LINUX PLUMBERS CONFERENCE | SEPTEMBER 20-24, 2021

Soufflé, Points-To Analysis and Data Layout Optimizations

Erick Ochoa
eochoa@gcc.gnu.org

Christoph Müllner
cmuellner@gcc.gnu.org

Philipp Tomsich
ptomsich@gnu.org

What is Soufflé?

https://souffle-lang.github.io/

What is Soufflé?

```
.decl edge(x:number, y:number)
```

What is Soufflé?

```
.decl edge(x:number, y:number)
.input edge
```

What is Soufflé?

```
.decl edge(x:number, y:number)
.input edge
.decl path(x:number, y:number)
```

What is Soufflé?

```
.decl edge(x:number, y:number)
.input edge
.decl path(x:number, y:number)
.output path
```

What is Soufflé?

```
.decl edge(x:number, y:number)
.input edge
.decl path(x:number, y:number)
.output path
path(x, y) :- edge(x, y).
```

What is Soufflé?

```
.decl edge(x:number, y:number)
.input edge
.decl path(x:number, y:number)
.output path
path(x, y) :- edge(x, y).
path(x, y) :- path(x, z), edge(z, y).
```

https://souffle-lang.github.io/simple

What is Soufflé?

Figure 2. Staged Compilation Framework for Synthesizing C++ Programs from a Datalog Specification

Scholz, Bernhard, et al. "On fast large-scale program analysis in datalog." Proceedings of the 25th International Conference on Compiler Construction. 2016.

What is Soufflé?

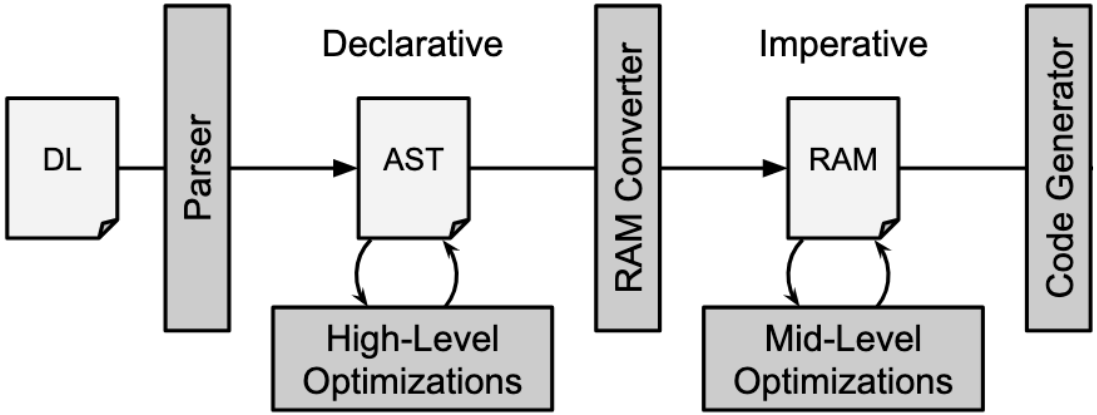


Figure 2. Staged Compilation Framework for Synthesizing C++ Programs from a Datalog Specification

Scholz, Bernhard, et al. "On fast large-scale program analysis in datalog." Proceedings of the 25th International Conference on Compiler Construction. 2016.

What is Soufflé?

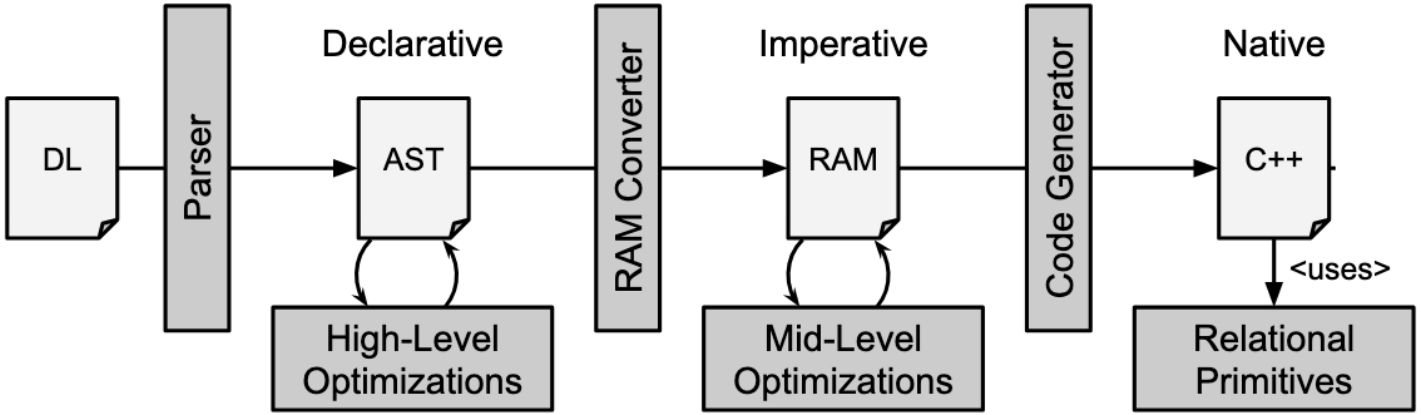


Figure 2. Staged Compilation Framework for Synthesizing C++ Programs from a Datalog Specification

Scholz, Bernhard, et al. "On fast large-scale program analysis in datalog." Proceedings of the 25th International Conference on Compiler Construction. 2016.

What is Soufflé?

