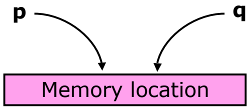
Points-to Analysis for the C Language

Amira Mensi, amira.mensi@mines-paristech.fr
PhD Advisors: Fabien Coelho, François Irigoin

Key Issue



- Determine the set of objects pointed to by a reference variable
- Provide a set of points-to relations:

(pointer, memory_location, approximation)

Dimensions

- Flow sensitivity
- Context sensitivity
- Path sensitivity
- Field sensitivity
- Heap modeling
- Interprocedurality

Client Analyses

- Proper memory effects
- Use-def chains
- Reaching definitions
- Liveness variables
- Constant propagation
- Dependences test
- …

State Of Art

Analysis	Framework	Idea	Flow Sensitivity	Context Sensitivity	Efficiency	Precision	Field Sensitivity	Inter procedurality
Andersen	GCC	Transforms pointer assignment into constraints and solves them to obtain a points-to graph	–	–	+	–	–	+
Steensgaard	LLVM	Uses type inference system to generate a shape storage graph	–	–	++	– –	–	+
Wilson	SUIF	Uses partial transfer function to compute points-to relations	+	+	–	+	–	++
Emami	McCat	Applies a specific rule for each pointer assignment pattern to compute Possible/Definitely points-to relations	+	++	–	+++	++	+++

Goal: Define and implement a general-purpose "points-to" analysis for C based on Emami's points-to analysis and Wilson's scheme at source level in PIPS framework

Our Approach

1. Computes points-to relations **(p, i, EXACT)** for any pointer assignment such as **p = &i** or **p->q->r = &j**
2. Transforms pointer dereferencing ***p** into array notation **p[0]**
3. Evaluates pseudo-array access **p[0]** using points-to relations to **i**
4. Updates points-to relations at each pointer value modification

Our Contributions

1. Constant memory accesses are used instead of temporary variables
2. All C instructions and operators are handled
3. Memory locations are modelized as a lattice
4. Errors are detected: uninitialized pointers, dangling pointers, memory leaks…
5. Context information is taken into account when modeling heap locations

Ongoing Work: Interprocedural Analysis

At each call site C

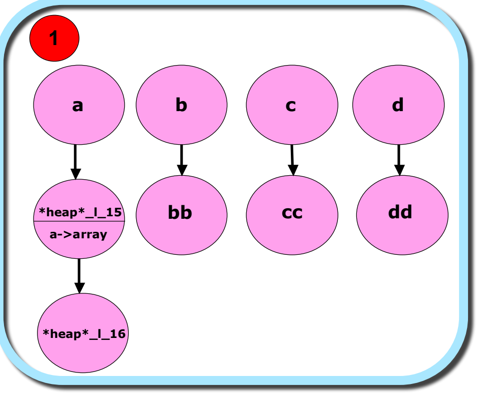
1. Combination of bottom-up and top-down analyses
2. Aliasing of formal parameters is checked
3. Binding B between effective and formal parameters is computed
4. Translation of the OUT points-to set for the callee using B to obtain the Gen set at C
5. Translation of the callee's written pointers to obtain the Kill set of C

An Example

```
void initialize(int cnt)
{  struct array_2D {
     int d1;
     int d2;
     int *array;
  };
int *b, *c, *d, bb[cnt], cc[cnt], dd[cnt], i = 0;
struct array_2D *a = (struct array_2D*) malloc(sizeof(struct array_2D)) ;
a->array = (int *) malloc(cnt * sizeof(int));
b = &bb[0];
c = &cc[0];
d = &dd[0];
for (i = 0; i < cnt; i++)
  a->array[i]  = d[i] + c[i] * d[i];
}  1
```

// Points-to relations at 1
(*HEAP*_l_15.array , *HEAP*_l_16[0] , EXACT)
(a, *HEAP*_l_15 , EXACT)
(b, bb[0] , EXACT)
(c, cc[0] , EXACT)
(d, dd[0] , EXACT)

Its Final Points-to Graph


1
a
b
c
d
*heap*_l_15
a->array
bb
cc
dd
*heap*_l_16