante las siguientes derivaciones:

S→aS→abAC→abcBC→abcdC→abcde

En este caso, cada uno de los símbolos de la secuencia de entrada abcde nos indica inequívocamente qué regla de la gramática debemos aplicar. El proceso de reconocimiento de la secuencia se lleva a cabo recorriendo la entrada de izquierda a derecha y para la derivación se utiliza la parte izquierda de cada una de las reglas de la gramática.

Por el contrario, los parsers ascendentes parten de la secuencia de entrada y aplicando las distintas reglas de la gramática llegan al símbolo inicial construyendo así un árbol de derivación de abajo a arriba. Se puede considerar este proceso como 'reducir' una secuencia de entrada al símbolo inicial de la gramática. En cada paso de reducción se sustituye una subsecuencia de la secuencia de entrada y que concuerda con el lado derecho de una producción por el símbolo del lado izquierdo de dicha producción hasta llegar al símbolo inicial.

Los parsers ascendentes se suelen llamar LR (Left - right) pues la entrada se lee de izquierda a derecha (Left to right) y la derivación se produce por la parte derecha de la gramática (Rightmost).

A continuación veremos un ejemplo de parsing ascendente en el que reducimos una secuencia de entrada hasta el símbolo inicial S.

Para la siguiente gramática:

$$S \rightarrow aABe$$
$$A \rightarrow Abc$$
$$A \rightarrow b$$
$$B \rightarrow d$$

La secuencia abbcde se puede reducir hasta el símbolo inicial S mediante los siguientes pasos:

$a\underline{bb}\underline{c}\underline{d}e \rightarrow a\underline{A}\underline{bc}\underline{d}e \rightarrow aA\underline{d}e \rightarrow \underline{aABe} \rightarrow S$

En la cadena inicial abbcde existen tres subsecuencias que son candidatas a poderse derivar: b,b y d. De estas tres subsecuencias derivamos la primera aparición de b debido a que es la situada más a la izquierda. Utilizando la regla A → b obtenemos una nueva cadena aAbcde. Esta cadena a su vez tiene tres subcadenas derivables Abc, b y d. De nuevo derivamos la subcadena situada más a la izquierda, utilizando la regla A → Abc conseguimos reducir la cadena Abc y obtenemos una nueva secuencia aAde. En la cadena aAde solo es posible reducir el carácter d, utilizando la regla B → d. Finalmente y mediante la utilización de la regla S → aABe se reduce la cadena aABe al símbolo inicial S.

Normalmente y debido a su eficiencia, se utilizan analizadores sintácticos descendentes implementados a mano para reconocer gramáticas de tipo LL. Los analizadores ascendentes, que pueden reconocen gramáticas de tipo LR mucho más complejas, se construyen mediante herramientas automatizadas. En la realización del reconocedor de matemáticas Braille definiremos una gramática LL implementada mediante un analizador sintáctico descendente.

3.3.1.5. Gramáticas LL(k) y LL(1)

El primer problema que aparece con el análisis sintáctico descendente es que a partir del nodo raíz, el analizador sintáctico puede no elegir correctamente las producciones adecuadas para llegar a la secuencia de símbolos que debe reconocer. En el momento en el que el analizador descubre que es imposible alcanzar esa secuencia de caracteres de entrada, debe deshacer las

últimas producciones aplicadas y volver al estado en el cuál era posible alcanzar esta secuencia de nuevo. Este proceso de recuperación de un estado anterior es conocido como backtracking. La utilización de backtracking durante el análisis sintáctico tiene un alto coste computacional ya que el analizador explora todos los posibles caminos hasta encontrar la solución. Sin embargo, esta técnica puede llegar a ser evitada si utilizamos un análisis descendente determinista. Un analizador será determinista cuando solo dependa del símbolo actual en la secuencia de entrada para decidir cómo expandir cada uno de los elementos no terminales.

Las gramáticas LL(k) son un conjunto de las gramáticas libres de contexto. Permiten un análisis descendente determinista (sin backtracking) por medio del reconocimiento de la secuencia de entrada de izquierda a derecha (left to right) derivando siempre por la izquierda (leftmost). Para realizar las sucesivas derivaciones únicamente se examinan k símbolos o tokens. En el caso de tener que examinar un único símbolo de entrada k=1 estaríamos hablando de una gramática LL(1). Las gramáticas LL(1) permiten construir un analizador determinista descendente únicamente examinando en cada momento el símbolo actual de la cadena de entrada. Además tienen varias propiedades distintivas, ninguna gramática ambigua o recursiva por la izquierda puede ser LL(1).

Para que una gramática con producciones sea LL(1), el conjunto de elementos primeros[1] de las producciones de sus no terminales tienen que ser disjuntos. Es decir, para cada A ϵ N o símbolo no terminal, sus alternativas comienzan con un símbolo terminal distinto:

$$\forall A \rightarrow \beta, A \rightarrow \beta' : \beta \neq \beta' \Rightarrow \text{Primeros}(A \rightarrow \beta) \cap \text{Primeros}(A \rightarrow \beta') = \varnothing$$

Por todos los motivos aquí explicados implementaremos el reconocedor de texto y matemáticas Braille mediante un analizador sintáctico descendente que defina una gramática LL(1). Cada uno de los elementos terminales de está gramática LL(1) serán los símbolos Braille.

3.3.1.6. Construcción de un Parser a partir de la descripción de una gramática en BNF

En las siguientes secciones definiremos una serie de gramáticas que permiten reconocer matemáticas y texto en Braille. Para la definición de estas gramáticas utilizaremos la notación BNF, pero las reglas que definen la gramática acabarán siendo un algoritmo iterativo dentro de un proyecto en Visual C++. Cada uno de los elementos no terminales de la gramática se transformarán en una nueva función en el código C++. El programa principal llamará a la función que implementa el símbolo no terminal inicial de la gramática y a partir de ahí se realizarán las llamadas a cada una de las funciones que codifican el resto de los elementos no terminales. En el apéndice D se detalla la forma de convertir una gramática definida mediante notación BNF en un algoritmo iterativo.

[1] Consultar Apéndice A: Definición de elementos Primeros y Siguientes

3.3.2. Primera gramática para el reconocimiento de matemáticas y texto en Braille

La primera versión de la gramática BNF para el reconocimiento de matemáticas Braille, es capaz de reconocer tanto expresiones matemáticas cómo texto escrito.
El subconjunto de operadores aritméticos reconocidos son suma, resta, multiplicación, división (en todas sus notaciones), raíces de cualquier orden, funciones exponenciales, subíndices, llamadas a funciones por ejemplo sen(x, y) o cos(y, x) además de paréntesis, corchetes, y los paréntesis invisibles utilizados en Braille para agrupar expresiones matemáticas complejas. La gramática creada permite que los operandos de las expresiones matemáticas puedan ser tanto numeros cómo variables. Un ejemplo de expresión reconocible por la gramática Braille, es la mostrada en figura 3.12.

$$\frac{\sqrt[n]{(x_1+2)^2}+\sqrt[n+1]{(x_2+2)^2}}{(x_3+2)^2}$$

Figura 3.12: Ejemplo de expresión reconocible por la gramática de la figura 3.13

<LINEA>::= <EXPR>{('=') <EXPR>}
<EXPR>::= <TERM>{('+' | '-') <TERM>}
<TERM>::= <FACT>{('∗'| '/' | ':' |'÷'|'·') <FACT>}
<FACT>::= (<FACT1>['∧'<FACT>] | 'raíz1' [<EXPR>]'raíz2' <FACT>)
<FACT1>::= <FACT2>['∨'<FACT1 >]
<FACT2>::= ('+' | '-') <FACT2>| ('('<EXPR>')' | '['<EXPR>']' |'{'<EXPR>'}'
|'<'<EXPR>'>' ||[Numerador] Num | String [FUNCION]
<FUNCION>::= '('[<EXPR>{,<EXPR>}]')'

Figura 3.13: Primera gramática para el reconocimiento de matemáticas y texto en Braille

Descripción de cada una de las reglas en la gramática de la figura 3.13:

- <LINEA>: La regla línea define una serie de expresiones separadas por la igualdad matemática (⠶).
- <EXPR>: Una expresión está formada por términos concatenados por los tokens suma y resta (⠖, ⠤).
- <TERM>: La regla que define un término reconoce factores concatenados por los tokens correspondientes a la multiplicación o división.
- <FACT>: Reconoce exponentes mediante el símbolo (⠰) y raíces de cualquier grado e índice. El símbolo utilizado para indicar una raíz se compone de dos caracteres braille. Entre el primero (⠜) y el segundo (⠱) se define el índice de la raíz que puede ser cualquier expresión. En el caso de la raíz cuadrada los dos símbolos aparecen seguidos, sin indicar el dos como índice. En tinta, el signo radical encierra bidimensionalmente mediante su trazo al 'radicando', la forma de indicar el fin del radicando en Braille es mediante la utilización de los paréntesis invisibles o auxiliares propios de Braille (⠣ ⠜).

- <FACT1>: Reconoce expresiones con subíndice. Este subíndice puede ser cualquier expresión. El símbolo Braille utilizado como predecesor de un subíndice es (⠰).

- <FACT2>: Permite expresiones entre paréntesis, corchetes, llaves y los paréntesis invisibles utilizados en Braille. También implementa el signo (+ , -) que precede a una expresión numérica. <FACT2>::=[Numerador] Num define la representación de fracciones explicada en la sección 2.2.2.2. Es importante notar que la división simplificada en Braille sólo la utilizamos con "fracciones numéricas enteras de denominador positivo", el denominador por tanto no tendrá signo. El terminal Numerador define a los números Braille representados en la parte inferior de la celda Braille. Mediante la regla <FACT2>:=String definimos identificadores o variables que pueden aparecer en un documento Braille tanto en secciones del documento con contenido matemático como texto Braille. Los textos no matemáticos que aparecen en los documentos suelen ser los enunciados de problemas, el nombre de los alumnos que han escrito el documento o la numeración de los ejercicios. Por otra parte esta misma regla, al analizar contenido matemático, permite reconocer las distintas variables que aparecen en expresiones matemáticas.

- <FUNCION>: Implementa la llamada a una función. La regla define a las expresiones entre paréntesis y separadas por comas. Por ejemplo de esta manera podemos reconocer expresiones como MCN(6,12) o sin(x).

El ejemplo de la Figura 3.15 muestra el árbol sintáctico correspondiente a la expresión de la Figura 3.14 utilizando la gramática Braille definida en la Figura 3.13. Se ha simplificado la notación en los nodos utilizando <L>,<E>,<T>,<F>,<F1>,<F2> para representar las reglas de la gramática <LINEA>, <EXPRESION>, <TERM>, <FACT>, <FACT1> y <FACT2> respectivamente.

$$\frac{(5^4 \cdot 5^2 \cdot 5^3)}{5^8} =$$

Figura 3.14: Expresión en tinta

3.3.2.1. Limitaciones de la primera gramática Braille

Aunque esta gramática interpreta gran parte de los operadores matemáticos descritos en la sección 2.2, codificación matemática Braille, tiene una serie de limitaciones que hacen que algunas expresiones matemáticas no se reconozcan correctamente.
Por una parte, en los documentos digitalizados podemos encontrar expresiones como las mostradas en la figura 3.16. En esta figura vemos el uso de un nuevo operador, la multiplicación implícita, un operador ampliamente utilizado en álgebra. La multiplicación implícita es utilizada cuando el lado izquierdo de una multiplicación es un número explícito y el lado derecho es una variable, función o constante. En esos casos podemos eliminar el signo multiplicación concatenando los factores. El uso muy común del operador multiplicación implícita en los documentos de ejemplo estudiados, provoca errores en nuestro analizador sintáctico ya que este tipo de expresiones no pueden ser reconocidas por la gramática. Será necesaria la definición de una nueva gramática Braille para interpretar este tipo de expresiones correctamente.