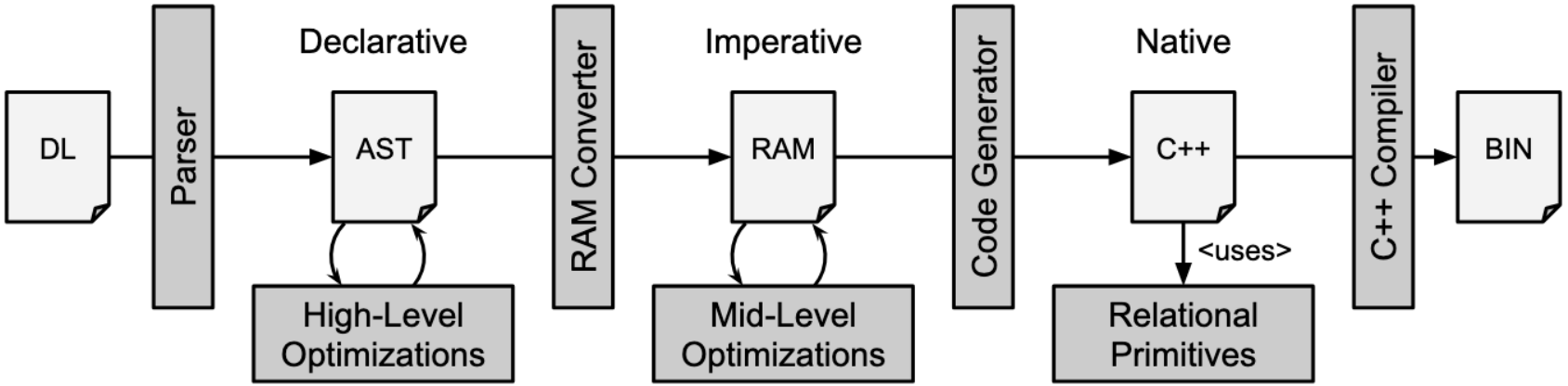


Figure 2. Staged Compilation Framework for Synthesizing C++ Programs from a Datalog Specification

Scholz, Bernhard, et al. "On fast large-scale program analysis in datalog." Proceedings of the 25th International Conference on Compiler Construction. 2016.

What is Soufflé?

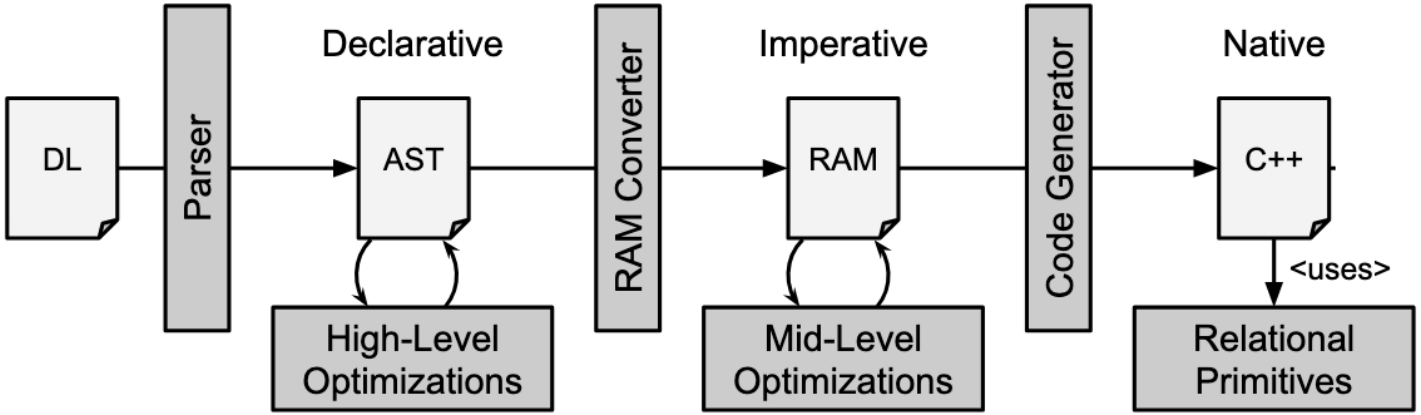


Figure 2. Staged Compilation Framework for Synthesizing C++ Programs from a Datalog Specification

Scholz, Bernhard, et al. "On fast large-scale program analysis in datalog." Proceedings of the 25th International Conference on Compiler Construction. 2016.

Why do we think Soufflé is interesting?

Why do we think Soufflé is interesting?

1. Datalog is a language well suited for program analysis.

Why do we think Soufflé is interesting?

1. Datalog is a language well suited for program analysis.
2. Parallelism via OpenMP.

Why do we think Soufflé is interesting?

1. Datalog is a language well suited for program analysis.
2. Parallelism via OpenMP.
3. Features such as provenance and profiling.

Why do we think Soufflé is interesting?

1. Datalog is a language well suited for program analysis.
2. Parallelism via OpenMP.
3. Features such as provenance and profiling.
4. High level language.

Can Soufflé programs be part of the GCC build process?

Can Soufflé programs be part of the GCC build process?

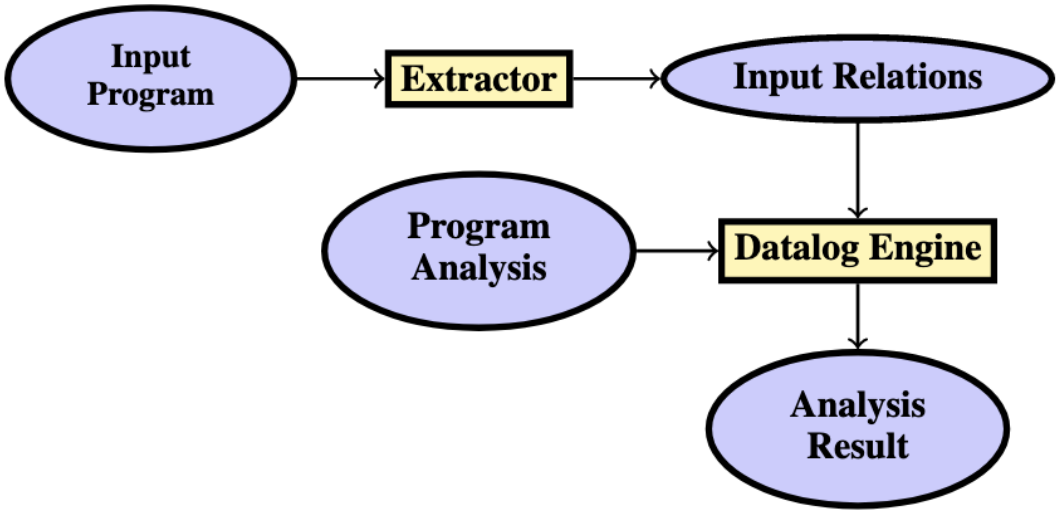
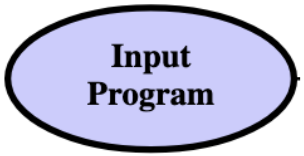


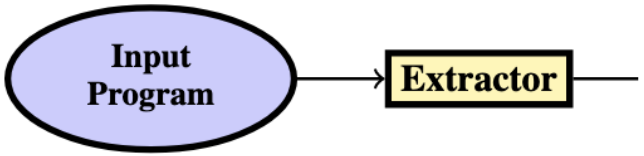
Figure 1. Typical Program Analysis in Datalog using an Extractor

Scholz, Bernhard, et al. "On fast large-scale program analysis in datalog." Proceedings of the 25th International Conference on Compiler Construction. 2016.

