UNIVERSIDADE DE SANTIAGO DE COMPOSTELA



ESCOLA TE´CNICA SUPERIOR DE ENXEN˜ AR´IA

Perldoop 2.0

Un compilador fuente-a-fuente Perl-Java

*Autor:*

César Piñeiro Pomar

*Directores:*

Juan Carlos Pichel Campos

José Manuel Abuín Mosquera

Grao en Enxeñeira Informática

Xulio 2016

Traballo de Fin de Grao presentado na Escola Técnica Superior de Enxeñería da universidade de Santiago de Compostela para a obtención do Grao en Enxeñaría Informática



**D. Juan Carlos Pichel Campos**, Profesor do Departamento de Electrónica e Computación da Universidade de Santiago de Compostela, e **D. José Manuel Abuín Mosquera**, Profesor do Departamento de Electrónica e Computación da Universidade de Santiago de Compostela,

INFORMAN:

Que a presenta memoria, titulada *Perldoop 2.0: Un compilador fuente-a-fuente Perl-Java*, presentada por **D. César Piñeiro Pomar** para superar os créditos correspondentes ao Traballo de Fin de Grao da titulación de Grao en Enxería Informática, realizouse baixo a nosa dirección no Departamento de Electrónica e Computación da universidade de Santiago de Compostela.

E para que así conste aos efectos oportunos , expiden o presente informe en Santiago de Compostela, a (Data):

O director, O codirector, O alumno,

Juan Carlos Pichel Campos José Manuel Abuín Mosquera César Piñeiro Pomar

Agradecementos

Me gustaría dar las gracias a D. Juan Carlos Pichel Campos y a D. José Manuel Abuín Mosquera por haber dirigido el presente trabajo, por su dedicación a lo largo de estos meses, por todo lo que he aprendido en este tiempo y por la cercanía con la que siempre me han tratado. También agradecer a D. Paulo Félix Lamas por ser la persona que me guio hasta ellos.

Por supuesto también agradecer a Alberto Seoane, Alex Suarez y sobre todo a Cristian Rodríguez, por habernos aguantado a los tres. ¡Gracias chicos! Por ser buenos compañeros y amigos.

Por ultimo a mi madre Hilda Pomar por su dedicación y apoyo.

Índice

[1. Introducción 1](#_Toc455592205)

[1.1. Perldoop Original 1](#_Toc455592206)

[1.2. Limitaciones de la versión original 2](#_Toc455592207)

[1.3. Objetivos del TFG 3](#_Toc455592208)

[2. Gestión del Alcance 4](#_Toc455592209)

[2.1. Descripción del Alcance 4](#_Toc455592210)

[2.2. Entregables 4](#_Toc455592211)

[2.3. Criterios de Aceptación 4](#_Toc455592212)

[2.4. Restricciones del Proyecto. 5](#_Toc455592213)

[2.5. Exclusiones del Proyecto 5](#_Toc455592214)

[2.6. Supuestos del Proyecto 5](#_Toc455592215)

[3. Casos de Uso 6](#_Toc455592216)

[4. Análisis de requisitos 12](#_Toc455592217)

[4.1. Requisitos funcionales 12](#_Toc455592218)

[4.2. Requisitos no funcionales 15](#_Toc455592219)

[5. Gestión del Proyecto 16](#_Toc455592220)

[5.1. Metodología 16](#_Toc455592221)

[5.1.1. Programación extrema 17](#_Toc455592222)

[5.2. Gestión de la Configuración 17](#_Toc455592223)

[5.2.1. Identificación de elementos de configuración 18](#_Toc455592224)

[5.2.1. Herramienta gestión de la configuración 18](#_Toc455592225)

[5.3. Planificación temporal 19](#_Toc455592226)

[5.3.1. Planificación y Alcance 20](#_Toc455592227)

[5.3.2. Formación 20](#_Toc455592228)

[5.3.3. Análisis 20](#_Toc455592229)

[5.3.4. Diseño 20](#_Toc455592230)

[5.3.5. Implementación 21](#_Toc455592231)

[5.3.6. Pruebas 21](#_Toc455592232)

[5.3.7. Documentación 21](#_Toc455592233)

[5.3.8. Cronograma 22](#_Toc455592234)

[5.3.9. Diagrama de Gantt 23](#_Toc455592235)

[5.4. Gestión de riesgos 24](#_Toc455592236)

[5.4.1. Especificación 25](#_Toc455592237)

[5.5. Análisis de costes 28](#_Toc455592238)

[6. Herramientas y Tecnologías 30](#_Toc455592239)

[6.1. IDEs de desarrollo 30](#_Toc455592240)

[6.2. Tecnologías 30](#_Toc455592241)

[6.3. Aplicaciones 31](#_Toc455592242)

[7. Arquitectura del software 33](#_Toc455592243)

[7.1. Interfaz de línea comandos 33](#_Toc455592244)

[7.2. Analizador Léxico 35](#_Toc455592245)

[7.3. Analizador Sintáctico 35](#_Toc455592246)

[7.4. Comprobación y generación de código 35](#_Toc455592247)

[7.5. Sistema de gestión de errores 36](#_Toc455592248)

[7.6. Sistema de mensajes 36](#_Toc455592249)

[8. Diseño e Implementación 37](#_Toc455592250)

[8.1. Diagrama de clases 37](#_Toc455592251)

[8.2. Diagramas de interacción 41](#_Toc455592252)

[8.3. Diseño tokens del analizador léxico 43](#_Toc455592253)

[8.3.1. Tokens Básicos 43](#_Toc455592254)

[8.3.2. Tokens de palabras reservadas 45](#_Toc455592255)

[8.3.3. Tokens de Etiquetas 46](#_Toc455592256)

[8.4. Gramática de Analizador Sintáctico 47](#_Toc455592257)

[8.5. Generación y Comprobación de código 53](#_Toc455592258)

[8.5.1. Tipos de datos 54](#_Toc455592259)

[8.5.1.1. Tipos asociados a etiquetas 54](#_Toc455592260)

[8.5.1.2. Tipos asociados a Contantes 55](#_Toc455592261)

[8.5.2. Casting automático 55](#_Toc455592262)

[8.5.2. Definición de funciones 56](#_Toc455592263)

[8.5.3. Asignación 58](#_Toc455592264)

[8.5.4. Ficheros 60](#_Toc455592265)

[8.5.5. Paquetes 61](#_Toc455592266)

[8.5.6. Funciones nativas 62](#_Toc455592267)

[8.5.7. Expresiones regulares 63](#_Toc455592268)

[8.5.8. Apache Hadoop 64](#_Toc455592269)

[8.5.8.1. Mapper 64](#_Toc455592270)

[8.5.8.2 Reducer 66](#_Toc455592271)

[8.6. Librería Java 68](#_Toc455592272)

[8.6.1. Clases para Tipos 69](#_Toc455592273)

[8.6.2. Clases para Funciones 70](#_Toc455592274)

[9. Validación y Pruebas 72](#_Toc455592275)

[9.1. Pruebas Unitarias 72](#_Toc455592276)

[9.1.1. Validación de requisitos. 73](#_Toc455592277)

[9.1.2. Pruebas librería Java 78](#_Toc455592278)

[9.2. Pruebas de Integración 82](#_Toc455592279)

[Conclusión 86](#_Toc455592280)

[Bibliografía 87](#_Toc455592281)

Índice Figuras

[Figura 1: Plantilla Perldoop 2](#_Toc455586212)

[Figura 2: Diagrama casos de uso 6](#_Toc455586213)

[Figura 3: Estructura de descomposición de trabajo (EDT) 19](#_Toc455586214)

[Figura 4: Diagrama Gantt parte 1 23](#_Toc455586215)

[Figura 5: Diagrama Gantt parte 2 24](#_Toc455586216)

[Figura 6: Arquitectura del sistema 33](#_Toc455586217)

[Figura 7: Ayuda Perldoop 2 34](#_Toc455586218)

[Figura 8: Ejemplo sistema de errores 36](#_Toc455586219)

[Figura 9: Clases con sus atributos y métodos 37](#_Toc455586220)

[Figura 10: Diagrama de clases 38](#_Toc455586221)

[Figura 11: Diagrama de secuencia de un análisis 41](#_Toc455586222)

[Figura 12: Error en el análisis 42](#_Toc455586223)

[Figura 13: Gramática analizador sintáctico 48](#_Toc455586224)

[Figura 14: Función perl con array y escalar 57](#_Toc455586225)

[Figura 15: Funcion Perl con referencia Array y escalar 57](#_Toc455586226)

[Figura 16: Ejemplo de tipado de funciones y su traducción 58](#_Toc455586227)

[Figura 17: Asignación entre listas 59](#_Toc455586228)

[Figura 18: Asignación entre lista y expresión 59](#_Toc455586229)

[Figura 19: Asignación entre lista y función 60](#_Toc455586230)

[Figura 20: Inicialización de colecciones 60](#_Toc455586231)

[Figura 21: Inicialización de colecciones optimizada 60](#_Toc455586232)

[Figura 22: Apertura de fichero 61](#_Toc455586233)

[Figura 23: Ejemplo de lectura de fichero 61](#_Toc455586234)

[Figura 24: Ejemplo de escritura de fichero 61](#_Toc455586235)

[Figura 25: Creación y uso de un paquete con su traducción 62](#_Toc455586236)

[Figura 26: Expresión regular match 64](#_Toc455586237)

[Figura 27: Mapper Perl 65](#_Toc455586238)

[Figura 28: Mapper java 66](#_Toc455586239)

[Figura 29:Reducer Perl 67](#_Toc455586240)

[Figura 30: Reducer java 68](#_Toc455586241)

[Figura 31: Clases para tipos 69](#_Toc455586242)

Índice Tablas

[Tabla 1: Tipos de documento 18](#_Toc455586243)

[Tabla 2: Cronograma 22](#_Toc455586244)

[Tabla 3: Tokens básicos 45](#_Toc455586245)

[Tabla 4: Tokens de palabras reservadas 46](#_Toc455586246)

[Tabla 5: Exprsiones regualres para etiquetas 46](#_Toc455586247)

[Tabla 6: Tokens de etiquetas 47](#_Toc455586248)

[Tabla 7: Relación etiqueta tipo de dato 54](#_Toc455586249)

[Tabla 8: Tabla de funciones auxiliares librería 71](#_Toc455586250)

# Introducción

Perldoop 2.0 es una herramienta que permite convertir código fuente escrito en Perl a código fuente escrito en Java. Esta herramienta es una reconstrucción desde cero del Perldoop original con el objetivo de eliminar limitaciones y añadir nuevas características.

## Perldoop Original

El trabajo realizado por los investigadores del CiTIUS dio como resultado la creación del programa Perldoop [1]. Su diseño permite transformar de forma sencilla y eficiente un script Perl [2] para el Procesamiento de Lenguaje Natural (PLN) en un programa java compatible con las tecnologías Big Data (en concreto, Hadoop[3]).

La herramienta hace uso de etiquetas, introducidas en el código original, para poder convertir automáticamente un código Perl secuencial en un código Java adaptado al paradigma MapReduce mejorando su ejecución en un clúster.

Perldoop está basado en la ejecución de expresiones regulares sobre el código para aplicar las transformaciones, por esta razón la herramienta requiere una estructura bien definida para poder funcionar, en particular se necesitan dos ficheros para generar el código java.

1. En primer lugar, como es de esperar, un archivo que contenga el script Perl a traducir. Este código debería:
   1. Seguir unas reglas estrictas de programación [4].
   2. Estar etiquetado correctamente [5].
2. Una plantilla Java es necesaria para asegurar el correcto funcionamiento del código generado por esta herramienta. La plantilla debería incluir el constructor de la clase, así como los import de las dependencias necesarias.

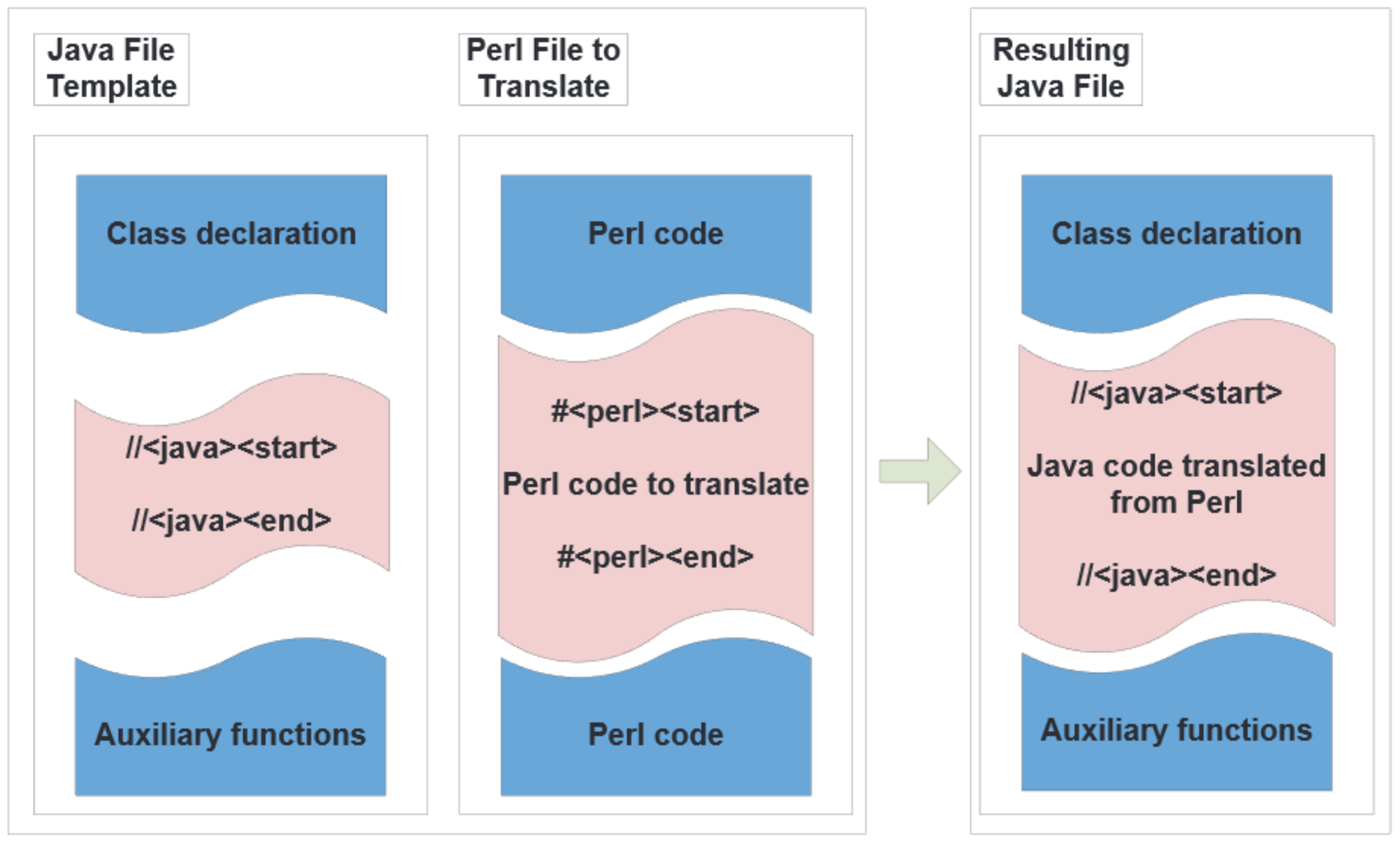


Figura 1: Plantilla Perldoop

La Figura 1 muestra el funcionamiento de Perldoop, la herramienta traduce el código entre las etiquetas <perl><start> y <perl><end> del código fuente Perl y pega esta traducción entre las etiquetas <java><start> y <java><end> de la plantilla.

## Limitaciones de la versión original

El principal problema de esta versión era la necesidad de una plantilla implementada previamente para poder ejecutar el código traducido. Una persona sin conocimientos de Java necesitaba que alguien elaborase la plantilla y en caso de cambios en el script, dependencias sobre todo, podía ser necesario tener que cambiar la plantilla. La fuente del problema es que, al usar patrones, solo se aplican transformaciones y el analizador no “entiende” el código con lo cual es imposible que sepa cuando hay que importar dependencias.

Otra limitación es que, como podemos ver en la Figura 1, las funciones auxiliares no son traducidas, un programador Java debe traducir el código de Perl manualmente, lo cual impide crear códigos modulares.

El etiquetado en algunos casos es redundante, algunas etiquetas no aportan información que no pudiera extrapolarse directamente del código. Un ejemplo es la etiqueta <perl> que precede a cualquier etiqueta escrita en código Perl o <var> que indica que estamos tratando una variable. Cuanta más información saquemos del código fuente Perl, más rápido será el etiquetado.

También existían limitaciones de sintaxis Perl tiene expresiones sintácticas que no están presentes en Java, por ejemplo, aplicar estructuras de control a una sentencia o evaluar condiciones no booleanas. Las reglas de programación para Perldoop especifican las alternativas a los casos anteriores para que el programador pueda adaptar su código.

## Objetivos del TFG

A continuación, detallamos los objetivos de este trabajo de fin de grado:

* Traducción de un subconjunto de la sintaxis y código Perl a Java con solo introducir una serie de etiquetas que permiten definir tipos de variables, tamaño de arrays, porciones de código, etc.
* Soporte para todos los tipos de datos básicos, así como para Array y Hash.
* Conversión para los operadores matemáticos, lógicos, binarios y de cadenas.
* Casting automático entre tipos de datos básicos de forma transparente al usuario, transformando número a cadenas o enteros a flotantes, entre otros.
* Soporte para las estructuras de control que afectan tanto a bloques como a líneas, entre ellas podemos destacar if, while, for, foreach, etc.
* Llamadas a funciones nativas de Perl que se determinen esenciales o que sean muy usadas en la mayoría de los scripts.
* Definición de funciones personalizadas, aunque con alguna limitación, que permitan separar el código en funciones para mejorar su legibilidad.
* Transformación de segmentos de código secuencial definidos en funciones mapper o reducer compatibles con Hadoop.
* Emulación básica de la aritmética de punteros de Perl, añadiendo la posibilidad de referenciar colecciones en vez de obligar a copiarlas.
* Sistemas de gestión de errores donde se informe al usuario de los errores producidos y las líneas donde han sido cometidos.
* Comprobaciones sobre el código para avisar de comportamientos no permitidos en Java como son el leer una variable sin inicializar o tener secciones muertas en código.
* Opciones de depuración para buscar errores en el análisis y facilitar correcciones en el traductor.

# Gestión del Alcance

En este capítulo se describe el resultado esperado como consecuencia de la realización de este proyecto.

## Descripción del Alcance

El producto resultante del desarrollo de este proyecto será un traductor fuente-a-fuente que permita traducir un subconjunto limitado de la sintaxis de Perl a Java. El software recibirá como entrada un script escrito en Perl y proporcionará como salida un código Java que pueda ser compilado y ejecutado directamente.

La herramienta se presenta como la versión mejorada de la versión inicial, la sintaxis Perl será aumentada en la medida de lo posible para reconocer los patrones más usados en los scripts de procesamiento de lenguaje natural (PLA).

Para terminar, al igual que la versión original, el producto final incluirá un manual de uso donde se especifiquen las características y la sintaxis de la herramienta.

## Entregables

* **Perldoop 2.0**: Sofware terminado y listo para ser usado.
* **Memoria de proyecto**: Aspectos relacionados con la gestión del proyecto.
* **Presentación**: Documento para la presentación del TFG en su defensa.
* **Manual de uso**: Documento para familiarizar a los usuarios con el uso y el funcionamiento de la herramienta.

## Criterios de Aceptación

En base a los objetivos especificados anteriormente, se logrará la aceptación de producto final cuando se cumplan los siguientes criterios.

La herramienta deberá poder traducir cualquier script Perl escrito con la sintaxis dentro del marco especificado en este Proyecto. Por otro lado, en caso de que la herramienta no pueda efectuar la traducción por un error sintáctico o semántico, el usuario será informado de la causa del problema y además de la posición aproximada del mismo dentro del fichero.

El proyecto deberá contar con una serie de pruebas que aseguren que se cumple lo especificado previamente, así como que el código fuente de la herramienta está libre de errores sintácticos típicos en unos lenguajes de programación no compilados.

Por último, se considerará el proyecto aceptado cuando ambos tutores den el visto bueno tanto a esta documentación como a la herramienta de traducción.

## Restricciones del Proyecto.

Debido a la diferencia del nivel entre los lenguajes de programación Perl y Java, se requiere complementar las sintaxis de Perl con etiquetas en los scripts a traducir.

## Exclusiones del Proyecto

La finalidad de esta herramienta es proporcionar un método rápido de traducción de scripts Perl a Java, en ningún momento se intenta conseguir un traductor con las capacidades del intérprete del Perl. Para empezar la sintaxis orientada a objetos de Perl no será soportada por no ser necesaria.

## Supuestos del Proyecto

En este proyecto asumiremos que los scripts han sido probados y validados por el intérprete de Perl, pues no tiene sentido perder tiempo en solucionar problemas para los cuales ya existe una solución oficial por parte de los creadores del lenguaje.

# Casos de Uso

Los casos de uso definen los posibles escenarios de iteración de un usuario con el software. Los diagramas muestran el comportamiento del programa sin entrar en detalles de implementación.

En este caso el programa tiene una interfaz por consola, el programa toma todos sus argumentos en la invocación y no requiere de más intervención por el usuario. Por este motivo las acciones del usuario se ven reducidas a los argumentos de invocación.

En la Figura 2 se muestra de manera gráfica las interacciones posibles entre el usuario y la herramienta traductora.

C:\Users\César\Desktop\casos de uso.png

Figura 2: Diagrama casos de uso

|  |  |
| --- | --- |
| Escala de importancia | |
| Alta | El caso de uso es crucial para el funcionamiento normal de la herramienta. |
| Media | El caso de uso puede resultar útil en la mayoría del uso de la herramienta, pero puede funcionar correctamente sin él. |
| Baja | El caso de uso complementa alguna acción sin mucha importancia, pero aumenta la versatilidad de la herramienta. |

|  |  |
| --- | --- |
| Caso de Uso 1 | |
| Nombre | Ejecutar Programa |
| Descripción | El sistema deberá comportarse como se describe en el siguiente caso de uso cuando un usuario trate de ejecutar el programa. |
| Importancia | Alta |
| Precondición | Ninguna |
| Secuencia Normal | |  |  | | --- | --- | | Paso | Acción | | 1 | El usuario establece los parámetros para la traducción que será ejecutada mediante línea de comandos. | | 2 | El sistema analiza los parámetros y configura el traductor. | | 3 | El sistema abre el código fuente Perl a traducir. | | 4 | El sistema guarda el código Java generado. | |
| Postcondición | En código java ha sido generado y almacenado en el disco duro. |
| Excepciones | |  |  | | --- | --- | | Paso | Excepción | | 1 | 1-A: Si alguno de los parámetros es incorrecto, se aborta la ejecución y se muestra un mensaje de error. | | 2 | 2-A: Si no se puede abrir el script Perl a traducir, se aborta la ejecución y se muestra un mensaje de error. | | 3 | 3-A: Si no se puede guardar el código Java, se aborta la ejecución y se muestra un mensaje de error. | |

|  |  |
| --- | --- |
| Caso de Uso 2 | |
| Nombre | Abortar en error |
| Descripción | El sistema deberá comportarse como se describe en el siguiente caso de uso cuando un usuario trate de ejecutar el programa. |
| Importancia | Baja |
| Precondición | Ninguna |
| Secuencia Normal | |  |  | | --- | --- | | Paso | Acción | | 1 | El usuario establece el parámetro | | 2 | El sistema encuentra un error en el código fuente. | | 3 | El sistema aborta la ejecución y no sigue analizando. | |
| Postcondición | Ninguna |
| Excepciones | Ninguna |

|  |  |
| --- | --- |
| Caso de Uso 3 | |
| Nombre | Interpretar comentario |
| Descripción | El sistema deberá comportarse como se describe en el siguiente caso de uso cuando un usuario trate de ejecutar el programa. |
| Importancia | Media |
| Precondición | Ninguna |
| Secuencia Normal | |  |  | | --- | --- | | Paso | Acción | | 1 | El usuario establece el parámetro | | 2 | El sistema encuentra un comentario en el código fuente | | 3 | El sistema comprueba si el comentario es valido | | 4 | El sistema traduce el comentario a java | |
| Postcondición | En código java contiene los comentarios escritos en Perl. |
| Excepciones | |  |  | | --- | --- | | Paso | Excepción | | 1 | 1-A: Si el comentario está escrito en medio de una sentencia, aborta la ejecución y se muestra un mensaje de error. | |

|  |  |
| --- | --- |
| Caso de Uso 4 | |
| Nombre | Emular paréntesis |
| Descripción | El sistema deberá comportarse como se describe en el siguiente caso de uso cuando un usuario trate de ejecutar el programa. |
| Importancia | Media |
| Precondición | Ninguna |
| Secuencia Normal | |  |  | | --- | --- | | Paso | Acción | | 1 | El usuario establece el parámetro | | 2 | El sistema encuentra una función sin paréntesis en sus argumentos. | | 3 | El sistema comprueba el número de parámetros de la función. | | 4 | El sistema añade manualmente los paréntesis contando las comas. | |
| Postcondición | Ninguna |
| Excepciones | |  |  | | --- | --- | | Paso | Excepción | | 1 | 1-A: Si la función no existe, no se añaden los paréntesis y se delega el error al analizador. | | 2 | 2-A: Si la función tiene menos parámetros de los requeridos, solo se añaden los que hay y se delega el error al analizador. | |

|  |  |
| --- | --- |
| Caso de Uso 5 | |
| Nombre | Modo depuración |
| Descripción | El sistema deberá comportarse como se describe en el siguiente caso de uso cuando un usuario trate de ejecutar el programa. |
| Importancia | Baja |
| Precondición | Ninguna |
| Secuencia Normal | |  |  | | --- | --- | | Paso | Acción | | 1 | El usuario establece el parámetro | | 2 | El sistema empieza a realizar la traducción | | 3 | El sistema imprime las tareas que va realizando | |
| Postcondición | Ninguna |
| Excepciones | Ninguna |

|  |  |
| --- | --- |
| Caso de Uso 6 | |
| Nombre | Optimizar código |
| Descripción | El sistema deberá comportarse como se describe en el siguiente caso de uso cuando un usuario trate de ejecutar el programa. |
| Importancia | Alta |
| Precondición | Ninguna |
| Secuencia Normal | |  |  | | --- | --- | | Paso | Acción | | 1 | El usuario establece el parámetro | | 2 | El sistema invierte tiempo en aplicar patrones al código para generar un código más legible y optimizado. | |
| Postcondición | Ninguna |
| Excepciones | Ninguna |

|  |  |
| --- | --- |
| Caso de Uso 7 | |
| Nombre | Buscar código muerto |
| Descripción | El sistema deberá comportarse como se describe en el siguiente caso de uso cuando un usuario trate de ejecutar el programa. |
| Importancia | Media |
| Precondición | Ninguna |
| Secuencia Normal | |  |  | | --- | --- | | Paso | Acción | | 1 | El usuario establece el parámetro | | 2 | El sistema comprueba si existen sentencias inalcanzables. | |
| Postcondición | Ninguna |
| Excepciones | |  |  | | --- | --- | | Paso | Excepción | | 1 | 1-A: Si encuentra sentencias inalcanzables, se informa al usuario con un error. | |

|  |  |
| --- | --- |
| Caso de Uso 8 | |
| Nombre | Generar main |
| Descripción | El sistema deberá comportarse como se describe en el siguiente caso de uso cuando un usuario trate de ejecutar el programa. |
| Importancia | Baja |
| Precondición | Ninguna |
| Secuencia Normal | |  |  | | --- | --- | | Paso | Acción | | 1 | El usuario establece el parámetro | | 2 | El sistema crea la función de arranque main con el código global del script. | |
| Postcondición | El código Java contiene la función main. |
| Excepciones | Ninguna |

|  |  |
| --- | --- |
| Caso de Uso 9 | |
| Nombre | Cambiar ruta de salida |
| Descripción | El sistema deberá comportarse como se describe en el siguiente caso de uso cuando un usuario trate de ejecutar el programa. |
| Importancia | Media |
| Precondición | Ninguna |
| Secuencia Normal | |  |  | | --- | --- | | Paso | Acción | | 1 | El usuario establece el parámetro | | 2 | El sistema genera la traducción | | 3 | El sistema guarda los ficheros en el directorio especificado | |
| Postcondición | Los archivos generados se encuentran en el directorio especificado. |
| Excepciones | |  |  | | --- | --- | | Paso | Excepción | | 1 | 1-A: Si el directorio no existe o no se tiene acceso, aborta la ejecución y se muestra un mensaje de error. | |

# Análisis de requisitos

En las reuniones con los tutores del proyecto se fueron definiendo una serie de requisitos que el software final deberá de cumplir. En esta sección se clasificarán los requisitos según el estándar IEEE830 [7].