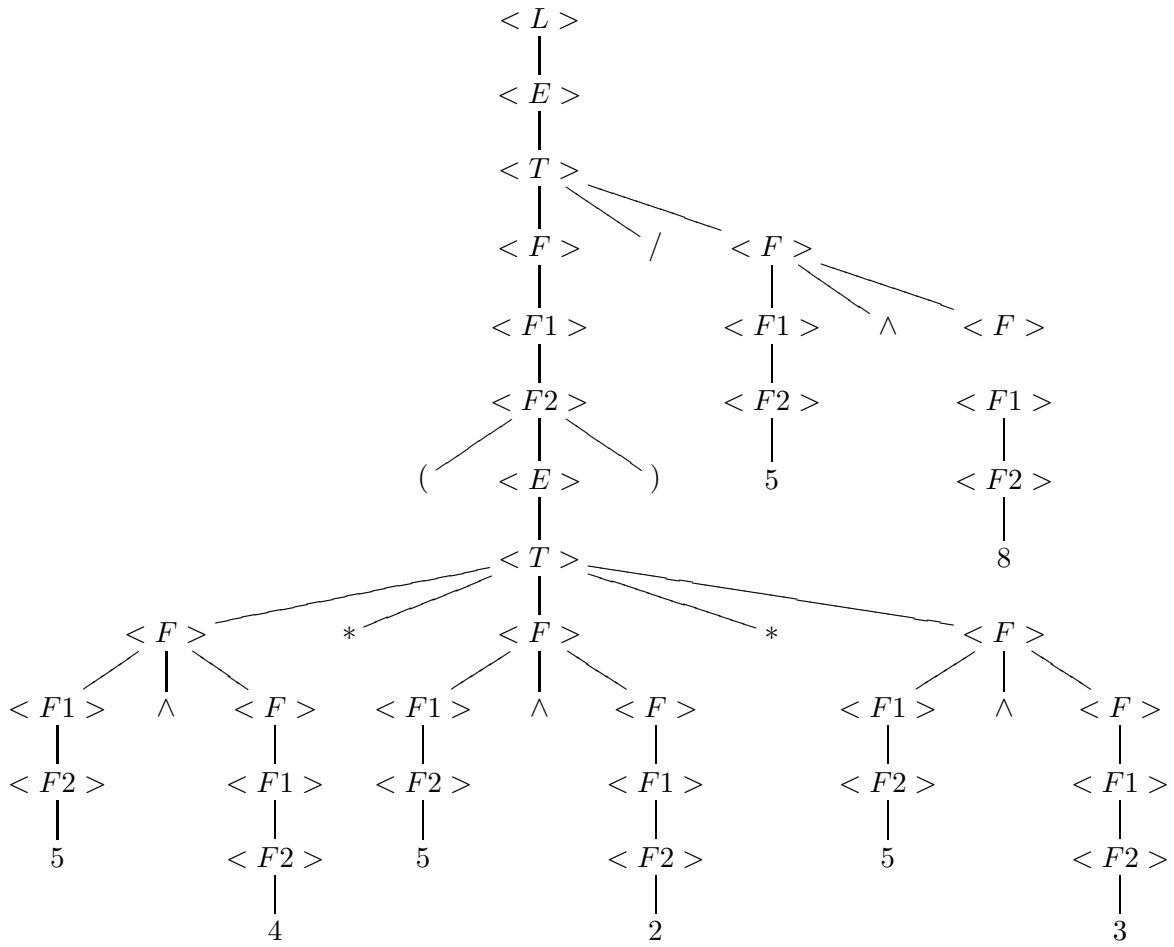


Figura 3.15: Árbol sintáctico correspondiente a la expresión del ejemplo 3.14

$4x^3 + 6x^2 + 2x + 1$
$3\sqrt{2} + 2\sqrt{3}$
$2sen(x)$
$0{,}5(x + 1)$

Figura 3.16: Ejemplos del uso de la multiplicación implícita

$$2xy + 2xy = z$$

Figura 3.17: Interpretación correcta de factores utilizando la definición de identificador de la primera gramática Braille

Otro de los problemas que presenta esta primera gramática es la definición de identificador como un string, tal y como se puede ver en la regla BNF <FACT2>. Con esta definición de identificador, se intenta simplificar la gramática y hacerla lo más genérica posible. Se pretende que la definición de identificador permita interpretar correctamente tanto texto Braille (enunciados de problemas, nombres de función como sen() o cos()) como variables en matemáticas (por ejemplo: (a + b) = c)
En la gramática, un identificador está definido como un string, es decir una serie de caracteres concatenados. Esta definición provoca algunos errores no deseados en la interpretación de matemáticas. Mientras que esta definición de identificador funciona perfectamente en algunos casos, como en la figura 3.17 donde los factores xy aparecen multiplicados, en otros casos esta definición puede resultar errónea. En el ejemplo de división de la Figura 3.18c se puede ver como el denominador 'cd' es interpretado erróneamente. El motivo es que el factor 'd' debería aparecer a la derecha de la expresión multiplicando al factor $\frac{ab}{c}$ mediante la multiplicación implícita, tal como se muestra en la figura 3.18b. Sin embargo, la definición de identificador como un string provoca la interpretación incorrecta de la expresión matemática y hace que el factor 'cd' sea reconocido como un único token indivisible.

En la siguiente sección definiremos una nueva gramática que implemente este nuevo operador para solucionar el problema de las matemáticas con variables.

3.3.3. Segunda gramática para el reconocimiento de matemáticas y texto en Braille

Tal como se ha discutido en la sección anterior, existen algunos puntos que mejorar en el reconocedor de matemáticas Braille.

La necesidad de tener en cuenta el operador multiplicación implícita comentado anteriormente implica una nueva gramática mejorada (Figura 3.19). Esta nueva gramática implementada en el analizador sintáctico no presentará errores al tratar de interpretar dos factores concatenados

(a)	$\frac{ab}{c}d$ (b)	$\frac{ab}{cd}$ (c)

Figura 3.18: (a) Expresión Braille (b) Interpretación correcta (c) Interpretación incorrecta

pero sin ningún operador que los separe. En ese caso interpreta que el operador entre ambos factores es la multiplicación implícita. Este nuevo operador se tiene en cuenta en la regla de la gramática <TERM>::= <FACT>{(<FACT>) }.

<LINEA>::= <EXPR>{('=') <EXPR>}
<EXPR>::= <TERM>{('+' | '-') <TERM>}
<TERM>::= <FACT>{(('∗'| '/' | ':' |'÷'|'·') <FACT>| <FACT>) }
<FACT>::= (<FACT1>['∧'<FACT>] | 'raíz1' [<EXPR>]'raíz2' <FACT>)
<FACT1>::= <FACT2>['∨'<FACT1>]
<FACT2>::= ('+' | '-') <FACT2>| ('('<EXPR>')' | '['<EXPR>']' |'{'<EXPR>'}'
|'<'<EXPR>'>' |[Numerador] Num | String [FUNCION]
<FUNCION>::= '('[<EXPR>{,<EXPR>}]')'

Figura 3.19: Grámatica no LL(1) para el reconocimiento de matemáticas y texto Braille

Al añadir esta nueva regla a la gramática aparece un problema, la gramática deja de ser LL(1) y se convierte en ambigua. Recordemos que una gramática LL(1) depende de un único símbolo de entrada para decidir la siguiente regla que debe aplicar. Al incorporar a la gramática la nueva regla <TERM>::= <FACT>{(<FACT>) } los operadores suma y resta pueden ser derivados utilizando la regla <EXPR> o bien la regla <FACT2>. Para evitar esta ambigüedad crearemos una nueva regla <FACT2'>que será utilizada al encontrar el operador multiplicación implícita. Esta nueva regla <FACT2'>es exactamente igual que <FACT2> excepto que no deriva en los operadores suma y resta. La gramática LL(1) completa y que permite reconocer el nuevo operador de multiplicación implícita la podemos ver en la figura 3.20.

En la figura 3.21 vemos el árbol sintáctico para la expresión 5(x+1)+2x utilizando la estructura de la nueva gramática de la figura 3.20. Como se puede ver, la expresión matemática contiene dos operadores de multiplicación implícita.

<LINEA>::= <EXPR>{('=') <EXPR>}
<EXPR>::= <TERM>{('+' | '-') <TERM>}
<TERM>::= <FACT>{(('∗'| '/' | ':' |'÷'|'·'|'·') <FACT>| <FACT'>) }
<FACT>::= (<FACT1>['∧'<FACT>] | 'raíz1' [<EXPR>]'raíz2' <FACT>)
<FACT'>::= (<FACT1'>['∧'<FACT'>] | 'raíz1' [<EXPR>]'raíz2' <FACT'>)
<FACT1>::= <FACT2>['∨'<FACT1>]
<FACT1'>::= <FACT2'>['∨'<FACT1'>]
<FACT2>::= (('+' | '-') <FACT2>| '('<EXPR>')' | '['<EXPR>']' |'{'<EXPR>'}'
|'<'<EXPR>'>' |[Numerador] Num | Letra | String [FUNCION])
<FACT2'>::= ('('<EXPR>')' | '['<EXPR>']' |'{'<EXPR>'}'
|'<'<EXPR>'>' |[Numerador] Num | Letra | String [FUNCION])
<FUNCION>::= '('[<EXPR>{,<EXPR>}]')'

Figura 3.20: Segunda grámatica para el reconocimiento de matemáticas y texto Braille

Esta nueva gramática incorpora un nuevo símbolo (⠒) utilizado en Braille para eliminar ambigüedades. Este nuevo símbolo ayuda a representar algunas expresiones en las que se necesita diferenciar entre la representación número y letra de un mismo carácter. Añadimos este nuevo

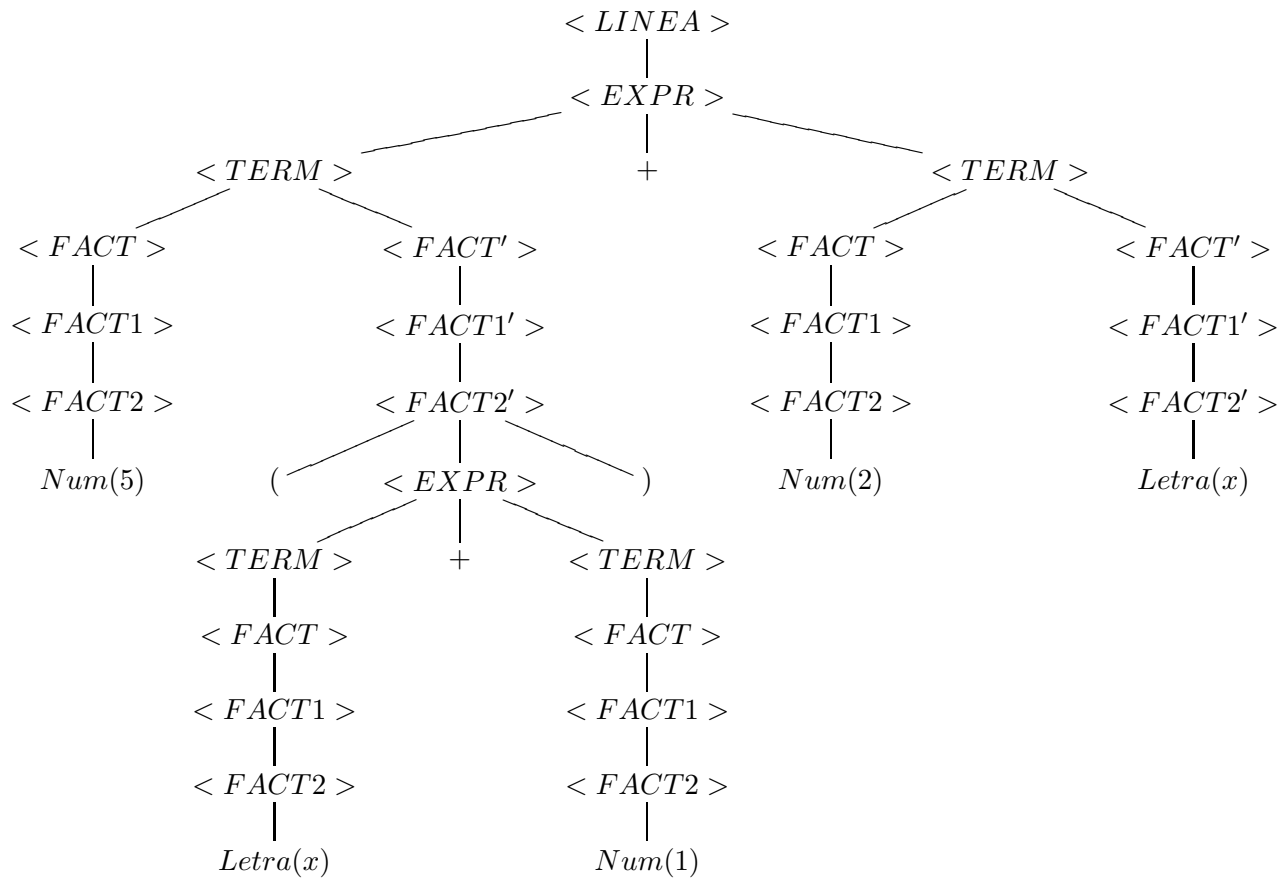


Figura 3.21: Árbol sintáctico correspondiente a la expresión 5(x+1)+2x utilizando la gramática de la Figura 3.20

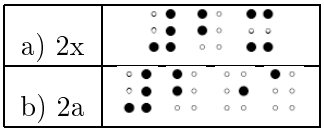


Figura 3.22: Uso del prefijo de letra minúscula

símbolo en la regla <TERM> al igual que la multiplicación implícita en esta segunda gramática Braille. Este nuevo símbolo se utiliza como prefijo de letra minúscula diferenciando un operando de tipo carácter de su equivalente numérico únicamente cuando es necesario. Mientras que en la Figura 3.22(a) no es necesario el uso de este símbolo debido a que el operando letra minúscula 'x' no tiene un equivalente numérico, en la Figura 3.22(b) el símbolo (⠰) nos ayuda a diferenciar el operando 'a' de su equivalente numérico '1'.