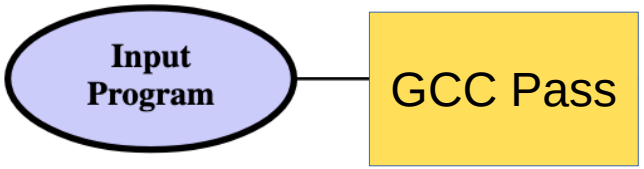
Can Soufflé programs be part of the GCC build process?



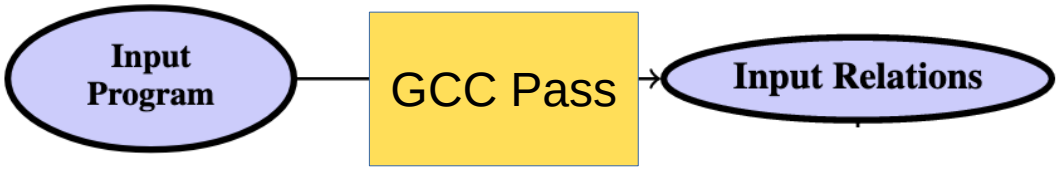
Can Soufflé programs be part of the GCC build process?



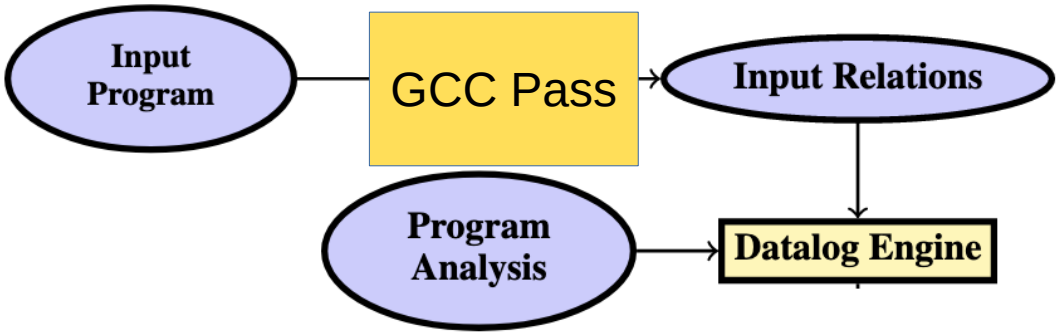
Can Soufflé programs be part of the GCC build process?



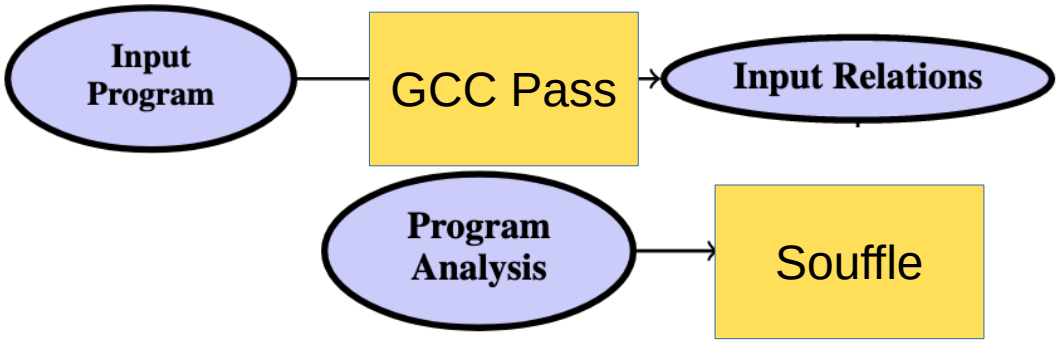
Can Soufflé programs be part of the GCC build process?



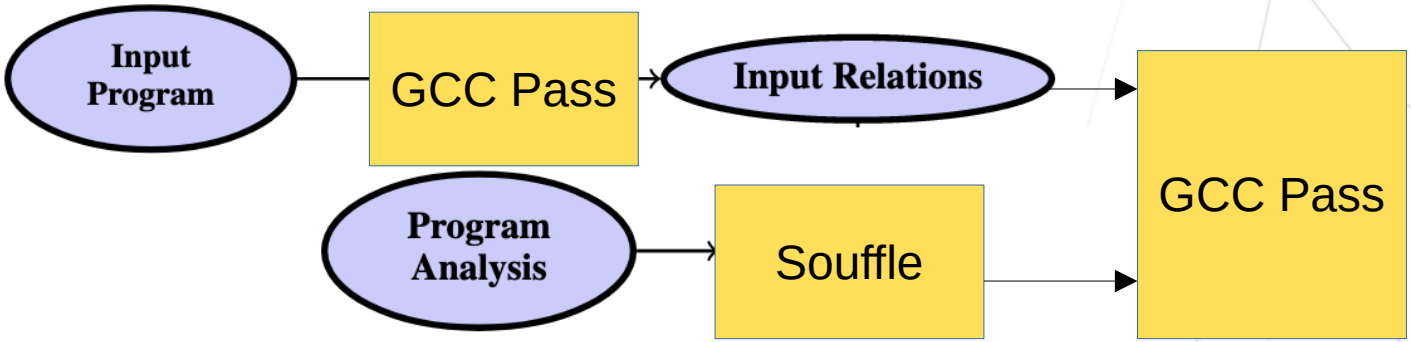
Can Soufflé programs be part of the GCC build process?