Los requisitos podemos clasificarlos atendiendo a dos criterios.

* Funcionales y no funcionales.
* Según su importancia:
  + Vital: Son necesario para la aceptación del proyecto.
  + Importante: El sistema puede funcionar correctamente sin ellos, pero el usuario esperara disponer de ellos.
  + Deseable: Añaden calidad al producto final.

## Requisitos funcionales

Los requisitos funcionales definen las funciones del sistema y sus componentes, a continuación, se enumeran los encontrados en este proyecto.

|  |  |
| --- | --- |
| RF-1 | |
| Nombre | Implementación de interfaz por consola |
| Descripción | La interacción con el usuario se llevará a través de una interfaz por consola, el usuario definirá los parámetros y luego ejecutará el programa. |
| Importancia | Vital |

|  |  |
| --- | --- |
| RF-2 | |
| Nombre | Definir scripts a traducir |
| Descripción | El usuario puede definir un conjunto de scripts que desea traducir a java. |
| Importancia | Vital |

|  |  |
| --- | --- |
| RF-3 | |
| Nombre | Definir carpeta de salida |
| Descripción | El usuario puede elegir una carpeta de salida distinta de la actual. |
| Importancia | Importante |

|  |  |
| --- | --- |
| RF-4 | |
| Nombre | Traducir variables básicas, array, hash y descriptores de fichero. |
| Descripción | El sistema debe poder interpretar las variables en el código, así mismo también inicializarlas y acceder a sus dimensiones. |
| Importancia | Vital |

|  |  |
| --- | --- |
| RF-5 | |
| Nombre | Traducción de operaciones |
| Descripción | El sistema deberá traducir las operaciones de tipo: Matemáticas, Lógicas, Binarias, Concatenación, Repetición y Asignación. |
| Importancia | Vital |

|  |  |
| --- | --- |
| RF-6 | |
| Nombre | Casting automático |
| Descripción | El sistema deberá convertir entre los distintos tipos de datos de forma transparente para el usuario. |
| Importancia | Vital |

|  |  |
| --- | --- |
| RF-7 | |
| Nombre | Traducción de estructuras de control |
| Descripción | El Sistema deberá traducir las siguientes estructuras de control:  if-elsif-else, while, do while, for, foreach |
| Importancia | Vital |

|  |  |
| --- | --- |
| RF-8 | |
| Nombre | Traducción funciones nativas Perl |
| Descripción | El sistema deberá traducir las siguientes funciones nativas Perl:  chomp, chop, close, defined, delete, die, each, exists, exit, join  keys, lc, lcfirst, length, open, pop, print, push, say, shirft  sort, splice, split, substr, system, uc, ucfirst, unshift, values |
| Importancia | Quedaría Bien |

|  |  |
| --- | --- |
| RF-9 | |
| Nombre | Declaración de funciones propias en Perl |
| Descripción | El usuario podrá definir funciones propias dentro del código Perl. |
| Importancia | Importante |

|  |  |
| --- | --- |
| RF-10 | |
| Nombre | Generar código Hadoop |
| Descripción | El sistema podrá convertir código secuencial definidos en funciones mapper o reducer compatibles con Hadoop. |
| Importancia | Vital |

|  |  |
| --- | --- |
| RF-11 | |
| Nombre | Implementar sistema de error |
| Descripción | El sistema deberá avisar al usuario cuando encuentre un error en el código fuente. |
| Importancia | Importante |

|  |  |
| --- | --- |
| RF-12 | |
| Nombre | Implementar sistema de depuración |
| Descripción | El sistema deberá poder informar de su funcionamiento interno para facilitar el mantenimiento de la misma. |
| Importancia | Deseable |

|  |  |
| --- | --- |
| RF-13 | |
| Nombre | Verificar código generado |
| Descripción | El sistema deberá buscar irregularidades en el código java para luego poder avisar al usuario. |
| Importancia | Deseable |

|  |  |
| --- | --- |
| RF-14 | |
| Nombre | Optimizar código generado |
| Descripción | El sistema deberá intentar optimizar el código final aun a coste del tiempo de traducción |
| Importancia | Deseable |

|  |  |
| --- | --- |
| RF-15 | |
| Nombre | Traducir comentarios Perl |
| Descripción | El sistema deberá conservar los comentarios del programador en el código fuente copiándolos en el código final. |
| Importancia | Deseable |

|  |  |
| --- | --- |
| RF-16 | |
| Nombre | Formatea código java |
| Descripción | El sistema deberá producir un código legible para el usuario, es importante que el código final este correctamente identado. |
| Importancia | Deseable |

|  |  |
| --- | --- |
| RF-17 | |
| Nombre | Generar paréntesis funciones |
| Descripción | El sistema debería permitir que las funciones especificadas sin paréntesis fueran traducidas sin la necesidad de añadírselos. |
| Importancia | Deseable |

|  |  |
| --- | --- |
| RF-18 | |
| Nombre | Generar main automático |
| Descripción | El sistema debería generar un punto de ejecución para el programa en caso de querer generar código secuencial. |
| Importancia | Deseable |

## Requisitos no funcionales

Los requisitos no funcionales definen características del sistema que pueden usarse para validar la aplicación una vez terminada. Por lo tantos se refirieren a todos los requisitos que no especifican la información a guardar ni la función a realizar, solo las características.

|  |  |
| --- | --- |
| RNF-1 | |
| Nombre | Portabilidad |
| Descripción | La aplicación deberá funcionar y comportarse del mismo modo en Windows, Linux y Mac. |
| Importancia | Importante |

|  |  |
| --- | --- |
| RNF-2 | |
| Nombre | Paradigma orientado a objetos |
| Descripción | La implementación del código fuente deberá desarrollarse con el paradigma orientado a objetos |
| Importancia | Importante |

|  |  |
| --- | --- |
| RNF-3 | |
| Nombre | Extensibilidad |
| Descripción | La aplicación deberá estar diseñada de forma que permita añadir nuevas funcionales sin afectar a las existentes. |
| Importancia | Importante |

|  |  |
| --- | --- |
| RNF-4 | |
| Nombre | Tecnología apropiada |
| Descripción | La aplicación deberá estar implementada usando la tecnología apropiada para traductores, es decir analizador léxico, sintáctico y semántico, en vez de aplicar expresiones regulares con trasformaciones directas. |
| Importancia | Vital |

# Gestión del Proyecto

En esta fase de gestión del proyecto se abordan las medidas necesarias para realizar un control sobre el proyecto durante su ciclo de vida. El principal objetivo de este apartado es cumplir con el alcance dentro de las restricciones temporales, económicas y de calidad.

## Metodología

Para evitar que el proyecto fracase, antes de transformar los requisitos es necesario la elección de una metodología que se adecue al proyecto y además permita satisfacer la planificación del mismo. La importancia de la metodología recae en dar solución a la hora de elegir la forma de estructura, planificar y gestionar el proceso de desarrollo de una aplicación concreta. Para su elección debemos analizar qué modelo de ciclo de vida se adapta mejor en nuestro caso.

El proyecto tiene un carácter individual, de modo que no existe la necesidad de coordinarse con un equipo o necesidad de reuniones entre los integrantes, lo importante es profundizar lo suficiente en el diseño o en la ingeniería de software lo necesario para aclarar lo que se va a desarrollar y el cómo sin profundizar en elementos que no aporten soluciones aplicables.

Por otro lado, el proyecto presenta una fase importante de investigación, y a pesar de que los requisitos no cambiaran durante el desarrollo del proyecto, es posible que sean necesarios cambios según se vaya profundizando en la muy permisiva sintaxis de Perl. Por este motivo es necesario un ciclo de desarrollo ágil y cíclico que permita adaptar el código a los posibles cambios.

Dadas las características anteriores se necesita una metodología que permita: flexibilidad en el desarrollo, procesos simples de terrollo y un proceso cíclico de revisión para poder aplicar correcciones en caso de ser necesario y poder cumplir con la fecha de entrega.

Por los motivos expuestos se ha seleccionado como metodología de desarrollo la Programación Extrema.

### 5.1.1. Programación extrema

La Programación extrema [9] es una metodología ágil de desarrollo de software cuyo principal objetivo es crear un software de calidad centrándose en el desarrollo de código. La filosofía se centra en un ciclo de desarrollo y prueba continuos, pudiendo así adaptarse a cambios de forma dinámica cuando cambian los requisitos o el ritmo de desarrollo es insuficiente. En este caso, como se dijo anteriormente, los requintos se mantendrán fijos durante todo el ciclo de vida, pero la sintaxis de Perl puede ocasionar que sean necesarias modificaciones para que la traducción sea válida. Por otro lado, la fecha de entrega del proyecto es inamovible y dada la complejidad y cantidad de código necesaria para implementar un traductor es importante invertir la mayor parte del tiempo posible en el desarrollo en vez de creando documentación. Este ciclo de vida permite utilizar el código como propia documentación cuando sea posible, reduciendo así la carga de los procesos de ingeniería de software.

El modelo de ciclo de vida software está pensado para equipos de trabajos pequeños o medianos y este caso no podría ser más pequeño, además del seguimiento de buenas prácticas. Otras de sus características es que se presta especial atención a la realimentación, el trabajo se divide en ciclos cortos, la validación se lleva a cabo mediante pruebas unitarias, diseños simples que permitan cumplir los requisitos de forma fácil, etc.

## Gestión de la Configuración

Se entiende por configuración en el desarrollo del software, todas las características descritas en la documentación técnica o que son alcanzadas por el producto. La gestión de configuración, tiene como objetivo establecer y mantener la integridad de todo producto creado durante el desarrollo a través de una serie de tareas. En primer lugar, hay que definir los elementos sobre los que se aplica la gestión de la configuración, luego hay que definir un proceso de cambios y por último auditorías de configuración para verificar el cumplimiento del mismo.

Un dato a tener en cuenta es que la única finalidad de la gestión de la configuración en este proyecto es mantener un control de versiones, ya que en ningún momento existirá otra persona distinta del alumno trabajando sobre el proyecto. Esto se ha tenido en cuenta la hora de realizar esta gestión de la configuración, adaptando solo las partes necesarias, aligerando el desarrollo de este proceso para evitar malgastar el tiempo.

Cabe añadir que la función del proceso de gestión de la configuración en este proyecto es meramente la de mantener un control de versiones, ya que no se realizará trabajo de forma concurrente por parte de varias personas: el alumno será el único que modifique los elementos de configuración. Esto se ha tenido en cuenta a la hora de realizar la planificación de la gestión de configuración, adaptándola a los fines que realmente tendrá, aligerando el peso de este proceso durante el desarrollo del proyecto.

### 5.2.1. Identificación de elementos de configuración

Los elementos de configuración detectados son los siguientes:

* Código fuente
* Código de pruebas
* Memoria de proyecto
* Manual de la aplicación
* Presentación del proyecto

Todos los elementos, excepto el código fuente y pruebas, deberán seguir el siguiente patrón:

<Tipo de documento>\_<AAMMDD>\_[v<versión>].<extensión>

Donde el tipo de documento pude ser:

|  |  |  |  |
| --- | --- | --- | --- |
| Tipo de documento | Descripción | Código | Extensión |
| Memoria de proyecto | Documento base con información acerca del proyecto | MEN | .pdf |
| Manual de la aplicación | Manual de usuario | MAN | .pdf |
| Presentación del proyecto | Diapositivas para la exposición del proyecto | PDP | .ptt |

Tabla 1: Tipos de documento

La fecha estará en formato AAMMDD corresponde a la fecha de modificación del elemento y la versión hace referencia a la versión interna del mismo, con cada modificación el número de versión incrementa.

Por último, la extensión hace referencia al formato que en principio siempre está presente en sistemas tipo Windows.

### 5.2.1. Herramienta gestión de la configuración

La elección de la herramienta de gestión de la configuración es un paso muy importante antes de iniciar un proyecto. La herramienta debe conservar las distintas versiones de los elementos y además permitir el trabajo de múltiples personas sobre el mismo trabajo.

Para alojar y controlar las distintas versiones del código fuente y los elementos, se ha considerado la elección de un repositorio tipo GIT. El repositorio deberá estar alojado en un servidor en la nube para asegurar la persistencia del trabajo en caso de que el tipo de trabajo se estropee y además permitir a los tutores del proyecto poder seguir el desarrollo del mismo.

Por todas estas razones se ha optado por alojarlo en la web BitBucket [9], esta web ofrece repositorios privados donde pueden trabajar 3 personas de forma gratuita, lo cual en este caso es el numero perfecto.

## Planificación temporal

En este apartado se aborda la planificación temporal comentado por un EDT (Estructura de descomposición de Trabajo) divididos en varios apartados según su especificación. El EDT es una representación jerárquica del proyecto en entregables o áreas de trabajo para determinar el ámbito del trabajo y poder planificar los objetivos propuestos.

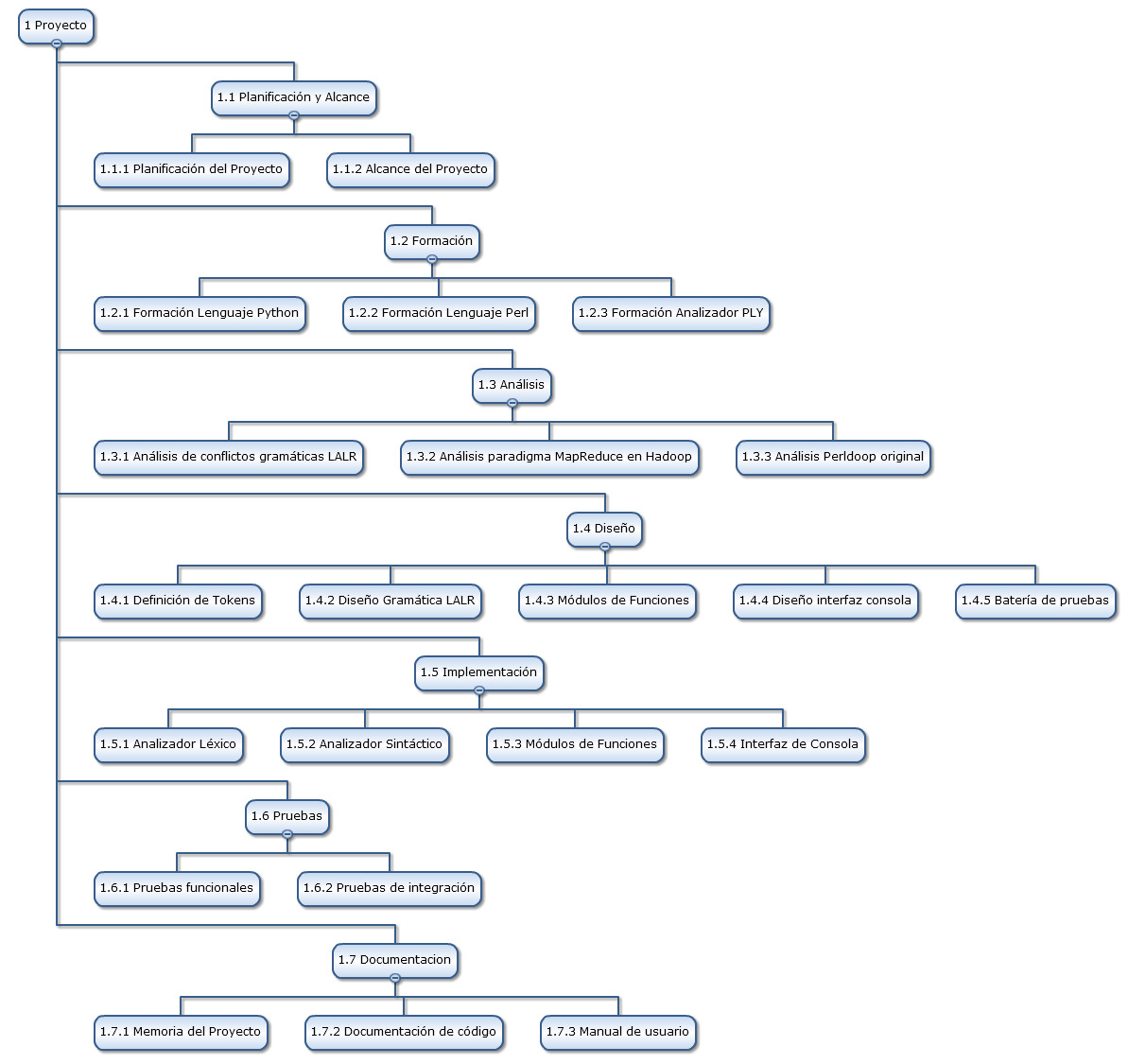
**

Figura 3: Estructura de descomposición de trabajo (EDT)

### 5.3.1. Planificación y Alcance

En esta etapa se realiza un estudio sobre el proyecto a realizar, definiendo los objetivos, requisitos, casos de uso e identificando los posibles riesgos que pueden afectar al proyecto. Esta fase es el punto de partida en la que se basan las fases posteriores.

* Tiempo estimado: Una Semana
* Rol: Analista-Programador

### 5.3.2. Formación

Para la realización del TFG es necesario el conocimiento en los lenguajes de Programación Java, Python y Perl. El primero se estudia con bastante profundidad en la carrera, pero los dos últimos requieren de una mejor formación.

Además, es necesario familiarizarse con la tecnología de construcción de traductores en Python llamada PLY [10].

* Tiempo estimado: Una Semana
* Rol: Analista-Programador

### 5.3.3. Análisis

Para poder diseñar correctamente un traductor es necesario conocer en profundidad las posibilidades de la gramática antes de comenzar con su diseño. Por otro lado, es necesario familiarizarse con el paradigma de Hadoop y analizar detenidamente la versión anterior de la herramienta para no repetir los mismos errores.

* Tiempo estimado: Una Semana
* Rol: Analista-Programador

### 5.3.4. Diseño

El proyecto requiere el diseño de 5 módulos principales:

1. **Tokens:** La primera parte del diseño consiste en definir los tokens léxicos de Perl que formaran parte de nuestro proyecto.
2. **Gramática**: La gramática es la parte crucial de este proyecto, se debe definir una gramática lo suficientemente amplia como para cumplir los objetivos del proyecto.
3. **Módulos de funciones**: Las funciones de generación de código deben diseñarse de forma que se pueda reutilizar el mayor código posible.
4. **Interfaz de consola**: La interacción con el usuario en la parte más importante de la herramienta, una mala interfaz puede perjudicar la opinión sobre la herramienta.
5. **Pruebas**: Es necesario validar la herramienta con los mínimos casos de pruebas que permitan eliminar los máximos errores posibles.

* Tiempo estimado: 1 Semana y media
* Rol: Analista-Programador

### 5.3.5. Implementación

Esta es la fase más larga en toda la planificación del proyecto. Consiste en la implementación de todo el proyecto empezando por el analizador léxico hasta la generación de código.

* Tiempo estimado: 7 Semanas
* Rol: Analista-Programador

### 5.3.6. Pruebas

Esta parte engloba el desarrollo de las pruebas especificadas en el apartado de diseño, además de las correcciones sobre el código fuente en caso de encontrarse algún error.

* Tiempo estimado: 2 Semanas
* Rol: Analista-Programador

### 5.3.7. Documentación

La última parte de la planificación que se ira realizando de manera incremental según se avanza en las demás fases, engloba la documentación del proyecto, el manual de usuario y la creación de este documento de memoria de proyecto.

* Tiempo estimado: 1 Semana y media
* Rol: Analista-Programador

### 5.3.8. Cronograma

La tabla siguiente muestra la planificación temporal del proyecto por semanas de igual forma que se hizo en el anteproyecto.

|  |  |
| --- | --- |
| Estimación temporal (en semanas) | |
| Planificación y Alcance | **1,00** |
| Planificación del Proyecto | 0,60 |
| Alcance del Proyecto | 0,40 |
| Formación | **1,00** |
| Formación Lenguaje Python | 0,45 |
| Formación Lenguaje Perl | 0,45 |
| Formación Analizador PLY | 0,10 |
| Análisis | **1,00** |
| Análisis de conflictos gramáticas LALR | 0,20 |
| Análisis paradigma MapReduce en Hadoop | 0,40 |
| Análisis Perldoop original | 0,40 |
| Diseño | **1,50** |
| Definición de Tokens | 0,30 |
| Diseño Gramática LALR | 0,70 |
| Módulos de Funciones | 0,10 |
| Diseño interfaz consola | 0,10 |
| Batería de pruebas | 0,30 |
| Implementación | **7,00** |
| Analizador Léxico | 0,60 |
| Analizador Sintáctico | 1,00 |
| Módulos de Funciones | 5,20 |
| Interfaz de Consola | 0,20 |
| Pruebas | **2,00** |
| Pruebas funcionales | 0,30 |
| Pruebas de integración | 1,70 |
| Documentación | **1,50** |
| Memoria del Proyecto | 1,10 |
| Documentación de código | 0,20 |
| Manual de usuario | 0,20 |
| Total | **15** |

Tabla 2: Cronograma

### 5.3.9. Diagrama de Gantt

El diagrama de Gantt ofrece una representación gráfica de la planificación de las actividades identificados en la Estructura de Descomposición de Trabajo. La granularidad de esta descomposición dependerá de la estimación del nivel necesario para una estimación fiable.

Como recurso asignado al proyecto, solo tenemos al alumno que trabajara todos los días durante unas 4 horas para terminar el proyecto en 15 semanas. El proyecto se desarrollará desde el lunes 28 de marzo hasta el domingo 3 de julio, un total de 105 días.