Otra de las mejoras implementadas en esta nueva gramática es la definición correcta de identificador. Como hemos comentado anteriormente, las variables utilizadas en expresiones matemáticas Braille no pueden ser reconocidas utilizando un string, ya que en ese caso aparecen problemas como los de la figura 3.18. Recordemos que con la gramática anterior, el factor 'cd' se reconocía como un único token indivisible lo que hacía que la interpretación de la expresión no fuese correcta. Para resolver este problema, definimos en la regla <FACT2> un identificador como una única letra o carácter. De esta manera y volviendo al ejemplo de la figura 3.18, 'cd' no se reconocerá como un único token 'cd'. Con la nueva definición de identificador se reconocen dos tokens de tipo Letra, 'c' y 'd' unidos mediante la multiplicación implícita definida en esta nueva gramática.

3.3.3.1. Limitaciones de la segunda gramática Braille

Aunque con esta última modificación en la gramática Braille hemos solucionado el problema de las variables utilizadas en matemáticas, aún nos falta por resolver la traducción de texto Braille no matemático.

En la primera gramática implementada, al tener definido un identificador como un string el texto se reconocía correctamente, pero con la nueva gramática en la que un identificador se define como una única letra interpretamos que cada uno de los caracteres que componen el texto a traducir es un token de tipo letra unido al siguiente mediante la multiplicación implícita.

El mismo problema ocurre con la interpretación de funciones matemáticas. Las funciones matemáticas están formadas por la concatenación de caracteres formando el nombre de la función, pero a diferencia del texto Braille, éstas aparecen dentro de contenido matemático. Por ejemplo, en la figura 3.23 vemos un ejemplo de la interpretación de la función MCN. Utilizando la gramática anterior este nombre de función se reconocía correctamente pero con la nueva gramática, la función MCN se interpreta como la multiplicación de tres operandos M, C y N

$$\frac{2}{3} + \frac{5}{7} + \frac{3}{5} = MCN(3, 7, 5)$$

Figura 3.23: Ejemplo del reconocimiento de funciones

Para solucionar los problemas comentados para la correcta traducción de texto Braille y con el reconocimiento de los nombres de funciones matemáticas, necesitamos un nuevo proceso que se ejecutará antes del análisis sintáctico. En la siguiente sección describiremos paso a paso el proceso que permite diferenciar en un documento Braille el contenido matemático del texto.

3.3.3.2. Preanálisis sintáctico: Cómo distinguir entre texto y matemáticas

El preanálisis sintáctico es un proceso mediante el cuál tratamos de identificar en un documento Braille las regiones con texto Braille exclusivamente. Normalmente en los documentos Braille estudiados, estas regiones corresponderán a los enunciados, a la numeración de los problemas o al nombre del alumno que escribió el documento. Para distinguir este tipo de contenido, se ejecuta el proceso de preanálisis como un paso previo a la utilización de la gramática definida en nuestro analizador sintáctico. De esta manera, el analizador sintáctico durante la interpretación puede distinguir entre los caracteres Braille que pertenecen a una expresión matemática y los que pertenecen a texto Braille.

De una forma similar, el preanálisis sintáctico también se encarga de buscar el nombre de funciones en el contenido matemático, como por ejemplo: sen(x) o cos(x). Este reconocimiento previo de nombres de función permitirá después al analizador sintáctico distinguir las variables utilizadas en matemáticas de los nombres de funciones.

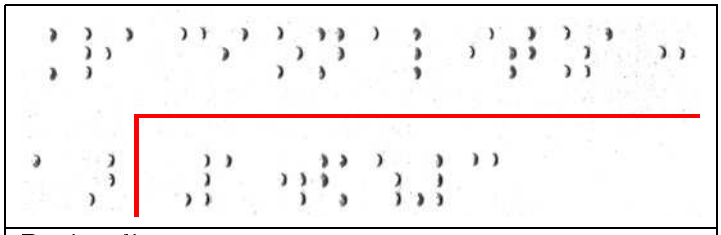
El preanálisis sintáctico reescribe el documento XML generado por el OBR y que fue explicado en la sección 3.2.2. Recordemos que este XML contiene la definición de cada uno de los caracteres Braille que aparecen en el documento original. El preanálisis sintáctico modifica este fichero XML, marcando las secciones no matemáticas y los nombres de función con nuevas etiquetas.

Para realizar la distinción entre matemáticas y texto, el preanálisis sintáctico particiona el documento XML utilizando como separador los espacios en blanco del documento. Cada una de las secciones obtenidas como resultado de esta partición, debe ser clasificada como texto o matemáticas Braille. Si en el conjunto de caracteres existentes entre dos espacios en blanco no existe ninguna operación matemática, se considera que esta sección está codificando únicamente texto. De la misma manera, la presencia de un operador matemático entre alguno de los caracteres que encontramos entre dos espacios en blanco, indican que la sección contiene matemáticas.

Un proceso similar se utiliza para distinguir dentro del contenido matemático el nombre de funciones como por ejemplo: sen(x) o cos(y). Se particiona el contenido matemático utilizando como separadores los distintos operadores y el signo paréntesis utilizado en la representación de funciones.

Para marcar y etiquetar las secciones del documento con texto Braille añadimos dos nuevas etiquetas <COMENTARIO> y <FUNCION> a las ya descritas previamente <PAGINA> <FILA> y <CARACTER>.

El preanalizador sintáctico utiliza la etiqueta <COMENTARIO> para etiquetar el conjunto de caracteres que pertenecen a texto no matemático. Con la etiqueta <FUNCION> el analizador especifica el conjunto de caracteres que forman el nombre de una función. Como resultado,



Racionalitza:
a) $\frac{1}{\sqrt{3}}$

Figura 3.24: Documentos en Braille y tinta con contenido matemático y no matemático

de cada nuevo nodo de tipo <COMENTARIO> o <FUNCION> colgarán uno o más nodos de tipo <CARACTER>.
Si observamos la figura 3.24 que corresponde a un documento Braille y su correspondiente traducción a tinta, podemos reconocer tres secciones claramente. Por una parte el texto no matemático del encabezado 'Racionalitza:' y del apartado del problema 'a)'. Por otra, el texto matemático '$\frac{1}{\sqrt{3}}$'.

A continuación veremos un ejemplo de cómo se modifica un fichero XML que corresponde a un documento Braille. En la Figura 3.25 tenemos la codificación XML realizada por el OBR para la imagen de la figura 3.24. El preanálisis sintáctico recorre cada uno de los nodos del documento XML identificando las secciones que no contienen operadores matemáticos. El resultado del preanálisis sintáctico puede verse en la Figura 3.26. Vemos como el documento Braille de la Figura 3.25 se ha reescrito identificando el texto braille no matemático 'Racionalitza:' y 'a)' dentro de una etiqueta <COMENTARIO>.

Contenido XML	Símbolo	Notación Braille
<XML version='1.0'>		
<PAGINA ALUMNE='GABRIEL' DATA='06-06-09'>		
<FILA NL='1' NC='14' XS='40' YS='11' XI='600' YI='44'>		
<CARACTER POSX='40'>000101</CARACTER>	MAYÚSCULA	
<CARACTER POSX='80'>111010</CARACTER>	r	
<CARACTER POSX='120'>100000</CARACTER>	a	
<CARACTER POSX='160'>100100</CARACTER>	c	
<CARACTER POSX='200'>010100</CARACTER>	i	
<CARACTER POSX='240'>101010</CARACTER>	o	
<CARACTER POSX='280'>101110</CARACTER>	n	
<CARACTER POSX='320'>100000</CARACTER>	a	
<CARACTER POSX='360'>111000</CARACTER>	l	
<CARACTER POSX='400'>010100</CARACTER>	i	
<CARACTER POSX='440'>011110</CARACTER>	t	
<CARACTER POSX='480'>101011</CARACTER>	z	
<CARACTER POSX='520'>100000</CARACTER>	a	
<CARACTER POSX='560'>010010</CARACTER>	:	
</FILA>		
<FILA NL='2' NC='10' XS='40' YS='48' XI='440' YI='81'>		
<CARACTER POSX='40'>100000</CARACTER>	a	
<CARACTER POSX='80'>001110</CARACTER>	)	
<CARACTER POSX='120'>000000</CARACTER>		
<CARACTER POSX='160'>001111</CARACTER>	NUM	
<CARACTER POSX='200'>100000</CARACTER>	1	
<CARACTER POSX='240'>010011</CARACTER>	/	
<CARACTER POSX='280'>110101</CARACTER>	raíz	
<CARACTER POSX='320'>100011</CARACTER>	raíz	
<CARACTER POSX='360'>001111</CARACTER>	NUM	
<CARACTER POSX='400'>100100</CARACTER>	3	
</FILA>		

Figura 3.25: Descripción XML del documento Braille de la figura 3.24

Contenido XML	Símbolo	N. Braille
<PAGINA ALUMNE='GABRIEL' DATA='06-06-09'>		
<FILA NL='1' NC='14' XS='40' YS='11' XI='600' YI='44'>		
<COMENTARIO>		
<CARACTER POSX='40'>000101</CARACTER>	MAYÚSCULA	
<CARACTER POSX='80'>111010</CARACTER>	r	
<CARACTER POSX='120'>100000</CARACTER>	a	
<CARACTER POSX='160'>100100</CARACTER>	c	
<CARACTER POSX='200'>010100</CARACTER>	i	
<CARACTER POSX='240'>101010</CARACTER>	o	
<CARACTER POSX='280'>101110</CARACTER>	n	
<CARACTER POSX='320'>100000</CARACTER>	a	
<CARACTER POSX='360'>111000</CARACTER>	l	
<CARACTER POSX='400'>010100</CARACTER>	i	
<CARACTER POSX='440'>011110</CARACTER>	t	
<CARACTER POSX='480'>101011</CARACTER>	z	
<CARACTER POSX='520'>100000</CARACTER>	a	
<CARACTER POSX='560'>010010</CARACTER>	:	
</COMENTARIO>		
</FILA>		
<FILA NL='2' NC='10' XS='40' YS='48' XI='440' YI='81'>		
<COMENTARIO>		
<CARACTER POSX='40'>100000</CARACTER>	a	
<CARACTER POSX='80'>001110</CARACTER>	)	
</COMENTARIO>		
<CARACTER POSX='160'>001111</CARACTER>	NUM	
<CARACTER POSX='200'>100000</CARACTER>	1	
<CARACTER POSX='240'>010011</CARACTER>	/	
<CARACTER POSX='280'>110101</CARACTER>	raíz	
<CARACTER POSX='320'>100011</CARACTER>	raíz	
<CARACTER POSX='360'>001111</CARACTER>	NUM	
<CARACTER POSX='400'>100100</CARACTER>	3	
</FILA>		

Figura 3.26: Preanálisis sintáctico (distinción entre texto y matemáticas) del documento Braille de la figura 3.25.

3.3.3.3. Ambigüedades en Braille:

Existen algunas ambigüedades en la definición del lenguaje Braille que pueden significar más de una traducción válida de una misma expresión. En esta sección hablaremos de algunas de ellas y de las implicaciones que puede tener su presencia durante el análisis sintáctico. A continuación comentaremos algunos de los caracteres que pueden tener distintas interpretaciones.

El símbolo (⠷) tiene dos posibles valores distintos en Braille. Puede representar tanto al carácter 'á' cómo al corchete abierto '['. Este tipo de ambigüedad no es importante ya que suponemos que cuando encontramos este símbolo al traducir matemáticas nos estamos refiriendo al símbolo '[' mientras que si estamos traduciendo texto probablemente nos referimos al símbolo 'á'. Esta interpretación se puede llegar a hacer gracias al proceso de preanálisis sintáctico explicado anteriormente y que identifica las secciones de un documento con contenido exclusivamente matemático.

El mismo problema se repite con el símbolo ⠌, mientras en matemáticas se utiliza como carácter prefijo de un subíndice, al traducir texto Braille se interpreta como el carácter 'í'.