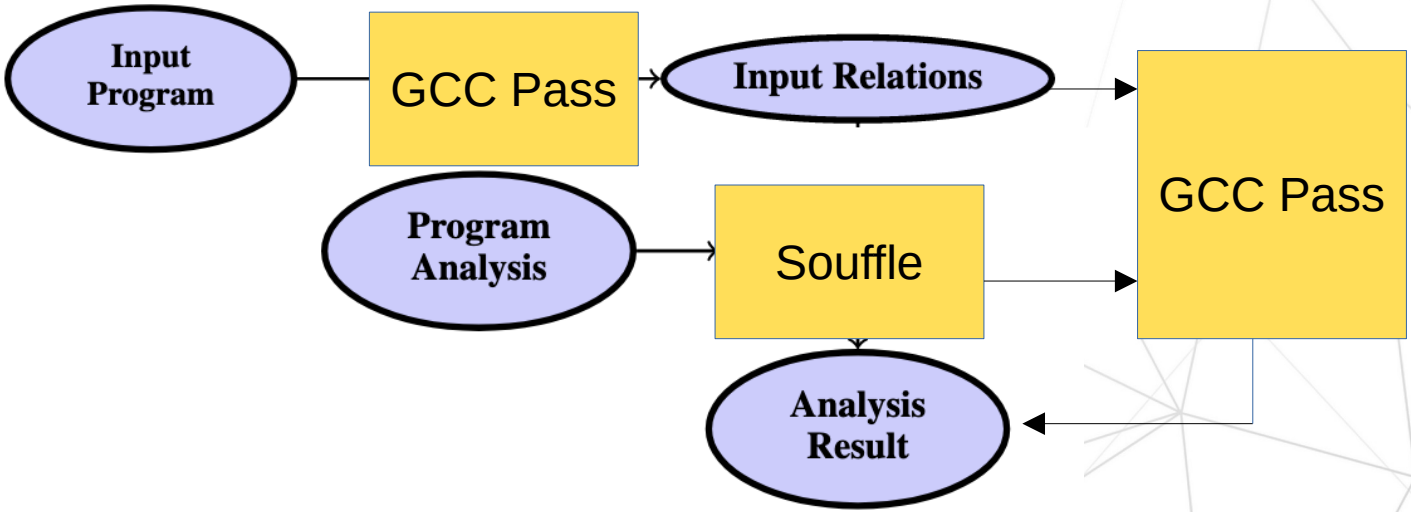
Can Soufflé programs be part of the GCC build process?



Can Soufflé programs be part of the GCC build process?



Can Soufflé programs be part of the GCC build process?



Can Soufflé programs be part of the GCC build process?

1. Generated C++ is C++17

Can Soufflé programs be part of the GCC build process?

1. Generated C++ is C++17
2. Compile with -fpermissive

Can Soufflé programs be part of the GCC build process?

1. Generated C++ is C++17
2. Compile with -fpermissive
3. Compile with -fopenmp

Can Soufflé programs be part of the GCC build process?

1. Generated C++ is C++17
2. Compile with -fpermissive
3. Compile with -fopenmp
4. Compile with -fexceptions

Motivation for LTO Points-To Analysis implementation in Soufflé

Motivation for LTO Points-To Analysis implementation in Soufflé

https://gcc.gnu.org/wiki/GCCSpec2017/mcf

Motivation for LTO Points-To Analysis implementation in Soufflé

https://gcc.gnu.org/wiki/GCCSpec2017/mcf

```
struct arc
{
  int id;
  cost_t cost;
  node_p tail, head;
  short ident;
  arc_p nextout, nextin;
  flow_t flow;
  cost_t org_cost;
};
```

Motivation for LTO Points-To Analysis implementation in Soufflé

https://gcc.gnu.org/wiki/GCCSpec2017/mcf

```
struct arc
{
  int id;
  cost_t cost;
  node_p tail, head;
  short ident;
  arc_p nextout, nextin;
  flow_t flow;
  cost_t org_cost;
};
```

```
struct node {
  cost_t potential;
  int orientation;
  node_p child, pred, sibling, sibling_prev;
  arc_p basic_arc, firstout, firstin, arc_tmp;
  flow_t flow;
  LONG depth;
  int number;
  int time;
};
```

Motivation for LTO Points-To Analysis implementation in Soufflé

https://gcc.gnu.org/wiki/GCCSpec2017/mcf

```
struct arc
{
  int id;
  cost_t cost;
  node_p tail, head;
  short ident;
  arc_p nextout, nextin;
  flow_t flow;
  cost_t org_cost;
};
```

```
struct node {
  cost_t potential;
  int orientation;
  node_p child, pred, sibling, sibling_prev;
  arc_p basic_arc, firstout, firstin, arc_tmp;
  flow_t flow;
  LONG depth;
  int number;
  int time;
};
```

Motivation for LTO Points-To Analysis implementation in Soufflé

https://gcc.gnu.org/wiki/GCCSpec2017/mcf

```
struct arc
{
  int id;
  cost_t cost;
  node_p tail, head;
  short ident;

  flow_t flow;
  cost_t org_cost;
};
```

```
struct node {
  cost_t potential;
  int orientation;
  node_p child, pred, sibling, sibling_prev;
  arc_p basic_arc, firstout, firstin, arc_tmp;
  flow_t flow;
  LONG depth;
  int number;
  int time;
};
```

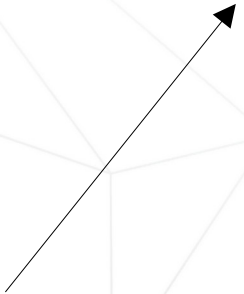
Motivation for LTO Points-To Analysis implementation in Soufflé

https://gcc.gnu.org/wiki/GCCSpec2017/mcf

```
struct arc
{
  int id;
  cost_t cost;
  node_p tail, head;
  short ident;

  flow_t flow;
  cost_t org_cost;
};
```

```
struct node {
  cost_t potential;
  int orientation;
  node_p child, pred, sibling, sibling_prev;
  arc_p basic_arc, firstout, firstin, arc_tmp;
  flow_t flow;
  LONG depth;
  int number;
  int time;
};
```



Motivation for LTO Points-To Analysis implementation in Soufflé

https://gcc.gnu.org/wiki/GCCSpec2017/mcf

```
struct arc
{
  int id;
  cost_t cost;
  node_p tail, head;
  short ident;

  flow_t flow;
  cost_t org_cost;
};
```

```
struct node {
  cost_t potential;
  int orientation;
  node_p child, pred, sibling, sibling_prev;
  arc_p basic_arc, firstout, firstin          ;
  flow_t flow;
  LONG depth;
  int number;
  int time;
};
```

Motivation for LTO Points-To Analysis implementation in Soufflé

https://gcc.gnu.org/wiki/GCCSpec2017/mcf

```
struct arc
{
  node_p tail, head;
  cost_t cost;
  cost_t org_cost;
  flow_t flow;
  int id;
  short ident;
};
```

```
struct node {
  cost_t potential;
  node_p child, pred, sibling, sibling_prev;
  arc_p basic_arc, firstout, firstin          ;
  flow_t flow;
  LONG depth;
  int orientation;
  int number;
  int time;
};
```

