Abstract

The ability to formally specify and verify compiler transformations (optimisations) is increasingly important to compiler designers as transformations become increasingly complex.

TRANS is a language for expressing compiler transformations. The language operates over the control flow graph (CFG) of a program. A TRANS transformation has two parts: the rewrite, which describes how to change the instructions at a node in the CFG and how to modify the edges of the graph to actually apply the transformation, and the side-condition which is a temporal logic formula which describes where in the CFG a transformation may be applied [Lacey, 2003, Kalvala et al., 2008]. The original presentation of TRANS demonstrated the power of the language by using a toy language, L_0, which contains only assignments, if statements, goto statements and return statements. All the variables in L_0 are of type integer.

The main contribution of this thesis is the compilation of a catalogue of compiler transformations written in TRANS which can be formally verified and may be used in the future by compiler writers who wish to use TRANS, as examples or extensions to their own transformation catalogues.

A number of extensions to L_0 and TRANS were implemented to allow more complex transformations to be written. TRANS is extended to support to nodes in the control flow graph being basic blocks instead of single instructions, which improves the efficiency of the implementation. L_0 is extended by adding syntax and semantics for supporting for arrays and function calls and TRANS is extended to support matching these new features. The TRANS syntax uses a recursive method to match each array subscript value and function parameters instead of requiring lists to be implemented in TRANS. The thesis also implements a number of array subscript analysis techniques in TRANS to support implementations of a number of loop based transformations.

Bibliography

Sara Kalvala, Richard Warburton, and David Lacey. Specifying and refining program transformations using temporal logic side conditions. *(in progress)*, 2008.

David Lacey. *Program transformation using temporal logic specification*. PhD thesis, Oxford University Computing Laboratory, 2003.