Existen sin embargo otras ambigüedades más difíciles de resolver que las anteriores, el símbolo (⠶) por ejemplo, utilizado para representar la suma en matemáticas al traducir texto equivale a los signos de exclamación '¡' y '!'. Lo mismo sucede con el símbolo (⠦) utilizado en matemáticas para representar a la multiplicación y que al traducir texto representa el signo de comillas dobles ' " '. Si nos basamos en la sola presencia de operadores matemáticos para deducir las secciones de un documento con contenido matemático estos símbolos pueden inducir al error. El Preanalizador sintáctico debe tener en cuenta el contexto en el que aparecen para decidir si están representado una operador matemático o sólo texto.

Existe otra ambigüedad en Braille que solo puede resolverse dependiendo del contexto en el que nos encontramos. En Braille, la representación de los números ordinales se realiza desplazando los dígitos a la parte baja de la celda Braille y añadiendo un signo específico a su derecha. Esta representación nos ayuda a definir los ordinales '1º' o '1ª' tal y como se puede ver en la figura 3.27. Sin embargo, existe un problema con este tipo de representación de ordinales que deriva en una ambigüedad entre las expresiones que contienen fracciones en su forma abreviada y las expresiones que contienen ordinales. Por ejemplo, la representación de 2ª y 2/1 al igual que 3ª y 3/1 utilizan la misma codificación Braille. La resolución a este problema queda pendiente para un trabajo futuro en el que se tendría que definir una gramática para el reconocimiento de texto braille que incluyera a los ordinales.

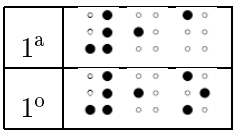


Figura 3.27: Representación de ordinales

3.3.3.4. Recuperación de errores:

La recuperación de errores es una fase fundamental para el analizador sintáctico de Braille debido a la naturaleza de los documentos que debe procesar. Por una parte, el documento XML

origen del analizador sintáctico puede contener errores debido al reconocimiento incorrecto de los caracteres Braille escaneados. Por otra, los propios alumnos pueden haber cometido distintos errores sintácticos a la hora de escribir el documento original con la máquina PERKINS. Por tanto, puede ser muy frecuente la aparición de errores sintácticos en los documentos, los cuales deben ser señalados y continuar con la ejecución del analizador sintáctico normalmente. Hasta ahora hemos conseguido que el analizador sintáctico de Braille reconozca expresiones matemáticas sintácticamente correctas pero la sola presencia de un símbolo inesperado en la secuencia de entrada haría detenerse al analizador.

Existen distintas técnicas para recuperar errores sintácticos, algunos de los principios en los que se basa el diseño del control de los errores sintácticos [18] son:
1) Un analizador sintáctico debe determinar que ha ocurrido un error tan pronto como sea posible.
2) Después de encontrar un error, el analizador sintáctico debe buscar un lugar apropiado en el que reanudar el análisis. Un analizador sintáctico siempre debe intentar analizar tanto código como sea posible, encontrando tantos errores reales como sea posible.
3) Un analizador sintáctico debería intentar evitar el problema de cascada de errores, en el cual un error genera una larga lista de falsos errores.
4) Un analizador sintáctico debe evitar bucles infinitos en los errores, en los que se genera una cascada sin fin de mensajes de error sin consumir ninguna entrada.

En el analizador sintáctico Braille, el método utilizado para la recuperación de errores es el llamado 'modo de alarma'. Esta técnica muy común, es aplicable a analizadores sintácticos descendentes. El nombre viene dado porque en algunos casos complejos, el analizador sintáctico puede llegar a consumir un numero muy grande de tokens en el intento de encontrar un lugar donde reanudar el análisis sintáctico. En el peor de los casos puede hacer que se acabe consumiendo el resto del programa. Por otra parte, asegura que el analizador sintáctico no caiga en un bucle infinito durante la recuperación de errores. La idea básica del 'modo de alarma' consiste en añadir un conjunto de tokens de sincronización a cada uno de los procedimientos recursivos que implementan la gramática Braille. En el momento en el que se produce un error, el analizador sintáctico explora hacia delante buscando alguno de los tokens que existen dentro del conjunto de sincronización. Los tokens utilizados para formar el conjunto de sincronización son los elementos primeros y siguientes[2] de cada una de las reglas de la gramática Braille definida anteriormente. Al entrar en cada uno de los procedimientos que implementan nuestra gramática siempre se tiene que cumplir que el token actual pertenezca al conjunto de primeros de esa regla o procedimiento. En caso contrario, muestra un error y el analizador de errores avanza hasta encontrar cualquiera de los elementos del conjunto siguientes.

A continuación reproduciremos el control de errores utilizado en el reconocedor de matemáticas Braille pero utilizando una gramática Braille mucho más sencilla que permitirá entenderlo con mayor facilidad, transformaremos la gramática en un algoritmo iterativo y añadiremos una serie de procedimientos que implementan el control de errores en modo alarma.

Definimos una gramática Braille simplificada que reconoce la aritmética en enteros con los

[2]Consultar Apéndice A: Definición de elementos Primeros y Siguientes

operadores suma, resta, multiplicación y división de la siguiente forma:

```
<expresión>::= <término>('+' |'-') <término>
<término>::= <factor>('*'|'/')<factor>
<factor>::= ('(' <expresión>')'| numero )
```

Utilizamos los algoritmos para el cálculo de los elementos primeros y siguientes (ver Apéndice A) para cada una de las reglas que forman la gramática con el siguiente resultado:

```
primeros(<expresión>) = {prefijo_de_número, '('};
siguientes(<expresión>) = {')'};
primeros(<término>) = {prefijo_de_número,'('};
siguientes(<término>) = {')','+','-'};
primeros(<factor>) = { prefijo_de_número,'('};
siguientes(<factor>) = {')','+','-','*','/'};
```

En la figura 3.28 vemos una implementación de la recuperación de errores en modo alarma para la gramática simplificada y el conjunto de primeros y siguientes comentado. Se utilizan tres procedimientos para codificar cada una de las tres reglas que definen la gramática: expresión, término y factor. El control de errores se realiza al inicio de cada uno de los procedimientos mediante las funciones checkinput y scanto. Estás dos funciones comprueban si el token actual en la secuencia de entrada pertenece al conjunto de primeros o siguientes (ver Apéndice A). En el caso de pertenecer al conjunto de primeros se ejecuta el procedimiento normalmente mientras que si el token actual pertenece al conjunto de siguientes no se ejecuta el cuerpo del procedimiento.

Veamos paso a paso un ejemplo de cómo la implementación de control de errores para la gramática anterior detecta y repara un error durante el análisis sintáctico de una expresión incorrecta. Si observamos la figura 3.29 existe un error sintáctico en la expresión debido a la aparición de un paréntesis cerrado después del operador suma. La función expresión de la gramática inicia el análisis sintáctico comprobando que el símbolo 'prefijo de número' pertenece al conjunto de los elementos primeros. Una vez comprobado y utilizando el símbolo suma como separador llama a las funciones termino para reconocer cada uno de los operandos de la suma. Para el primer operando a la izquierda de la suma, la función termino verifica que el símbolo 'prefijo de número' pertenece al conjunto de primeros y por tanto continúa al análisis correctamente llamando a la función factor que a su vez también comprueba que 'prefijo de numero' pertenece a su conjunto de primeros. Al llegar a este punto se ha interpretado correctamente el número 3 como el primer término de la suma.
Sin embargo, en el reconocimiento del segundo operando de la suma el intérprete detecta un error al inicio de la función termino. Esta función detecta que el paréntesis cerrado pertenece al conjunto de siguientes en vez de al conjunto de primeros. Para la función, encontrar este paréntesis cerrado indica que falta un operando después de la suma y por eso no ejecutará el cuerpo de la función para tratar de interpretarlo. Una vez recuperado este error la función inicial expresion es llamada de nuevo por el analizador sintáctico para continuar analizando el resto de la expresión matemática. En este caso cada uno de los dos términos serán reconocidos correctamente y el análisis sintáctico finalizará reportando un único error.

En está sección hemos expuesto utilizando una gramática simplificada la implementación

```
procedure scanto(siguientes)
begin
    while not(token in siguientes) do
        getToken;
    end while;
end scanto;

procedure checkinput(primeros, siguientes, token)
begin
    if not(token in primeros) then
        error;
        scanto(siguientes);
    end if;
end checkinput;



procedure expresion()
begin
primeros = {prefijo_de_número, '('};
siguientes = {')'};
checkinput(primeros, siguientes, token);
if not (token in siguientes)
    termino();
    while (1)
        if (token = '+') then
            match('+');
            termino();
        else if (token = '-') then
            match('-');
            termino();
        else return;
        end if;
    end while;
end if;
end expresion;
```

```
procedure termino()
begin
primeros = {prefijo_de_número,'('};
siguientes = {')','+','-'};
checkinput(primeros, siguientes, token);
if not (token in siguientes)
    factor();
    while (1)
        if (token = '*') then
            match('*');
            factor();
        else if (token = '/') then
            match('/');
            factor();
        else return;
        end if
    end while;
end if;
end termino;


procedure factor()
begin
primeros =  {prefijo_de_número,'('};
siguientes = {')','+','-','*','/'};
checkinput(primeros, siguientes, token);
if not (token in siguientes)
    if (token = '(') then
        match('(');
        expresion();
        match(')');
    else if (token = prefijo_de_número) then
        match('numero');
    else return;
    end if;
end factor;
```

Figura 3.28: Ejemplo de implemtentación de control de errores

de la recuperación de errores en un analizador sintáctico descendente. El método utilizado en el reconocedor de matemáticas Braille funciona del mismo modo, para ello se han calculado el conjunto de elementos primeros y siguientes para cada una de las reglas de la segunda gramática Braille para el reconocimiento de matemáticas y texto. Este cálculo detallado se puede encuentrar en el Apéndice B de esta memoria.

3.4. MathML

En esta sección hablaremos de la salida generada por nuestro analizador Braille. El conjunto de matemáticas generadas por el analizador tienen que ser mostradas por pantalla y para esa tarea se eligió una herramienta que permitiera representar contenido matemático de la manera más sencilla posible.

El Mathematical Markup Language o MathML es una aplicación XML creada por el W3C

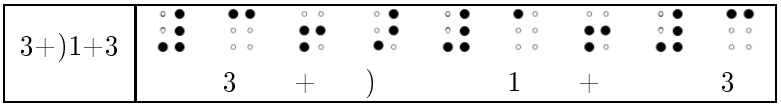


Figura 3.29: Error sintáctico en expresión matemática debido a la presencia del paréntesis.

(World Wide Web Consortium) para describir notación matemática. MathML fue creado para facilitar el uso de contenido matemático y científico en internet y en otras aplicaciones. El objetivo principal es que las matemáticas sean servidas, recibidas y procesadas de la misma manera que HTML proporcionó esa funcionalidad al texto. La primera versión de MathML 1.1 aparece en Julio de 1999 y la versión estable actual, MathML 2.0 en Octubre del 2003. Actualmente se está trabajando en la versión 3.0 cuyo último borrador aparece en Noviembre del 2008. MathML intenta solucionar el vacío que dejaba HTML respecto a la notación matemática. Hasta ahora, la manera de presentar contenido de este tipo con HTML era mediante la utilización de imágenes exportadas de otros programas. Al basarse en XML, MathML permite a los navegadores web renderizar expresiones matemáticas. Para ello nos proporciona una serie de tags que permiten describir expresiones en términos de su presentación y su semántica (figura 3.30). MathML no está pensado para ser editado a mano, sino para ser utilizado por herramientas como editores de ecuaciones o para ser exportado desde otros paquetes matemáticos.
En nuestro caso será creado por el propio reconocedor de escritos Braille a medida que avanza el análisis sintáctico.

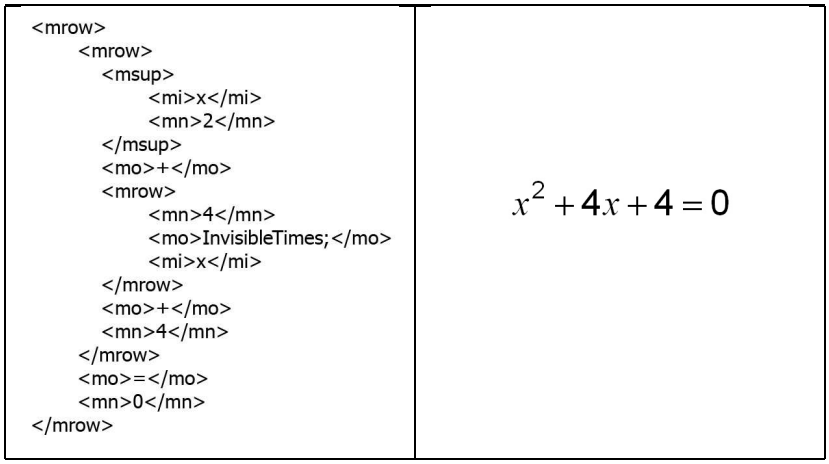


Figura 3.30: Representación de la expresión $x^2+4x+4=0$ en MathML utilizando las etiquetas msup, mrow, mi y mn

3.4.1. Elementos de presentación en MathML

Esta sección explica los diferentes elementos de presentación que utiliza MathML, los cuales permiten describir la estructura de la notación matemática. Los elementos de presentación se corresponden a los 'constructores' tradicionales en notación matemática. Es decir, el tipo de símbolos y estructuras que construyen la notación matemática tradicional.