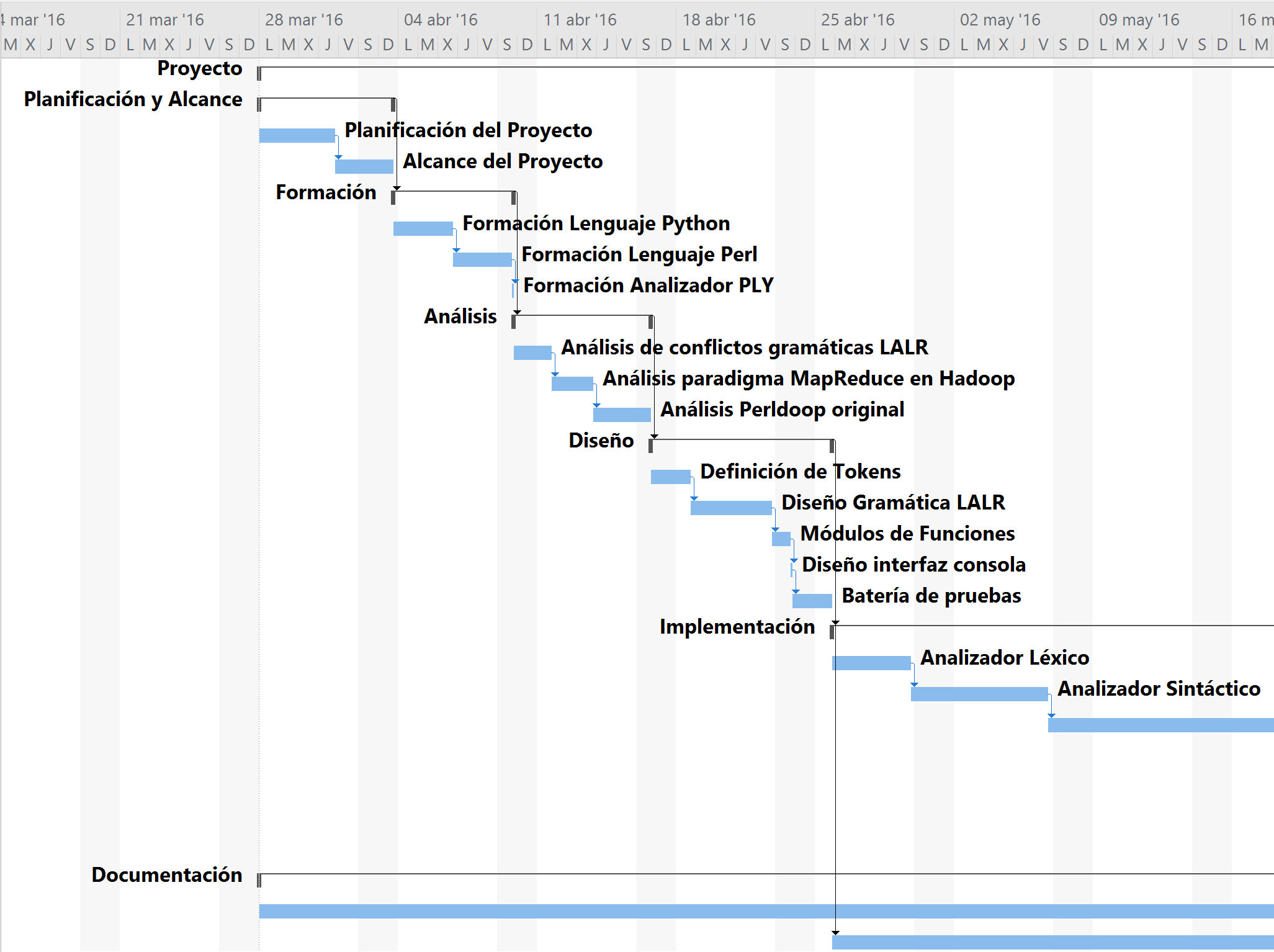


Figura 4: Diagrama Gantt parte 1

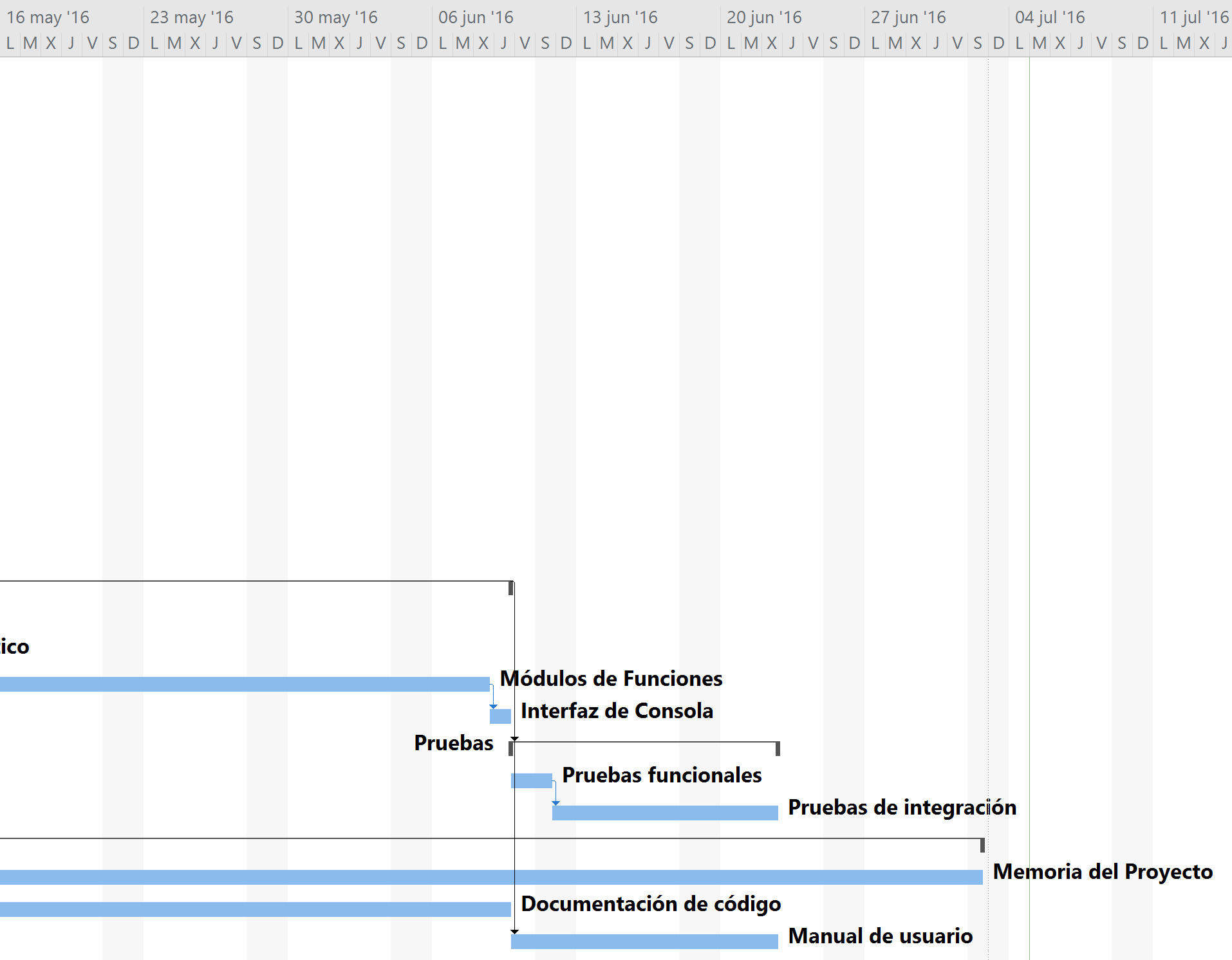


Figura 5: Diagrama Gantt parte 2

## Gestión de riesgos

Los riegos son sucesos inciertos de naturaleza positiva o negativa que pueden manifestarse durante el desarrollo del proyecto, en este caso solo identificaremos los riesgos de carácter negativo. Esta tiene como objetivo identificar, analizar y cuantificar los posibles riesgos que puedan presentarse. Una vez identificados se procederá a clasificarlos siguiendo la siguiente clasificación:

Repercusión que tendría un riesgo en caso de darse sobre las posibilidades de completar el proyecto de forma correcta en el tiempo planificado.

|  |  |
| --- | --- |
| ESCALA DE IMPACTO | |
| Alta | El impacto podría afectar gravemente a la calidad final del producto o causar un retraso importante en la planificación. |
| Media | El impacto podría afectar levemente la calidad final del producto o causar un retraso leve en la planificación. |
| Baja | El impacto podría afectar a la planificación pero no lo suficiente como para retrasar la fecha de finalización. |

Estimación de la probabilidad que tiene un riego en aparecer.

|  |  |
| --- | --- |
| ESCALA DE PROBABILIDAD | |
| Alta | Probabilidad igual o mayor que 70% |
| Media | Probabilidad entre el 30% y 70% |
| Baja | Probabilidad igual o menor que 30% |

### 5.4.1. Especificación

Se procede a la identificación y clasificación de riegos siguiendo la metodología descrita en el PMBOK [6].

**Riesgos del Proyecto**

Se definen los riesgos del proyecto a aquellos riegos que afectan a la gestión del proyecto, en caso de producirse, afectaran a la planificación temporal del proyecto y al coste del mismo.

|  |  |
| --- | --- |
| ID | RP-1 |
| Nombre | Retraso respecto a la planificación |
| Descripción | Puede darse el caso de que alguna tarea se haya planificado mal y esto casusa un retraso en la fecha de entrega. |
| Probabilidad | Media |
| Impacto | Alto |
| Plan de prevención | Realizar reuniones periódicas con los tutores de proyecto para vigilar el avance del proyecto. |
| Plan de continencia | Aumentar el número de horas diarias hasta alcanzar la planificación inicial. |

|  |  |
| --- | --- |
| ID | RP-2 |
| Nombre | Malgastar tiempo documentando |
| Descripción | El caso de cambios en el desarrollo del proyecto, puede ser necesario adaptar la documentación perdiendo tiempo de desarrollo. |
| Probabilidad | Media |
| Impacto | Medio |
| Plan de prevención | No documentar en profundidad las partes sujetas a cambios hasta asegurar su permanencia. |
| Plan de continencia | Aumentar el número de horas diarias para reparar el documento. |

|  |  |
| --- | --- |
| ID | RP-3 |
| Nombre | Enfermedad |
| Descripción | Si el alumno cae enfermo habrá un periodo de tiempo donde no se pueda seguir la planificación. |
| Probabilidad | Baja |
| Impacto | Medio |
| Plan de continencia | Restructurar la planificación para repartir las horas perdidas en el resto de días. |

**Riesgos Técnicos**

Los riegos técnicos, son aquellos que pueden amenazar la calidad final de producto y el tiempo de desarrollo del software que pueden repercutir en la planificación temporal.

|  |  |
| --- | --- |
| ID | RT-1 |
| Nombre | Diseño incorrecto de la gramática del analizador sintáctico |
| Descripción | La gramática creada en la fase de diseño contiene conflictos y no puede ser implementada. |
| Probabilidad | Media |
| Impacto | Alto |
| Plan de prevención | Diseñar la gramática directamente sobre el analizador para evitar conflictos. |
| Plan de continencia | Rediseñar la gramática hasta eliminar los conflictos. |

|  |  |
| --- | --- |
| ID | RT-2 |
| Nombre | Tiempo de depuración prolongado |
| Descripción | La depuración del traductor consume mucho más tiempo del esperado con la aparición de muchos errores. |
| Probabilidad | Alta |
| Impacto | Alto |
| Plan de prevención | Implementar las pruebas antes de la herramienta para evitar la mayoría de los fallos. |
| Plan de continencia | Invertir horas extras en pruebas hasta volver a la planificación. |

|  |  |
| --- | --- |
| ID | RT-3 |
| Nombre | Perdida del trabajo de un día |
| Descripción | Un fallo en el sistema evita el acceso a los datos almacenados en el ordenador. |
| Probabilidad | Baja |
| Impacto | Alto |
| Plan de prevención | Comenzar a trabajar con el repositorio y actualizarlo al menos una vez al día. |
| Plan de continencia | Repetir el trabajo lo antes posible para recordar exactamente lo que se ha perdido. |

|  |  |
| --- | --- |
| ID | RT-4 |
| Nombre | Requisito imposible de implementar |
| Descripción | Un requisito especificado para el proyecto es imposible de Implementar. |
| Probabilidad | Bajo |
| Impacto | Alto |
| Plan de prevención | Diseño preliminar de una aproximación en la implantación para cada requisito del sistema. |
| Plan de continencia | Reunión con los tutores del proyecto para buscar solución al problema. |

|  |  |
| --- | --- |
| ID | RT-5 |
| Nombre | Traducciones incorrectas |
| Descripción | Las traducciones de la herramienta no son baldías tanto por no poder ser compiladas como por no hacer lo mismo que el código fuente. |
| Probabilidad | Medio |
| Impacto | Alto |
| Plan de prevención | Probar cada traducción segunda se va implementando de forma unitario y en conjunto con el resto. |
| Plan de continencia | Esperar a la fase de pruebas para corregir los errores de traducción. |

|  |  |
| --- | --- |
| ID | RT-6 |
| Nombre | Errores sintácticos en Python |
| Descripción | Los lenguajes interpretados pueden contener errores sintácticos en líneas que no son ejecutadas. |
| Probabilidad | Alta |
| Impacto | Bajo |
| Plan de prevención | Verificar que todas las líneas son interpretadas. |
| Plan de continencia | Corregir los errores en cuanto aparezcan mientras se continua la implementación o con el plan e pruebas. |

|  |  |
| --- | --- |
| ID | RT-7 |
| Nombre | Interfaz de consola poco usable |
| Descripción | La interfaz de consola es poco intuitiva y es difícil familiarizarse con ella |
| Probabilidad | Baja |
| Impacto | Alto |
| Plan de prevención | Pedir opinión a otras personas acerca de la interfaz y realizar cambios con sus sugerencias. |
| Plan de continencia | Añadir un manual para mitigar una interfaz complicada. |

## Análisis de costes

Una vez terminada la planificación temporal del proyecto, es necesario estimar los costes que acarrearía el proyecto en una situación real. En esta fase se incluye el coste humano (el alumno) así como las herramientas y recursos necesario para la creación del proyecto. Todos los programas utilizados durante el desarrollo están disponibles de manera gratuita en internet y pueden descargarse desde su página oficial. El único material usado es un ordenador portátil propiedad del alumno con un coste de 1.800€ comprado al inicio del curso 2015-2016, con lo cual el coste total del proyecto viene dado por las horas invertidas por el alumno en la realización del mismo más el precio del portátil.

Para realizar el cálculo de la mano de obra se han consultado varias webs como InfoJobs [11] o la calculadora de contratos de la USC para buscar el sueldo medio de un ingeniero informático en proyectos de este tipo. Se ha estipulado un sueldo aproximado de 17,2€ la hora.

En la siguiente tabla se muestra una división de costes por cada tarea del proyecto.

|  |  |
| --- | --- |
| **Tarea** | Coste |
| **Planificación y Alcance** | **460,10 €** |
| Planificación del Proyecto | 276,06 € |
| Alcance del Proyecto | 184,04 € |
| **Formación** | **460,10 €** |
| Formación Lenguaje Python | 207,05 € |
| Formación Lenguaje Perl | 207,05 € |
| Formación Analizador PLY | 46,01 € |
| **Análisis** | **460,10 €** |
| Análisis de conflictos gramáticas LALR | 92,02 € |
| Análisis paradigma MapReduce en Hadoop | 184,04 € |
| Análisis Perldoop original | 184,04 € |
| **Diseño** | **690,15 €** |
| Definición de Tokens | 138,03 € |
| Diseño Gramática LALR | 322,07 € |
| Módulos de Funciones | 46,01 € |
| Diseño interfaz consola | 46,01 € |
| Batería de pruebas | 138,03 € |
| **Implementación** | **3.220,70 €** |
| Analizador Léxico | 276,06 € |
| Analizador Sintáctico | 460,10 € |
| Módulos de Funciones | 2.392,52 € |
| Interfaz de Consola | 92,02 € |
| **Pruebas** | **920,20 €** |
| Pruebas funcionales | 138,03 € |
| Pruebas de integración | 782,17 € |
| **Documentacion** | **690,15 €** |
| Memoria del Proyecto | 506,11 € |
| Documentación de código | 92,02 € |
| Manual de usuario | 92,02 € |
| **Total** | **6.901,50 €** |

En resumen, si tenemos en cuenta todas las áreas:

|  |  |
| --- | --- |
| Fuente | Coste |
| Estación de trabajo | 1.800,0€ |
| Software utilizado | 0,0€ |
| Repositorio web | 0,0€ |
| Mano de obra | 6.901,50 € |
| Total | **8.701,5€** |

# Herramientas y Tecnologías

En este apartado se describirán las diferentes herramientas y tecnologías utilizadas para la creación de este proyecto, entendemos por herramienta a cualquier aplicación usada para crear algún elemento del proyecto y las tecnologías todo aquello en que se apoya el proyecto para funcionar.

## IDEs de desarrollo

* **Eclipse** [12]: Se ha utilizado este IDE para el desarrollo del código fuente del traductor, eclipse tiene un plugin muy completo para el desarrollo de Python3, al contrario que Netbeans. Todo el código fuente y la depuración del mismo fueron realizados con él.
* **Netbeans** [13]: Se ha utilizado este IDE para el desarrollo de una librería auxiliar java que simplifica el código de traducción. La razón de no haber usado Eclipse es la costumbre de haber realizado muchos proyectos Java en él y conocer muchos atajos rápidos para un desarrollo lo mas rápido posible. En caso de que existiese un plugin python3 para este IDE, todo el desarrollo se hubiera realizado con él.

## Tecnologías

* **Python**: Se ha optado la realización de traductor en este lenguaje de programación por ser un lenguaje de alto nivel, rápido de programar y que ofrece portabilidad entre distintas plataformas. El proceso de traducción requiere concatenar y dividir constantemente arrays y cadenas, lo cual se puede hacer con un simple operador. Otros lenguajes como Java o C no tienen soporte tan directo y se necesitaría más tiempo de desarrollo.
* **PLY** [10]: Entre los distintos analizadores léxicos y sintácticos disponibles para Python, se ha optado por PLY debido a su simplicidad y a su integración directa en el código Python sin necesidad de invocar a un preprocesador que transforme su código a Python. El analizador sintáctico de PLY es ascendente y permite definir gramáticas LALR(1) lo cual es suficiente para cumplir los objetivos del proyecto.
* **Perl**: En el desarrollo de un traductor Perl-Java es imprescindible contar con el intérprete Perl para probar ficheros de ejemplo y verificar que tanto ellos como sus tradiciones ofrecen las mismas salidas una vez ejecutados.

## Aplicaciones

* **Notepad++**: Aunque para el desarrollo de código se han usado IDE, los scripts Perl y el fichero que contiene la definición de la gramática se han creado con este editor de texto. Por otro lado, el editor contiene funciones muy útiles como la de grabar y repetir acciones que ayuda a replicar y maquetar código de formas que el IDE no soporta.
* **SourceTree** [14]: Como ya se ha especificado en el apartado de gestión de la configuración, para el control de versiones del código fuente se ha optado por un repositorio GIT, alojado en la página BitBucket. Para el acceso y las gestiones se han considerado diversos clientes gráficos, pero al final se ha optado por SourceTree por tener una interfaz muy completa y contar con el apoyo de muchos usuarios. La herramienta permite realizar todas las tareas necesarias para el proyecto, permite realizar commit de forma rápida, ver el historial de commits en forma de árbol (o línea en nuestro caso) , comparar diferentes versiones del mismo fichero y volver atrás seleccionando el nodo en el árbol de commits.
* **Microsoft Office**: Se ha utilizado la suite de ofimática para la creación de la memoria del proyecto, más concretamente las versiones Word, Excel y Project. Se ha optado elegir esta versión frente a las versiones libres por ser mucho más completas en cuanto a opciones y además ya estaban instaladas en el sistema debido a su uso en las prácticas de la carrera. Aunque se trata de un programa de pago, esto no implica ningún coste debido a que se amortizaron durante toda la carrera y la versión Project se usó la versión de demostración suficiente para este proyecto.
* **Draw.io** [15]: Para crear todos los diagramas de la memoria del proyecto se ha optado por esta herramienta online que permite crear todo tipo de diagramas y además almacenarlos en la nube.
* **WBS TooL**: Esta herramienta web se ha usado para la creación de la Estructura de descomposición de trabajo, requerido como porte importante de la gestión del proyecto. Se ha optado por usar esta herramienta por su familiarización en las asignaturas de Ingeniería del Software y gestión de proyecto. La herramienta ofrece todas las funciones necesarias para generar un EDt de forma gráficas y luego permite exportar como xml para poder usarlo como base en la construcción de un diagrama de Gantt.
* **Microsoft Project**: La gestión del proyecto requiere de la construcción de un diagrama Gantt con la planificación estipulada por el proyecto que abarque las horas de un trabajo de fin de grado. Se ha optado por la versión de Microsoft frente a Project Libre debido a que al contrario que en las prácticas de las asignaturas, no podemos tolerar fallos que suelen ocurrir con la versión libre tales como error en los cálculos después de guardar y volver a cargar un proyecto. Además, el proyecto tiene un horario distinto del laboral donde se requiere definir un calendario nuevo, esta característica solo está disponible en la versión Microsoft porque la versión libre contiene un bug que ignora los calendarios.

# Arquitectura del software

En este apartado se discute acerca de la arquitectura en la que se ha basado la construcción del software, identificando cada uno de los componentes que lo forman, así como una breve descripción del papel de cada uno.

Los componentes son: Una interfaz de consola, el analizador léxico, el analizador sintáctico, el comprobar y generador de código, el sistema de mensajes, el sistema de gestión de errores y el propio sistema de ficharos del sistema operativo. La disposición entre ellos puede verse en el diagrama de la Figura 6.

D:\Users\Informatica\cuarto\Proyecto\arquitectura.png

Figura 6: Arquitectura del sistema

## Interfaz de línea comandos

Para la interacción con el usuario se ha optado por la utilización de una interfaz de consola, este tipo de interfaz es la más habitual que podemos encontrar en traductores y compiladores. La línea de comandos es la interfaz que permite más flexibilidad en su uso, además en caso de querer automatizar las tareas, es muy fácil invocar a la aplicación dentro de un script, lo cual sería imposible con una interfaz gráfica.

La idea principal de interacción con el usuario es que una vez se haya descargado la aplicación, la añada al path y luego pueda utilizarla simplemente invocando Perldoop desde la consola.

En la Figura 7 podemos ver un ejemplo de la invocación de la aplicación, en este caso se ha optado por el parámetro –h, que despliega la documentación de aplicación mostrando los parámetros existentes y una breve descripción de lo que hacen.

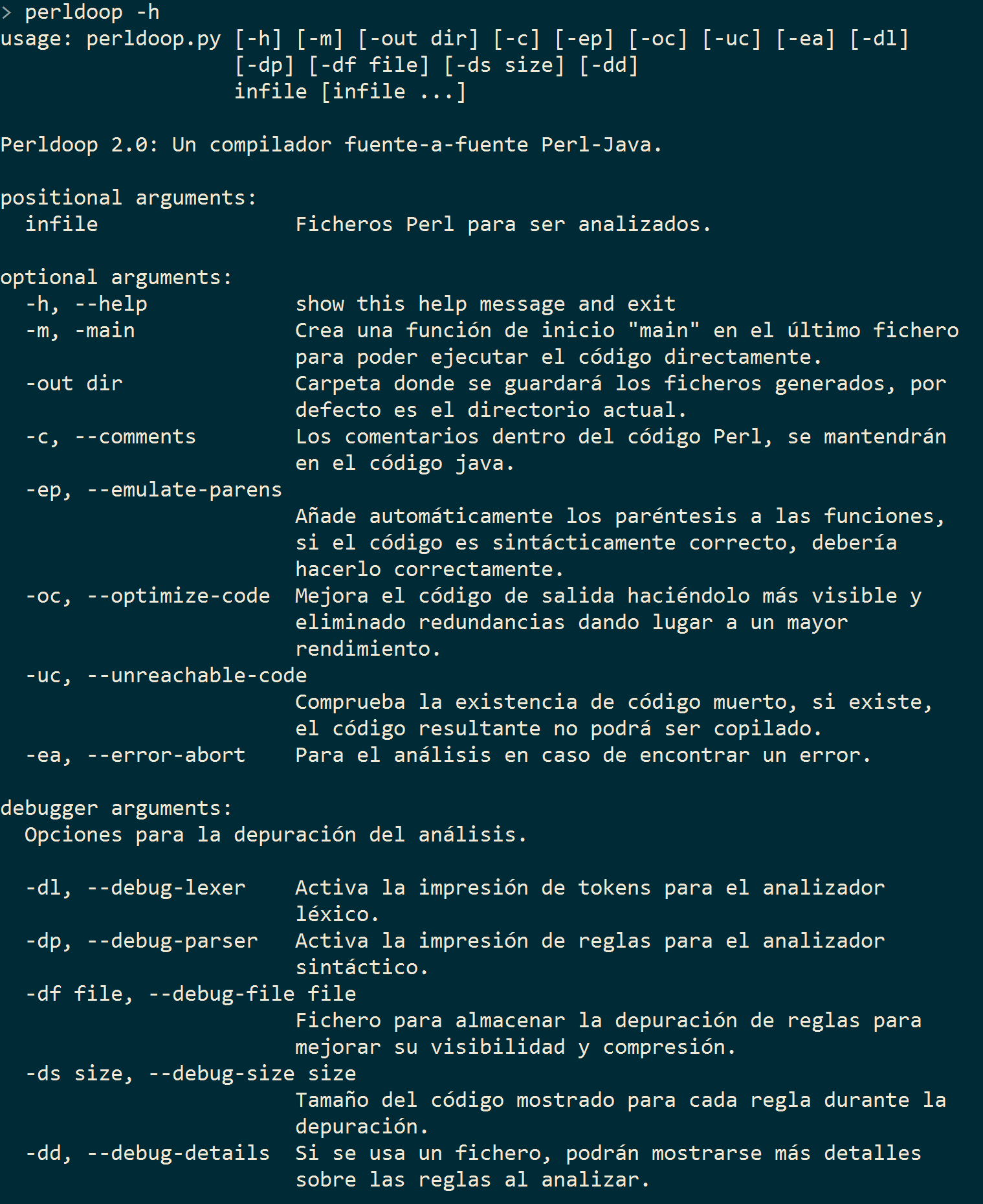


Figura 7: Ayuda Perldoop 2

## Analizador Léxico

El analizador léxico es la primera fase en un proceso de compilación o traducción. El analizador léxico recibe como entrada la secuencia de caracteres del código fuente y produce como salida una lista de tokens. Los tokens son un conjunto de caracteres que tienen un significado coherente dentro del lenguaje de programación en cuestión, por ejemplo ‘=’, ‘&&’,’while’ son tokens comunes que podemos encontrar.

El analizador léxico trabaja bajo demanda del analizador sintáctico devolviéndole un componente léxico según el analizador sintáctico lo va necesitando para avanzar en la gramática.

En este caso, el analizador sintáctico será el encargado de crear y pasarle el código fuente al analizador léxico.

## Analizador Sintáctico

El analizador sintáctico es el encargado de comprobar si una cadena de tokens (componentes léxicos) puede ser generada por una determinada gramática. En este caso se ha intentado definir una gramática lo suficientemente amplia para analizar la mayoría de los scripts usados en el CITIUS en el campo de procesamiento de lenguaje natural.

El ritmo del análisis lo lleva el analizador sintáctico, él es quien llama al analizador léxico y cuando tiene suficiente información llama al generador de código para ir creando la traducción. Una vez que el análisis ha terminado, la traducción ya ha sido generada completamente y será retornada.

Normalmente los analizadores sintácticos son los encargados de generar un árbol sintáctico para luego verificar el código fuente. Para simplificar este proyecto y adaptarlo al tiempo requerido por un trabajo de fin de grado, se optado por fusionar el analizador sintáctico con el semántico y el generador de código, da como resultado un traductor con menos código y más rápido de implementar.

## Comprobación y generación de código

La comprobación y la generación de código, está formado por un conjunto de funciones agrupadas en módulos que son invocadas por el analizador sintáctico. La tarea de estas funciones es validar los parámetros recibidos del analizador sintáctico y si son correctos, generar una cadena con la traducción Java correspondiente.

Las funciones reciben como parámetro todos los datos necesarios para realizar su tarea como, por ejemplo, la tabla de símbolos. La generación de código se basa en una simple concatenación de los argumentos y en algunos casos la conversión de un tipo de dato a otro.

## Sistema de gestión de errores

Cada vez que una fase del traductor falla, se invoca al sistema de error para informar al usuario de su naturaleza y donde se cometió. Los primeros errores que pueden ocurrir están a cargo de la interfaz de consola, estos corresponden a problemas con el sistema de archivos: fichero no existe, sin permiso de lectura, error de lectura, etc.

