Capítulo 5

Conclusiones

El correspondiente trabajo final de Licenciatura en Ciencias de la Computación, en los tres primeros capítulos se explicaron conceptos relacionados a la temática de Comprensión de Programas (CP) y Análisis de Identificadores (ids). El estudio de estos temas tiene como objetivo ubicar al lector en el contexto de estas temáticas y además brindar un estado de arte acorde a las mismas. A partir del estudio del estado del arte de las técnicas de análisis de ids, se detectó que no todas estas técnicas están implementadas en herramientas automáticas. Construir una herramienta con tales características facilita entender el propósito de los ids en los códigos. Cuando una persona ajena a un código de software, entiende con facilidad el significado de los ids, comprende mejor el sistema de estudio, por ende la construcción de esta herramienta es un aporte directo al área de la CP. Teniendo en cuenta esta ausencia de implementaciones, se llevó a cabo el desarrollo de una herramienta, que a través de una interfaz amigable le ayuda al usuario a analizar los ids presentes en los códigos, mediante la ejecución de algunas de las técnicas estudiadas. Esta herramienta llamada IDA fue descripta en el capítulo anterior, en este último capítulo se describirán aspectos generales sobre esta herramienta, se brindarán algunas conclusiones y se proponen trabajos futuros que se pueden realizar.

5.1. Herramienta Identifier Analizer (IDA)

En este apartado, se describen características generales de la herramienta IDA que servirán de introducción para detallar distintas propuestas relacionadas a trabajos futuros.

Se ha elaborado una herramienta IDA (Identifier Analizer), la misma permite extraer con facilidad los elementos estáticos presentes en los códigos escritos en JAVA. Estos elementos son, los ids como objetos principales, los comentarios y literales. IDA captura estos elementos por medio de un Analizador Sintáctico (AS) (ver capítulo 4 para más detalles).

Debido a la complejidad que demanda construir este tipo de AS, para acotar las tareas en el desarrollo del mismo, unicamente se programó al AS para que extraiga los ids en su punto de declaración (Ejemplo: int i;) y no en la referencias del mismo (Ejemplo: i=i+1;). En el caso de los comentarios y literales, el AS los extrae en forma completa. La aclaración hecha en este párrafo es para proponer una mejora al respecto, la misma se describe en la próxima sección.

Finalizada la captura de los elementos por medio del AS, en el próximo paso IDA le permite al usuario escoger entre dos técnicas de división de ids, que se describen a continuación:

Algoritmo Greedy: Lo primero que realiza el algoritmo es dividir al id, agregando un espacio en blanco en posiciones del ids (si existen), que se destaquen por una marca de separación entre dos palabras (hard-words). Ejemplo: el guión bajo in_put → in put, o camel-case inPut → in put. El resto de las palabras resultantes (softwords), se someten a un proceso recursivo de división. Este proceso, se lleva a cabo con dos rutinas una comienza desde el principio de la palabra (**buscarPrefijo**) y el otra desde el final (**buscarSufijo**). Luego cada una va tomando de a un carácter en forma progresiva y consultando si esa porción de la palabra pertenece a algún diccionario. En caso de ser afirmativo, procede a separarla del resto y continuar con el análisis. El resultado de la rutina (entre **buscarPrefijo** y **buscarSufijo**) que mayor divisiones tenga será escogido como candidato (ver capítulo 3 - sección **??**).

Algoritmo Samurai: De manera similar a la técnica Greedy, al principio se divide al id con espacios en blanco, en lugares donde haya marcas de separación (si existen) que destaquen la división entre dos palabras del id (guión bajo, camel-case). El resto de las palabras se procesan por dos rutinas, la primera se encarga de buscar si el nombre del id posee la variante camel-case. Ejemplo: INPut → in put. La segunda rutina itera sobre la palabra tomando de a dos partes de forma progresiva desde el primer caracter en adelante, luego una función de score (puntaje) le dará un puntaje a cada parte, si son lo suficientemente altos se procederá a dividir entre ambas partes, sino continuará analizando el resto (ver capítulo 3 - sección **??**).