A continuación mostramos una relación de los distintos elementos o etiquetas que permitirán construir matemáticas con MathML. Sólo en la capa de presentación de MathML existen 28 elementos, los cuales aceptan más de 50 atributos. Por ese motivo, en esta sección vamos a describir únicamente los símbolos que se han implementado en el reconocedor de escritos Braille.

El W3C (World Wide Web Consortium) distingue cuatro categorías en el conjunto de los elementos de presentación, dependiendo del uso de cada uno de los elementos. Los elementos de presentación que se han utilizado en el analizador sintáctico de Braille pertenecen a una de estas categorías: Token Elements, General Layout Schemata, Script and Limit Schemata y Tables and Matrices.

Las etiquetas pertenecientes a la categoría Token Elements pueden contener cualquier secuencia de 0 o más caracteres en su interior. Las etiquetas pertenecientes a esta categoría son <mi>, <mn>, <mo>y <mtext>

En la categoría General Layout Schemata encontramos elementos que permiten representar fracciones y radicales. Forman parte de esta categoría las siguientes etiquetas <mfrac>, <msqrt>y <mroot>.

Por último, en de la categoría Script and Limit Schemata tenemos <msub>y <msup>, etiquetas que sitúan expresiones matemáticas alrededor de una base.

Así, el conjunto de etiquetas MathML utilizadas por el analizador sintáctico es el siguiente:

- <mi>: Identificador: Representa un nombre simbólico o un texto arbitrario. Los identificadores pueden incluir variables, nombres de funciones y constantes simbólicas.
- <mn>: Numero: Representa una secuencia de dígitos.
- <mo>: Permite representar un operador o un separador.
- <mtext>: Es usado para representar texto arbitrario, texto o comentarios sin ningún significado matemático.
- <mrow>: Agrupa cualquier numero de expresiones horizontalmente.
- <mfrac>: Permite formar una fracción a partir de dos subexpresiones que serán el numerador y el denominador.
- <msqrt>: Muestra una raíz cuadrada, recibe como argumento la base, ya que el índice de la raíz es siempre dos.
- <mroot>: Forma una raíz con cualquier índice. Por tanto necesita dos argumentos para especificar el índice y la raíz.
- <msub>: Permite añadir un subíndice a una base.
- <msup>: Añade un exponente a una base.

A continuación veremos algunos ejemplos reales de expresiones matemáticas que se encuentran en los documentos Braille escaneados y cómo estos se representan utilizando MathML en el reconocedor de matemáticas Braille.

En el ejemplo de la figura 3.31 se muestra el uso de las etiquetas <mrow>, <msup>, <mfrac>y <mn>para representar la expresión $\frac{5^6}{5^2}$.
En el ejemplo, se puede comprobar como utilizamos la etiqueta <mfrac>para representar la división, <msup>para representar los distintos exponentes y cómo cada uno de los operandos numéricos están dentro de una etiqueta <mn>.
Por otra parte, la etiqueta <mrow>se encarga de agrupar expresiones, de esta manera al operador <mfrac>, que representa fracciones, podemos indicarle el numerador y el denominador agrupados por este operador <mrow>.

<mfrac> <mrow> <msup> <mn>5</mn> <mn>6</mn> </msup> </mrow> <mrow> <msup> <mn>5</mn> <mn>2</mn> </msup> </mrow> </mfrac>	$\frac{5^6}{5^2}$	<mfrac> <mrow> <mroot> <mi>x</mi> <mn>5</mn> </mroot> </mrow> <mrow> <mroot> <mi>x</mi> <mn>3</mn> </mroot> </mrow> </mfrac>	$\frac{\sqrt[5]{x}}{\sqrt[3]{x}}$

Figura 3.31: Representación de las expresiones $\frac{5^6}{5^2}$ y $\frac{\sqrt[5]{x}}{\sqrt[3]{x}}$ en MathML utilizando las etiquetas mfrac, mroot, msup, mrow, mn y mi

Otra de las expresiones que aparecen en los documentos escaneados es $\frac{\sqrt[5]{x}}{\sqrt[3]{x}}$ puede verse su conversión a MathML en la figura 3.31. En la figura se observa el uso de la etiqueta <mroot>para representar raíces de cualquier índice. Por otra parte, utilizamos la etiqueta <mfrac>para definir una fracción dónde el numerador y el denominador aparecen agrupados utilizando una etiqueta <mrow>. La etiqueta <mn>nos permite definir un valor numérico y <mi>un identificador.

Capítulo 4

Resultados

En esta sección analizamos los resultados obtenidos en la interpretación de documentos originales escritos por alumnos utilizando una máquina Perkins. Para ello se han estudiado 6 documentos completos con matemáticas y texto (correspondiente a enunciados y numeración de problemas). Algunas de estas traducciones se pueden encontrar en el Apéndice C.
En los resultados obtenidos se muestra la importancia del OBR en el proceso definitivo de traducción de documentos. Se han realizado dos traducciones para cada uno de los 6 documentos, la primera traducción se realiza a partir de un archivo XML con el contenido de la imagen codificado sin ningún error en el reconocimiento de los caracteres. Los errores sintácticos que pueden aparecer en esta traducción sólo pueden ser originados por expresiones matemáticas definidas incorrectamente. La segunda de las traducciones se hace utilizando el fichero XML generado por el OBR a partir del reconocimiento de cada uno de los caracteres en la imagen. Esta interpretación añade errores a la traducción final, debidos al reconocimiento incorrecto de algunos caracteres y al ruido en la imagen. El OBR utilizado para realizar estas traducciones aun no está completamente finalizado, por lo que el número de errores en traducciones producidos por el reconocimiento incorrecto de caracteres será menor en un futuro.

Para valorar la calidad en las traducciones necesitamos diferenciar entre la traducción de texto y la traducción de contenido matemático, para ello definimos:

- Palabras: Son las regiones del documento Braille que el analizador sintáctico trata como texto. En los ejemplos estudiados corresponde a las palabras que forman los enunciados y los apartados de problemas.

- Expresiones matemáticas: Expresiones matemáticas completas que se han traducido mediante el parser de matemáticas Braille. Para separar una expresión matemática de otra se utiliza la igualdad matemática '=', ya que es el elemento terminal derivable a partir de la primera regla de la gramática Braille definida.

Utilizaremos estas dos definiciones para valorar la calidad en la traducción de la figura 4.1 a modo de ejemplo. Esta figura contiene dos traducciones de un mismo documento Braille, una a partir de un reconocimiento correcto de todos los caracteres de la figura (izquierda) y otra en la que algunos de los caracteres no han sido reconocidos correctamente por el OBR (derecha). Valoramos la calidad de ambas traducciones de la siguiente manera:

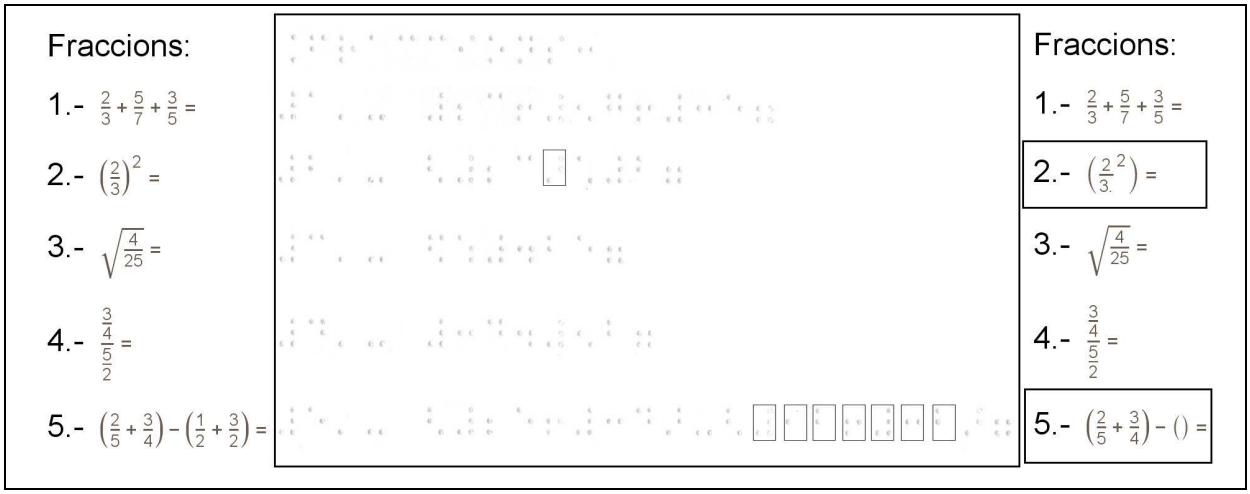


Figura 4.1: Traducción realizada a partir de la imagen sin errores en el reconocimiento de caracteres (izquierda) y con errores (derecha)

- Reconocimiento de texto: La traducción a partir del reconocimiento de caracteres hecho por el OBR muestra 6 palabras correctamente (fraccions:, 1.-, 2.-, 3.-,4.-, 5.-) de las 6 posibles.
- Expresiones matemáticas: El análisis a partir del reconocimiento de caracteres hecho por el OBR (derecha) indica que sólo 3 de las 5 (60 %) expresiones son correctas.

En la figura 4.2 se muestra como los errores cometidos por el OBR durante el proceso de reconocimiento de caracteres hace que dos expresiones matemáticas no se acaben traduciendo correctamente.

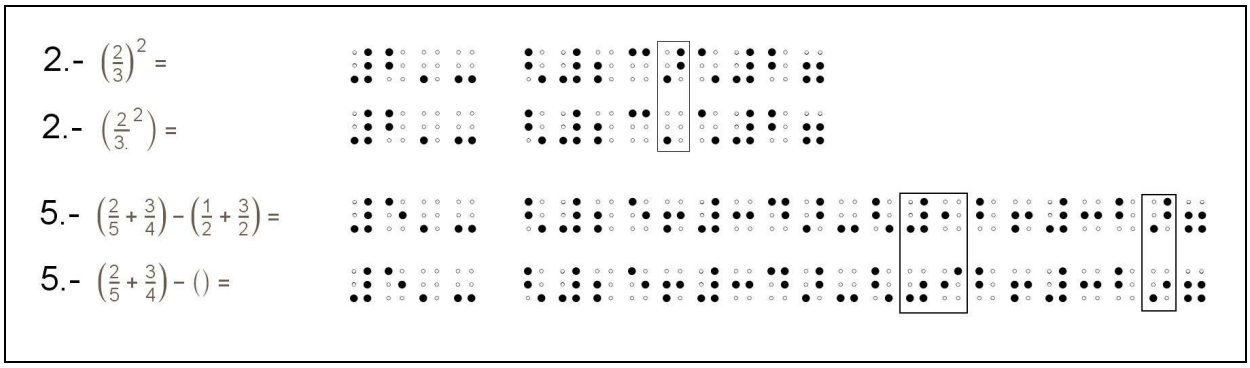


Figura 4.2: Expresiones traducidas incorrectamente debido al reconocimiento erróneo de algunos de los caracteres.

En la figura 4.3 vemos los resultados de la traducción de 6 páginas Braille que contienen 98 expresiones matemáticas y 171 palabras. Para realizar la valoración en la traducción se comparan los resultados obtenidos en la traducción sin errores de reconocimiento con los obtenidos utilizando el reconocimiento hecho por el OBR.

- Reconocimiento de texto: La traducción a partir del reconocimiento hecho por el OBR reconoce 162 palabras correctamente de las 171 que aparecen en los documentos. Es decir el 94 % del texto que aparece en los documentos se ha reconocido y traducido correctamente.
- Expresiones matemáticas: En la traducción de matemáticas se reconocen y traducen 90 expresiones correctamente de las 98 que contienen los 6 documentos, lo que significa que el 91 % del contenido matemático de los documentos se ha traducido de forma correcta.

Nº	Total Palabras	Palabras correctas	Palabras incorrectas	Total Expr. mat.	Expr. mat. correctas	Expr. mat. incorrectas
01	33	31	2	14	14	0
02	27	26	1	22	22	0
03	36	34	2	16	16	0
04	24	21	3	20	15	5
05	37	36	1	16	15	1
06	14	14	0	10	8	2
total	171	162	9	98	90	8

Figura 4.3: Resultados obtenidos en la traducción de 6 páginas en Braille que contienen 171 palabras y 98 expresiones matemáticas.

