Capítulo 5

Conclusiones

La motivación de desarrollar la herramienta IDA, esta dada por la ausencia de herramientas con similares características. No abundan herramientas que posean interfaz amigable con el usuario y ejecuten técnicas que analicen ids de un código pasado entrada. Tampoco existen muchas implementaciones que extraigan ids, dividan ids y expandan abreviaturas en nombres de ids.

El uso del parser construido con ANTLR, permite extraer con facilidad los elementos estáticos presentes en el código que son necesarios para los algoritmos que analizan de ids. Estos elementos son, los ids como objetos principales y luego los comentarios, literales y documentación JAVA doc.

Cabe destacar que la herramienta IDA tiene implementada dos técnicas de división, Greedy y Samurai. La primera necesita consultar un diccionario de palabras en Inglés y un listado genérico de abreviaciones conocidas para llevar a cabo sus tareas. Ambas listas ocupan mucho espacio de almacenamiento y se utiliza una base de datos para hacer las consultas más eficientes.

En cambio, el algoritmo Samurai divide los ids mediante la utilización de recursos propios del código. Estos recursos son, los comentarios, los literales y documentación JAVA Doc que son extraídos mediante el parser antes mencionado. Con estos recursos, se arma un listado de frecuencias de aparición de palabras que son usadas en la función de scoring (ver sección nn). Por otro lado, suele ocurrir que estos recursos son escasos, por ende los autores decidieron armar un listado de palabras perteneciente a un conjunto amplio

de programas escritos en JAVA. Este listado, no solo ocupa menos espacio que los diccionarios de Greedy sino que están constituidos con palabras más adecuadas al ámbito de las ciencias de la computación. Esto implica que la división sea más eficiente y por ende que después la expansión sea más precisa.

Por otro lado, el algoritmo de expansión básico emplea los mismos diccionarios de palabras que utiliza Greedy, pero con la diferencia que consulta previamente la lista de frases capturadas del código, dando la preferencia a esta lista primero. La lista de frases se arma en función de los comentarios, literales y documentación JAVA Doc extraídos con el parser explicado al principio. Este algoritmo tiene el problema que ante múltiples alternativas de expansión, no sabe elegir una única opción.

Como trabajo futuro se planea implementar nuevas técnicas de análisis de ids en IDA. Una de ellas es AMAP (Automatically Mining Abbreviation Expansions in Programs). 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. También, resuelve el problema que posee el algoritmo básico cuando no sabe que opción elegir ante muchas opciones de expansión. Para lograrlo esto, el algoritmo prioriza la frecuencia de aparición de las palabras por cercanía de alcance estático partiendo del lugar donde se encuentre el id analizado. También AMAP permite entrenarse con con conjunto de programas pasado como entrada para recopilar más palabras y mejorar aún más la precisión de la expansión.

Otra mejora futura para la herramienta IDA es la posibilidad de acoplarla como plugin a un entorno de desarrollo 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.

Bibliografía