Lo primero que hace el sistema de error es invocar al sistema de mensaje para obtener el texto que se le mostrara al usuario, luego calculará la línea y la columna usando la posición del carácter y para terminar obtiene un fragmento del código fuente para mostrárselo al usuario. En la Figura 8 podemos ver un ejemplo de varios errores de diferente tipo en el código fuente.

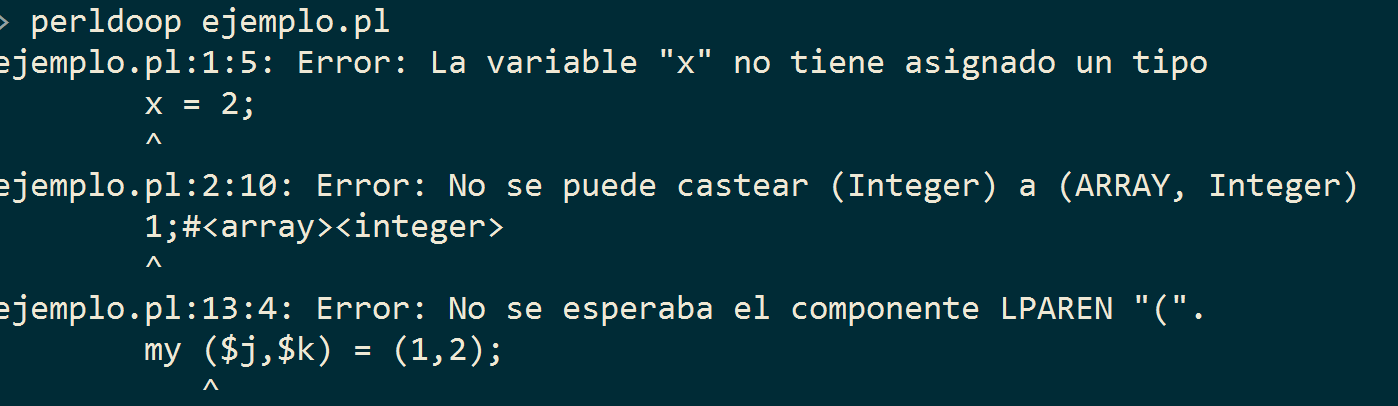


Figura 8: Ejemplo sistema de errores

## Sistema de mensajes

El sistema de mensajes contiene todos los mensajes que son mostrados al usuario en algún momento, estos abarcan desde la ayuda de la interfaz de consola hasta los errores. Gracias a este sistema los mensajes no están mezclados con el código fuente y es fácil cambiarlos y adaptarlos según va evolucionando la aplicación.

Los mensajes están almacenados en una tabla en formato clave valor, donde la clave es un código usado para referenciar al mensaje dentro del código fuente. A su vez la tabla está asociada a un idioma dentro de una tabla de mayor nivel, de esta forma pueden crearse muchas tablas distintas en distintos idiomas. La elección del idioma debe cambiarse de manera manual en este fichero editando la constante LANGUAGE.

# Diseño e Implementación

Una vez definida la arquitectura del sistema, es el momento de detallar el diseño a seguir durante su implementación. En este apartado se explica en detalle el funcionamiento y las distintas clases del sistema, así como interactúan entre ellas.

## Diagrama de clases



Figura 9: Clases con sus atributos y métodos

La Figura 9 muestra todas las clases del sistema, en ella podemos ver todos los atributos y métodos de los que está compuesta cada una, pero no sus relaciones. La mayoría de las clases están compuestas por muchos métodos simples lo cual hace que su representación en el diagrama sea muy grande, por esa razón en primer lugar se muestra la composición de las clases y luego en la Figura 10 se muestran las relaciones sin su contenido.

Siguiendo la filosofía de la programación extrema, el código fuente también se considera documentación y en caso de ser posible es mejor documentar el propio código a generar documentación externa. Por esa razón, todos los métodos y atributos mostrados en la Figura 9 están descritos en profundidad en el propio código y pueden ser consultados en el fichero .py correspondiente.

C:\Users\César\Desktop\diagramaclases.png

Figura 10: Diagrama de clases

Para simplificar el diagrama se ha agrupado las clases que contienen solo métodos estáticos y que nunca son instanciadas. A continuación, se hará un breve resumen de la finalidad y uso de cada clase:

* **Perldoop:** Clase que inicia todo el sistema, es la encargada de interpretar los argumentos y crear un analizador sintáctico por cada fichero de entrada.
* **Options:** Contiene las opciones pasadas como argumento a la aplicación y que el analizador sintáctico heredará. De esta forma tenemos un fichero que contiene las opciones por defecto y que un usuario podría ajustar sin perjudicar el resto de variables del Parser.
* **Parser:** Analizador sintáctico, tiene definida toda la gramática en su interior y las llamadas a las funciones de comprobación y generación.
* **Lexer:** Analizador léxico, tiene definidos los tokens y las palabras reservadas.
* **Declare:** Almacena la pre declaración de tipo de una variable hasta que esta es instanciada. La clase desaparece en cuanto la variable es declarada.
* **Messages:** Almacena todos los mensajes que se muestran en la aplicación y es la encargada de imprimir los errores del sistema.
* **Code:** Almacena el código que es pasado en la reducción del analizador sintáctico, contiene todos los datos necesarios para decidir cuando existen varias traducciones posibles.
* **Access:** Almacena los accesos a variables hasta que la variable es usada, esto es necesario porque en caso de un HashMap no sabemos si habrá que realizar un get o un put hasta el último momento.
* **Position:** Guarda los datos relacionados con la posición dentro del fichero, estos datos son usados en caso de error para mostrarle fila y columna al usuario junto con una porción de código.
* **Variable:** Almacena una variable declarada, contiene los datos de tipo, nombre y acceso en el caso de que el fichero fuera marcado como paquete.
* **Funcion:** Contiene los datos de las funciones declaradas por el usuario dentro del código, aquí se almacenan los tipos de los argumentos y de sus retornos.
* **Package:** Una vez terminado el análisis sintáctico de un fichero, si está marcado como paquete, se almacenarán aquí los atributos y funciones para que puedan ser invocados en otro fichero.
* **Casting:** Contiene las funciones que permiten cambiar de tipo las expresiones en el código, los cambios se realizan mediante un casting directo o la invocación a las funciones Java correspondientes.
* **Auxiliary:** Funciones varias que suelen ser usadas por todos los módulos, entre ellas se encuentran la identación de código o la creación de los imports.
* **Variables:** Tiene las funciones de gestión de variables de la tabla de símbolos y la generación de variables auxiliares reservadas para conversiones indirectas del código.
* **DataTypes:** Contiene las constantes de todos los datos y tipos, se usa una clase para no escribir las constantes como cadenas en las que podemos cometer errores.
* **Statements:** Genera el código de las sentencias Perl, declara las variables y valida si el código es necesario que sea traducido. Por ejemplo, el código “1+1;” al no almacenar el resultado no generara ninguna traducción.
* **Functions:** Almacena la gramática de las funciones nativas de Perl que el analizador sintáctico heredará. Las funciones de Perl tienen una sintaxis que entra en conflicto con el tipado, por eso deben ser definidas y tratadas aparte.
* **Blocks:** Genera el código de todos los bloques de código, desde los bloques de control a la clase que contiene toda la traducción.
* **Operations:** Genera el código de todas las operaciones que pueden realizarse con expresiones, ya sean matemáticas, lógicas, comparaciones, etc.
* **Hadoop:** Contiene la gramática con las transformaciones de código para traducir un código secuencias por la estructura MapReduce paralela que usa Hadoop. Se realiza en una clase aparte para no condicionar el resto de la herramienta y permitir nuevas transformaciones en el futuro.
* **Collections:** Contiene el código para gestiones las colecciones, aquí es donde se inicializan y se verifica si los accesos son correctos en tipo.

## Diagramas de interacción

Los diagramas de interacción representan como se relaciona un usuario con el sistema y además como los objetos intervienen entre sí para realizar una función. En un traductor, las interacciones pueden ser muy largas y dependen mucho del código que estamos analizando así pues a continuación se mostrará el diagrama de interacción de un código simple que permita extrapolar el funcionamiento del sistema.

El código elegido será “my $x=2; #<string>” que declara una variable de tipo cadena y almacena en ella el valor numérico 2, para el cual tendrá que realizar una transformación de tipo.

C:\Users\César\Desktop\secuencia1.png

Figura 11: Diagrama de secuencia de un análisis

Como podemos ver en la Figura 11, un fragmento tan pequeño de código, ha creado un diagrama de secuencia considerablemente grande. El sistema sigue el mismo patrón para todos los tipos de código, primero va leyendo tokens y cuando tiene los suficientes datos para realizar una reducción, realiza una llamada al módulo de generación y comprobación de código y luego reduce. En la secuencia 27, vemos el comportamiento ante el final de sentencia, donde sus variables son declaradas y termina su procesado, todas las sentencias son luego concatenadas una vez que se crear el bloque que las contiene. En este caso al no existir ningún bloque son englobadas dentro de la clase en un método reservado para ello.

Un detalle es que una vez terminada la traducción y esta es retornada por la función parser, el main antes de escribirla en el fichero, procede a la identacion de código para hacerlo más visible al usuario. El usuario no es notificado de ninguna manera durante el proceso de este análisis, igual que otros compiladores como ‘gcc’, sólo se informará al usuario en caso de encontrar un error y si el usuario no recibe ningún mensaje quiere decir que la traducción se ha generado correctamente.

En el siguiente diagrama se muestra un fragmento de una iteración donde sucede un error porque se intenta leer una variable que no existe:

C:\Users\César\Desktop\secuencia2.png

Figura 12: Error en el análisis

En la Figura 12 podemos el proceso seguido en caso de entrar un error. En primer lugar, debemos notificar al usuario que el código fuente contiene un error y luego avisar al analizador de que el código contiene errores. Una vez hecho esto, es necesario subsanar el error para poder continuar el análisis, en este caso crear la variable. El código Java ya no será impreso en un fichero al terminar, pero si queremos seguir el código en busca de más errores es necesario subsanarlo o puede producir una cascada de errores. Un ejemplo de esto es, una variable que no existe, nunca fue asignada, con lo cual si se intenta leer saldrán ambos errores si el primero no crea una variable libre de otros errores. El resto de errores siguen la misma filosofía y siempre se intenta que un mismo error nunca sea notificado de dos formas distintas.

## Diseño tokens del analizador léxico

En este apartado definiremos los tokens que pueden ser reconocidos en el código fuente y el sistema para identificarlos. Según su origen podemos dividir los tokens en tres conjuntos diferentes: Básicos, Palabras reservadas y Etiquetas.

### 8.3.1. Tokens Básicos

Los tokens básicos no todos tienen un valor fijo y son reconocidos por expresiones regulares en la mayoría de los casos. En la Tabla 3 podemos ver todos los tokens definidos, la expresión regular que los reconoce y una pequeña descripción aclaratoria.

En algunos lenguajes de programación se puede definir los caracteres como tokens para simplificar su definición de forma que ‘+’ usa su valor ASCII como token. En Python al ser un lenguaje de muy alto nivel, es necesario definir una constante para todos los tokens.

|  |  |  |
| --- | --- | --- |
| Id | Expresión regular | Descripción |
| ID | [A-Za-z\_][\w\_]\* | Identificadores |
| DOLLAR | \$ | Escalar |
| AT | @ | Array |
| PERCENTAGE | % | HashMap/Modulo |
| INT\_NUMBER | (([1-9]\d\*)|  (o[xX][a-fA-F\d]+)|  (o[bB][01]+)|  (0[0-7]\*)) | Todos los números enteros en formato: Decimal, Hexadecimal, Octal y Binario |
| FLOAT | ((\d\*\.\d+)|(\d+\.\d\*)) | Números decimales |
| FLOAT\_NUMBER | (INT\_NUMBER | FLOAT)  [eE](\+-)?INT\_NUMBER | Números en notación científica. |
| STRING\_QUOTE | '([^'\n]|(\.))\*' | Cadenas simples |
| STRING\_DOUBLE\_QUOTE | “([^”\n]|(\.))\*” | Cadenas dobles |
| CMD | `.+` | Cadenas ejecutables |
| PLUS | \+ | Suma |
| MINUS | - | Resta |
| TIMES | \\* | Multiplicación |
| DIVIDE | / | División |
| POW | \\*\\* | Elevado |
| OR | | | Or binario |
| AND | & | And binario |
| NOT | ~ | Not binario |
| XOR | \^ | Xor binario |
| LSHIFT | << | Desplazar bit izquierda |
| RSHIFT | >> | Desplazar bit derecha |
| LOR | || | OR lógico |
| LAND | && | And lógico |
| LNOT | ¡ | Not lógico |
| NUM\_LT | < | Menor que |
| NUM\_GT | > | Mayor que |
| NUM\_LE | <= | Menor/Igual que |
| NUM\_GE | >= | Mayor/Igual que |
| NUM\_EQ | == | Igual que |
| NUM\_NE | ¡= | No igual que |
| STR\_REX | =~ | Reconocido por regex |
| STR\_NO\_REX | !~ | No reconocido por regex |
| SMART\_EQ | ~~ | Comparador Inteligente |
| CMP\_NUM | <=> | Comparador números |
| EQUALS | = | Asignación |
| TIMESEQUAL | \\*= | Asignar multiplicación |
| DIVEQUAL | /= | Asignar división |
| MODEQUAL | %= | Asignar modulo |
| POWEQUAL | \\*\\*= | Asignar elevado |
| PLUSEQUAL | \+= | Asignar suma |
| MINUSEQUAL | -= | Asignar resta |
| LSHIFTEQUAL | <<= | Asignar desp izquierda |
| RSHIFTEQUAL | >>= | Asignar desp derecha |
| ANDEQUAL | &= | Asignar And binario |
| OREQUAL | |= | Asignar Or binario |
| XOREQUAL | \^= | Asignar Xor binario |
| LANDEQUAL | &&= | Asignar And lógico |
| LOREQUAL | ||= | Asignar Or lógico |
| PERIODEQUAL | \.= | Asignar Concatenación |
| XEQUAL | x= | Asignar Repetición |
| PLUSPLUS | \+\+ | Incrementar |
| MINUSMINUS | -- | Decremento |
| LPAREN | \( | Paréntesis derecho |
| RPAREN | \) | Paréntesis izquierdo |
| LBRACKET | \[ | Corchete derecho |
| RBRACKET | \] | Corchete izquierdo |
| COMMA | (, | =>) | Los dos tipos de comas |
| PERIOD | \. | Concatenación |
| TWO\_PERIOD | \.\. | Generador de rango |
| QUEST\_CLOSE | \? | Operador ternario |
| SEMI | ; | Punto y coma |
| COLON | : | Else del Operador ternario |
| TWO\_COLON | :: | Acceso a paquete |
| LBRACE | \{ | Llave izquierda |
| RBRACE | \} | Llave derecha |
| BACKSLASH | \\ | Barra invertida/Escape |
| POINTED | -> | Acceso a puntero |
| REGEX\_MOD | (i|s|m|x|o|p|d|a|  u|l|g|cg|e)? | Modificador expresiones regulares |
| M\_REGEX | m/.\*/ REGEX\_MOD | Regex tipo match |
| S\_REGEX | s/.\*/.\*/REGEX\_MOD | Tegex tipo sustitución |
| Y\_REGEX | (y|tr)/.\*/.\*/REGEX\_MOD | Regex tipo transposición |
| STDIN | <STDIN> | Entrada estándar |
| COMMENT | \# | Inicio comentario |
| COMMENT\_SPECIAL | \#!.\* | Comentarios especiales |
| COMMENT\_IGNORE\_LINE | .\*\#<ignore-line> | Ignorar la línea |
| COMMENT\_IGNORE\_BLOCK | \#<ignore-block>\n  (.|\n)  \*\#<ignore-block> | Ignorar bloque de código |
| COMMENT\_CODE\_IMPORT | \#<java-import>.\* | Añadir import java |
| COMMENT\_CODE\_LINE | \#<java-line>.\* | Añadir sentencia java |

Tabla 3: Tokens básicos

### 8.3.2. Tokens de palabras reservadas

Las palabras reservadas son identificadoras que el lenguaje de programación Perl tiene reservado para un uso especial. Para diferenciarlos, cada vez que el analizador encuentra un identificador, consulta un hashMap que contiene todas las palabras reservadas y le asigna el tipo de la misma. En Perl todas las variables son precedidas por $, @, % por lo tanto nunca son variables reservadas. Por otro lado, las funciones no tienen ningún símbolo precedido con lo cual son las que deben diferenciarse de las palabras reservadas.

La Tabla 4 contiene todas las palabras reservadas definidas en el analizador, el valor de la palabra reservada corresponde al id en minúsculas a no ser que se especifique lo contrario.

|  |  |
| --- | --- |
| Id | Descripción |
| MY | Crea variables |
| SUB | Define métodos |
| OUR | Crea variables accesibles en paquete |
| PACKAGE | Crea un paquete |
| WHILE | Bucle while |
| DO | Bucle do while |
| FOR (for,foreach) | Bucle for y for each, por convención de Perl se usan indistintamente. |
| UNTIL | Condicional if negada |
| IF | Condicional if |
| ELSIF | Contracción de else if |
| ELSE | Bloque else |
| UNLESS | Bucle while negado |
| LAST | Rompe el bucle |
| NEXT | Sala a la próxima iteración del bucle |
| CMP | Compara dos cadenas |
| X | Repite una cadena un número de veces |
| UNDEF | Valor indefinido, similar a null java |
| LLAND (and) | And lógico de baja precedencia |
| LLOR (or) | Or lógico de baja precedencia |
| LLXOR (xor) | Xor lógico de baja precedencia |
| LLNOT (not) | Not lógico de baja precedencia |
| STR\_LT (lt) | Menor que para cadenas |
| STR\_GT (gt) | Mayor que para cadenas |
| STR\_LE (le) | Menor/Igual que para cadenas |
| STR\_GE (ge) | Mayor/igual que para cadenas |
| STR\_EQ (eq) | Igual que para cadenas |
| RSTR\_NE (ne) | No igual que para cadenas |

Tabla 4: Tokens de palabras reservadas

A parte de las palabras reservadas a las que nos tienen acostumbrados todos los lenguajes de programación, Perl nos añade operadores de baja precedencia (para evitar entre paréntesis sus argumentos) y operadores de comparación de cadenas.

### 8.3.3. Tokens de Etiquetas

Las etiquetas son el punto clave para el funcionamiento de Perldoop, con ellas podemos traducir un lenguaje sin tipo a un lenguaje con tipo y realizar diversas transformaciones en el código.

Las etiquetas se definen dentro de comentarios para no afectar al correcto funcionamiento del script con su interprete nativo. Las etiquetas se definen como caracteres alfanuméricos encerrados entre ‘<’ y ‘>’. Como vimos en el apartado de tokens básicos, el comentario esta definido solo por ‘#’, al reconocer este carácter el analizador léxico entra en un estado para reconocer etiquetas. Si se encuentra un elemento con sintaxis de etiqueta, se procederá a comprobar si existe, en caso afirmativo se devolverá como token, en caso contrario se añadirá al comentario.

En la Tabla 5 podemos ver los patrones de las etiquetas existentes:

|  |  |  |
| --- | --- | --- |
| Id | Expresión regular | Descripción |
| LABEL\_ID | < [A-Za-z\_][\w\_]\*> | Identificador de etiquetas |
| LABEL\_SIZE | <[1-9][0-9]\*> | Etiquetas de tamaño |
| LABEL\_VAR | <(\$|@|%) [A-Za-z\_][\w\_]\*> | Etiquetas de variables |

Tabla 5: Exprsiones regualres para etiquetas

Las etiquetas de tamaño y variables sirven para hacer referencia a variables en predeclaracion y para definir tamaños en caso de colecciones. Los identificadores sin embargo siguen la filosofía del reconocimiento de palabras reservas, se comprueba y solo se devuelve si existe. A continuación, se muestran las etiquetas que son reconocidas por estas 3 expresiones regulares:

|  |  |
| --- | --- |
| Id | Descripción |
| VAR | Todas las variables reconocidas con LAVEL\_VAR |
| SIZE | Todos los tamaños reconocidos con LABEL\_SIZE |
| L\_ARRAY | Identificador del tipo de dato Array |
| L\_HASH | Identificador del tipo de dato Hash |
| L\_LIST | Identificador del tipo de dato List |
| REF | Identificador del tipo de dato Referencia |
| TYPE | Identificador de los tipos de dato básicos |
| ARGS | Identificador de los argumentos de una función. |
| RETURNS | Identificador de los retornos de una funcion |
| MAPPER\_CODE | Identificador de un Mapper |
| MAPPER\_LOOP | Identificador del bucle a paralelizar en un Mapper |
| HADOOP\_PRINT | Identificador de las impresiones en Hadoop |
| REDUCER\_CODE | Identificador de un Reducer |
| REDUCER\_OP | Identificador de la operación de Reduccion |
| REDUCER\_CHANGE | Identificador de la operación de cambio de clave en Reducer. |
| REDUCER\_KEY | Identificador de la variable clave en Reducer |
| REDUCER\_VALUE | Identificador de la variable valor en Reducer |
| REDUCER\_VAR | Identificador de las variables a declarar en Reducer |

Tabla 6: Tokens de etiquetas

## Gramática de Analizador Sintáctico

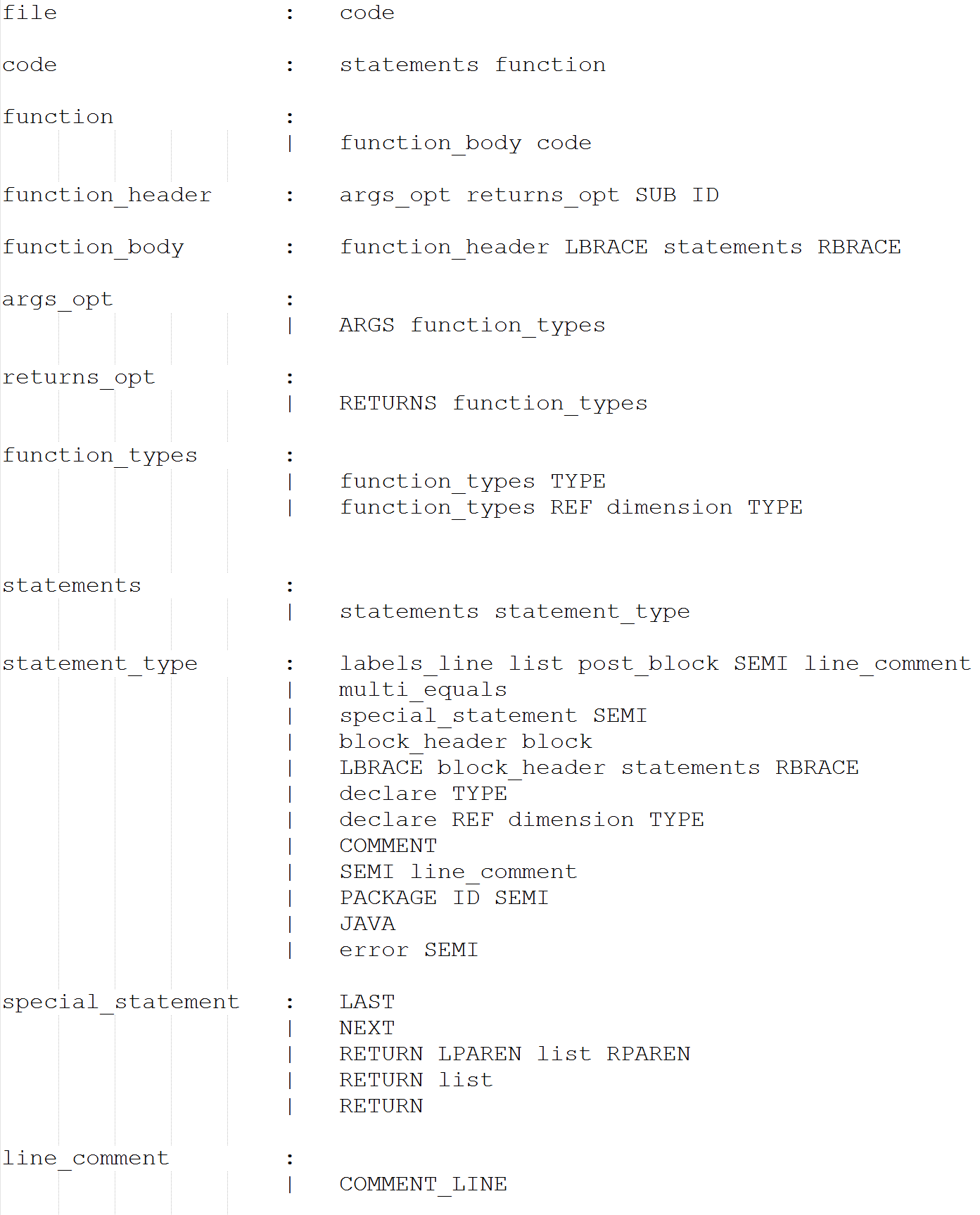
El analizador sintáctico es el encargado de la validación de los tokens para comprobar si la sintaxis es correcta. Existen diversos tipos de analizadores, ascendentes y descendentes, así como distintos tipos de gramáticas para cada uno de ellos que proporcionan las capacidades para definir un comportamiento determinado.