Capítulo 5

Conclusiones y trabajo futuro

El Sistema Braille es escaso en símbolos: con sólo 6 puntos por cada celda Braille únicamente se pueden representar $2^6 = 64$ caracteres distintos. Este limitado conjunto de caracteres permite representar texto, química, música o matemáticas. Debido a la limitación en el número de símbolos del alfabeto Braille es normal su combinación, es frecuente que un mismo carácter tenga distintas interpretaciones y que su traducción dependa del contexto en el que se encuentre. Es necesaria la definición de una gramática específica para un tipo de contenido que permita una correcta traducción del mismo.
En este proyecto hemos creado un traductor de texto y matemáticas Braille para incorporarlo en una herramienta de traducción automática de Braille a tinta. El objetivo de esta herramienta es la ayuda en la integración de alumnos invidentes en centros escolares, para ello el intérprete que hemos creado tiene las siguientes características:

- Es capaz de traducir texto Braille y matemáticas en un mismo documento automáticamente: El analizador identifica en un documento las secciones con contenido matemático o texto y traduce correctamente cada una de ellas.

- Traduce un subconjunto de operadores matemáticos: El analizador traduce documentos Braille escritos por niños con conocimientos básicos en matemáticas. Los alumnos, más adelante, necesitarán nuevos operadores y notación matemática, pero utilizarán un ordenador en vez de una máquina Perkins para la redacción, por lo que no será necesaria esta aplicación para el reconocimiento de sus textos. Aunque el número de operadores implementados ha resultado ser suficiente en las pruebas realizadas, la incorporación de nuevos operadores seguiría el procedimiento explicado en esta memoria.

- Es robusto a errores cometidos por el alumno: En los documentos con contenido matemático pueden existir errores sintácticos causados por los alumnos. Estos errores al traducir matemáticas Braille son señalados por el intérprete que continúa con la traducción del resto del documento. De forma similar, el intérprete señala también los errores sintácticos originados por el reconocimiento incorrecto de algún caracter por parte del OBR (Optical Braille Recognition).

Debido a los resultados de traducción muy positivos obtenidos mediante la herramienta desarrollada, podemos concluir que el proyecto se ha realizado con éxito.

Como en toda aplicación, podríamos crear nuevas versiones de la misma con nuevos requerimientos que surgen a partir del análisis de los resultados obtenidos y que serían origen de un trabajo futuro:

- Se deberían detectar las posibles ambigüedades en Braille discutidas en la sección 3.3.3.3 y mostrar cada una de las interpretaciones válidas de un mismo documento. Un Ejemplo de estas ambigüedades los encontramos en la representación de ordinales. La aplicación debería mostrar las distintas interpretaciones y permitir al usuario decidir la opción correcta.

- La escritura posicional que se revisó en la sección 2.2.3 consiste en una notación simplificada de Braille utilizada en algoritmos para el cálculo numérico. La interpretación de este tipo de expresiones no ha sido realizada por el intérprete y podrían ser objeto de un trabajo futuro.

Apéndice A: Primeros y siguientes

A lo largo de esta memoria hemos hecho referencia en numerosas ocasiones a los elementos que pertenecen al conjunto de primeros o siguientes. La definición de las funciones primeros y siguientes nos ayudan a construir un analizador sintáctico, a continuación se explica el concepto y los algoritmos que permiten su cálculo.

Primeros

Si α es una cadena de símbolos gramaticales, se considera PRIMEROS(α) al conjunto de terminales (ω) que inician las cadenas derivadas de α. Definimos como cadena vacía (λ) a aquella cadena que no contiene ningún elemento terminal. Si $\alpha \rightarrow \lambda$ entonces λ también pertenece al conjunto de primeros(α). Los elementos que derivan en λ son denominados anulables.

Primeros: concepto $\forall \alpha \epsilon\ (T|N)^+$
Primeros(α) = { ω | $\alpha \Rightarrow \omega$ ... y $\omega \epsilon$ T}

Cálculo del conjunto de símbolos anulables:
Un símbolo A ϵ N es anulable si A $\Rightarrow \lambda$
cálculo del conjunto de símbolos anulables:
Anulables$_0$ = {A ϵ N | A $\rightarrow \lambda$ }
repetir
anulables$_{n+1}$ = anulables$_n$ $\cup$ { A ϵ N | A $\rightarrow \alpha_1\alpha_2...\alpha_m$ $\forall$ i ϵ 1..m $\alpha_i \epsilon$ anulables$_n$ }
Hasta que anulables$_n$==anulables$_{n+1}$

Cálculo del conjunto de primeros de los no terminales:
$\forall$ A ϵ N
Si A ϵ anulables **entonces** P$_0$(A)={λ}
sinó P$_0$(A)=$\varnothing$
Repetir $\forall$ producción A $\rightarrow \alpha$ de G
P$_{n+1}$(A)=P$_n$(A) $\cup$ PrimerosDe(α)
Hasta que $\forall$ A ϵ N : P$_n$(A) == P$_{n+1}$(A)

PrimerosDe(λ)={λ}
PrimerosDe(aβ)={a}, aϵT
PrimerosDe(Aβ) = **si** $\lambda \epsilon$ P$_n$(A) **entonces**

(P_n(A)-{λ}) $\cup$ PrimerosDe(β)
sinó P_n(A)
finsi
,A ϵ N

Siguientes

Se definen Siguientes(A) como el conjunto de elementos terminales ω que pueden aparecer inmediatamente a la derecha de A. Es decir, no existe ningún símbolo entre A y ω o si ha existido en algún momento ha acabado derivando a la cadena vacía λ.

Siguientes: concepto $\forall$A ϵ N
Siguientes(A) ={ω | S $\Rightarrow$ αAγ
ω ϵ Primeros(γ) y
S es el símbolo inicial y
ω ϵ T }

Cálculo del conjunto de siguientes:
$\forall$ A ϵ N **Si** A es el inicial **entonces**
S_0(A) = {λ}
sinó S_0(A) = $\varnothing$
finsi
Repetir $\forall$ producción A $\rightarrow$ α**B**β de G
$\mathbf{S_{n+1}(B)}$ = S_n(B) $\cup$ PrimerosDe(β) $\cup$
si λ ϵ PrimerosDe(β) **entonces**
S_n(A)
sinó $\varnothing$
finsi
Hasta que $\forall$ A ϵ N : S_n(A)== S_{n+1}(A)

Apéndice B: Primeros y Siguientes de la gramática Braille

En este apéndice calculamos el conjunto de elementos primeros y siguientes para la gramática explicada en la sección 3.3.3 y que permite el reconocimiento de texto y matemáticas Braille (figura 5.1). La figura 5.2 muestra el paso de esta gramática en notación BNF a producciones, este cambio de notación es necesario para el cálculo de los elementos Primeros (figura 5.3) y Siguientes (figura 5.4). Los algoritmos utilizados para el cálculo del conjunto de elementos primeros y siguientes se encuentran en el Apéndice A.

<LINEA>::= <EXPR>{('=') <EXPR>}
<EXPR>::= <TERM>{('+' | '-') <TERM>}
<TERM>::= <FACT>{(('∗'| '/' | ':' |'÷'|'·'|'·') <FACT>| <FACT'>) }
<FACT>::= (<FACT1>['∧'<FACT>] | 'raíz1' [<EXPR>]'raíz2' <FACT>)
<FACT'>::= (<FACT1'>['∧'<FACT'>] | 'raíz1' [<EXPR>]'raíz2' <FACT'>)
<FACT1>::= <FACT2>['∨'<FACT1>]
<FACT1'>::= <FACT2'>['∨'<FACT1'>]
<FACT2>::= (('+' | '-') <FACT2>| '('<EXPR>')' | '['<EXPR>']' |'{'<EXPR>'}'
|'<'<EXPR>'>' |[Numerador] Num | Letra | String [FUNCION])
<FACT2'>::= ('('<EXPR>')' | '['<EXPR>']' |'{'<EXPR>'}'
|'<'<EXPR>'>' |[Numerador] Num | Letra | String [FUNCION])
<FUNCION>::= '('[<EXPR>{,<EXPR>}]')'

Figura 5.1: Gramática para el reconocimiento de matemáticas y texto Braille

<linea>	→	<expr><linea_1>	<fact2_1>	→	Letra
<expr>	→	<term><expr_1>	<fact2_1>	→	<fact2_3> Num
<linea_1>	→	'=' <expr><linea_1>	<fact2_1>	→	'<' <expr> '>'
<linea_1>	→	λ	<fact2_1>	→	'{' <expr> '}'
<term>	→	<fact> <term_1>	<fact2_1>	→	'[' <expr> ']'
<expr_1>	→	<expr_2> <term> <expr_1>	<fact2_1>	→	'(' <expr> ')'
<expr_1>	→	λ	<fact2_1>	→	<fact2_2> <fact2>
<fact>	→	<fact_1>	<fact'_3>	→	<expr>
<term_1>	→	<term_2> <term_1>	<fact'_3>	→	λ
<term_1>	→	λ	<fact1'>	→	<fact2'> <fact1'_1>
<expr_2>	→	'-'	<fact'_2>	→	'exponente' <fact'>
<expr_2>	→	'+'	<fact'_2>	→	λ
<fact_1>	→	'raiz1' <fact_3> 'raiz2' <fact>	<funcion>	→	'(' <funcion_1> ')'
<fact_1>	→	<fact1> <fact_2>	<fact2_3>	→	Numerador
<term_2>	→	<fact'>	<fact2_3>	→	λ
<term_2>	→	<term_3> <fact>	<fact2_2>	→	'-'
<fact_3>	→	<expr>	<fact2_2>	→	'+'
<fact_3>	→	λ	<fact2'>	→	<fact2'_1>
<fact1>	→	<fact2> <fact1_1>	<fact1'_1>	→	'subindice' <fact1'>
<fact_2>	→	'exponente' <fact>	<fact1'_1>	→	λ
<fact_2>	→	λ	<funcion_1>	→	<expr> <funcion_2>
<fact'>	→	<fact'_1>	<funcion_1>	→	λ
<term_3>	→	'prefijo_de_letra'	<fact2'_1>	→	String_Nombre_Funcion <funcion>
<term_3>	→	'div4'	<fact2'_1>	→	Letra
<term_3>	→	'div3'	<fact2'_1>	→	<fact2'_2> Num
<term_3>	→	'div2'	<fact2'_1>	→	'<' <expr> '>'
<term_3>	→	'/'	<fact2'_1>	→	'{' <expr> '}'
<term_3>	→	'*'	<fact2'_1>	→	'[' <expr> ']'
<fact2>	→	<fact2_1>	<fact2'_1>	→	'(' <expr> ')'
<fact1_1>	→	'subíndice'<fact1>	<funcion_2>	→	',' <expr> <funcion_2>
<fact1_1>	→	λ	<funcion_2>	→	λ
<fact'_1>	→	'raiz1' <fact'_3> 'raiz2' <fact'>	<fact2'_2>	→	Numerador
<fact'_1>	→	<fact1'> <fact'_2>	<fact2'_2>	→	λ
<fact2_1>	→	String_Nombre_Funcion <funcion>			

