Compilers Project Report

Chintak Sheth   Geetika Bangera   Ubaid Ullah   Siddhartha Gupta

Type Checking

It's implemented using two virtual functions const Type* typeCheck() and void typePrint() for each of the subclasses of the Ast and STEClasses classes. isSubType method of the Type object checks if two types are compatible. Type checking for Operators (OpNode class), Expressions (ExprNode class and its subclasses), Statements(StmtNode and its subclasses) and Rules(RuleNode class) have been implemented.

While Loop and Break Statement

Grammar specification for While statement is implemented. Further, labeled break statements can appear inside while loops. Check that break labels are valid and break statements occur only inside loops are also done.

Memory Allocation

Registers are used for holding variables except for function parameters and local variables which are allocated on stack. A function to get the next available register for floats and ints each is implemented in MemAlloc.C. Further, a mapping for variables to registers and vice-versa has been created. Two registers are reserved for special use. R000 for Stack Pointer address and R001 for Base Pointer Address.

Functions

Following sequence of actions take place during function calls:
Initialize stack pointer
Caller pushes actual parameters in right to left order on stack
Caller pushes return address on stack
Execute jump instruction to callee's code
Callee saves return address of caller
Callee pushes local variables on stack
Callee pops parameters by incrementing stack pointer
Callee jumps to the return address of caller

Events

Events are input using the IN instruction in the assembly code. All event names have only a single character, so it will be easy to input event names using the IN instruction that returns just a single byte; an event takes only integer and floating point arguments. Depending upon event parameter declarations, IN operations are used to input event arguments and convert them to integers or floating point numbers.

Intermediate Code Generation

Intermediate code is represented using Quadruples or 3-address code. For each of the subclasses of AST and STEClasses classes, a codegen method is implemented which generates the intermediate code. Label class handles labels for jump instructions. Arg class encapsulates different arguments for 3 address intermediate code.

Basic Blocks

Basic blocks are created using Basic Block class which wraps an instruction generated by Intermediate Code gen. Any block containing jump instruction, is a predecessor to the instruction corresponding to jump target label. Similarly, that instruction is a successor to the former block. Further, for each block gen and kill sets are pre-computed.

Optimization

Dataflow analysis using Live Variable Analysis is done on Intermediate code generated for VariableEntry class nodes. During each iteration of this phase, in and out sets are computed using backward dataflow equations until there is no change in these sets.