Stage-5 实验报告

吴垒 2020010916

实验内容

明显比 stage-4 简单，可能因为完成 stage-4 需要对整个框架做修改，改完就完全理解了。

Step11：

在 frontend/lexer/lex.py 中添加 L/RBracket 用于匹配数组的中括号。

在 frontend/ast/tree.py 中添加 IndexExpr 作为数组节点，子属性 base 存储其 Identifier，index 同时完成对数组声明（各维度）的存储与对单个数组元素下标的存储。

在 frontend/parser/ply_parser.py 修改 p_declaration 添加可生成 empty 的非终结符 arrayindex 用于匹配数组定义的维度部分。添加 p_postfix_index 用于匹配表达式中的数组元素下标部分。同时将 p_binary_expression 的 assignment 产生式使其左值可匹配数组元素。

在 frontend/type/array.py 中为 ArrayType 添加 getdim(k)方法获取其第 k 维度的深度，并直接修改 multidim 的第二个参数为 list 类型用于直接通过 intliteral 的 list 建立 type。

在 frontend/typecheck/namer.py 中添加 visitIndexExpr 用于对表达式中数组的元素访问进行类型检查并匹配符号栈中的符号。在各个 visit 函数中将每个表达式的类型用 setattr 设置为其 type 属性，用于进行类型检查。修改 visitDeclaration，依次判断声明的是数组还是变量、是否为全局变量、是否有初始值并进行访问，设置 type、symbol。修改 visitAssignment 判断左值类型，并设置 type。

在 utils/tac/tacinstr.py 中添加 Alloc 类表示为数组分配空间的 tac 指令。

在 utils/tac/funcvisitor.py 中添加 visitAlloc 用于根据数组 size 生成分配空间的 tac 指令。

在 frontend/tacgen/tacgen.py 中修改 visitDeclaration，添加判断声明对象是否为数组并调用 visitAlloc 为数组分配空间。添加 visitIndexExpr 用于生成表达式中的数组元素对应的 tac 码；先根据 symbol 的 dim 用 add、mul 计算其基址偏移量，之后同时生成两个寄存器，一个用于存储其地址，用 setattr 设置为 addr 属性，一个用于存储其值，用 setattr 设置为 val 属性，分别针对该元素出现在左值及右值的情况，避免再添加特判。修改 visitAssignment，分别判断其是左值数组元素还是标识符、是否为全局变量并设置其值。

在 utils/riscv.py 中添加 Alloc 的 Riscv 指令，用于传递 tac 码中分配地址的信息。

在 backend/riscv/riscvasmemitter.py 的 generateGlobal 函数中将原本 bss 段固定的保留内存长度 4 改为 symbol.type.size，用于同时为标识符、数组分配内存空间。在 riscvasmemitter.riscvinstrselector 中添加 visitAlloc 将 tac 指令 Alloc 直接翻译为 Riscv 指令，由于涉及到栈空间的分配，此步暂时不翻译为实际 RV 指令，交由 emitNative 处理。在 RiscvSubroutineEmitter 中添加 emitAllocStack 方法用于在栈上分配数组，只需将当前 offset 与 sp 相加作为数组基址、并将 nextLocalOffset 加上数组空间即可。

在 backend/reg/bruteregalloc.py 的 allocForLoc 中添加对 loc 指令的判断，若指令类型为 Riscv.Alloc，调用 subemitter.emitAllocStack 分配空间，否则直接 emitNative 即可。

Step2：

在 frontend/parser/ply_parser.py 中修改 p_paramlist、p_paramlist_single 在尾部添加非终结符 paramindex 表示可能存在的数组下标，用于匹配数组传参，并添加对 paramindex 的语法匹配；修改 p_declaraction 为固定匹配未初始化的 Identifier，p_declaraction_array 匹配未初始化的数组声明，添加 p_declaration_array_init 匹配初始化的数组声明，并对尾部非终结符 arrayInit、其包含的 integerList 进行相应语法匹配。

在 frontend/symbol/varsymbol.py 中修改 setInitValue 使数组 value 可初始化为 list。

在 frontend/typecheck/namer.py 中修改 visitParameter 完成对形参类型的设置，修改 visitDeclaration 完成对数组的初始化，需要判断是否为全局变量类型。对函数形参若为数组类型且其第一维为空，则将其第一维初始化为-1。

在 frontend/typecheck/typer.py 的 visitCall 中添加对函数参数列表的类型检查。

修改 frontend/type/array.py 中 Arraytype 的__eq__方法，当某一维为-1 时说明为通配属性，可判定为该维相等。

在 frontend/tacgen/tacgen.py 中修改 transform 中对每个 function 的 param 的访问，判定其若为数组则应设置 baseTemp 属性而不是 Temp。修改 visitIdentifier 判断其是否为 Call 的数组传参，若为数组则设置其 val 为首地址。修改 visitDeclaration 添加对数组的初始化 tac 码，用 visitStoreTemp 依次将初始化值装载。

在 backend/riscv/riscvasmemitter.py 的 riscvasmemitter 的 generateGlobal 方法中添加将完成初始化的全局数组放到 data 段，并将值依次用.word 填入。

思考题

Step11：可变长度数组

假设当建立动态数组时设该数组 size 存在 t0，则需要添加 Riscv 的 sw, t1, sp 存储当前栈顶位置作为数组基址并添加 add sp, sp, -t0 将数组空间压栈，这样就导致了 sp 在一个函数内的变化；然而在变化期间仍需要通过 sp+offset 访问局部变量，故需要进入某一函数后固定 fp、将所有变量的访问方式改为通过 fp + nextLocalOffset 的方式访问，更换计算 nextLocalOffset 的方式，这将是一项比较麻烦的工程，因为 stage9 的函数栈空间分配我就想了很久没有想明白。

Step12：数组第一维为空的设计

由于数组任一元素的地址计算不依赖于该数组第一维的元素的值（如访问 a[n1][n2]的元素 a[i][j]偏移为 i+n2*j），因此不需要知道第一维即可实现数组内容的访问；可理解为第一维指向若干固定间隔的指针，而指针间隔的计算是依赖于之后的维度的；因此访问数组参数时第一维会直接衰减成数组指针，便于进行之后的代码分析与转换。在此增加对第一维大小的检查虽然合法，但只是凭空增加编译器的编写难度，是意义不大的。