Figura 5.2: Paso a producciones de la gramática BNF

P(<linea>)	Numerador '+' '-' String_Nombre_Funcion Letra Num '<' '(' '[' '(' 'raiz1'
P(<linea_1>)	'=' λ
P(<expr>)	'-' '+' Numerador '(' '[' '(' '<' Num Letra String_Nombre_Funcion 'raiz1'
P(<expr_1>)	'-' '+' λ
P(<term>)	Numerador '+' '-' String_Nombre_Funcion Letra Num '<' '(' '[' '(' 'raiz1'
P(<expr_2>)	'+' '-'
P(<term_1>)	Numerador '(' '[' '(' '<' Num Letra String_Nombre_Funcion 'raiz1' '*' '/' 'div2' 'div3' 'div4' 'prefijo_de_letra' λ
P(<term_2>)	Numerador String_Nombre_Funcion Letra Num '<' '(' '[' '(' 'raiz1' 'prefijo_de_letra' 'div4' 'div3' 'div2' '/' '*'
P(<fact>)	'-' '+' Numerador '(' '[' '(' '<' Num Letra String_Nombre_Funcion 'raiz1'
P(<term_3>)	'*' '/' 'div2' 'div3' 'div4' 'prefijo_de_letra'
P(<fact'>)	Numerador '(' '[' '(' '<' Num Letra String_Nombre_Funcion 'raiz1'
P(<fact_1>)	Numerador '+' '-' String_Nombre_Funcion Letra Num '<' '(' '[' '(' 'raiz1'
P(<fact_2>)	'exponente' λ
P(<fact1>)	'-' '+' Numerador '(' '[' '(' '<' Num Letra String_Nombre_Funcion
P(<fact_3>)	Numerador '+' '-' String_Nombre_Funcion Letra Num '<' '(' '[' '(' 'raiz1' λ
P(<fact'_1>)	Numerador String_Nombre_Funcion Letra Num '<' '(' '[' '(' 'raiz1'
P(<fact'_2>)	'exponente' λ
P(<fact1'>)	Numerador '(' '[' '(' '<' Num Letra String_Nombre_Funcion
P(<fact'_3>)	Numerador '+' '-' String_Nombre_Funcion Letra Num '<' '(' '[' '(' 'raiz1' λ
P(<fact1_1>)	'subíndice' λ
P(<fact2>)	Numerador '+' '-' String_Nombre_Funcion Letra Num '<' '(' '[' '('
P(<fact1'_1>)	'subíndice' λ
P(<fact2'>)	Numerador String_Nombre_Funcion Letra Num '<' '(' '[' '('
P(<fact2_1>)	'-' '+' Numerador '(' '[' '(' '<' Num Letra String_Nombre_Funcion
P(<fact2_2>)	'+' '-'
P(<fact2_3>)	Numerador λ
P(<funcion>)	'('
P(<fact2'_1>)	Numerador '(' '[' '(' '<' Num Letra String_Nombre_Funcion
P(<fact2'_2>)	Numerador λ
P(<funcion_1>)	Numerador '+' '-' String_Nombre_Funcion Letra Num '<' '(' '[' '(' 'raiz1' λ
P(<funcion_2>)	',' λ

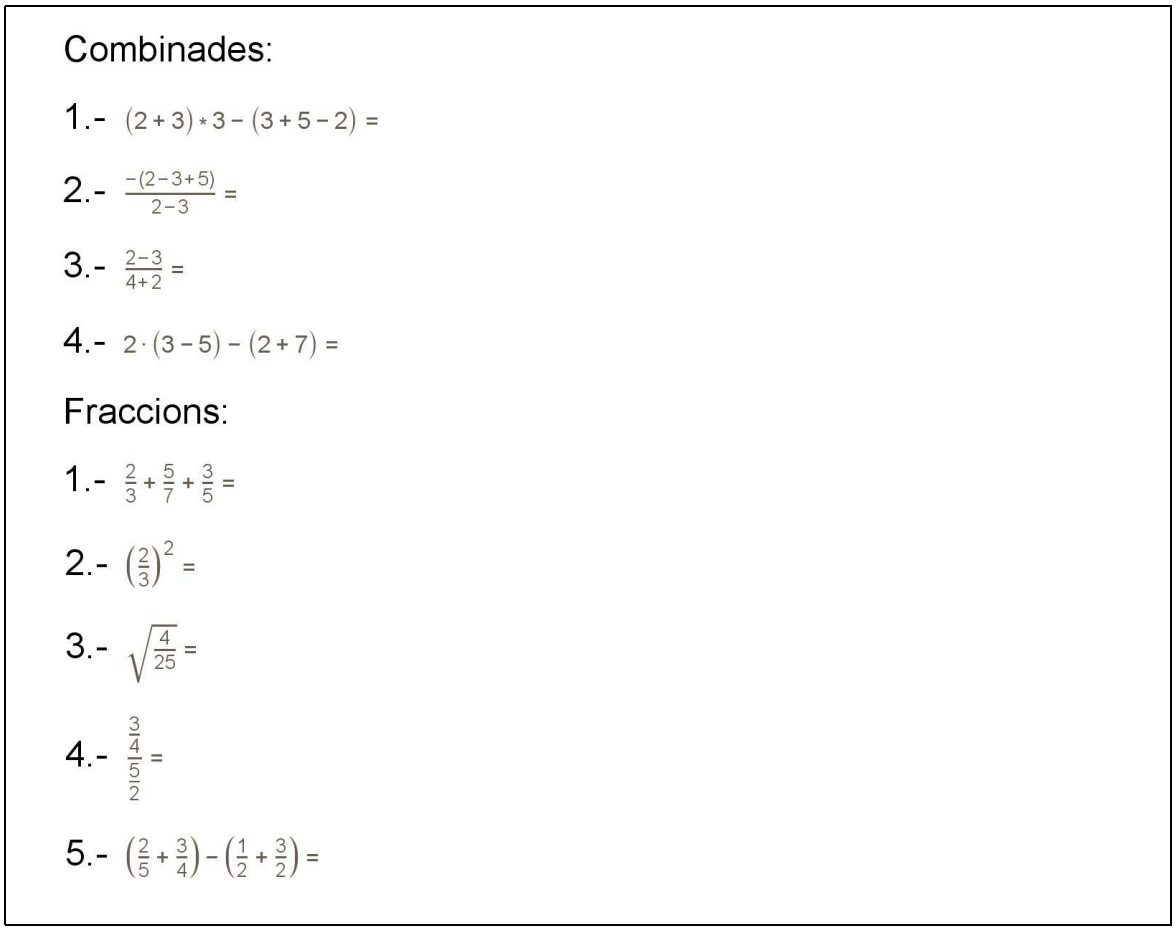
Figura 5.3: Cálculo del conjunto de elementos Primeros para los no terminales

S(<linea>)	EOF
S(<linea_1>')	'=' EOF
S(<expr>)	'=' EOF ',' 'raiz2' ')' ']' '}' '>'
S(<expr_1>)	'>' '}' ']' ')' 'raiz2' ',' EOF '='
S(<Term>)	'=' EOF ',' 'raiz2' ')' ']' '}' '>' '+' '-'
S(<expr_2>)	'raiz1' '(' '[' '{' '<' Num Letra String_Nombre_Funcion '-' '+' Numerador
S(<Term_1>)	'>' '}' ']' ')' 'raiz2' ',' EOF '=' '-' '+'
S(<Term_2>)	'>' '}' ']' ')' 'raiz2' ',' EOF '=' '-' '+' 'prefijo_de_letra' 'div4' 'div3' 'div2' '/' '*' 'raiz1' String_Nombre_Funcion Letra Num '<' '{' '[' '(' Numerador
S(<fact>)	'=' EOF ',' 'raiz2' ')' ']' '}' '>' '+' '-' 'prefijo_de_letra' 'div4' 'div3' 'div2' '/' '*' 'raiz1' String_Nombre_Funcion Letra Num '<' '{' '[' '(' Numerador
S(<Term_3>)	'raiz1' String_Nombre_Funcion Letra Num '<' '{' '[' '(' Numerador '+' '-'
S(<fact'>)	'=' EOF ',' 'raiz2' ')' ']' '}' '>' '+' '-' Numerador '(' '[' '{' '<' Num Letra String_Nombre_Funcion 'raiz1' '*' '/' 'div2' 'div3' 'div4' 'prefijo_de_letra'
S(<fact_1>)	'>' '}' ']' ')' 'raiz2' ',' EOF '=' '-' '+' Numerador '(' '[' '{' '<' Num Letra String_Nombre_Funcion 'raiz1' '*' '/' 'div2' 'div3' 'div4' 'prefijo_de_letra'
S(<fact_2>)	'=' EOF ',' 'raiz2' ')' ']' '}' '>' '+' '-' 'prefijo_de_letra' 'div4' 'div3' 'div2' '/' '*' 'raiz1' String_Nombre_Funcion Letra Num '<' '{' '[' '(' Numerador
S(<fact1>)	'>' '}' ']' ')' 'raiz2' ',' EOF '=' '-' '+' 'exponente' 'Numerador' '(' '[' '{' '<' Num Letra String_Nombre_Funcion 'raiz1' '*' '/' 'div2' 'div3' 'div4' 'prefijo_de_letra'
S(<fact_3>)	'raiz2'
S(<fact'_1>)	'>' '}' ']' ')' 'raiz2' ',' EOF '=' '-' '+' 'prefijo_de_letra' 'div4' 'div3' 'div2' '/' '*' 'raiz1' String_Nombre_Funcion Letra Num '<' '{' '[' '(' Numerador
S(<fact'_2>)	'=' EOF ',' 'raiz2' ')' ']' '}' '>' '+' '-' Numerador '(' '[' '{' '<' Num Letra String_Nombre_Funcion 'raiz1' '*' '/' 'div2' 'div3' 'div4' 'prefijo_de_letra'
S(<fact1'>)	'>' '}' ']' ')' 'raiz2' ',' EOF '=' '-' '+' 'exponente' 'prefijo_de_letra' 'div4' 'div3' 'div2' '/' '*' 'raiz1' String_Nombre_Funcion Letra Num '<' '{' '[' '(' Numerador
S(<fact'_3>)	'raiz2'
S(<fact1_1>)	'=' EOF ',' 'raiz2' ')' ']' '}' '>' '+' '-' 'prefijo_de_letra' 'div4' 'div3' 'div2' '/' '*' 'raiz1' String_Nombre_Funcion Letra Num '<' '{' '[' '(' Numerador 'exponente'
S(<fact2>)	'>' '}' ']' ')' 'raiz2' ',' EOF '=' '-' '+' 'exponente' Numerador '(' '[' '{' '<' Num Letra String_Nombre_Funcion 'raiz1' '*' '/' 'div2' 'div3' 'div4' 'prefijo_de_letra' 'subindice'
S(<fact1'_1>)	'=' EOF ',' 'raiz2' ')' ']' '}' '>' '+' '-' Numerador '(' '[' '{' '<' Num Letra String_Nombre_Funcion 'raiz1' '*' '/' 'div2' 'div3' 'div4' 'prefijo_de_letra' 'exponente'
S(<fact2'>)	'>' '}' ']' ')' 'raiz2' ',' EOF '=' '-' '+' 'exponente' 'prefijo_de_letra' 'div4' 'div3' 'div2' '/' '*' 'raiz1' String_Nombre_Funcion Letra Num '<' '{' '[' '(' Numerador 'subindice'
S(<fact2_1>)	'=' EOF ',' 'raiz2' ')' ']' '}' '>' '+' '-' 'prefijo_de_letra' 'div4' 'div3' 'div2' '/' '*' 'raiz1' String_Nombre_Funcion Letra Num '<' '{' '[' '(' Numerador 'exponente' 'subindice'
S(<fact2_2>)	'(' '[' '{' '<' Num Letra String_Nombre_Funcion '-' '+' Numerador
S(<fact2_3>)	Num
S(<funcion>)	'>' '}' ']' ')' 'raiz2' ',' EOF '=' '-' '+' 'exponente' 'prefijo_de_letra' 'div4' 'div3' 'div2' '/' '*' 'raiz1' String_Nombre_Funcion Letra Num '<' '{' '[' '(' Numerador 'subindice'
S(<fact2'_1>)	'=' EOF ',' 'raiz2' ')' ']' '}' '>' '+' '-' Numerador '(' '[' '{' '<' Num Letra String_Nombre_Funcion 'raiz1' '*' '/' 'div2' 'div3' 'div4' 'prefijo_de_letra' 'exponente' 'subindice'
S(<fact2'_2>)	Num
S(<funcion_1>)	')'
S(<funcion_2>)	')'

Figura 5.4: Cálculo del conjunto de elementos Siguientes para los no terminales

Apéndice C: Ejemplos de interpretación de documentos

Este apéndice muestra algunas de las traducciones realizadas por el intérprete de matemáticas y texto Braille. Los documentos han sido escritos utilizando una máquina Perkins por alumnos Centro de Recursos Educativos Joan Amades, en ellos se recogen los principales operadores matemáticos implementados en la aplicación desarrollada.