Una vez que fueron divididos los ids, las distintas partes resultantes se someten a un proceso de expansión por medio de una técnica, que se explica a continuación:

Algoritmo de Expansión Básica: Este algoritmo se encarga de tomar palabras que resultaron producto de la separación de ids. En caso de que estas palabras estén abreviadas, el algoritmo de expansión las expande a su correspondiente palabra completa, para ello se utilizan los comentarios o literales capturados del código por medio del AS, si los mismos son escasos, se recurre a un diccionario de palabras en Inglés como último recurso (ver capítulo 3 - sección **??**).

Los resultados conseguidos, producto de las técnicas descriptas anteriormente se muestran en tablas para que el usuario pueda visualizar y sacar conclusiones. Los casos de estudio del capítulo 4 mostraron que la división Samurai tiene mejor comportamiento que la Greedy, esto es lógico debido que este último, es un algoritmo muy primitivo[**?**, **?**, **?**]. Con respecto a las expansiones, el algoritmo correspondiente, si bien también es sencillo y no cuenta con procesos complejos, expande los ids de manera aceptable, de acuerdo a la información que captura el AS por medio de los comentarios y literales. Las palabras completas producto de la expansión de ids abreviados, brindan información sobre los conceptos del Domino del Problema ubicados en el

programa analizado (ver Casos de Estudio - capítulo 4). Esta información es crucial para entender el propósito de los ids en el código y por ende facilita entender el programa de estudio. De esta manera, se hace un aporte al área de la CP en la búsqueda del principal objetivo, que es relacionar el Domino del Problema con el Dominio del Programa.

5.2. Trabajos Futuros

En esta sección se describen propuestas vinculadas a trabajos futuros. Se tomará como puntapié inicial el actual desarrollo de la herramienta IDA para proponer mejoras y/o expansiones a la misma, a continuación se describen cada una de ellas:

- Ampliar Captura del Analizador Sintáctico.
- Implementar otro Algoritmo de Expansión.
- Acoplar a Entornos de Desarrollos.
- Traducción de Identificadores en Código.

5.2.1. Ampliar Captura del Analizador Sintáctico

En la sección anterior se explicó que el AS únicamente captura ids en su punto de declaración, mientras que en el resto de los lugares en código de estudio se ignoran. Si se amplia el AS para que capture la totalidad de los ids presentes en un archivo JAVA, por un lado se le brindará al usuario mayor información estática asociada a los ids, y por el otro se perfeccionarán las técnicas de análisis de ids. Sin duda, la técnica más beneficiada en este sentido será Samurai, este beneficio viene gracias a las tablas de frecuencias que este algoritmo utiliza en la función de score (ver capítulo 3 - sección **??**). Las dos tablas de frecuencias de aparición de tokens[1] que son utilizadas

[1]El concepto "token", en este contexto, significa las palabras que están contenidas en literales, comentarios e ids, este último necesita un proceso especial, para más detalles ver cap. 3 - sección **??**

por el Algoritmo Samurai, son la tabla de frecuencias local y la tabla de frecuencias global. Ambas poseen una doble entrada, una es el token y la otra es la frecuencia de aparición de cada token. En la tabla local se considera la frecuencia de aparición de tokens en el archivo de estudio actual, mientras que la global se asocia a un conjunto grande de programas. A continuación, se explican las mejoras que recibiría cada tabla, si el AS se amplia:

Tabla de Frecuencia Local: Intuitivamente, dentro esta tabla, cada token correspondiente a un id en el código de estudio tendrá la frecuencia de aparición local mucho más precisa, dado que se capturan todas las apariciones del id.