Motivation for LTO Points-To Analysis implementation in Soufflé

https://gcc.gnu.org/wiki/GCCSpec2017/mcf

```
struct arc
{
  node_p tail, head;
  cost_t cost;
  cost_t org_cost;
  flow_t flow;
  int id;
  short ident;
};
```

72

```
struct node {
  cost_t potential;
  node_p child, pred, sibling, sibling_prev;
  arc_p basic_arc, firstout, firstin          ;
  flow_t flow;
  LONG depth;
  int orientation;
  int number;
  int time;
};
```

104

Motivation for LTO Points-To Analysis implementation in Soufflé

https://gcc.gnu.org/wiki/GCCSpec2017/mcf

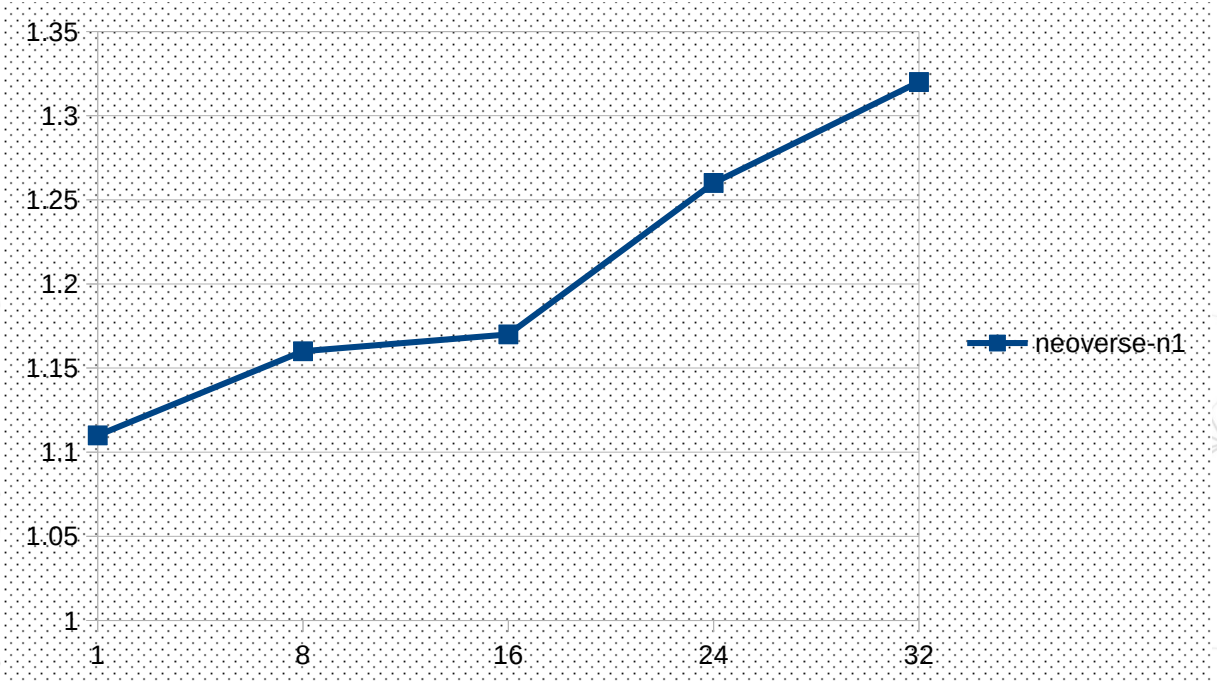
```
struct arc
{
  node_p tail, head;
  cost_t cost;
  cost_t org_cost;
  flow_t flow;
  int id;
  short ident;
};
```

72 → 56

```
struct node {
  cost_t potential;
  node_p child, pred, sibling, sibling_prev;
  arc_p basic_arc, firstout, firstin          ;
  flow_t flow;
  LONG depth;
  int orientation;
  int number;
  int time;
};
```

104 → 96

Normalized MCF score



Type Escape vs Points-To Analysis

Points To Analysis	Type Escape Analysis
Can be applied at an alias-level granularity.	Can only be applied to all instances of a given type.
Harder to implement.	Easier to implement.

Majority of current implementation

```
PointsTo (pointer, pointee) :- PA (pointer, _, pointee, _).

PointsTo (pointer, pointee) :- PP (pointer, _, temp, _)
  , PointsTo (temp, pointee).

PP (left, 0, right, 0) :- PS (left, _, temp, _)
  , PointsTo (temp, right).

PP (left, 0, right, 0) :- SP (temp, _, right, _)
  , PointsTo (temp, left).

PA (left, 0, pointee, 0) :- SA (temp, _, pointee, _)
  , PointsTo (temp, left).

PS (left, 0, right, 0) :- SS (temp, _, right, _)
  , PointsTo (temp, left).
```

Majority of current implementation

PointsTo (pointer, pointee) :-
PA (pointer, _, pointee, _).

pointer = &pointee

Majority of current implementation

```
pointer = temp;
```

Majority of current implementation

:- PP (pointer, _, temp, _)

pointer = temp;

Majority of current implementation

:- PP (pointer, _, temp, _)

temp = &pointee;
pointer = temp;

Majority of current implementation

```
:- PP (pointer, _, temp, _)
, PointsTo (temp, pointee).
```

```
temp = &pointee;
pointer = temp;
```

Majority of current implementation

```
PointsTo (pointer, pointee)
  :- PP (pointer, _, temp, _)
  , PointsTo (temp, pointee).

temp = &pointee;
pointer = temp;
```

Majority of current implementation

```
left = *temp;
```

Majority of current implementation

:- PS (left, _, temp, _)

left = *temp;

Majority of current implementation

```
:- PS (left, _, temp, _)
, PointsTo (temp, right).
```

```
temp = &right;
left = *temp;
```

Majority of current implementation

```
PP (left, 0, right, 0)
  :- PS (left, _, temp, _)
  , PointsTo (temp, right).

temp = &right;
left = *temp;
// left = right;
```

Majority of current implementation

```
PointsTo (pointer, pointee) :- PA (pointer, _, pointee, _).

PointsTo (pointer, pointee) :- PP (pointer, _, temp, _)
  , PointsTo (temp, pointee).

PP (left, 0, right, 0) :- PS (left, _, temp, _)
  , PointsTo (temp, right).

PP (left, 0, right, 0) :- SP (temp, _, right, _)
  , PointsTo (temp, left).

PA (left, 0, pointee, 0) :- SA (temp, _, pointee, _)
  , PointsTo (temp, left).

PS (left, 0, right, 0) :- SS (temp, _, right, _)
  , PointsTo (temp, left).
```

1. Soufflé is an exciting possible new direction for program analysis

1. Soufflé is an exciting possible new direction for program analysis
2. There needs to be a dialogue on the possibility of integrating Soufflé generated code to GCC's code base

1. Soufflé is an exciting possible new direction for program analysis
2. There needs to be a dialogue on the possibility of integrating Soufflé generated code to GCC's code base
3. We are currently working on a points-to analysis to resolve concerns on our previous data layout optimization implementation.