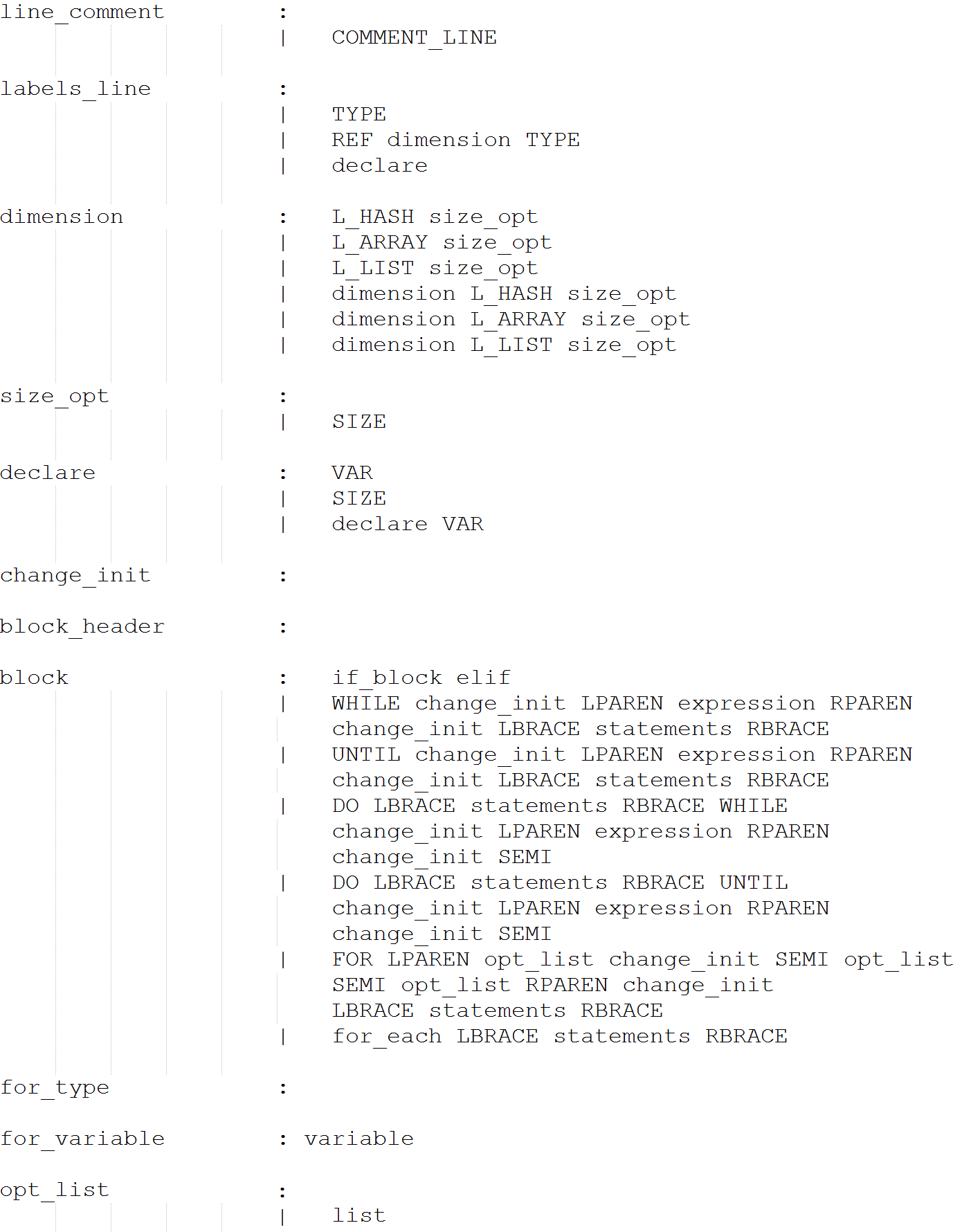
Este traductor hace uso de un analizador ascendente con una gramática LALR(1). Esta gramática permite definir analizadores rápidos y que no consumen tanta memoria como los LR.

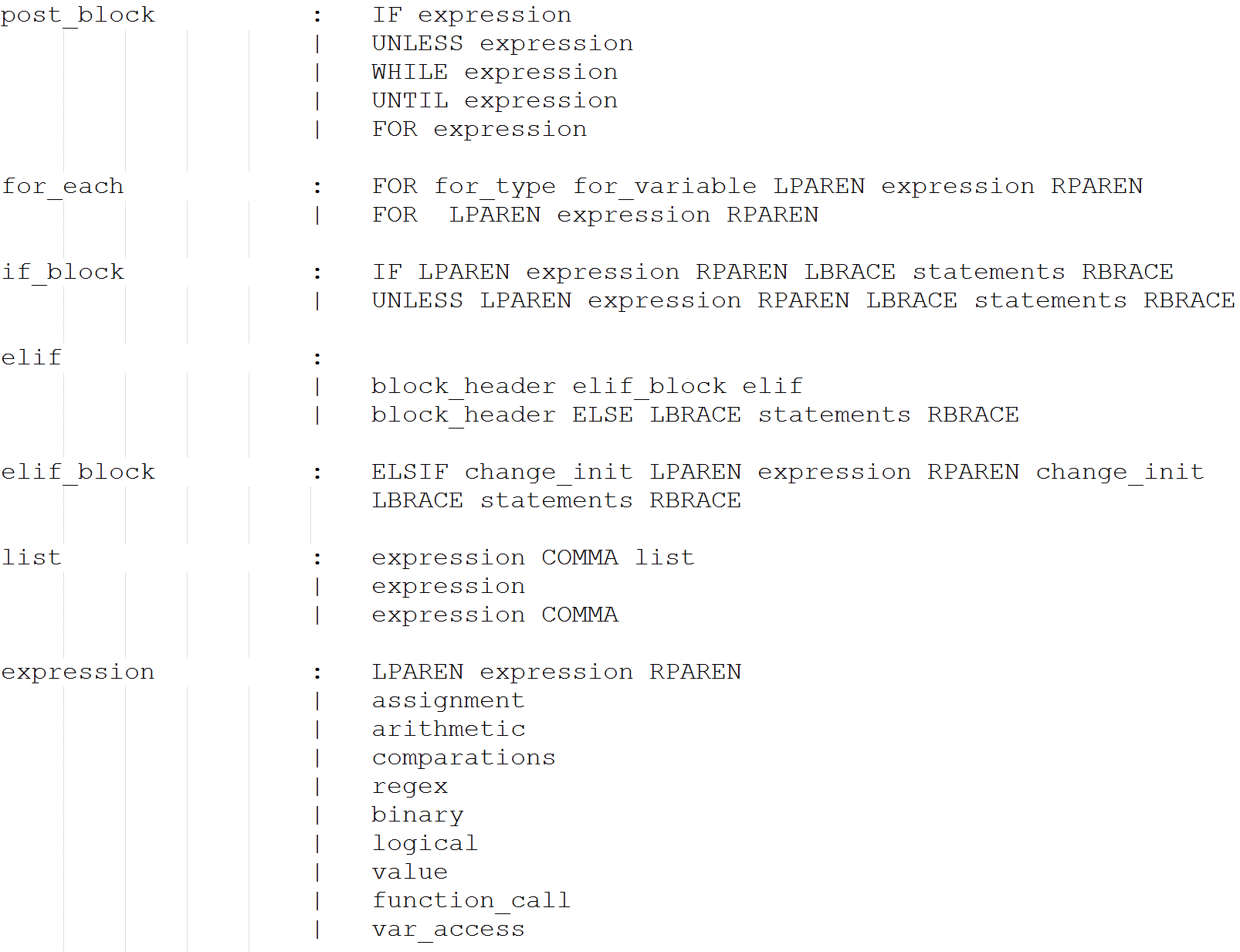
Perl tiene definido una sintaxis increíblemente grande y según su documentación, no puede ser reconocido por un analizador sintáctico, pero para problemas acotados como son los scripts de procesamiento de texto, una gramática LALR(1) es más que suficiente para realizar esta tarea.

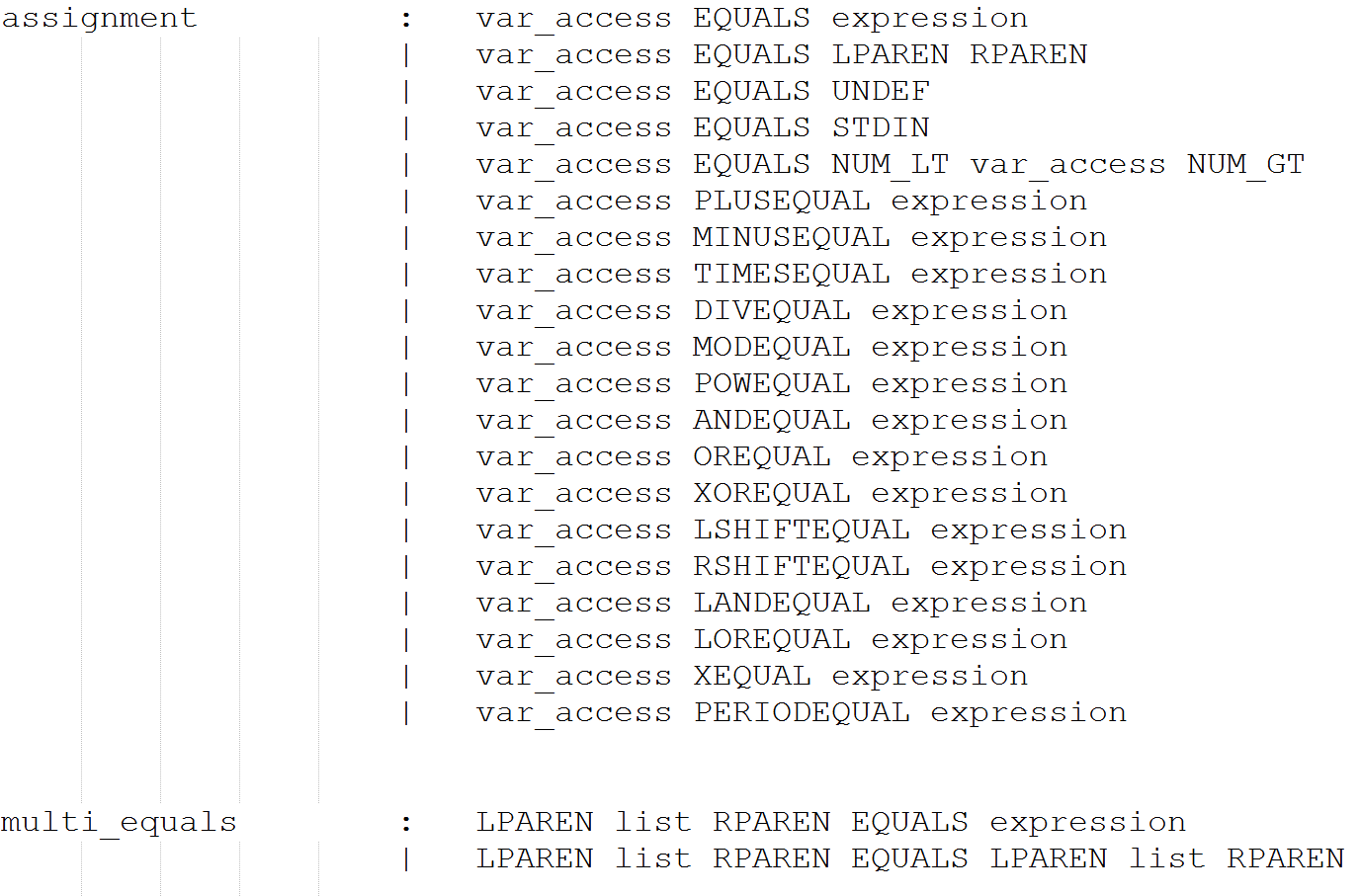
En la Figura 13 se representa la gramática definida en el analizador léxico, los componentes terminales de la gramática se representan en mayúsculas y corresponden a los tokens definidos en el apartado de definición de analizador léxico. Por otro lado, los símbolos no terminales de la gramática se escriben en minúscula y corresponden a la parte izquierda de las reglas gramaticales.

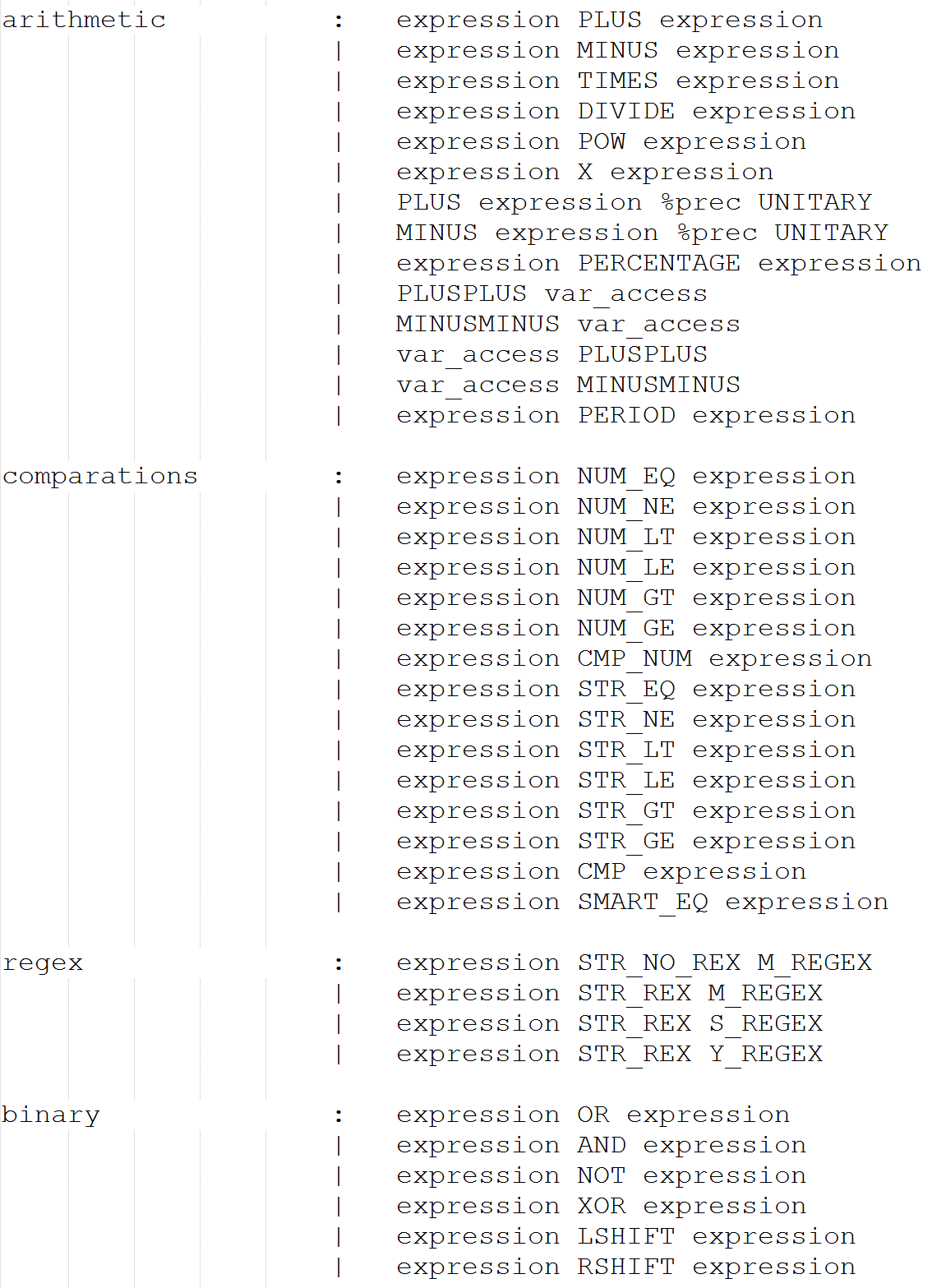
Figura 13: Gramática analizador sintáctico

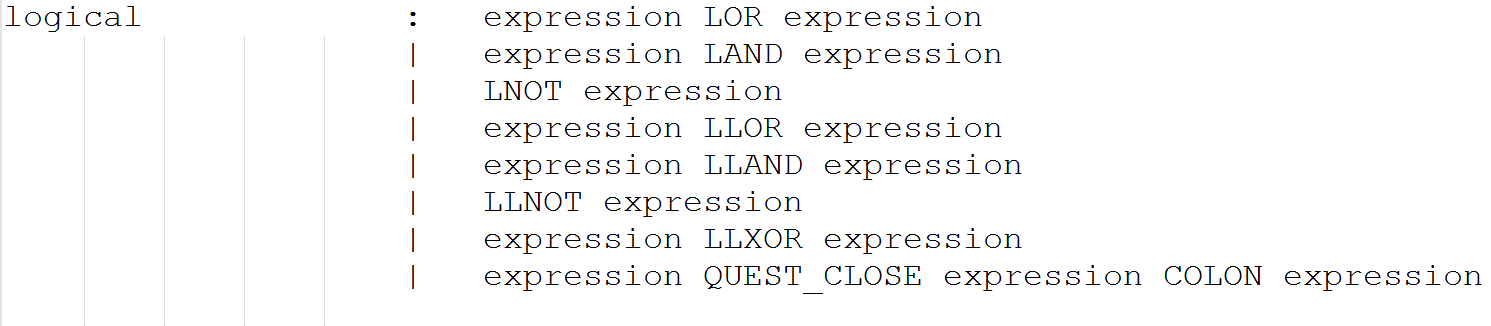


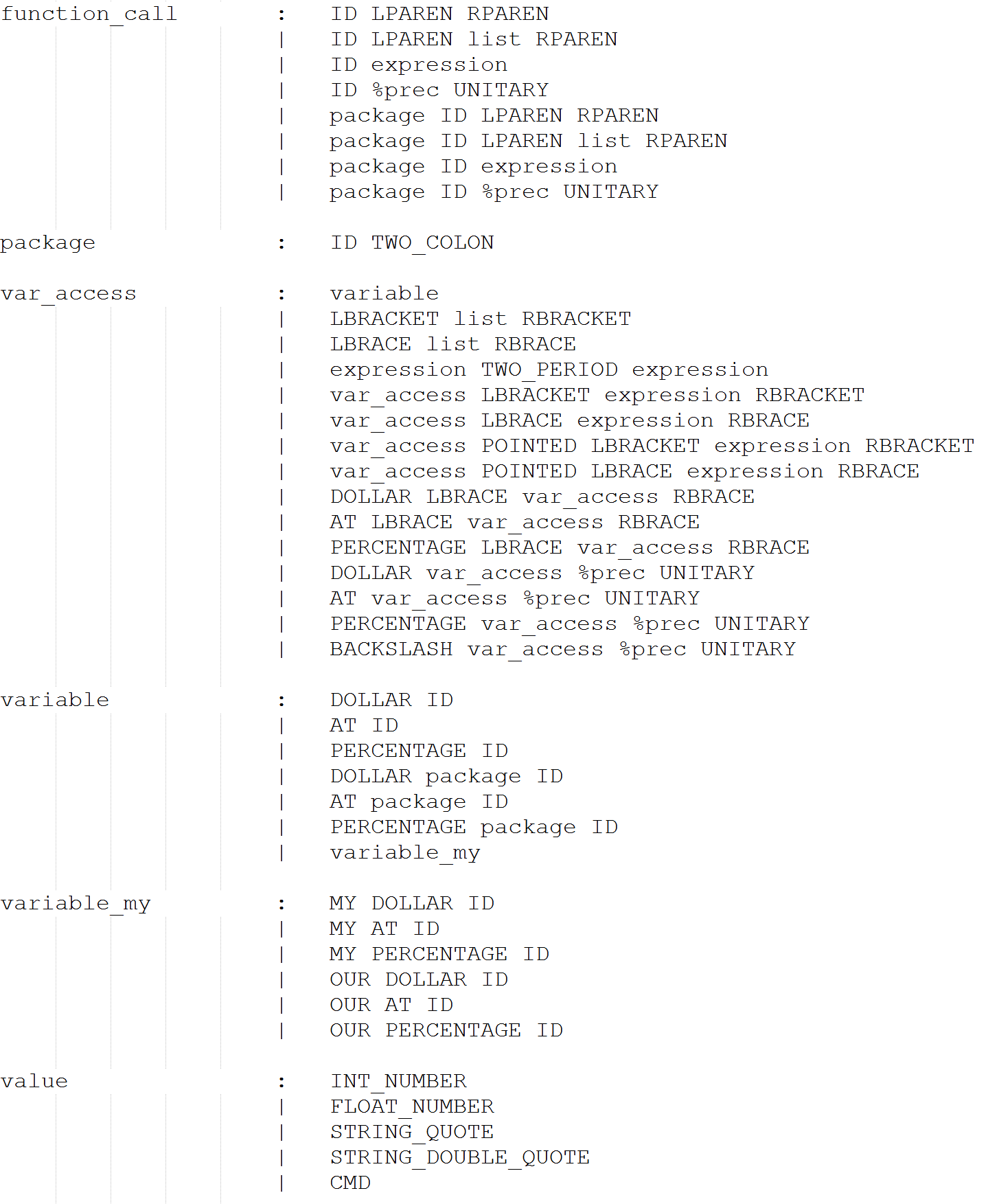












Para el funcionamiento de la gramática anterior, es necesario establecer una asociatividad y una precedencia para poder eliminar los conflictos en el análisis.

La asociatividad indica por que lado se empieza a reducir una regla. Existen tres tipos: derecha, izquierda o nula. Por ejemplo, la operación ‘2+3+1’ pude realizarse de dos maneras, por la derecha (2+4) o por la izquierda (5+1) en este caso el resultado sería el mismo, pero aun así hay que establece un criterio firme a realizar.

La precedencia indica la prioridad entre operadores, igual que el caso anterior existen tres tipos disjuntos de precedencias, mayor, menor e igual. Cuando dos operadores se encuentran juntos deberá darle prioridad uno al otro y en caso de tener igual prioridad se hará uso de la asociatividad para resolver la situación. Por ejemplo, en las generaciones matemáticas las multiplicaciones tienen prioridad frente a las sumas.

En la Figura 14 podemos ver la asociatividad de las operaciones adaptada de la documentación oficial de Perl [16] definida en la sintaxis de PLY.



Las precedencias están ordenadas de menos presente a mas precedente, de forma que cuanto más abajo se encuentre un símbolo, más precedencia tendrá.

## Generación y Comprobación de código

En este apartado hablaremos sobre las particularidades a alto nivel de la comprobación y generación de código.

### 8.5.1. Tipos de datos

Los tipos de datos son atributos que indican a Java la clase de dato que va a usar, para obtener este tipo de dato se usan las etiquetas presentes en el código cuando se trata de variables o funciones, o es deducido en caso de las contantes.

#### 8.5.1.1. Tipos asociados a etiquetas

|  |  |
| --- | --- |
| Etiqueta | Tipo de dato java |
| <boolean> | Boolean |
| <interger> | Integer |
| <long> | Long |
| <float> | Float |
| <double> | Double |
| <string> | String |
| <file> | PerlFile |
| <array> | [] |
| <list> | List<> |
| <hash> | Map<> |
| <ref> | Ref<> |

Tabla 7: Relación etiqueta tipo de dato

Las etiquetas de tipos de dato básico fueron diseñadas con los nombres de los tipos de datos java para una traducción más cercana al usuario. Estos son los tipos de datos asociados a los escalares de Perl.

Los arrays de Perl pueden traducirse por Arrays o List Java, la razón de realizar esta diferenciación es mejorar la eficiencia de acceso a cada uno de ellos. Por ejemplo, si se necesita un array estático que siempre tendrá un tamaño fijo, se deberá usar la etiqueta <array>, pero si no se conoce el tamaño o se van usar funciones para añadir elementos como push o pop, es mas rápido usar <list>. Aparte de esto ambas se pueden usarse indistintamente siendo la única penalización el rendimiento.

Para simular las referencias, se ha creado una clase Ref que almacena una colección simulando el mismo comportamiento. La razón de que sólo se pueden usar colecciones es que los tipos de dato básico son inmutables.

Los ficheros son unificados con una clase llamada PerlFile que permite tanto abrir ficheros para lectura como para escritura, de esta forma no es necesario especificar varios tipos de variables para ficheros.

#### 8.5.1.2. Tipos asociados a Contantes

Los tipos de datos asociados a constantes, se evalúan de la misma forma que lo haría Java si estuviese escrito en su propio código.

Los números sin decimales son siempre interpretados por Integer, estos pueden estar escrito en cualquier formato, Binario, Octal, Decimal o hexadecimal. Por otro lado, los números decimales y en notación científica son por defecto Double.

Las cadenas en Perl pueden especificarse de tres formas diferentes:

* Comillas simples (‘’): Las constantes entre comillas simples son traducidas directamente cambiándolas por comillas dobles puesto que la semántica es la misma.
* Comillas dobles (“”): Las comillas dobles indican que en Perl la constante puede contener variables que deberán ser interpoladas, pero como eso no está soportado, la traducción se hace sin examinar la cadena.
* Comillas inclinadas (``): Las comillas inclinadas indican que la cadena deberá ser ejecutada por el sistema y luego almacenar el resultado. La traducción realiza el mismo comportamiento generando una función que retorna un String con el resultado de la llamada al sistema.

### 8.5.2. Casting automático

El traductor cuenta con un sistema de casting automático para ayudar a lidiar con los problemas que aparecen al asignar tipos al código fuente.

Por defecto todos los tipos básicos pueden ser casteados directamente entre sí de igual forma que lo hace Perl. Esto no quiere decir que siempre sea semánticamente correcto, pero esa parte recae en el usuario.

Con las colecciones, el casting no se podrá hacer directamente igualando como los tipos básicos, el programador deberá recorrer la colección y copiar elemento a elemento. La única excepción a esta regla es entre un Array y una Lista del mismo tipo, dado que esencialmente son el mismo y debe fomentarse el uso entre ellos por factores de rendimiento.

A continuación, se muestran varias tablas con la forma de transformar para cada tipo de dato, las tablas solo incluyen las transformaciones validas, las obviadas darán error.

|  |  |
| --- | --- |
| Integer | |
| Boolean | ?1:0 |
| Long | (int) |
| Float | (int) |
| Double | (int) |
| String | Integer.parseInt |
| File | 1 |
| Array | length |
| List | size() |
| Map | size() |
| Ref | 1 |

|  |  |
| --- | --- |
| Long | |
| Boolean | ?1l:0l |
| Integer | = |
| Float | (long) |
| Double | (long) |
| String | Long.parseLong |
| File | 1l |
| Array | length |
| List | size() |
| Map | size() |
| Ref | 1l |

|  |  |
| --- | --- |
| Boolean | |
| Integer | != 0 |
| Long | != 0 |
| Float | != 0 |
| Double | != 0 |
| String | != “” && != “0” |
| File | true |
| Array | length != 0 |
| List | size() != 0 |
| Map | size() != 0 |
| Ref | != null |

|  |  |
| --- | --- |
| Double | |
| Boolean | ?1d:0d |
| Integer | = |
| Long | = |
| Float | = |
| String | Double.parseDouble |
| File | 1d |
| Array | length |
| List | size() |
| Map | size() |
| Ref | 1d |

|  |  |
| --- | --- |
| String | |
| Boolean | String.valueOf |
| Integer | String.valueOf |
| Long | String.valueOf |
| Float | String.valueOf |
| Double | String.valueOf |
| File | “1” |
| Array | String.valueOf(  length |
| List | String.valueOf(  size() |
| Map | String.valueOf(  size() |
| Ref | “1” |

|  |  |
| --- | --- |
| Float | |
| Boolean | ?1f:0f |
| Integer | = |
| Long | = |
| Double | (float) |
| String | Float.parseFloat |
| File | 1f |
| Array | length |
| List | size() |
| Map | size() |
| Ref | 1f |

|  |  |
| --- | --- |
| List | |
| Array | toList |

|  |  |
| --- | --- |
| Array | |
| List | toArray |

En algunos casos, se ha introducido valores constantes debido a la imposibilidad de aplicar el casting. Por ejemplo, si casteamos una referencia a un entero, deberíamos obtener la dirección de memoria donde se almacena el valor, con lo cual su valor no es importante.

