GoLite CodeGen First Plan

Akshay Gopalakrishnan and Stuart Mashall

March 2019

1 Status of Semantics

1. We still have a few typecheck errors to rectify.
2. Errors such as type equality will be dealt by adding line numbers to the AST itself followed by modifying the symbol table to record line numbers (optional) for each record in it.
3. The typecheck code should be done correctly by this weekend

2 Code Generation Status

1. We will use C as our target langauge with the reason mostly being we both know C relatively well.
2. We ensured we both know it enough so that we could leverage the language features to deal with upcoming weird cases during codegen

3 Selected Areas with their Description in both source and target language

3.1 Identifiers

3.1.1 Go

Golite in totality has three types of identifiers

1. Normal Identifiers
2. Blank identifiers
3. Keywords can also be identifiers

3.1.2 C

1. We can mimic the same normal identifiers in C
2. For blank identifiers, we decided to append a number or pre-pend a text to it to ensure each blank identifier is unique.
3. Having them unique solves our issue of using multiple blank identifiers in C and still ensuring the code is valid
4. We will adhere to same strategy of append or prepend in case of keywords as C does not allow you to define identifiers with names as keywords

3.2 Types

Golite has four base types which are

1. Base types
2. Structs
3. Arrays
4. Slices

Base types and structs follow similar style as that of C. The existence of blank identifiers has been resolved so it is easy to map it to C equivalent structs.

Arrays also adhere to the same style. But array bounds checking need to be done explicitly as C does not ideally do any bound checking when indexing arrays. For this whenever an array is indexed in Golite, we will convert it to an array whcih indexes an element after the index value is passed through a function that ensures its bound checked.

Slices do not ideally exist in C. So, we've to make our own data structure to define slices. We will do this by creating a 'struct' in C that has a header and an array to index and can be extended. (using memory allocation) Other functionalities like len and cap shall be dealt with as variables in the data structure itself. Similar for arrays, slice indexing will also pass through our bound-checking function.

In terms of equality, in Golite, the types are not resolved to base types to check equality. However in C, the types are resolved to check to base type to check equality. This allows us to do assignments of on variable of one type and another of different type but whose base types are same. However, since our typechecking phase would catch an invalid Golite program in terms of this, we do not need to take care of equality here. Also due to the fact that type equaltiy in C otherwise is same as GoLite

In terms of tagged vs untagged, C and Golite both allow defining variables with untagged types. For slices our data structure with its init function would be used. In terms of untagged equality, Golite does consider all untagged types of the same structure to be the same. However in C it is not the case that it consideres untagged types to be the same. (eg: structs). To deal with it, we decided to tag them during codegen itself so as to not face this issue in C.

3.3 Functions

3.3.1 Pass-by Semantics

GoLite works using Pass-by Value semantics. In C this can be done easily. For slices, too, passing the header of the slice would ensure same behavior as GoLite. The only main issue would be that of arrays. Arrays are by default pass-by reference in C. For this, we've decided to explicitly make a copy of the array and pass that copy to the function so that the original one does not change.

In C, we can have untagged parameters, but the compiler gives a warning, in that it says that no value can be passed as this parameter as its type would never match the untagged one. This is correct in terms of what we saw of C earlier of keeping untagged definitions unique. But since we decided to tag all untagged types, we will not face this issue.

3.3.2 Return Semantics

The return semantics is same for C as well as GoLite. That is, return by value is allowed. So we will keep it as is. The issue of untagged or tagged returns wont come in C due to us tagging all untagged types out there.

4 Coding of codegen status

We have not yet started coding for the Codegen phase. We are still trying out examples and charting out plans to deal with each of them.