4

7

e) $\frac{\sqrt{a*b}}{\sqrt[3]{a*b}} = \frac{\sqrt[6]{a^3*b^3}}{\sqrt[6]{a^2*b^2}} = \sqrt[6]{\frac{a^3*b^3}{a^2*b^2}} = \sqrt[6]{a*b}$

f) $\frac{\sqrt[6]{a^3}}{\sqrt{a^2}} = \frac{\sqrt[6]{a^3}}{\sqrt[6]{a^4}} =$

$= \sqrt[6]{\frac{a^3}{a^4}} = \sqrt[6]{\frac{1}{a}} = \frac{1}{\sqrt[6]{a}}$

g) $\frac{\sqrt[4]{a^3*b^5*c}}{\sqrt{a*b^3*c^3}} =$

$= \frac{\sqrt[4]{a^3*b^5*c}}{\sqrt[4]{a^2*b^6*c^6}} = \sqrt[4]{\frac{a^3*b^5*c}{a^2*b^6*c^6}} = \sqrt[4]{\frac{a}{b \cdot c^5}}$

h) $\sqrt[3]{2} * \sqrt[5]{2} = \sqrt[15]{2^5} * \sqrt[15]{2^3} =$

$= \sqrt[15]{2^5 * 2^3} = \sqrt[15]{2^8}$

7. Realitza la següent operació:

$\sqrt{18} + \sqrt{50} - \sqrt{2} - \sqrt{8}$

Sol:

$\sqrt{18} + \sqrt{50} - \sqrt{2} - \sqrt{8} = \sqrt{2*3^2} + \sqrt{2*5^2} - \sqrt{2} - \sqrt{2^3} = 3 \cdot \sqrt{2} + 5 \cdot \sqrt{2} - \sqrt{2} - 2 \cdot \sqrt{2} =$
$(3 + 5 - 1 - 2) \cdot \sqrt{2} =$

$= 5 \cdot \sqrt{2}$

1. Realitza els càlculs de les següents operacions amb potències:

a) $7^2 * 7^3 * 7 =$

b) $(-4)^2 * (-4)^4 =$

c) $\left(\frac{3}{5}\right)^3 * \left(\frac{3}{5}\right)^2 =$

d) $\frac{5^6}{5^2} =$

e) $\frac{\left(\frac{2}{3}\right)^5}{\left(\frac{2}{3}\right)^3} =$

f) $5^2 * 8^2 =$

g) $\left(\frac{1}{3}\right)^3 * \left(\frac{1}{3}\right)^3 =$

h) $(-2)^7 * (-5)^3 =$

i) $(3^2)^5 =$

j) $\left((-5)^2\right)^4 =$

k) $(-5)^3 * (-5)^2 * (-5) =$

l) $(-8)^3 * 2^3 =$

m) $\frac{\left(2^5 * 3^5\right)}{6^4} =$

n) $\frac{\left(\frac{2}{3}\right)^3}{\left(\frac{2}{3}\right)^2} =$

c) $(-8)^{\frac{2}{3}} =$

d) $\left\{\left(\sqrt{a+\sqrt{a}}+\sqrt{a-\sqrt{a}}\right) * \left(\sqrt{a+\sqrt{a}}-\sqrt{a-\sqrt{a}}\right)\right\}^2 =$

e) $2\cdot\sqrt{27}-3\cdot\sqrt{48}+\frac{1}{5}\cdot\sqrt{75} =$

f) $\left(\sqrt[3]{9}-\sqrt[3]{4}\right)\cdot\left(\sqrt[3]{3}+3\cdot\sqrt[3]{2}\right) =$

g) $\sqrt[6]{27}+\sqrt[4]{4}-\sqrt[6]{8} =$

h) $\left(x\cdot\sqrt{y}+y\cdot\sqrt{x}\right)\cdot\sqrt{x\cdot y} =$

i) $\sqrt{8-\frac{4}{25}}$

j) $\sqrt{\frac{5}{18}+\frac{5}{12}} =$

k) $\sqrt{\frac{4\cdot x^2}{y^4}}+\sqrt{\frac{y^4}{4}\cdot x^2}$

l) $\sqrt{5+2\cdot\sqrt{6}}\cdot\sqrt{5-2\cdot\sqrt{6}} =$

m) $(2+\sqrt{3})^2$

n) $(5\cdot\sqrt{2}+4\cdot\sqrt{3})\cdot(5\cdot\sqrt{2}-4\cdot\sqrt{3}) =$

ñ) $\frac{\sqrt{x\cdot y}}{\left(\sqrt{\frac{x}{y}}-\sqrt{\frac{y}{x}}\right)} =$

o) $100^{0,5}+81^{0,25}-16^{0,75} =$

p) $\sqrt[\frac{3}{4}]{\sqrt[4]{\frac{8}{27}}} =$

q) $\sqrt[3]{2\cdot\sqrt{2}} =$

Apéndice D: Construcción de un parser a partir de la notación BNF

En este apéndice se puede ver cómo transformar una gramática en notación BNF a un algoritmo iterativo. La gramática Braille se ha transformado en un algoritmo iterativo siguiendo estas reglas.

- **Programa principal**
  símbolo SLA; /* símbolo leído */
  main()
  { SLA = leer_simbolo();
  Inicial();
  If (SLA!=1) Error();
  }

  El programa principal siempre llamará a la misma función inicial (elemento no terminal) de la gramática.

- **<no terminal>::= expresión BNF**
  no_terminal() { Parser[expresión BNF] }

  Un símbolo no terminal definido en la parte izquierda de una regla BNF, pasará a ser una función en el algoritmo iterativo. La parte derecha de la regla BNF pasará a estar en el cuerpo de la nueva función.

- **Parser[α_1 | α_2 | ... | α_n]**
  switch (SLA)
  {
  case(Primeros(α_1)): Parser[α_1] break;
  case(Primeros(α_2)): Parser[α_2] break;
  ...
  case(Primeros(α_n)): Parser[α_n] break;
  default: error();
  }

  Un operador de unión en una gramática BNF pasará a ser una estructura de control de tipo switch - case.

- **Parser[α_1 α_2 ... α_n]**
  {
  Parser[α_1]
  Parser[α_2]
  ...
  Parser[α_n]
  }

  Un operador de tipo secuencia se transforma en una concatenación de todos los elementos de la secuencia en el nuevo código iterativo.

- **Parser[{α_n}]**
  {
  while (SLA Primeros(α))
  Parser[α_n]
  }

  El operador de tipo repetición {} en la gramática BNF pasa a ser un bucle en el algoritmo iterativo.

- **Parser[<no_terminal>]**
  {
  No_terminal()
  }

  Un símbolo no terminal en la parte derecha de una producción pasará a ser una llamada a una función en el nuevo algoritmo iterativo.

- **Parser[<terminal>]**
  {
  If (SLA==Terminal)
  leer_simbolo()
  else error()
  }

Un símbolo terminal llamará a la función leer_simbolo() que se encargará de avanzar hasta el siguiente token.

En el ejemplo de la figura 5.5 se puede ver la transformación de la regla en notación BNF <EXPR>::= <TERM>{('+' | '-') <TERM>} a un algoritmo iterativo. Las transformaciones realizadas para ello han sido las siguientes:

- El elemento no terminal <EXPR>situado a la izquierda de la regla, pasa a definir una nueva función expr() en el algoritmo iterativo.

- Los elementos no terminales <TERM>situados a la derecha de la regla, pasan a ser llamadas a la función term() en el algoritmo iterativo.

```
<EXPR>::= <TERM>{('+' | '-') <TERM>}
procedure expr()
begin
        term();
        while (1)
            if (token = '+') then
                leer_simbolo('+');
                term();
            end if;
            else if (token = '-') then
                leer_simbolo('-');
                term();
            end if;
            else return;
        end while;
end expr;
```

Figura 5.5: Construcción del Parser para la regla BNF Expr de nuestra gramática

- El operador de repetición {} pasa a ser un bucle While.

- El operador unión | se transforma en un estructura de control de tipo if then else con los elementos terminales como condición.
- Los símbolos terminales '+' y '-' llaman a la función leer_simbolo que leerá el siguiente token a procesar.

Bibliografía

[1] F. Schwebel. BraMaNet logiciel de traduction des mathématiques en braille, 2003.

[2] M. Batusic, K. Miesenberger, B. Stöger. Labradoor, a contribution to making mathematics accessible for the blind. In A. Edwards, A. Arato, W. Zagler, editors, Proc. ICCHP 98 (6th. International Conference on Computers Helping People with Special Needs), Oldenbourg, Wien, München., 1998.

[3] Crombie, D., Leonor, R., McKenzie, N., Barker, A: math2braille: Opening Access to Mathematics. In: Proc. ICCHP 2004 (International Conferences on Computers Helping People with Special Needs). Lecture Notes in Computer Science, Vol. 3118. Paris, France, 2004, pp. 670-677.

[4] Suzuki, M., Kanahori, T., Ohtake, N., and Yamaguchi, K. (2004). .ªn Integrated OCR Software for Mathematical Documents and Its Output with Accessibility.Ïn Miesenberger, K., Klaus, J., Zagler, W., and Burger, D., editors, Proc. ICCHP 2004 (9th International Conference on Computers Helping People with Special Needs), volume 3118 of LNCS, Berlin. Springe, 2004, pp. 648-655.

[5] Annamalai, A., Gopal, D., Gupta, G., Guo, H., Karshmer, A.: INSIGHT: a comprehensive system for converting Braille based mathematical documents to LaTeX. In: Stephanidis, C. (ed.): Universal Access in HCI- Inclusive Design in the Information Society, Vol. 4. Mahwah, NJ, 2003, pp. 1245-1249.

[6] A. I. Karshmer, G. Gupta, S. Geiiger, C. Weaver. Reading and writing mathematics: the mavis project. In International ACM Conference on Assistive Technologies (ASSETS), Marina del Rey, CA, USA, 1998, pp. 136-143.

[7] Hamid Reza Shahbazkia, Telmo Taveres Silva, and Rui Miguel Guerreiro. Automatic Braille Code Translation System. Universidade do Algarbe, Faro-Portugal, 2005.

[8] Disponible en URL: http://blatmat.sourceforge.net/

[9] Fernando Alonso, José L. Fuertes, Ángel L. González, and Loïc A. Martínez. SBT: A Translator from Spanish Braille to MathML, Facultad de informática, Universidad Politécnica de Madrid, 2006.

[10] Moço, V.,Archambault, D.: Automatic translator for mathematical Braille. In: Stephanidis, C. (ed.): Universal Access in HCI- Inclusive Design in the Information Society, Vol. 4. Mahwah, NJ, 2003, pp. 1335-1339.

[11] Archambault, D., Fitzpatrick, D., Gupta, G. Karshmer, A. I., Miesenberger, K., Pontelli, E.: Towards a universal maths conversion library. In: Proc. ICCHP 2004 (Intenational Conference on Computers Helping People wi Special Needs). Lecture Notes in Computer Science, Col. 3118. Paris, France, 2004, pp. 664-669.

[12] D. Archambault, D. Burger. The vickie project. In K. Miesenberger, J. Klaus, W. Zagler, editors, Proc. ICCHP 2002 (International Conference on Computers Helping People with Special Needs), Vol. 2398 of LNCS, pp. 90 - 97, Linz, Austria. Springer.

[13] B. Stöger, M. Batusié, C. Fahrengruber, K. Miesenberger, D. Archambault: Mathematical Working Environment, MAWEN, 2007.

[14] Nicotra, G.: LAMBDA Project. Disponible en URL: http://www.lamdaproject.org, 2006.

[15] Jose Enrique Fernández del campo. Braille y matemática, Organización Nacional de Ciegos Españoles (ONCE), Madrid, 2004.

[16] Alfred V.Aho, Ravi Sethi, Jeffrey D.Ullman. Compiladores, Principios, Técnicas y Herramientas, 2001.

[17] Philippe Charles, A Practical method for constructing Efficient LALR(k) Parsers with Automatic Error Recovery, 1991.

[18] Kenneth C. Louden, Construcción de compiladores, ed. Thomson, 2004.

[19] W3C Math Home, Disponible en URL: http://www.w3.org/Math/

Resumen. El trabajo expuesto en la presente memoria, forma parte de un proyecto de colaboración entre el Centro de Visión por Computador de la UAB y el Centro Joan Amades (ONCE), cuyo objetivo es la creación de recursos educativos que faciliten la integración de niños invidentes en las aulas.
Se presenta el proceso de implementación de un intérprete y traductor de documentos escritos en Braille con contenido matemático y de texto, que permite a un profesor que no conozca el sistema Braille, la lectura de documentos creados por alumnos invidentes.
Dicho intérprete forma parte de una herramienta que permite el reconocimiento de documentos escritos con una máquina Perkins.

Resum. El treball exposat a la següent memòria, forma part d'un projecte de col·laboració entre el Centre de Visió per Computador de la UAB i el Centre Joan Amades (ONCE), que té per objectiu la creació de recursos educatius que facilitin la integració de nens invidents a les aules.
Es presenta el procés de implementació d'un intèrpret i traductor de documents escrits en Braille amb contingut matemàtic i de text, que permet a un professor que no conegui el sistema Braille, la lectura de documents creats per alumnes invidents.
Aquest intèrpret forma part d'una eina que permet el reconeixement de documents escrits amb una màquina Perkins.

Abstract. The work outlined in this report is part of a collaborative project between the Computer Vision Center (UAB) and the Joan Amades Center (ONCE), which aims to create educational resources that facilitate the integration of blind children in classrooms.
We present the implementation process of an interpreter and translator of documents written in Braille to mathematical content and text, which allows a teacher who doesn't know Braille system, reading documents created by blind pupils.
This interpreter is part of a tool that allows the recognition of documents written in a Perkins machine.