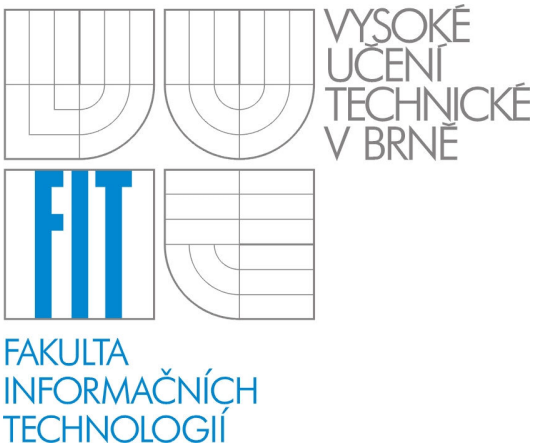

VYPe

(Compiler Construction)

Interprocedural Analysis (abstract)

Bc. Zdeněk Tisoň – xtison00
Bc. Pavel Wollný - xwolln00

1. 11. 2011

In this article we will examine basic concepts, why and when we wish to use interprocedural analysis. We discuss the importance of interprocedural analysis by showing some important optimization problems that cannot be solved by intraprocedural analysis.

Intraprocedural analysis is done one procedure at a time and assumes that procedures invoked may alter the state of all variables visible to the procedures and that they may create all possible side effects, such as modifying any of the variables visible to the procedure or generating exceptions that cause the unwinding of the call stack. Intraprocedural analysis is relatively simple but often imprecise and insufficient for some optimizations.

On the other hand an interprocedural analysis operates across an entire program, transfer information from the caller to its callees and vice versa. One relatively simple but useful technique is to inline procedures where possible. But this method is applicable only if we know the target of the procedure call. In some situation we do not know the target of call for example when procedures are invoked indirectly through a pointer or via the method dispatch mechanism, which is used in object-oriented programming languages. The disadvantages of inline procedures is that inlining can expand the code size exponentially. If we do not know the target of procedures call we need to do pointer analysis to specify interprocedural analysis. Therefore we will introduce call graphs.

Graphs that represent calling relationships between procedures in a program. In general, the presence of references or pointers to functions or methods requires us to get a static approximation of the potential values of all procedure parameters, function pointers, and receiver object types. To make an accurate approximation, interprocedural analysis is necessary.

Interprocedural analysis is also challenging because the behavior of each procedure is dependent upon the context in which it is called – context sensitivity. We introduce one simplistic but extremely inaccurate approach to interprocedural analysis, know as Context-insensitive analysis which treats each call and return statement as goto operations. For more accurate results we introduce context-sensitive methods like Cloning-based Context-sensitive Analysis and Summary-based Context-sensitive Analysis, which keep tracks of the context in which each procedure was called. A calling context is defined by the contents of the entire call stack. We refer to the string of call sites on the stack as the call string.

In the next part we use constant propagation to illustrate interprocedural analysis. This interprocedural optimization is neither readily applicable nor particularly beneficial. However, there are many reasons why interprocedural analysis is essential. We describe several important applications of

interprocedural analysis as Virtual Method Invocation, Pointer Alias Analysis, Parallelization, SQL Injection, and other.