Tabla de Frecuencia Global: Esta tabla originalmente fue construida por los autores, a partir del análisis de 9000 programas JAVA (ver capítulo 3 - sección **??**), y la misma no esta disponible en la web. Por lo tanto, se tomó la iniciativa de construir una aproximación. Esta aproximación se efectuó corriendo el mismo AS que se utiliza en IDA para 20 programas JAVA tomados como muestra. Teniendo en cuenta que este AS se va ampliar para que capture más ids, también se mejorará la aproximación en la construcción de la tabla de frecuencias globales. Una propuesta extra de mejora en la tabla de frecuencias globales, es ejecutar este AS ampliado, pero sobre un lote de muestra superior a 20 programas, de esta manera se obtendrá una mejor aproximación a la original.

5.2.2. Implementar otro Algoritmo de Expansión

Esta propuesta consiste en implementar una nueva técnica de análisis de ids en IDA, más precisamente un nuevo algoritmo de expansión. Este algoritmo es AMAP (Automatically Mining Abbreviation Expansions in Programs) descripto en el capítulo 3 sección **??**. Esta técnica, no necesita de diccionarios con palabras en Inglés como el caso de el algoritmo básico de expansión y observa gradualmente en el código los comentarios y literales presentes partiendo desde el lugar del id que se desea expandir. Además, AMAP no tiene el problema que posee el algoritmo básico cuando no sabe que opción elegir

ante muchas posibilidades de expansión, ya que posee criterios de selección ante múltiples opciones de expansión. AMAP brinda la posibilidad de entrenarse con un conjunto de programas pasado como entrada para recopilar más palabras y mejorar aún más la precisión de la expansión. Esta propuesta en IDA mejora las expansiones y brinda al usuario nuevas opciones de análisis de ids.

5.2.3. Traducción de Identificadores en Código

Para ubicarse en el contexto de esta mejora, IDA tiene un panel en donde se muestra el código leído del archivo, para que el usuario lo visualice. Este código resalta con color los ids cuando se seleccionaba un id en la tabla correspondiente (también lo hace con los comentarios y los literales, ver capítulo 4 para más detalles). Una propuesta de mejora en la herramienta IDA, consiste en traducir los ids que se muestran en este código y luego generar un nuevo archivo con el nuevo código. Esta traducción implica reemplazar cada id que se ubica en código por la expansión que fue llevada a cabo, dado que puede existir más de una alternativa de expansión por cada id, lo que se permitirá es que el usuario pueda elegir entre las distintas alternativas la que mejor le parezca. De esta forma, se obtendrá un código más legible y ayudará a comprenderlo más fácilmente. Luego el nuevo código con los ids expandidos se podrá guardar en un nuevo archivo de salida JAVA, este nuevo archivo será funcionalmente equivalente al original. Esta idea de traducción y creación de un nuevo archivo, fue tomada de una de las características que posee la herramienta Identifier Restructuring Tool cuyos autores son Tonella y Caprile (descripta en el capitulo 3 sección **??**), ya que esta herramienta realiza una traducción similar de ids en un nuevo código de salida.

5.2.4. Acoplar a entornos de desarrollos

Una extensión futura interesante para la herramienta IDA, es la posibilidad de acoplarla en forma de extensión (plugin) a un entorno de desarrollo integrado como es el caso de NetBeans o Eclipse. Esto permitiría que el usuario abra un proyecto JAVA e inmediatamente con IDA expanda los ids para

mejorar la comprensión. Esta propuesta en parte es similar a una de la características de la herramienta Identifier Dictionary (IDD), que fue desarrollada por Deissenboeck y Pizka (descripta en el capítulo 3 sección **??**). La herramienta IDD es un plugin de eclipse que al compilar un proyecto en JAVA, automáticamente captura y enumera dentro de una tabla los ids presentes en el proyecto, luego el usuario puede renombrar cada id a una forma más comprensiva.

La herramienta IDA tendría un plugin similar a IDD, solo que el renombre de los ids es automático a palabras completas mediante la implementación de la técnicas de análisis, y el usuario solo deberá intervenir para determinar que expansión es la más adecuada.