### 8.5.2. Definición de funciones

En este apartado se discutirá la transformación de las funciones Perl a funciones Java, los principales problemas y las soluciones encontradas.

En Perl las funciones reciben sus argumentos en forma de array, el resultado de ese array es la concatenación de todos los elementos pasados como argumentos. El primer problema reside cuando uno de los argumentos es también un array, a continuación, se ilustra el comportamiento del mismo.

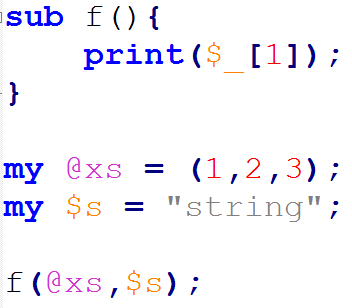


Figura 14: Función perl con array y escalar

El comportamiento esperado por una persona que no conoce el lenguaje Perl es pensar que la función imprimirá la cadena “string” pero en realidad se imprimirá el número 2. La razón de este comportamiento es que Perl concatena la lista con escalar creando un nuevo array con cuatro argumentos. En Java no podemos permitir este comportamiento debido a que, al no conocer la longitud de los arrays en tiempo de compilación, es imposible saber dónde empieza y donde acaba un array, lo cual impide realizar el casting al tipo correspondiente.

Para solucionar este problema, es necesario que Perl no concatene los argumentos y que si la función recibe dos argumentos, la función reciba un array de dos elementos. La solución pasa por obligar al usuario a pasar todas las colecciones por referencia en vez de por valor, de eso modo Perl no concatenara y el usuario puede des referenciar luego en la función.

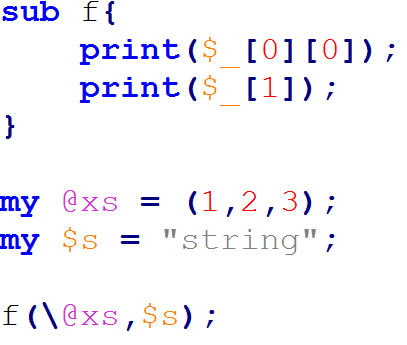


Figura 15: Funcion Perl con referencia Array y escalar

En la Figura 15 podemos ver cómo el array y el escalar se mantienen separados en la función, el primer elemento es un array al que podemos acceder a sus posiciones y el segundo es un escalar. Con esto el problema de las funciones queda solucionado y lo único que obligamos a hacer al usuario es añadir un ‘\’ antes de las colecciones en la llamada a las funciones.

Ahora falta abordar el tipado de las funciones tanto para los argumentos como para el retorno. Para ello se hará uso del mismo método que para el tipado de variables, a continuación, se muestra la sintaxis con varios ejemplos de funciones y su traducción.

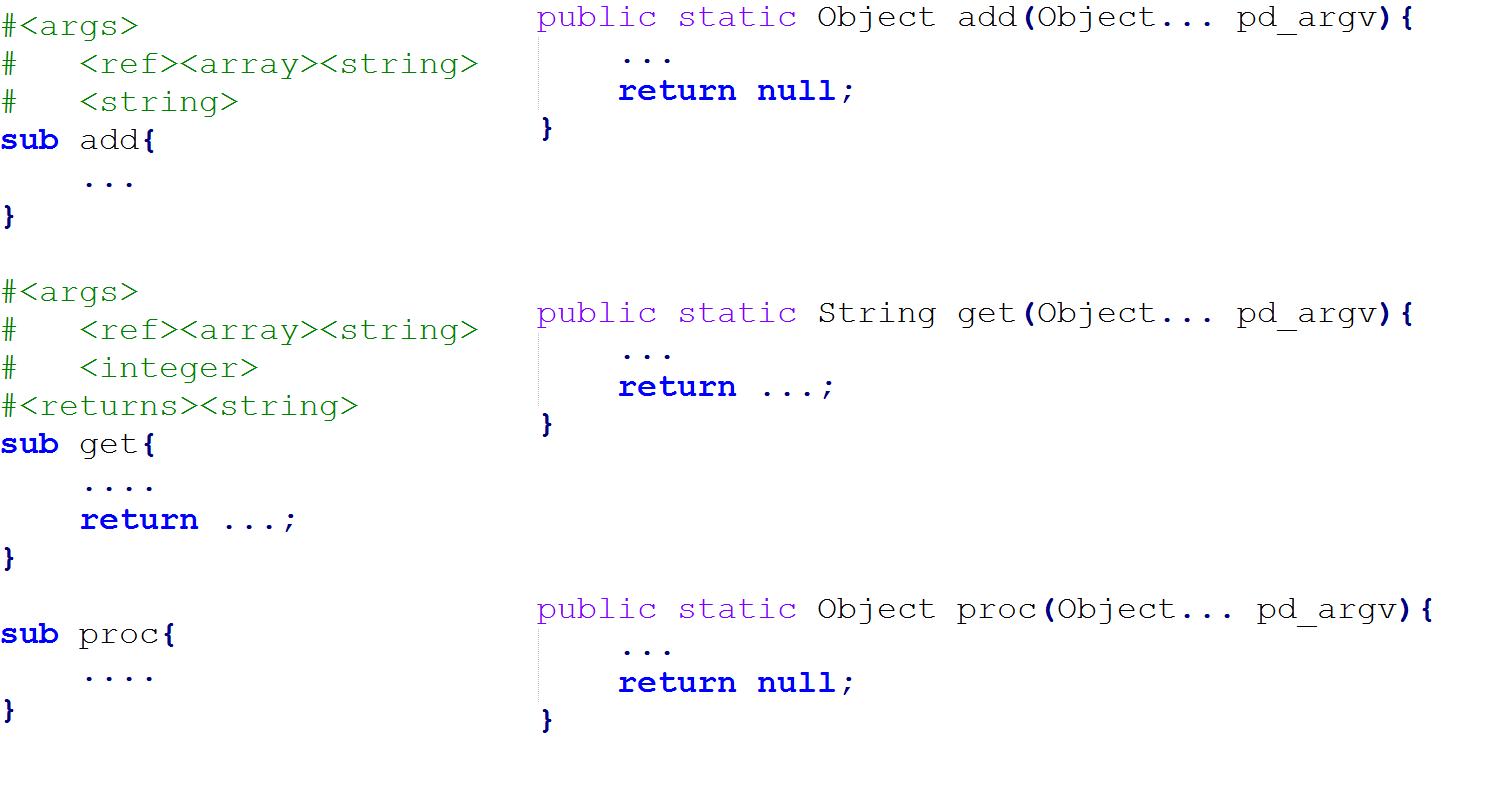


Figura 16: Ejemplo de tipado de funciones y su traducción

Como podemos ver en primer lugar el tipado de las funciones se realiza encadenando los tipos uno detrás de otro predecidos <args> para los argumentos y <returns> para los retornos. La posición de las etiquetas en varias líneas es meramente visual, puede optarse por poner todo en la misma línea.

Por otro lado, la traducción de las funciones es prácticamente igual para las tres funciones. Todas las funciones reciben un array de Object donde estarán los argumentos en caso de que existan. Cuando el usuario obtiene un argumento se le aplica el cast en función de las etiquetas de la cabecera. Los retornos siguen un procedimiento parecido. Si una función retorno un elemento, se le asignará ese tipo al retorno y si no retorna ninguno se conservará Object por razones técnicas relativas al anidamiento de funciones.

Como nota final, Perl permite retornar más de un elemento en una función, pero ese caso es el mismo que para los argumentos, se retorna un array de object y se usa la cabecera de la función para aplicar el casting en la asignación.

### 8.5.3. Asignación

En Perl, el operador igual ‘=’ puede usarse en cuatro contextos distintos, en este apartado se explicará el diseño para cada una de sus traducciones.

**Expresión = Expresión**

El primer caso es el contemplado por todos los lenguajes de programación, una expresión es almacenada en una variable. Por esa razón la traducción de este tipo es directa.

**Lista = Lista**

En el segundo caso, una lista de expresiones es asignada atómicamente a una lista de variables, por esa razón puede usarse para intercambiar variables y realizar acciones en grupo.

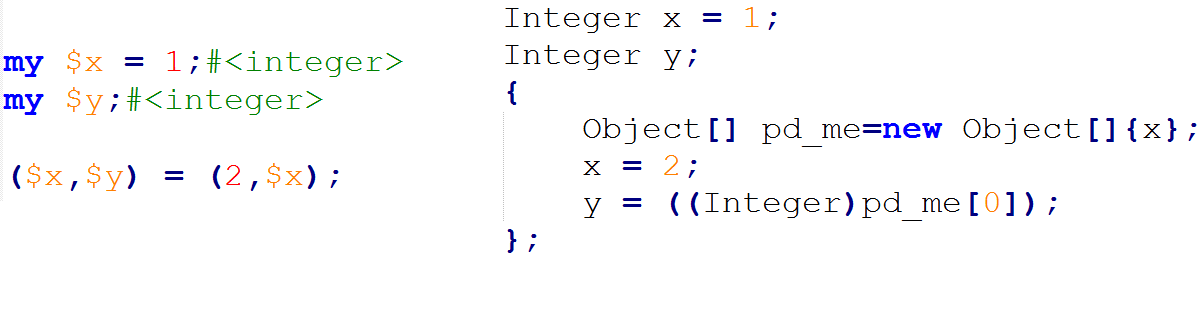


Figura 17: Asignación entre listas

Como podemos ver en la Figura 17, queremos asignar el valor de un número y una variable a dos variables. La estrategia para realizar esta asignación es la siguiente:

* Abrimos un bloque para no conservar la variable auxiliar pd\_me y poder reutilizarla.
* Guardamos todas las variables del lado derecho en un array.
* Generamos una lista de asignaciones donde las constantes se copian directamente y las variables se sacan del array devolviéndoles su tipo.

Siguiendo estos pasos, las asignaciones se realizan de la misma forma que se haría en Perl conservando la atomicidad.

**Lista = Expresión**

El tercer caso es el usado para asignar a una lista de variables los elementos consecutivos almacenados en un array.

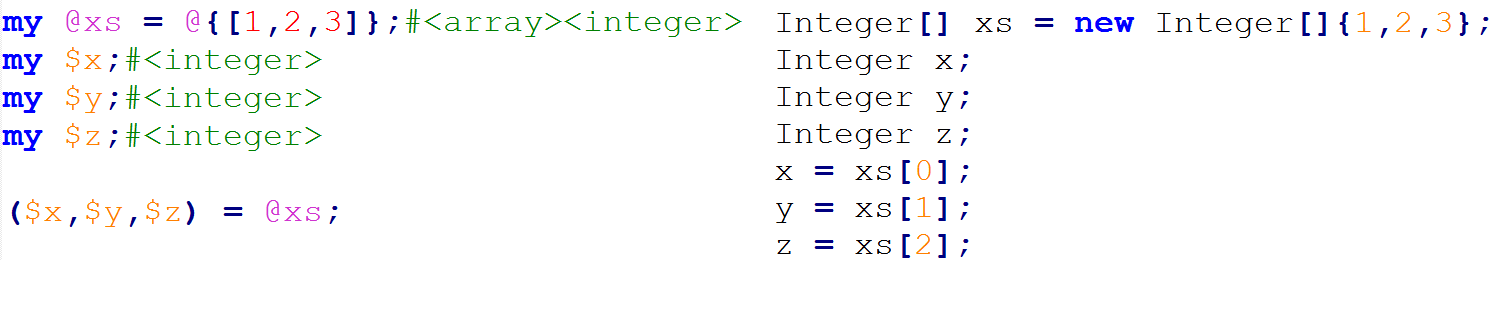


Figura 18: Asignación entre lista y expresión

Como podemos ver en la Figura 18, tenemos un array de tres elementos que queremos asignar a tres variables distintas. La traducción es este caso es bastante sencilla, se va accediendo a las posiciones del array y luego se va realizando la asignación.

Pero qué pasaría si el array no estuviera almacenado en una variable como en este caso, por ejemplo, en el retorno de una función.

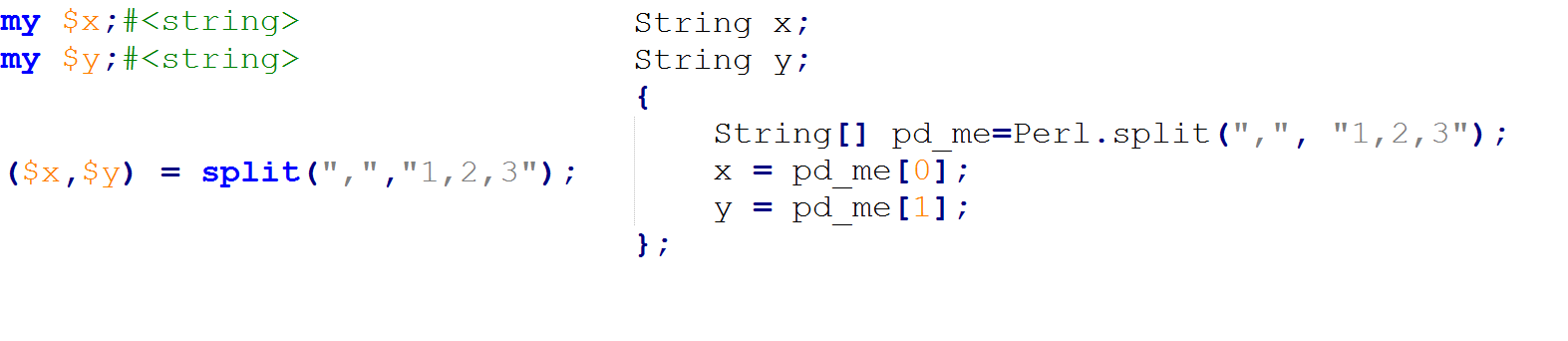


Figura 19: Asignación entre lista y función

La Figura 19 muestra el comportamiento en caso de no poder asegurar que el array sea una variable: se crea una variable auxiliar para realizar la asignación.

**Expresión = Lista**

El cuarto caso es el más complicado y el que no está soportado directamente: las listas en Perl pueden indicar la inicialización de un array o de un hash. Por esa razón no se conoce el tipo hasta su asignación. Así se ha optado al igual que en las funciones por hacer uso de los punteros. En Perl los punteros a array se inicializan entre [] y los punteros a hash se inicializan entre {}. Por esa razón las inicializaciones se realizarán creando un puntero y accediendo a él.

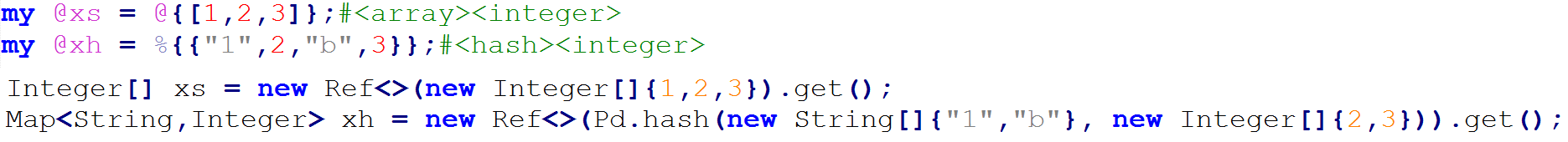


Figura 20: Inicialización de colecciones

Como podemos ver en la Figura 20, la traducción parece un poco ineficiente, pero este es uno de los casos contemplados en las optimizaciones del traductor (Acceso a una referencia recién creada).

Si ejecutamos el traductor con la opción de optimizar el resultado será el siguiente.

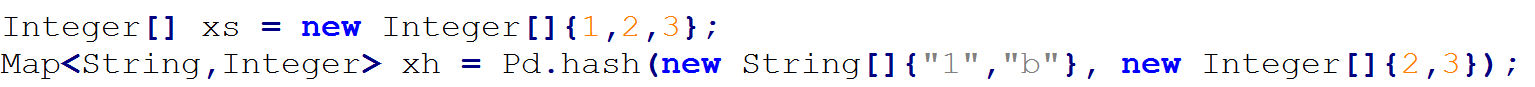


Figura 21: Inicialización de colecciones optimizada

Como podemos ver las referencias en el código Java han sido eliminadas y el código es igual de eficiente que si hubiesen sido declaradas directamente.

### 8.5.4. Ficheros

La gestión de ficheros es un asunto importante en los scripts de procesamiento de lenguaje natural. En Perl existen diferentes formas de gestionar los ficheros, pero para simplificar las traducciones se ha optado por el más usado. Las traducciones hacen uso de la librería Java que se explicaría más adelante pero aun así no hace falta para entender las traducciones.

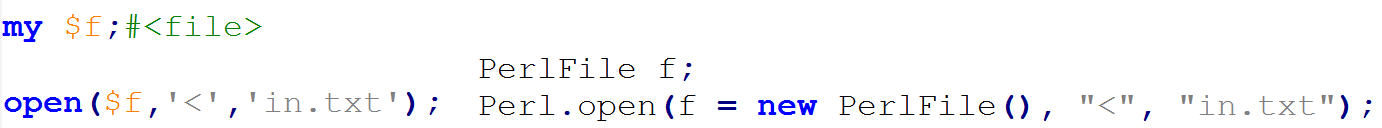


Figura 22: Apertura de fichero

Como podemos ver en la Figura 22, la traducción se parece mucho al código nativo Perl, la variable se inicializa en el argumento de la función y es java en tiempo de ejecución quien realiza el resto del proceso.

En este caso el fichero se abriría para lectura por el uso del modificador ‘<’, los otros modificadores disponibles son ‘>’ y ‘>>’ para abrir un fichero para escritura para truncar y concatenar respectivamente.

La lectura de un fichero se puede realizar de dos maneras, línea a línea usando un escalar o entero usando un array.

A continuación, se muestra un ejemplo de su uso:

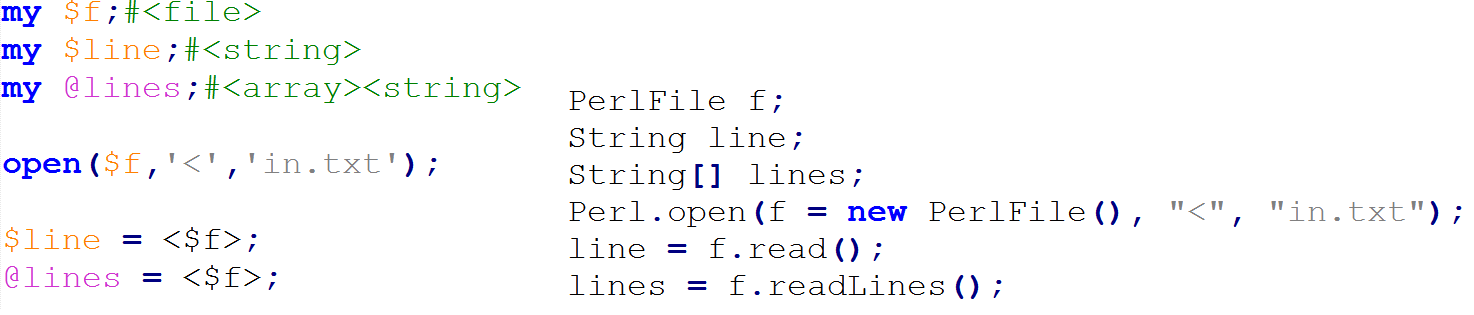


Figura 23: Ejemplo de lectura de fichero

Por otro lado, también es necesario especificar un formato para escribir en un fichero, a continuación, se muestra su uso usando la función print.

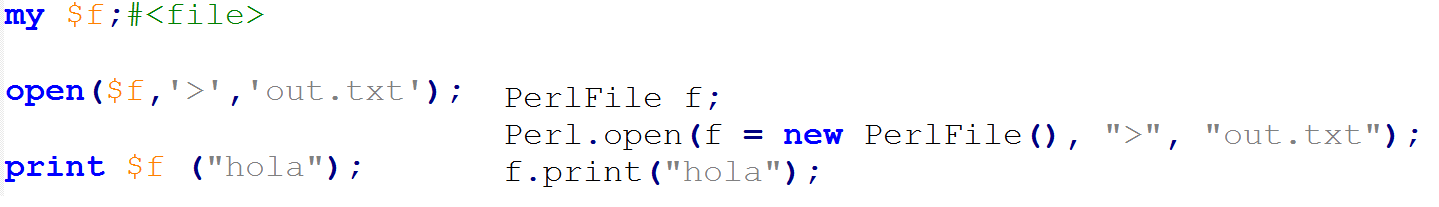


Figura 24: Ejemplo de escritura de fichero

Para terminar, es importante acordarse de cerrar los ficheros, para cerrar un fichero se llama a la función close, lo cual es trivial y no hace falta un ejemplo.

### 8.5.5. Paquetes

El soporte para definir paquetes también esta soportado por el proceso de traducción. La inclusión de otros scripts se hace mediante la directiva ‘use’ que permite incluir un paquete definido en otro fichero.

El funcionamiento interno es muy simple, se requiere al programador que introduzca los ficheros de forma ordenada en el traductor. Luego según va finalizando el análisis de un fichero, el traductor conserva todos los métodos y variables globales en un hash usando como clave el nombre del paquete. De esta forma cuando se analiza el siguiente fichero y se hace mención al paquete solo hace falta buscar en las las variables y métodos.

Un paquete corresponde a una clase Java, las variables globales a atributos y los métodos a métodos estáticos. A continuación, se muestra un ejemplo práctico del funcionamiento de este proceso.

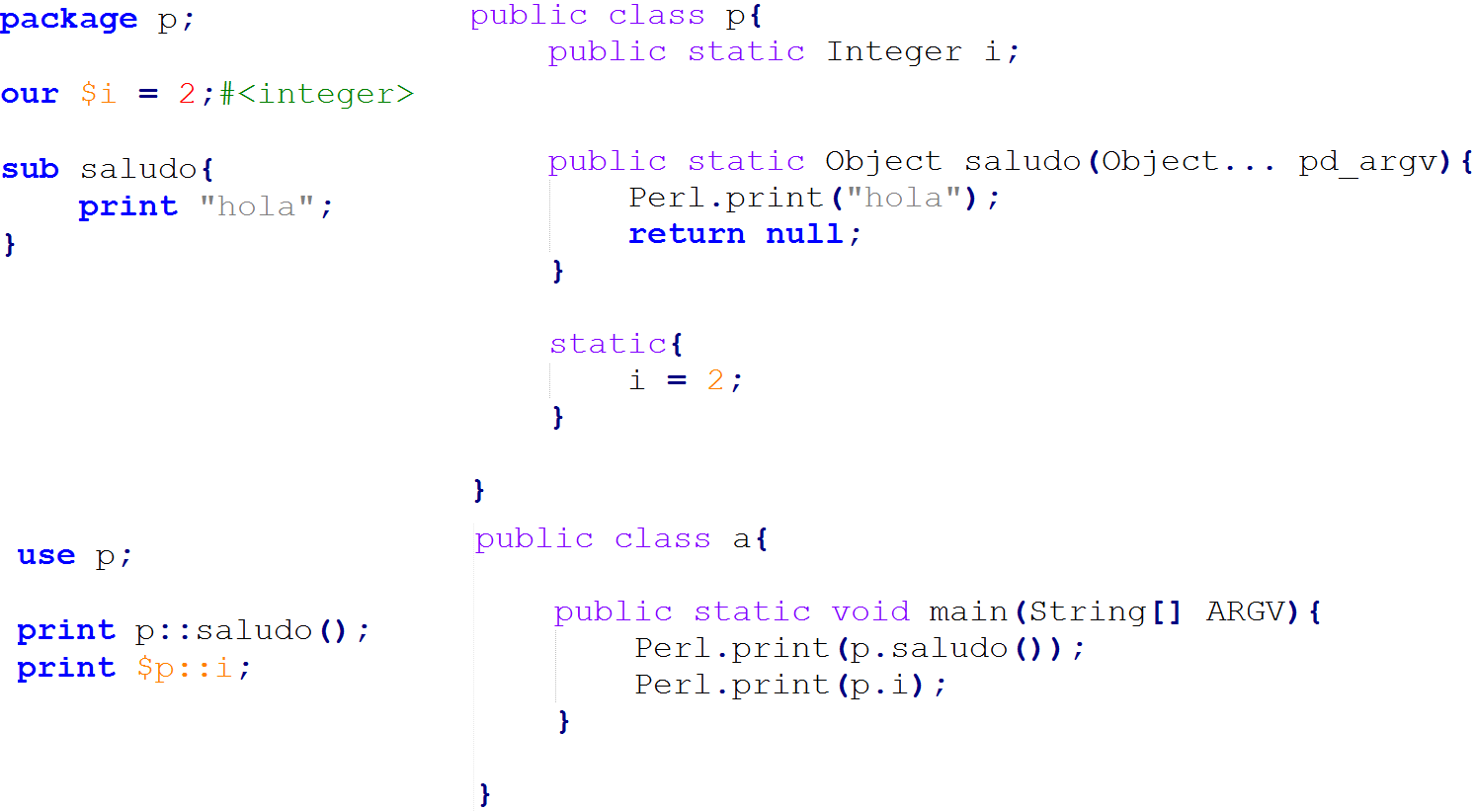


Figura 25: Creación y uso de un paquete con su traducción

### 8.5.6. Funciones nativas

Para una mejor integración con Perl, es necesario dar soporte a las funciones nativas. Para ello se han escogido las funciones más usadas y se ha definido una lista con ellas.