Bibliography

1. Golovanevsky, Olga, and Ayal Zaks. "Struct-reorg: current status and future perspectives." Proceedings of the GCC Developers' Summit. 2007.
2. Zhao, Peng, et al. "Forma: A framework for safe automatic array reshaping." ACM Transactions on Programming Languages and Systems (TOPLAS) 30.1 (2007): 2-es.
3. Yong, Suan Hsi, Susan Horwitz, and Thomas Reps. "Pointer analysis for programs with structures and casting." ACM SIGPLAN Notices 34.5 (1999): 91-103.
4. Balatsouras, George, and Yannis Smaragdakis. "Structure-sensitive points-to analysis for C and C++." International Static Analysis Symposium. Springer, Berlin, Heidelberg, 2016

Questions

Erick Ochoa
eochoa@gcc.gnu.org

Materials for Questions

Why not extend current IPA-PTA instead of implementing a new one?
What does a points-to analysis that handles cast look like?
How does type escape analysis differs from a points-to analysis?
How does GCC feeds constraints to the souffle program?
How are the constraints generated?
Can you show more of the Souffle code?
How is cast-aware points-to analysis implemented?

Comparing Points-To Analysis

GCC IPA-PTA	PTA in Forma Paper
Inter-procedural	
Field Sensitive	
Flow Insensitive	
Context Insensitive	
Inclusion-Based	Unification-Based
Cast Unaware	Cast Aware

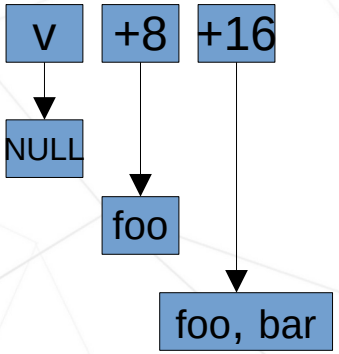
Cast-(Un)Aware

```
struct s  { long *a; long *b; long *c;};

struct s v;

v.a = NULL;                     // v +  0 ⊇ NULL
v.b = &foo;                     // v +  8 ⊇ foo
v.c = cond ? &foo : &bar;       // v + 16 ⊇ foo
                                // v + 16 ⊇ bar


// What does v.b points-to?
```



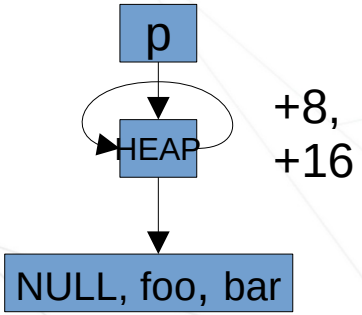
Cast-Unaware

```
struct s  { long *a; long *b; long *c;};

struct s *p = malloc (sizeof(struct s));
                                  //  p +  0 ⊇ HEAP


p->a= NULL;                       // *p +  0 ⊇ NULL
p->b= &foo;                       // *p +  8 ⊇ foo
p->c= cond ? &foo : &bar;         // *p + 16 ⊇ foo
                                  // *p + 16 ⊇ bar


// What does p->b points-to?
```



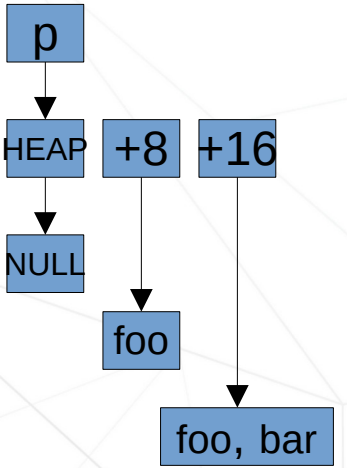
Cast-Aware

```
struct s  { long *a; long *b; long *c;};

struct s *p = malloc (sizeof(struct s));
                                  //  p +  0 ⊇ HEAP


p->a= NULL;                       // *p +  0 ⊇ NULL
p->b= &foo;                       // *p +  8 ⊇ foo
p->c= cond ? &foo : &bar;         // *p + 16 ⊇ foo
                                  // *p + 16 ⊇ bar


// What does p->b points-to?
```



VS

VS

```
struct s  { long a; long b; long c;};
```

VS

```
struct s  { long a; long b; long c;};

struct s r1;
struct s r2;
```

VS

```
struct s  { long a; long b; long c;};

struct s r1;
struct s r2;

r1.a = argc;
r2.b = argc;
```

VS

```
struct s  { long a; long b; long c;};

struct s r1;
struct s r2;

r1.a = argc;
r2.b = argc;

printf(“%ld\n”, r1.a * r2.b);
```

VS

```
struct s  { long a; long b; long c;};

struct s r1; // delete fields (b & c) or c?
struct s r2; // delete fields (a & c) or c?

r1.a = argc;
r2.b = argc;

printf(“%ld\n”, r1.a * r2.b);
```

Interface

```
souffle::SouffleProgram *prog = souffle::ProgramFactory::newInstance("points_to");

souffle::Relation *var = prog->getRelation ("Variable");
souffle::Relation *constraints_rel = prog->getRelation ("RawConstraints");
```

Interface

```
souffle::tuple _tuple (r);
souffle::RamUnsigned _yl(_lhs_t.k1());
souffle::RamUnsigned _el(_lhs_e.k1());
souffle::RamUnsigned _fl(_lhs.k1());
souffle::RamUnsigned _sl(_lhs.k2());
souffle::RamUnsigned _tl(_lhs.k3());
souffle::RamUnsigned _ql(_lhs.k4());
souffle::RamUnsigned _ol(_lhs_o);
souffle::RamUnsigned _yr(_rhs_t.k1());
souffle::RamUnsigned _er(_rhs_e.k1());
souffle::RamUnsigned _fr(_rhs.k1());
souffle::RamUnsigned _sr(_rhs.k2());
souffle::RamUnsigned _tr(_rhs.k3());
souffle::RamUnsigned _qr(_lhs.k4());
souffle::RamUnsigned _or(_rhs_o);
_tuple << _yl << _el << _fl << _sl << _tl << _ql
        << _ol << _yr << _er << _fr << _sr << _tr << _qr << _or;
r->insert (_tuple);
```

