2018

Project 1

CS 408.01 – PROFESSOR RAHEJA

JOSHUA J. CAMACHO

Sets using ADT in Java and C++

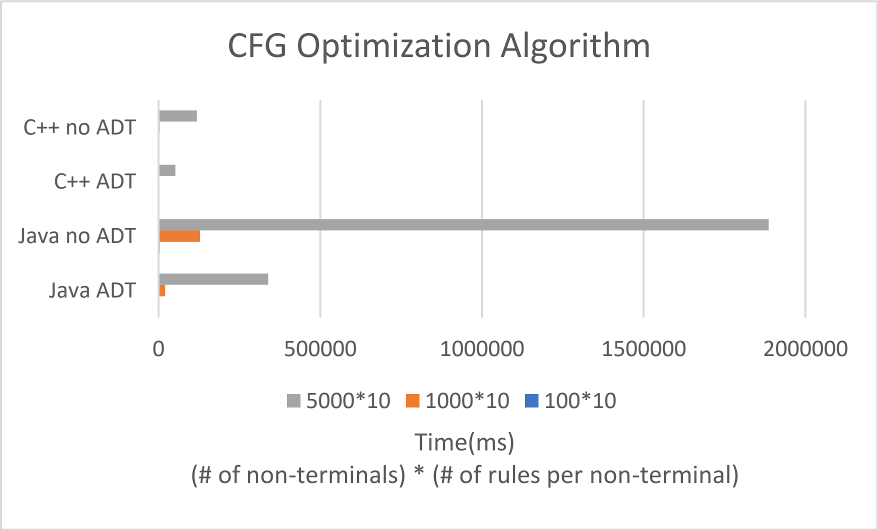
For this project I implemented set functionality, including support for the union, difference, and intersection. This was implemented in Java and C++ both using abstract data types and not using abstract data types. All four combinations were timed and compared using an application of sets.

Some background information before I go into the details of the application. From formal language theory, a context-free grammar (CFG) is a certain type of formal grammar: a set of production rules that describe all possible strings in a given formal language. It is described with a set of non-terminal symbols which have their respective rules, consisting of terminal and non-terminal symbols which can replace the respective terminal symbol in a string. Replacing all non-terminal symbols results in a string which is part of the language said to be described by the context-free grammar.

Optimization of the CFG is desirable for using CFGs in lexical analyzers. One such optimization is removing unreachable non-terminals, or non-terminals that can never be generated starting from the beginning non-terminal. For the application I implemented an algorithm to check for unreachable non-terminals. This is done by beginning at the starting non-terminal and adding each non-terminal to a Set of non-terminals that can be generated. At the end you find the unreachable non-terminals by doing a set difference SET (Unreachable Non-Terminals) = SET(ALL Non-Terminals) – SET(Reachable Non-terminals).

Timing results (in ms) for this application is as follows:

Data Size	Java ADT	Java no ADT	C++ ADT	C++ no ADT
100*10	3093	3792	45	54
1000*10	20283	128086	1847	3574
5000*10	338691	1886613	52111	118502



The results came at somewhat of a surprise. I was expecting the non-ADT implementations to be slower than the ADT implementations. Further testing will have to be done but my initial hypothesis is that the low amount of actual set operations in my algorithm is why the ADT implementations are faster. If more calls to the set operations would have been made, perhaps the non-ADT implementations would be faster. Java being slower than C++ was expected, as Java is a hybrid compile and interpreted language. C++ is a compiled language, so runtime is expected to be faster.