A continuación, se muestra una tabla con las funciones permitidas y una breve descripción del comportamiento de cada una de ellas.

|  |  |  |
| --- | --- | --- |
| Nº | Nombre | Descripción |
| 1 | chomp | Limpia una cadena y retorna el número de caracteres eliminados |
| 2 | chop | Quita el ultimo carácter a una cadena y lo retorna |
| 3 | close | Cierra un fichero y retorno su tuvo éxito |
| 4 | defined | Comprueba si una variable está asignada |
| 5 | delete | Borra un elemento de un Hash |
| 6 | die | Aborta el programa mostrando un mensaje de error |
| 7 | each | Recorre un Hashmap retornado de cada vez un clave-valor |
| 8 | exists | Comprueba si existe una clave en un HashMap |
| 9 | exit | Finaliza el programa |
| 10 | join | Une varias cadenas usando un separador |
| 11 | keys | Retorna las claves de un HashMap |
| 12 | lc | Transforma la cadena a minúsculas |
| 13 | lcfirst | Transforma la primera letra a minúsculas |
| 14 | length | Calcula la longitud de una candena |
| 15 | open | Abre un fichero |
| 16 | pop | Quita el último elemento de un array y lo retorna |
| 17 | print | Imprime |
| 18 | push | Añade un elemento al final de un array |
| 19 | shirft | Quita el primer elemento de un array y lo retorna |
| 20 | sort | Ordena un array |
| 21 | splice | Subdivide o remplaza segmentos de un array |
| 22 | split | Retorna una cadena dividida según un separador |
| 23 | substr | Subdivide o remplaza segmentos de una cadena |
| 24 | system | Realiza una llama al sistema |
| 25 | uc | Transforma la cadena a mayúsculas |
| 26 | ucfirst | Transforma la primera letra a mayúsculas |
| 27 | unshift | Añade un elemento al inicio de un array |
| 28 | values | Retorna los valores de un HashMap |

### 8.5.7. Expresiones regulares

En este apartado se tratará el tema de la traducción de las expresiones regulares. En Perl las expresiones regulares forman parte del código nativo sin necesidad de la invocación de ninguna función.

Las expresiones regulares pueden ser de tres tipos match, sustitución y transliteración; y a su vez contener modificadores que pueden cambiar su comportamiento. Por ejemplo, existen modificadores para cambiar el ámbito de línea a multilinea o de caso no sensitivo a caso sensitivo, entre otras.

Por esa razón se ha preferido conservar la máxima compatibilidad posible y dejar que sea Perl quien evalúe las expresiones regulares siendo invocado el intérprete desde Java. Estas invocaciones, se llevan a cabo mediante funciones que invocan a la librería de Perldoop, por esa razón si en un futuro es posible ofrecer un soporte nativo Java para la evaluación de expresiones regulares, no será necesario cambiar nada en el traductor.

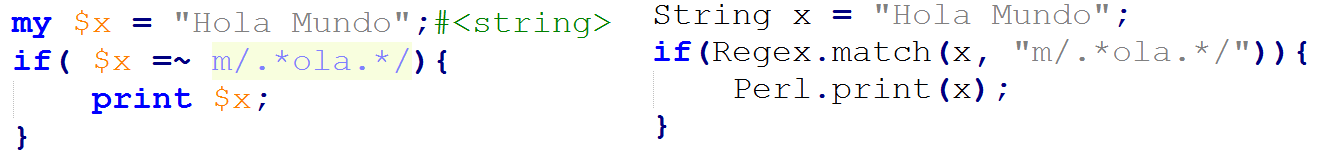


Figura 26: Expresión regular match

Como podemos ver en la Figura 26, se envía a java la variable con la cadena a valida y la expresión regular escrita tal cual para poder reconstruirla en una llamada en el intérprete a Perl.

### 8.5.8. Apache Hadoop

En este apartado se explicará el desarrollo para poder convertir un código Perl secuencial en un código Java adaptado al paradigma MapReduce mejorando su ejecución en un clúster usando la tecnología Hadoop. MapReduce es una técnica de procesamiento y un modelo de computación distribuida, el algoritmo está dividido en dos operaciones: mapeo y reducción. La operación de mapeo toma un conjunto de datos y lo convierte en otro conjunto de datos en el que los elementos se dividen en pares clave/valor. Por otro lado, la operación de reducir, que toma la salida del mapeo, combina las tuplas clave/valor en un conjunto más pequeño de tuplas. La operación de reducción se realiza siempre después de la operación de mapeo.

Para lograr este objetivo es necesario introducir nuevas etiquetas que permitan diferenciar un código Perl secuencial normal de uno que debe ser transformado. Por otro lado, también debe haber un compromiso por parte del programador. El manual define unas directrices en cuanto a la disposición del código para así poder aplicar la transformación. Por ejemplo, la versión secuencial de un Mapper, puede entenderse como un bucle que realiza el procesado secuencialmente.

#### 8.5.8.1. Mapper

El Mapper es el encargado de procesar un conjunto de datos, que por lo general se encuentran en forma de un fichero o un directorio y luego es almacenado en el sistema de ficheros HDFS de Hadoop. El fichero es lo que recibe la función mapper fragmentado línea a línea, despues del procesado, los datos son convertidos en tuplas que almacenan los datos en pares clave/valor.

En primer lugar, es necesario acotar el fragmento de código que pertenece al mapper separándolo del resto del código fuente. Para esto se usará la etiqueta con nombre <mapper\_code> seguida de un bloque que encerrará el código fuente. Con esta sintaxis, indicamos al analizador sintáctico que debe crear una clase que extienda de la clase Mapper, crear una sobrescritura de su método map y copiar la traducción en su interior.

En segundo lugar, la transformación de código secuencial a paralelo se realiza transformando un bucle que recibe sus datos de la entrada estándar. El criterio de transformación es eliminar el bucle por completo y asignar a la variable que obtenía los datos de la entrada estándar al argumento de la función map.

Por último, es necesario guardar los datos en algún lugar después del procesado. En Perl el criterio seguido es imprimir los pares clave/valor usando dos separadores, uno para separar la clave del valor y otros para separar las tuplas. En Hadoop, existe un elemento llamado contexto, en su interior existe una función llamada write que recibe como argumento esa tupla. Para la transformación, emplearemos dos elementos, una llamada a la función print y la etiqueta <hadoop\_print>. La transformación se basa en el siguiente principio: si obligamos al programador a imprimir sus datos usando el siguiente formato en la función print (clave, separador1, valor, separador2), es muy fácil realizar la transformación, debido a que solo debemos coger los argumentos 1 y 3 para luego pasárselos a la función write.

Para ilustrar todo los explicado en este apartado, se hará uso a continuación del famoso ejemplo WordCount en Perl junto con su traducción Java.

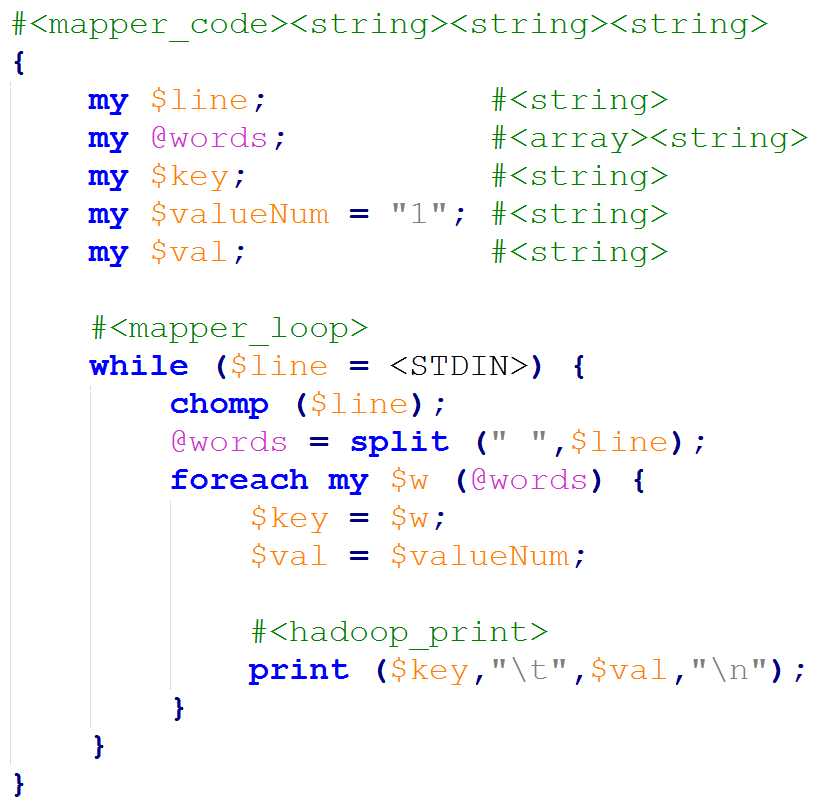


Figura 27: Mapper Perl

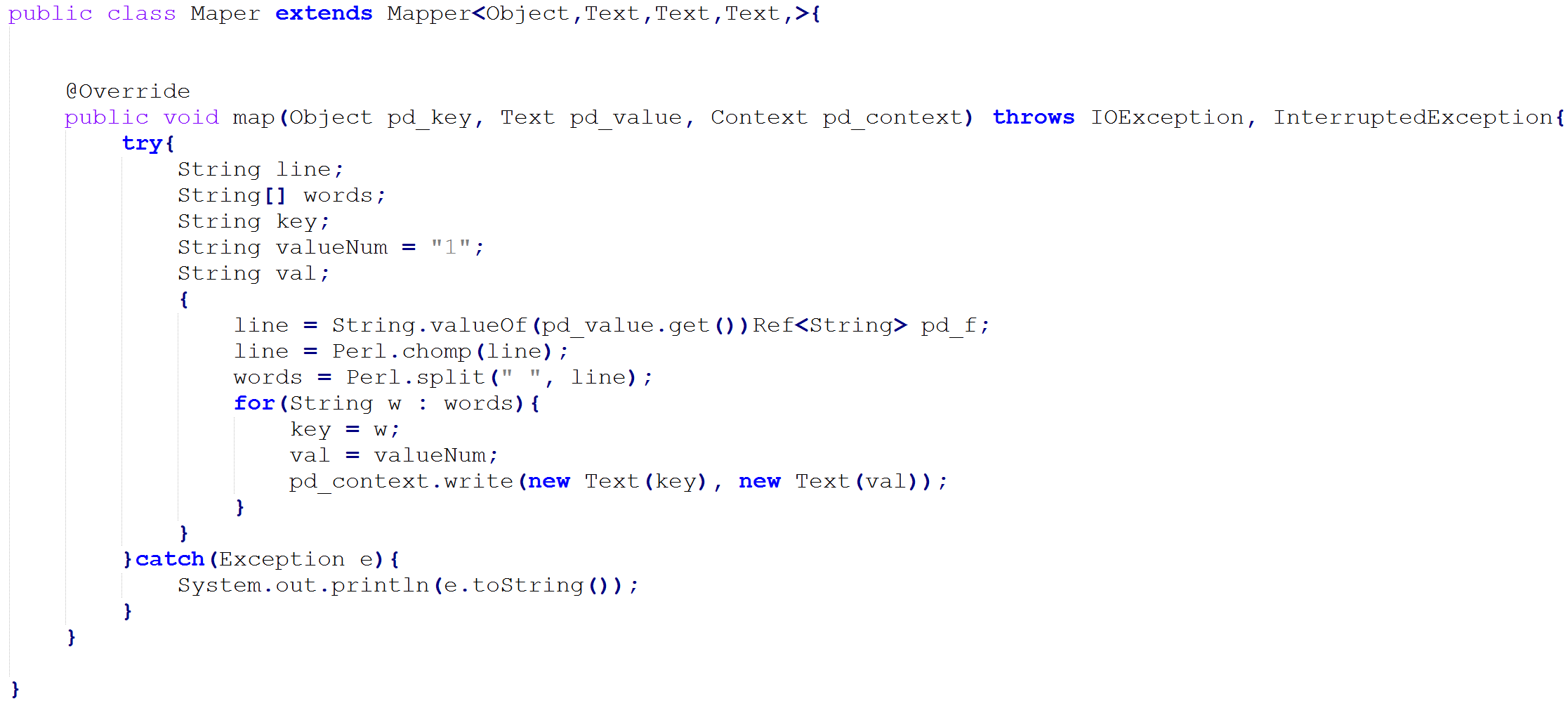


Figura 28: Mapper java

Como añadido final, junto a la etiqueta <mapper\_code>, hay definidos tres tipos. Esos tipos son opcionales y puede ignorarse, su función es cambiar el tipo de entrada y salida del Mapper. Por defecto si no se especifica nada se establecerán todos los tipos como cadenas(Text), el primer tipo hace referencia a cada línea leída del fichero, el segundo y el tercero hacen referencia a la clave y el valor respectivamente de la tupla de salida.

#### 8.5.8.2 Reducer

El Reducer es el encargado de combinar las tuplas clave/valor generadas por el Mapper con el objetivo de generar un resultado final. Una vez terminado se genera un nuevo conjunto de producción que es almacenado en el sistema de ficheros HDFS de Hadoop.

Al igual que ocurrió con el Mapper, el Reducer también necesita etiquetas para adaptar el código fuente. El principal problema que presenta el Reducer frente al Mapper es que su sintaxis es variable y es necesario proporcionar más libertad a la hora de su creación.

En primer lugar, debemos acotar el código del Reducer dentro de un bloque al igual que en el caso anterior: en este caso la etiqueta usada debe ser <reducer\_code> y el proceso de traducción es exactamente igual que el caso anterior, cambiando el nombre de la clase padre y su método.

Para traducir los scripts de reducción, se han identificado tres regiones clave para transformar el código secuencial al Recucer Hadoop. Una zona de definición de variables, la operación de reducción y por último las acciones una vez terminado con una clave y antes de empezar con la siguiente.

La primera región identificada con la etiqueta <reducer\_var> indica cuales son las variables que usaremos dentro del Reducer, la razón de esta definición es eliminar variables necesarias en Perl para aplicar transformaciones fuera de las operaciones de reducción y cambio de clave. En esta región es necesario identificar dos variables con una etiqueta especial cada una, <reducer\_key> y <reducer\_value>, que corresponden a la variable que almacenara la calve y el valor respectivamente. Con esta región logramos identificar las variables a escribir en el código final y ademas las variables que habrá que inicializar con clave y valor para cada iteración sobre una tupla en cada reducción.

A continuación, con la etiqueta <reducer\_op> definimos la región que contiene las operaciones de reducción, o lo que quiere decir lo mismo, todas las acciones que son realizadas para todos los valores con la misma clave.

Para terminar, la etiqueta <reducer\_change> define la región con el código ejecutado al realizar todas las operaciones de reducción con una clave. En la mayoría de los casos esta zona es usada para guardar el resultado de la operación de reducción antes de empezar con la siguiente. Al igual que con el Mapper, el Reducer también puede hacer uso del contexto para guardar sus datos, la sintaxis es idéntica a la del Mapper.

Al igual que el Mapper, el Reducer también puede cambiar el tipo de dato de su entrada y salida, que por defecto son cadenas. En este caso sería necesario definir cuatros tipos, uno más que el Mapper, los dos primeros para clave/valor de entrada y los otros dos para clave/valor salida.



Figura 29:Reducer Perl

En la Figura 29 podemos ver claramente la definición de un Reducer del famoso ejemplo WordCount que toma la salida de la operación Mapper y cuenta cuantos valores están asociados a cada clave.

Lo más importante antes de ver la traducción es darse cuenta de la importancia de la definición de regiones. Si nos fijamos la mayoría del código es necesario para limpiar líneas, separar clave de valor y gestionar el cambio de clave, lo cual en Hadoop ya no es necesario porque ya lo realizá el en nuestro lugar.

La traducción del Reducer es más compleja que la del Mapper. En este caso se almacena la traducción de cada uno de las regiones según se va procesando el fichero. Una vez terminado el bloque, se concatenan cada una de las regiones ignorando el código resultante de la traducción natural del bloque. Es importante realizar esta traducción porque, aunque el código secuencial ejecutado en paralelo produciría el mismo resultado, si nos fijamos ya en el código anterior, existen bloques condicionales que nunca se ejecutarán. Por ejemplo, oldkey eq newKey, si cada Reducer trabaja con una clave distinta, en el mismo Reducer nunca habra dos claves distintas y nunca se ejecutará el bloque correspondiente al else.

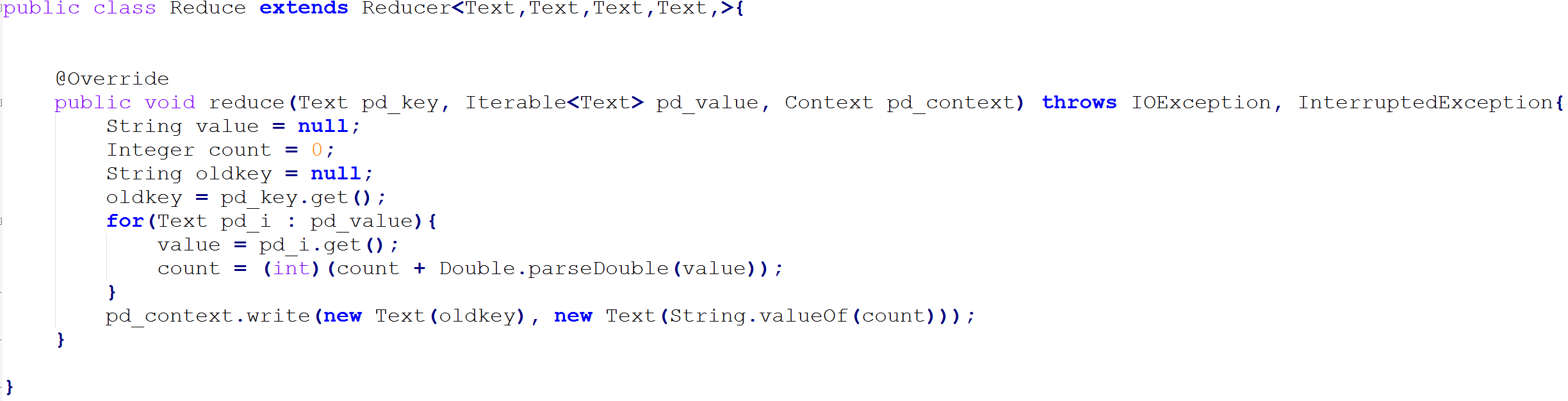


Figura 30: Reducer java

La Figura 30 muestra un Reducer muy compacto comparado con su versión secuencial, si no fijamos es fácil apreciar donde se han situado cada una de las regiones definidas con etiquetas. Las variables han sido copiadas al inicio de la función sin aplicar ningún cambio significativo. La operación de reducción ha sido introducida dentro de un bloque que recorre todos los valores asociados a un clave y para terminar la operación de cambio de clave se realiza antes de finalizar la función Reducer.

## Librería Java

Como se ha comentado en secciones anteriores, se ha creado una librería de soporte para el traductor, en este apartado se comentará su composición y las funcionalidades que contiene para hacer de soporte al traductor.

En primer lugar, la razón de la creación de una librería Java es proporcionar un nivel de abstracción entre el comportamiento nativo de Perl y el código necesario para imitarlo en Java. En otros casos era necesario la creación de una función para poder insertar una operación dentro de una expresión y en java son necesarias más operaciones como instanciación de objetos que obligarían a partir la expresión en varias líneas.

La librearía está formada por 9 clases que pueden ser agrupadas en dos tipos. El primer tipo ofrece un conjunto de funciones estáticas que son invocadas directamente sin necesidad de ninguna instanciación. El segundo grupo son objetos que son instanciados para simular tipos de datos de Perl que no existen en Java.

La implementación y diseño de la librería es muy simple y en esta sección se mostrará la función que realiza cada clase o los métodos que proporciona.

### 8.6.1. Clases para Tipos

Las Clases para tipos imitan el comportamiento de tipos básicos para Perl que en Java no existen como tal. La necesidad de estas clases es poder traducir una función sobre un tipo de dato por una función del mismo nombre y que realiza el mismo comportamiento. La otra razón importante es que la mayoría de las clases nativas Perl requieren generar una serie de Import para cada una de sus dependencias y de esta forma podemos ocultar todos ellos bajo la propia librería.

C:\Users\César\Desktop\Clases de Tipos.png

Figura 31: Clases para tipos

Como podemos ver en la Figura 31, existen 5 clases para la gestión de los tipos de Perl: PerlList, HashPer y PerlFile. Las clases Fread y Fwrite solo existen para dar soporte a los ficheros actuando PerlFile como fachada de ambos.

PerlFile y HashPerl existen sólo para reproducir el comportamiento de Perl en la asignación. Cuando Perl guarda un valor en una lista hash, la asignación devuelve el mismo elemento para volver a usarlo, en Java el comportamiento por defecto es devolver el elemento que existía en el mismo lugar antes de la asignación y en caso de no ser así devolver null. Las clases de la librería extienden las clases nativas de Java heredando el método para insertar y retornando el valor que añaden, dejando el comportamiento igual que en Perl.

Por otro lado, PerlFile, es el tipo de dato creado cuando se define una variable como fichero, luego según el uso que se le de a ese fichero, se creara un Fread si se abre para lectura o Fwrite si es para escritura. Los métodos de cada uno de ellos siguen la filosofía de trabajo con fichero en Java, primero abren el fichero y luego crea un buffer para aumentar el rendimiento, el proceso es realmente sencillo y no requiere de más acciones.

### 8.6.2. Clases para Funciones

Las clases para funciones son un conjunto de métodos estáticos agrupados en tres clases Regex, Pd, Perl.

La clase Regex contiene la funcionalidad para la ejecución de las expresiones regulares, es decir, un método para match, otro para sustitución y otro para transliteración. En cada función, se realiza una llamada al sistema para que Perl las evalúe la expresión y el resultado es retornado por la función. Este sistema fue adoptado por el tiempo requerido para estudiar toda la sintaxis de las expresiones regulares de Perl y buscar una librería Java para imitarlas. En el futuro, si se encuentra una solución mejor, sólo habría que remplazar el código dentro de las funciones sin más complicaciones.

La clase Pd, contiene las funciones relativas a comportamientos de Perl que no existen en Java de forma nativa o que necesaria evaluar la expresión en tiempo de ejecución debido a que no se puede llevar a cabo en tiempo de ejecución.

|  |  |
| --- | --- |
| Nombre | Descripción |
| toInteger | Castea un número a integer\* |
| toLong | Castea un número a Long\* |
| toFloat | Castea un número a Float\* |
| toDouble | Castea un número a Double\* |
| repeat | Operador Perl x que repite una cadena un numero de determinado de veces. |
| and | Realiza lo operación and con números enteros, retornando el segundo si el primero es falso o el primero en caso contrario. |
| or | Realiza lo operación and con números enteros, retornando el primero se es cierto o el segundo si el primero es falso. |
| xor | Realiza la opción lógica xor.\*\* |
| hash | Crea un hash desde un conjunto de expresiones. |
| range | Operador Perl .. que crea una lista entre dos intervalos que pueden ser números o cadenas. |
| copy | Realiza una copia superficial de una colección, usado cuando se igualan dos colecciones. |
| array | Crea un array desde una lista. |
| eval | Comprueba si una expresión es cierta. \*\* |

Tabla 8: Tabla de funciones auxiliares librería

\*Las funciones de casting son necesarias ante la imposibilidad de convertir un número sin saber si es un tipo primitivo o un objeto. Las cuatro funciones reciben como argumento un Number que representa cualquier número y al cual java puede convertir cualquier tipo número sea del tipo que sea, luego cada función retorna intValue(), longValue(), floatValue() o doubleValue().

\*\*Tanto eval como xor no pueden escribirse en el código porque requieren duplicar la expresión para realizar la operación y en algunos casos no podemos asegurar que la misma no implicaría un alto coste o que se obtendrá el mismo resultado. La función xor tiene que realizarse mediante or y and y en el caso de validar si algo es cierto es necesario realizar múltiples comprobaciones, por ejemplo, para la cadena se necesita comprobar si no es nula, vacía o es “0”. En el caso de operaciones del estilo “x++” o retornos de funciones, nunca se podrá replicar la expresión.

Por ultimo queda la clase Perl, esta clase contiene todas las funciones nativas especificadas en la sección 8.4.7. Su implementación no requiere diseño alguno, toda pueden realizarse mediante objetos y funciones ya definidas en Perl y que por claridad no deberían escribirse directamente en la traducción. Por ejemplo, si un script de Perl contiene ‘print(“Hola mundo”)’ la traducción seria ‘Perl.print(“Hola mundo”)’, si un programador de Perl mira el código Java, le será más fácil de entender que ver ‘System.out.print(“Hola mundo”)’.

# Validación y Pruebas

En este apartado tiene como finalidad abarcar la fase de validación y pruebas para comprobar que el sistema se comporta de forma correcta respecto a las especificaciones requeridas y además cumple con las expectativas de los directores de proyecto, es decir, nuestro cliente. Para poder lograr estos objetivos es necesario diseñar y poner en práctica un conjunto de casos de prueba que permitan validar si se cumplen cada uno de los requisitos funcionales y no funcionales.

La metodología de programación extrema tiene como uno de sus puntos importantes la fase de validación y pruebas. Las pruebas pueden dividirse en dos grandes grupos:

* **Pruebas Unitarias**: Las pruebas unitarias son aquellas creadas con la finalidad de verificar que el software se comporta de la manera esperada y siguiendo los requisitos, es decir, cada requisito se comprueba por separado sin interferencias de los demás.
* **Pruebas de Integración**: Las pruebas de integración permiten validar el funcionamiento del sistema en conjunto.

Las pruebas son un punto clave en la metodología de Programación Extrema. Uno de los motivos por los cuales se ha seleccionado esta metodología es que en la construcción de un traductor es muy importante comprobar que el código generado es válido y que tiene el mismo comportamiento que el código original. Además, debido a la dependencia entre módulos, es muy difícil detectar un error si no se identifica claramente al módulo al que pertenece.

## Pruebas Unitarias

Las pruebas unitarias tienen como objetivo comprobar el correcto funcionamiento del código y además validar que se cumplen cada uno de los requisitos especificados. Por esa razón se ha decidido separar las pruebas unitarias en dos bloques, el primer bloque estará dedicado a validar cada uno de los requisitos con pruebas sobre el traductor. Por otro lado, el segundo bloque estará orientado a la librería Java, estas pruebas estarán orientadas a comprobar que las funciones que ofrece al traductor tienen un comportamiento esperado según su especificación en el diseño.