Implementation

Pointer/Pointee	Unique Identifier
Global Variable	<G, varpool_node*, x, x, x>
Function	<F, cgraph_node*, x, x, x>
Local Variable	<L, cgraph_node*, id, x, x>
SSA Variable	<S, cgraph_node*, id, x, x>
Abstract Variable	<A, NULL, id, x, x>
String Literal (DECL_INIT)	<C, varpool_node*, x, x, x>
String Literal (gassign)	<C, cgraph_node*, bb_idx, stmt_idx, x>
String Literal (args)	<C, cgraph_node*, bb_idx, stmt_idx, arg_idx>
Heap Variable	<H, cgraph_node*, bb_idx, stmt_idx, x>

Implementation

Expression	Name
A = B	Copy Constraint
A = &B	Address-Of Constraint
A = *B	Scalar-Deref Constraint
*A = B	Deref-Scalar Constraint

Implementation

Expression	Name
A + off_l = B + off_r	Copy Constraint
A + off_l = &B + off_r	Address-Of Constraint
A + off_l = *B + off_r	Scalar-Deref Constraint
*A + off_l = B + off_r	Deref-Scalar Constraint

Implementation

Expression	Name
C + off	Scalar Expression
&C + off	Address-Of Expression
*C + off	Dereference Expression

Implementation

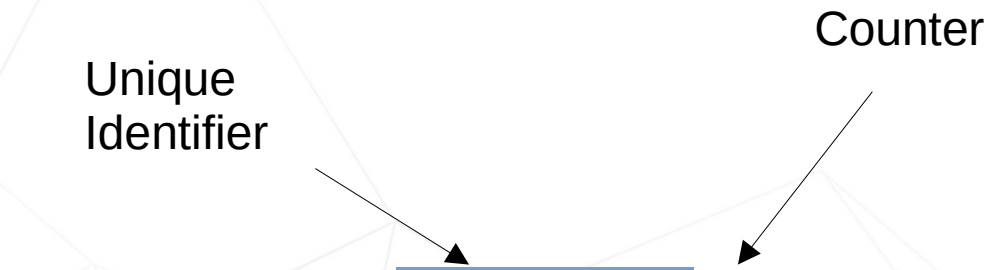
Constraint Record
LHS Expression
LHS Variable Representation
LHS Offset
RHS Expression
RHS Variable Representation
RHS Offset

Soufflé

Unique identifier
(e.g., <G, varpool_node*, x, x, x>

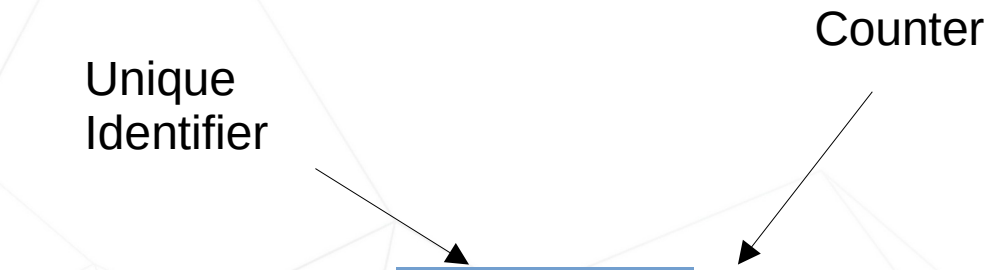
```
.decl Variable (y: v_t, f :f_id, s: s_id, t: t_id, q: q_id)
.input Variable
```

Soufflé



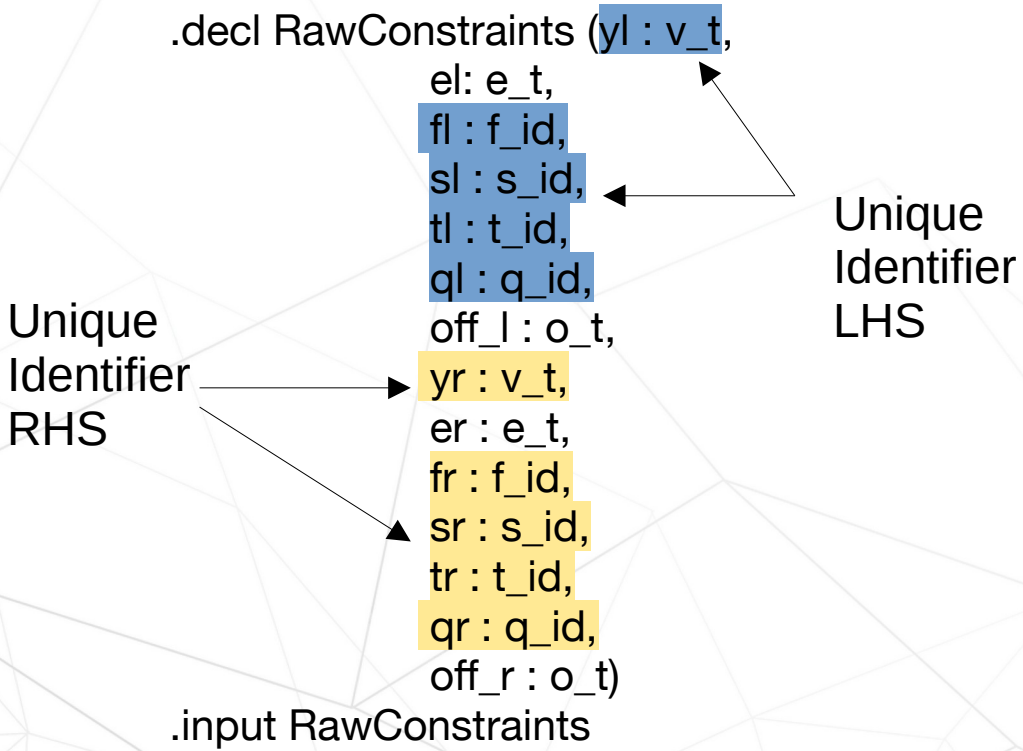

```
VariableMap (y, f, s, t, q, $)
:- Variable (y, f, s, t, q).
```

