Progess Report 2

Presentor: Xie Li

April 25, 2021

Overview

- Running SV-COMP cases on SMACK.
- Memory model and its construction.
- Insert assertion to Boogie.

Memory Model and Region: DSA

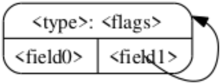
Each DS node represent a set of dynamic memory object (may be infinite), different nodes represent disjoint sets of objects.

Definition (Data Structure Graph)

A DS graph for a function F is $G(F) = \langle N, E, E_V, N_{call} \rangle$.

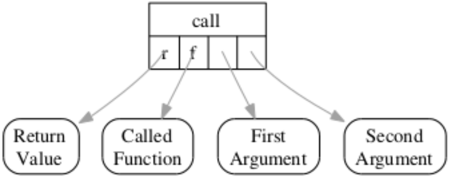
- N is a set of nodes.
- E is a set of edges in the graph, E is a function of the type $\langle n_s, f_s \rangle \to \langle n_d, f_d \rangle$, where $n_s, n_d \in N$ and $f_s \in field(T(n_s)), f_d \in field(T(n_d))$.
  This node-field pair is called a *cell*, non-pointer compatible and register variables are mapped to $\langle null, 0 \rangle$
  E is a function.
- E_V is a partial function for pointer-compatible variables in $vars(f)$. i.e. $vars(f) \to \langle n, f \rangle$.
- N_{call} is the set of call nodes which is a subset of N. Every call node is a tuple of node-field pairs: $\langle r, f, a_1, \ldots, a_k \rangle$. Each element can also be regarded as a points-to edge in the graph.

Visualization of DSG



DS node

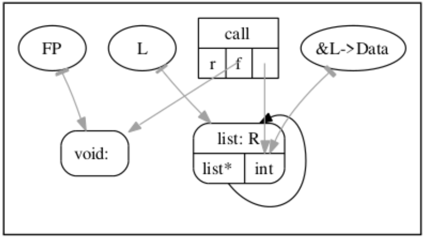
Variable

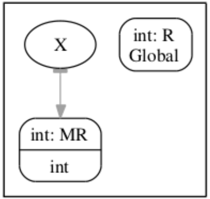


Call Node

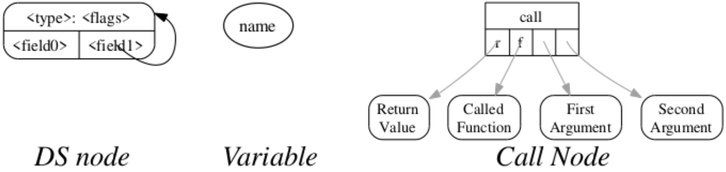
Visualization of DSG

```
typedef struct list { struct list *Next;
                      int Data; } list;
int Global = 10;
void do_all(list *L, void (*FP)(int*)) {
  do { FP(&L->Data);
       L = L->Next;
  } while(L);
}
void addG(int *X) { (*X) += Global; }
void addGToList(list *L) { do_all(L, addG); }
list *makeList(int Num) {
  list *New = malloc(sizeof(list));
  New->Next = Num ? makeList(Num-1) : 0;
  New->Data = Num; return New;
}
int main() {   /* X & Y lists are disjoint */
  list *X = makeList(10);
  list *Y = makeList(100);
  addGToList(X);
  Global = 20;
  addGToList(Y);
}
```





Graph Node and Field



There are three pieces of information of a DS Node n.

- $T(n)$.
- $G(n)$ is a set of global objects represented by n.
- $flag(n) \subseteq \{\mathbf{H}, \mathbf{S}, \mathbf{G}, \mathbf{U}, \mathbf{A}, \mathbf{M}, \mathbf{R}, \mathbf{C}, \mathbf{O}\}$

Meaning of the Flags

- Storage class flags: **H**eap, **S**tack, **G**lobal, **U**nknown.
- Whether a memory object is loaded or stored: **R**eferenced, **M**odified.
- **C**omplete.
- C**O**llapsed: nodes representing multiple, incompatible types of objects: type homogenous, *use* of a object.
- **A**rray flag.

Construction of the DSG

Three steps:

- Construct the DSG for each function.
- Bottom-up analysis.
- Top-down analysis.

Two important properties:

- The DSG is correct even if only a portion of the callers and callees are consider and incorperated into the graph.
- The order of the graph inlining does not modify the final result.

Problem: How to understand these two properties?

Primitive Graph Operation

TODO: include graphics here.

- `mergeCells(c1, c2)`
- `cloneGraphInto(G1, G2)`
- `resolveCallee` and `resolveCaller`
- `resolveArguments` and `markComplete`

Local Analysis Phase

Bottom-Up Analysis Phase

- Non-recursive calls.
- Recursive calls.
- Recursive calls with function pointer.

Insert Self-defined Assertion