### 9.1.1. Validación de requisitos.

Para la identificación de las pruebas se ha decidido usar el identificador P seguido del identificador del requisito a probar. Por ejemplo, el caso de prueba del requisito funcional 1 (RF-1), el caso de prueba se llamará PRF-1.

|  |  |
| --- | --- |
| PRF-1 | |
| Nombre Requisito | Implementación de interfaz por consola |
| Descripción prueba | Comprobar el medio de interacción del usuario con el traductor. |
| Resultado esperado | Toda la interacción del usuario con el traductor, tanto para realizar traducciones como para especificar opciones se realizan mediante una interfaz de consola. |
| Estado | Cumplido, el traductor hace uso del paquete argsparse de python para definir una interfaz por consola totalmente funcional. |

|  |  |
| --- | --- |
| PRF-2 | |
| Nombre Requisito | Definir scripts a traducir |
| Descripción prueba | Comprobar cómo se especifican los scripts a traducir. |
| Resultado esperado | El usuario puede seleccionar los scripts, independientemente de donde estén situados y de forma intuitiva. |
| Estado | Cumplido, la consola del traductor toma como argumento posicional la localización de cada uno de los archivos a traducir, la localización de los mismos es indiferente para el traductor. |

|  |  |
| --- | --- |
| PRF-3 | |
| Nombre Requisito | Definir carpeta de salida |
| Descripción prueba | Comprobar que el traductor permite elegir una localización para almacenar el resultado distinto de una carpeta por defecto. |
| Resultado esperado | El usuario puede elegir cualquier carpeta del sistema para almacenar la traducción. |
| Estado | Cumplido, en la interfaz por consola existe el paramento -out que permite seleccionar una carpeta para almacenar la traducción. |

|  |  |
| --- | --- |
| PRF-4 | |
| Nombre Requisito | Traducir variables básicas, array, hash y descriptores de fichero. |
| Descripción prueba | Comprobar que existe una traducción válida para cada uno de los anteriores tipos de datos. |
| Resultado esperado | El usuario puede definir cualquiera de los anteriores tipos de datos en sus scripts. |
| Estado | Cumplido, el traductor cuenta con una etiqueta para cada uno de los tipos de datos anteriores, además de incluir un tipo nuevo llamado lista para mejorar el acceso en caso de concatenación de arrays. |

|  |  |
| --- | --- |
| PRF-5 | |
| Nombre Requisito | Traducción de operaciones |
| Descripción prueba | Comprobar que existe una traducción válida para las operaciones Matemáticas, Lógicas, Binarias, Concatenación, Repetición y Asignación. |
| Resultado esperado | El usuario puede usar las operaciones anteriores en sus scripts. |
| Estado | Cumplido, el traductor puede traducir los operadores anteriores listados en la documentación de Perl [17]. |

|  |  |
| --- | --- |
| PRF-6 | |
| Nombre Requisito | Casting automático |
| Descripción prueba | El casting entre tipos se realiza de forma automática |
| Resultado esperado | El usuario puede asignar valores de tipos distintos sin tener que hacer ningún tipo de acción diferente a que fueran del mismo tipo. |
| Estado | Cumplido, el traductor contiene un módulo Casting con las funciones para convertir entre tipos, cada vez que haga falta una conversión de tipo solo hace falta invocar a la Liberia con la expresión, su tipo y el tipo destino. |

|  |  |
| --- | --- |
| PRF-7 | |
| Nombre Requisito | Traducción de estructuras de control |
| Descripción prueba | Existe una traducción válida para las estructuras de control: if-elsif-else, while, do while, for, foreach |
| Resultado esperado | El usuario puede usar las estructuras de control anteriores en sus scripts. |
| Estado | Cumplido, El analizador no solo puede traducir las estructuras de control típicas de los lenguajes de programación, sino que además tiene soporte para las estructuras únicas de Perl como son unless, until y do until que son las opuestas a if, while y do while. |

|  |  |
| --- | --- |
| PRF-8 | |
| Nombre Requisito | Traducción funciones nativas Perl |
| Descripción prueba | Existe una traducción válida para las funciones nativas especificadas en el FR-8. |
| Resultado esperado | El usuario puede llamar a las funciones nativas en Perl en sus scripts y en la traducción tendrá el mismo comportamiento sin necesidad de ningún trabajo por parte del usuario. |
| Estado | Cumplido, las llamadas a las funciones nativas en Perl se traducen por llamadas a una función con mismo nombre y comportamiento definidas en una librería java incluida con el traductor. |

|  |  |
| --- | --- |
| PRF-9 | |
| Nombre Requisito | Declaración de funciones propias en Perl |
| Descripción prueba | Comprobar si el usuario puede definir sus propias funciones en el código. |
| Resultado esperado | El usuario puede definir funciones en el código fuente y luego invocarlas. |
| Estado | Cumplido, el traductor proporciona unas etites para poder asignar tipos a las funciones del mismo modo que se hace con las variables, hay limitaciones en cuanto a la forma de pasar los argumentos, pero estas no limitan sus capacidades. |

|  |  |
| --- | --- |
| PRF-10 | |
| Nombre Requisito | Generar código Hadoop |
| Descripción prueba | El sistema puede generar código Hadoop paralelizando algoritmos secuenciales. |
| Resultado esperado | El usuario puede elegir una parte de un algoritmo para ser transformada en un Mapper o Reducer de Hadoop para ejecutar su código en dicha plataforma. |
| Estado | Cumplido, el traductor proporciona una serie de etiquetas para definir las áreas que se desean traducir a código Hadoop, el usuario solo tiene que etiquetas su código y el traductor realizara el resto. |

|  |  |
| --- | --- |
| PRF-11 | |
| Nombre Requisito | Sistema de error |
| Descripción prueba | Comprobar las formas en que el sistema informa al usuario en caso de ocurrir un error. |
| Resultado esperado | El usuario es consciente de que ha cometido un error, conoce su origen y el lugar donde ha ocurrido. |
| Estado | Cumplido, el traductor en caso de error, informará al usuario informando del fichero, línea, columna y causa del error. Además, se mostrará un fragmento del código señalando el inicio de la palabra que contiene el error siempre que proceda. |

|  |  |
| --- | --- |
| PRF-12 | |
| Nombre Requisito | Implementar sistema de depuración |
| Descripción prueba | El sistema deberá poder informar de su funcionamiento interno para facilitar el mantenimiento de la misma. |
| Resultado esperado | El sistema muestra el estado del analizador léxico o sintáctico con el fin de facilitar su verificación. |
| Estado | Cumplido, la interfaz de consola contiene comandos para la depuración del traductor, pudiendo mostrar tokens o fragmentos de código generado según se especifiquen las opciones. |

|  |  |
| --- | --- |
| PRF-13 | |
| Nombre Requisito | Verificar código generado |
| Descripción prueba | El analizador debe detectar la irregularidad de código muerto que, aunque sea correcto sintácticamente, el compilador de java no aceptara. |
| Resultado esperado | El traductor informa del error al usuario indicándole que su script contiene código muerto. |
| Estado | Cumplido, la interfaz de consola presenta una opción para verificar el código muerto a fin de realizar un mayor análisis sobre el código aun a costa del rendimiento de generación. |

|  |  |
| --- | --- |
| PRF-14 | |
| Nombre Requisito | Optimizar código generado |
| Descripción prueba | Comprobar si se puede generar una mejor traducción eliminados patrones de código redundante. |
| Resultado esperado | La traducción es más atractiva visualmente o tienen menos código redundante que la traducción normal. |
| Estado | Cumplido, la interfaz de consola tiene una opción para habilitar una serie de expresiones regulares para eliminar código redundante, usar notación diamante y eliminar paréntesis innecesarios. |

|  |  |
| --- | --- |
| PRF-15 | |
| Nombre Requisito | Traducir comentarios Perl |
| Descripción prueba | Comprobar si los comentarios en el código Perl se conservan en la traducción. |
| Resultado esperado | Los comentarios son interpretados por el traductor y adaptados a comentarios java. |
| Estado | Cumplido, la consola permite habilitar una opción para interpretar los comentarios como tokens del analizador y generar comentarios java con el mismo contenido. |

|  |  |
| --- | --- |
| PRF-16 | |
| Nombre Requisito | Formatea código java |
| Descripción prueba | Comprobar la salida del código generado por el traductor. |
| Resultado esperado | El traductor genera código identado los bloques encerrados entre llaves para mejorar la visualización. |
| Estado | Cumplido, una vez generada la traducción, el traductor escanea las llaves en el código fuente para ir identando el código antes de imprimirlo. |

|  |  |
| --- | --- |
| PRF-17 | |
| Nombre Requisito | Generar paréntesis funciones |
| Descripción prueba | Comprobar si el analizador interpreta funciones con los argumentos sin paréntesis. |
| Resultado esperado | El analizador realiza la misma acción que si los argumentos tuvieran paréntesis. |
| Estado | Cumplido, el analizador comprueba el número de argumentos necesarios por la función y añade los paréntesis antes de enviar los tokens al analizador sintáctico. |

|  |  |
| --- | --- |
| PRF-18 | |
| Nombre Requisito | Generar main automático |
| Descripción prueba | Comprobar si el código global puede ser ejecutado en un main. |
| Resultado esperado | El código de salida tiene la traducción del código global en un main. |
| Estado | Cumplido, la interfaz por consola tiene una opción para activar el main, en caso contrario el código global se generar en un bloque estático. |

|  |  |
| --- | --- |
| PRNF-1 | |
| Nombre Requisito | Portabilidad |
| Descripción prueba | Comprobar si el traductor se comporta igual en Windows, Linux y Mac |
| Resultado esperado | El sistema operativo no influye en el traductor. |
| Estado | Cumplido, el traductor está programado en Python, al ser un lenguaje interpretado no es dependiente del sistema operativo. |

|  |  |
| --- | --- |
| PRNF-2 | |
| Nombre Requisito | Paradigma orientado a objetos |
| Descripción prueba | Comprobar cómo están organizadas los distintos segmentos del código. |
| Resultado esperado | La implementación del código fuente debería estar desarrollada con el paradigma orientado a objetos. |
| Estado | Cumplido, el código está programado usando el paradigma orientado a objetos. |

|  |  |
| --- | --- |
| PRNF-3 | |
| Nombre Requisito | Extensibilidad |
| Descripción prueba | Comprobar si la herramienta permite añadir nuevas capacidades en el futuro. |
| Resultado esperado | Los cambios futuros se pueden aplicar sin afectar a las funcionalidades existentes. |
| Estado | Cumplido, para añadir una nueva funcionalidad sólo es necesario añadir la regla en la gramática, su comprobación y generación de condigo. |

|  |  |
| --- | --- |
| PRNF-4 | |
| Nombre Requisito | Tecnología apropiada |
| Descripción prueba | Comprobar las tecnologías usadas para el desarrollo |
| Resultado esperado | El traductor sigue los principios de construcción de un traductor, es decir, analizador léxico sintáctico, etc. |
| Estado | Cumplido, el traductor este construido con un módulo python para la construcción de analizadores léxicos y sintácticos llamado PLY. |

### 9.1.2. Pruebas librería Java

La librería incorpora funciones que el traductor usa para simplificar el código generado, algunas de las funciones son muy simples y no requieren pruebas para validar que funcionan correctamente.

Las funciones nativas necesitan pasar una validación para asegurar su correcto funcionamiento, en algunos casos es necesario trabajar dividiendo cadenas o arrays donde es muy fácil equivocarse con los índices de corte. Por esa razón se han definido casos de prueba con Junit para comprobar si la entrada y la salida de la función coincide con la implementación nativa de Perl.

Dado que un requisito del traductor es que el código debe funcionar correctamente en el intérprete de Perl, tenemos asegurado que los argumentos de las funciones siempre serán válidos. Por otro lado, las funciones no contienen bloques condicionales lo que nos facilita la definición de las pruebas, una prueba es suficiente para validar toda la cobertura del código.

Para la especificación del diseño de pruebas se ha optado por definir una tabla con la entra y la salida, en la entrada se especifica el valor de los argumentos y en la salida, el retorno y el nuevo valor si algún argumento cambia su valor.

|  |  |
| --- | --- |
| print | |
| Entrada | args: (“cad1”, “cad2”, “cad3”) |
| Salida | return: 1 |

|  |  |
| --- | --- |
| split | |
| Entrada | regex=”,” cad=”1,2,3” |
| Salida | return: (“1”,”2”,”3”) |

|  |  |
| --- | --- |
| chop | |
| Entrada | cad=”Tardis” |
| Salida | return: “T”, cad=”ardis” |

|  |  |
| --- | --- |
| chomp | |
| Entrada | cad=”Tardis\r\n” |
| Salida | return: 2, cad=”Tardis” |

|  |  |
| --- | --- |
| System | |
| Entrada | cad=”echo 1” |
| Salida | return: 0 |

|  |  |
| --- | --- |
| System | |
| Entrada | cad=”asdf” |
| Salida | return: -1 |

|  |  |
| --- | --- |
| Sort | |
| Entrada | Array=(2,3,1) |
| Salida | return: (1,2,3) |

|  |  |
| --- | --- |
| Sort | |
| Entrada | Array=(2,3,1), cmp=(a,b)->{return b>a;} |
| Salida | return: (3,2,1) |

|  |  |
| --- | --- |
| uc | |
| Entrada | cad=”hola” |
| Salida | return: “HOLA” |

|  |  |
| --- | --- |
| lc | |
| Entrada | cad=”HOLA” |
| Salida | return: “hola” |

|  |  |
| --- | --- |
| ucfirst | |
| Entrada | cad=”hola” |
| Salida | return: “Hola” |

|  |  |
| --- | --- |
| lcfirst | |
| Entrada | cad=”HOLA” |
| Salida | return: “hOLA” |

|  |  |
| --- | --- |
| delete | |
| Entrada | hash={“key”=>”value”}, key=”key” |
| Salida | return: “value”, hash={} |

|  |  |
| --- | --- |
| join | |
| Entrada | sep=”,”,array={“1”,”2”,”3”} |
| Salida | return: “1,2,3” |

|  |  |
| --- | --- |
| keys | |
| Entrada | hash={“a”=>”1”, “b”=>”2”, “c”=>”3”,} |
| Salida | return: (“a”,”b”,”c”)\* |

\*Solo se predice elemento no su orden

|  |  |
| --- | --- |
| values | |
| Entrada | hash={“a”=>”1”, “b”=>”2”, “c”=>”3”,} |
| Salida | return: (“1”,”2”,”3”)\* |

\*Solo se predice elemento no su orden

|  |  |
| --- | --- |
| length | |
| Entrada | cad=”HOLA” |
| Salida | return: 4 |

|  |  |
| --- | --- |
| substr | |
| Entrada | cad=”hola mundo”, init=4 |
| Salida | return: “ mundo” |

|  |  |
| --- | --- |
| substr | |
| Entrada | cad=”hola mundo”, init=4, len=1 |
| Salida | return: “ ” |

|  |  |
| --- | --- |
| substr | |
| Entrada | cad=”hola mundo”, init=4, len=1, repl=” en el ” |
| Salida | return: “ ”, cad=“hola en el mundo” |

|  |  |
| --- | --- |
| push | |
| Entrada | array=(2,3,4), elem=5 |
| Salida | return: (2,3,4,5) |

|  |  |
| --- | --- |
| pop | |
| Entrada | array=(2,3,4,5) |
| Salida | return: (2,3,4) |

|  |  |
| --- | --- |
| unshift | |
| Entrada | array=(2,3,4), elem=1 |
| Salida | return: (1,2,3,4) |

|  |  |
| --- | --- |
| shift | |
| Entrada | array=(1,2,3,4) |
| Salida | return: (2,3,4) |

|  |  |
| --- | --- |
| splice | |
| Entrada | array=(1,2,3,4), init=1 |
| Salida | return: (2,3,4), array=(1,) |

|  |  |
| --- | --- |
| splice | |
| Entrada | array=(1,2,3,4), init=1,len=1 |
| Salida | return: (2,), array=(1,3,4) |

|  |  |
| --- | --- |
| splice | |
| Entrada | array=(1,2,3,4), init=1,len=1, repl=(5,6) |
| Salida | return: (2,), array=(1,5,6,3,4) |

## Pruebas de Integración

Las pruebas de integración verifican que las diferentes partes del algoritmo realizan su tarea cuando trabajan en equipo. Las pruebas unitarias no se usaron directamente sobre el código fuente del traductor puesto que la disposición de los módulos los aísla en cuento a errores. Por ejemplo, un fallo en el módulo Casting, hará que el código no esté bien casteado, pero a ojos de otro módulo se asume que el código ya contiene un tipo correcto. De esta forma los módulos trabajan de forma independiente por un resultado común, de forma que probándolos juntos o separado, no interferirían entre ellos.

Para asegurarnos de que se genera código correctamente, es necesario probar todas las reglas del analizador sintáctico, que son las encargadas de llamar a los distintos módulos, de forma que si se genera código correcto podemos asumir que el módulo está correcto. Para asegurarse realmente de que el código generado es correcto, se compilara cada traducción con javac, si el compilador genera el .class correspondiente podemos asumir que la prueba ha sido satisfactoria.

Las diferentes pruebas de integración son identificadas de manera secuencial mediantelas siglas PI-N donde N se incrementa con cada prueba. Los ficheros de prueba se almacenarán en el repositorio en una carpeta perdoop2-test.

|  |  |
| --- | --- |
| PI-1 | |
| Nombre prueba | Prueba Acceso |
| Descripción prueba | Se crea un script con los accesos a las distintas tipas de colecciones con un distinto nivel de profundidad: Array, Array de Array, Hash, Hash de Hash, List… |
| Fichero de prueba | testAcceso.pl |
| Estado | Traducción correcta y Compilado |

|  |  |
| --- | --- |
| PI-2 | |
| Nombre prueba | Prueba Bloques |
| Descripción prueba | Se crea un script donde se definen todos los tipos de bloques: if-elsif-else, unless, while, until, do while, do until, for, foreach. |
| Fichero de prueba | testBloques.pl |
| Estado | Traducción correcta y Compilado |

|  |  |
| --- | --- |
| PI-3 | |
| Nombre prueba | Prueba Casting |
| Descripción prueba | Se crea un script donde existe una asignación entre cada tipo de dato para probar todos los castings posibles. |
| Fichero de prueba | testCasting.pl |
| Estado | Traducción correcta y Compilado |

|  |  |
| --- | --- |
| PI-4 | |
| Nombre prueba | Prueba Copia |
| Descripción prueba | Se crea un script donde se crean asignaciones entre colecciones para comprobar cuando hay que referenciar colecciones y cuando realizar una copia. |
| Fichero de prueba | testCopia.pl |
| Estado | Traducción correcta y Compilado |

|  |  |
| --- | --- |
| PI-5 | |
| Nombre prueba | Prueba Ficheros |
| Descripción prueba | Se crea un script para probar el manejo de ficheros, abrir, cerrar, leer, escribir… |
| Fichero de prueba | testFicheros.pl |
| Estado | Traducción correcta y Compilado |

|  |  |
| --- | --- |
| PI-6 | |
| Nombre prueba | Prueba Funciones Usuario |
| Descripción prueba | Se crea un script donde se definen varias funciones personalizadas y luego son invocadas. |
| Fichero de prueba | testFuncionesUsuario.pl |
| Estado | Traducción correcta y Compilado |

|  |  |
| --- | --- |
| PI-7 | |
| Nombre prueba | Prueba Inicialización |
| Descripción prueba | Se crea un script donde se inicializan colecciones de todas las formas posibles. |
| Fichero de prueba | testInicializacion.pl |
| Estado | Traducción correcta y Compilado |

|  |  |
| --- | --- |
| PI-8 | |
| Nombre prueba | Prueba MultiAsignacion |
| Descripción prueba | Se crea un script donde se realizan multiasignaciones para probar si se traducen atómica y correctamente. |
| Fichero de prueba | testMultiAsignacion.pl |
| Estado | Traducción correcta y Compilado |

|  |  |
| --- | --- |
| PI-9 | |
| Nombre prueba | Prueba Nativas |
| Descripción prueba | Se crea un script donde se hace una llamada a todas las posibles funciones nativas para comprobar si se traduce correctamente la llamada. |
| Fichero de prueba | testNativas.pl |
| Estado | Traducción correcta y Compilado |

|  |  |
| --- | --- |
| PI-10 | |
| Nombre prueba | Prueba OpAritmetica |
| Descripción prueba | Se crea un script donde se prueban todas las operaciones aritméticas. |
| Fichero de prueba | testOpAritmetica.pl |
| Estado | Traducción correcta y Compilado |

|  |  |
| --- | --- |
| PI-11 | |
| Nombre prueba | Prueba OpAsignacion |
| Descripción prueba | Se crea un script donde se prueban todas las operaciones de asignación. |
| Fichero de prueba | testOpAsignacion.pl |
| Estado | Traducción correcta y Compilado |

|  |  |
| --- | --- |
| PI-12 | |
| Nombre prueba | Prueba OpBinario |
| Descripción prueba | Se crea un script donde se prueban todas las operaciones binarias. |
| Fichero de prueba | testOpBinario.pl |
| Estado | Traducción correcta y Compilado |

|  |  |
| --- | --- |
| PI-13 | |
| Nombre prueba | Prueba OpComparacion |
| Descripción prueba | Se crea un script donde se prueban todas las operaciones de comparación. |
| Fichero de prueba | testOpComparacion.pl |
| Estado | Traducción correcta y Compilado |

|  |  |
| --- | --- |
| PI-14 | |
| Nombre prueba | Prueba OpLogico |
| Descripción prueba | Se crea un script donde se prueban todas las operaciones lógicas. |
| Fichero de prueba | testOpLogico.pl |
| Estado | Traducción correcta y Compilado |

|  |  |
| --- | --- |
| PI-15 | |
| Nombre prueba | Prueba PostBloques |
| Descripción prueba | Se crea un script donde se pruebas los bloques que actúan sobre sentencias: if, unless, while, until, foreach |
| Fichero de prueba | testPostBloques.pl |
| Estado | Traducción correcta y Compilado |

|  |  |
| --- | --- |
| PI-16 | |
| Nombre prueba | Prueba Regex |
| Descripción prueba | Se crea un script donde se prueban los tres tipos de expresiones regulares. |
| Fichero de prueba | testRegex.pl |
| Estado | Traducción correcta y Compilado |

|  |  |
| --- | --- |
| PI-17 | |
| Nombre prueba | Prueba Tipos |
| Descripción prueba | Se crea un script donde se prueban todas las etiquetas de tipo en la inicialización de variables en todas las posiciones permitidas. |
| Fichero de prueba | testTipos.pl |
| Estado | Traducción correcta y Compilado |

|  |  |
| --- | --- |
| PI-18 | |
| Nombre prueba | Prueba Mapper |
| Descripción prueba | Se crea un script donde se transforma un código secuencial a un Mapper. |
| Fichero de prueba | testMapper.pl |
| Estado | Traducción correcta y Compilado |

|  |  |
| --- | --- |
| PI-19 | |
| Nombre prueba | Prueba Reducer |
| Descripción prueba | Se crea un script donde se transforma un código secuencial a un Reducer. |
| Fichero de prueba | testMapper.pl |
| Estado | Traducción correcta y Compilado |

# Conclusión

En este trabajo de fin de grado se ha planteado la creación de una herramienta que permita traducir un script escrito en Perl a código fuente Java. La diferencia entre ambos lenguajes ha resultado en una etapa de análisis de alcance y diseño muy dura. Por un lado, tenemos a Perl, un lenguaje interpretado y libre de tipos que, en la mayoría de los casos, aunque el código sea ilógico, el siempre encuentra la forma de salir del paso. Por otro lado, esta Java un lenguaje que requiere una compilación y con una fuerte asociación de tipos siendo además es muy severo en cuanto a su ejecución, a la mínima envía una excepción.

Por otro lado, no todo el código fuente debía ser traducido a Java, uno de los objetivos principales de este proyecto es transformar código secuencial para el Procesamiento de Lenguaje Natural (PLN) en un programa java compatible con las tecnologías Big Data (Hadoop). Los códigos debían funcionar correctamente sin tener que obligar al programador a conocer el funcionamiento de la plataforma en cuanto a implementación y quedándole como una responsabilidad el compilar el código resultante para luego proceder con su ejecución.

El principal objetivo durante todo el desarrollo del proyecto, ha sido dar soporte a toda la sintaxis que fuera posible abarcar. Muchas características, sobre todo las añadidas en último lugar, no gozan de toda la potencia y versatilidad que podrían haber tenido con más tiempo de desarrollo. Aun así, fue posible añadir todas las características que se esperaban y con una calidad más que aceptable dentro del ámbito abarcado.

Durante el desarrollo de la implementación, han surgido diversos problemas debido a que es imposible traducir un código ambiguo en el que solo en el momento de su ejecución es posible averiguar cuál es su funcionamiento. Por esa razón en muchos casos se ha optado por limitar la sintaxis a una de las opciones, obviando las demás u obligando al programador a especificarlas de una forma distinta.

Por último, es importante considerar que la herramienta no terminará su desarrollo con la finalización de este proyecto. Los objetivos especificados en el anteproyecto solo abarcaban el tiempo disponible, y en un futuro podrían introducirse nuevas mejoras cuyo tiempo de desarrollo fuese demasiado complejo como para ser abordado por este trabajo de fin de grado.

# Bibliografía

1. Perldoop [En línea]. Available: https://github.com/citiususc/perldoop
2. Tom Christiansen, brian d foy & Larry Wall with Jon Orwant, Programming Perl 4h Edition, O’REILLY
3. Tom White, Hadoop The Definitive Guide, O’REILLY
4. Manual Perldoop, Sección 2.2 [En línea]. Available: https://github.com/citiususc/perldoop/blob/master/doc/UserManual\_Perldoop\_EN.pdf
5. Manual Perldoop, Sección 2.3 [En línea]. Available: https://github.com/citiususc/perldoop/blob/master/doc/UserManual\_Perldoop\_EN.pdf
6. PMBOK Guides,A Guide To The Project Management Body Of Knowledge Project, Management Institute, 2004.
7. IEEE830. recommended practice for software requirements speciﬁcations. Technical report, 1998
8. Douglas Bell. Software Engineering for Students. Addison-Wesley, 2005.
9. Atlassian Software, «BitBucket,» [En línea]. Available: https://bitbucket.org/
10. PLY (Python Lex-Yacc) [En línea]. Available: http://www.dabeaz.com/ply/
11. InfoJobs, «InfoJobs,» [En línea]. Available: https://www.infojobs.net/.
12. Eclipse IDE [En línea]. Available: https://eclipse.org/
13. Netbeans IDE [En línea]. Available: https://netbeans.org/
14. SourceTree [En línea]. Available: https://www.sourcetreeapp.com
15. Draw.io [En línea]. Available: https://www.draw.io/
16. Perl 5 documentation [En línea]. Available: http://perldoc.perl.org/perlop.html
17. Perl 5 Operators [En línea]. Available: http://www.tutorialspoint.com/perl/perl\_operators.htm