Soufflé



1. <G, foo, x, x, x>
2. <G, bar, x, x, x>
3. <F, main, x, x, x>

Soufflé


.decl RawConstraints (yl : v_t,
el: e_t,
fl : f_id,
sl : s_id,
tl : t_id,
ql : q_id,
off_l : o_t,
yr : v_t,
er : e_t,
fr : f_id,
sr : s_id,
tr : t_id,
qr : q_id,
off_r : o_t)
.input RawConstraints
Unique Identifier LHS
Unique Identifier RHS

Soufflé

```
.decl RawConstraints (yl : v_t,
                      el: e_t,
                      fl : f_id,
                      sl : s_id,
                      tl : t_id,
                      ql : q_id,
                      off_l : o_t,
                      yr : v_t,
                      er : e_t,
                      fr : f_id,
                      sr : s_id,
                      tr : t_id,
                      qr : q_id,
                      off_r : o_t)
.input RawConstraints
```

Expression Type LHS

Expression Type RHS

Soufflé

```
.decl RawConstraints (yl : v_t,
                      el: e_t,
                      fl : f_id,
                      sl : s_id,
                      tl : t_id,
                      ql : q_id,
                      off_l : o_t,
                      yr : v_t,
                      er : e_t,
                      fr : f_id,
                      sr : s_id,
                      tr : t_id,
                      qr : q_id,
                      off_r : o_t)
.input RawConstraints
```

Offset LHS

Offset RHS

Soufflé

```
Constraints (el, gl, off_l, er, gr, off_r)
  :- RawConstraints (yl, el, fl, sl, tl, ql, off_l, yr, er, fr, sr, tr, qr, off_r)
  , VariableMap (yl, fl, sl, tl, ql, gl)
  , VariableMap (yr, fr, sr, tr, qr, gr)
```

```
SELECT gl
WHERE unique_id = <yl, fl, sl, tl, ql>
FROM VariableMap
```

Soufflé

```
Constraints (el, gl, off_l, er, gr, off_r)
  :- RawConstraints (yl, el, fl, sl, tl, ql, off_l, yr, er, fr, sr, tr, qr, off_r)
  , VariableMap (yl, fl, sl, tl, ql, gl)
  , VariableMap (yr, fr, sr, tr, qr, gr)
```

```
SELECT gr
WHERE unique_id = <yr, fr, sr, tr, qr>
FROM VariableMap
```

Soufflé

```
Constraints (el, gl, off_l, er, gr, off_r)
  :- RawConstraints (yl, el, fl, sl, tl, ql, off_l, yr, er, fr, sr, tr, qr, off_r)
  , VariableMap (yl, fl, sl, tl, ql, gl)
  , VariableMap (yr, fr, sr, tr, qr, gr)
```

Soufflé

```
Constraints (el, gl, off_l, er, gr, off_r)
  :- RawConstraints (yl, el, fl, sl, tl, ql, off_l, yr, er, fr, sr, tr, qr, off_r)
  , VariableMap (yl, fl, sl, tl, ql, gl)
  , VariableMap (yr, fr, sr, tr, qr, gr)
```

```
el gl + off_l = er gr + off_r
PLAIN gl + off_l = ADDRESS_OF gr + off_r
A + 0 = &B + 0
```

Soufflé

PP (gl, off_l, gr, off_r) :- Constraints (e_type_plain, gl, off_l, e_type_plain, gr, off_r).

PA (gl, off_l, gr, off_r) :- Constraints (e_type_plain, gl, off_l, e_type_addressof, gr, off_r).

PS (gl, off_l, gr, off_r) :- Constraints (e_type_plain, gl, off_l, e_type_star, gr, off_r).

SP (gl, off_l, gr, off_r) :- Constraints (e_type_star, gl, off_l, e_type_plain, gr, off_r).

SA (gl, off_l, gr, off_r) :- Constraints (e_type_star, gl, off_l, e_type_addressof, gr, off_r).

SS (gl, off_l, gr, off_r) :- Constraints (e_type_star, gl, off_l, e_type_star, gr, off_r).

Soufflé

PP (gl, off_l, gr, off_r) // A + 0 = B + 0

PA (gl, off_l, gr, off_r) // A + 0 = &B + 0

PS (gl, off_l, gr, off_r) // A + 0 = *B + 0

SP (gl, off_l, gr, off_r) // *A + 0 = B + 0

SA (gl, off_l, gr, off_r) // *A + 0 = &B + 0

SS (gl, off_l, gr, off_r) // *A + 0 = *B + 0

Soufflé

```
PointsTo (pointer, pointee) :- PA (pointer, _, pointee, _).

PointsTo (pointer, pointee) :- PP (pointer, _, temp, _)
  , PointsTo (temp, pointee).

PP (left, 0, right, 0) :- PS (left, _, temp, _)
  , PointsTo (temp, right).

PP (left, 0, right, 0) :- SP (temp, _, right, _)
  , PointsTo (temp, left).

PA (left, 0, pointee, 0) :- SA (temp, _, pointee, _)
  , PointsTo (temp, left).

PS (left, 0, right, 0) :- SS (temp, _, right, _)
  , PointsTo (temp, left).
```

Soufflé

```
PointsToFeedback (yl, fl, sl, tl, ql, yr, fr, sr, tr, qr) :- PointsTo (gl, gr)
 , VariableMap (yl, fl, sl, tl, ql, gl)
 , VariableMap (yr, fr, sr, tr, qr, gr)
 .
```

```
SELECT unique_id = <yl, fl, sl, tl, ql>
WHERE gl
FROM VariableMap
```

```
SELECT unique_id = <yr, fr, sr, tr, qr>
WHERE gr
FROM VariableMap
```

Bibliography

1. Golovanevsky, Olga, and Ayal Zaks. "Struct-reorg: current status and future perspectives." Proceedings of the GCC Developers' Summit. 2007.
2. Zhao, Peng, et al. "Forma: A framework for safe automatic array reshaping." ACM Transactions on Programming Languages and Systems (TOPLAS) 30.1 (2007): 2-es.
3. Yong, Suan Hsi, Susan Horwitz, and Thomas Reps. "Pointer analysis for programs with structures and casting." ACM SIGPLAN Notices 34.5 (1999): 91-103.
4. Balatsouras, George, and Yannis Smaragdakis. "Structure-sensitive points-to analysis for C and C++." International Static Analysis Symposium. Springer, Berlin, Heidelberg, 2016