A Verified Garbage Collector for Gallina

Shengyi Wang, Anshuman Mohan, Aquinas Hobor

APLAS NIER
November 18, 2019

Broad Problem

Verify graph-manipulating programs
written in executable C
with machine-checked correctness proofs

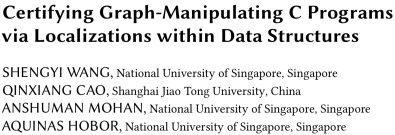
Broad Problem

Verify graph-manipulating programs
written in executable C
with machine-checked correctness proofs

Ubiquitous in critical areas

Broad Solution

Verified
Software
Toolchain



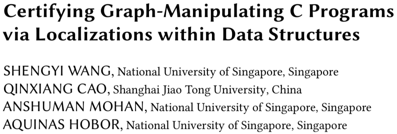
Certifying Graph-Manipulating C Programs via Localizations within Data Structures

SHENGYI WANG, National University of Singapore, Singapore
QINXIANG CAO, Shanghai Jiao Tong University, China
ANSHUMAN MOHAN, National University of Singapore, Singapore
AQUINAS HOBOR, National University of Singapore, Singapore

VST + CompCert + 40000 LOC library

Broad Solution

Verified
Software
Toolchain



Certifying Graph-Manipulating C Programs via Localizations within Data Structures

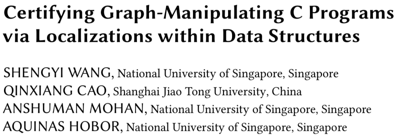
SHENGYI WANG, National University of Singapore, Singapore
QINXIANG CAO, Shanghai Jiao Tong University, China
ANSHUMAN MOHAN, National University of Singapore, Singapore
AQUINAS HOBOR, National University of Singapore, Singapore

VST + CompCert + 40000 LOC library

Powerful enough to verify executable code
against realistic specifications
expressed with mathematical graphs

Broad Solution

Verified
Software
Toolchain



Certifying Graph-Manipulating C Programs via Localizations within Data Structures

SHENGYI WANG, National University of Singapore, Singapore
QINXIANG CAO, Shanghai Jiao Tong University, China
ANSHUMAN MOHAN, National University of Singapore, Singapore
AQUINAS HOBOR, National University of Singapore, Singapore

VST + CompCert + 40000 LOC library

Powerful enough to verify executable code
against realistic specifications
expressed with mathematical graphs

[Wang *et. al.*, PACMPL OOPSLA 2019]

This Talk

Gallina ⇝ CompCert C ⇝ Assembly

This Talk

Gallina ⇝ CompCert C ⇝ Assembly

This Talk

Gallina $\rightsquigarrow$ CompCert C $\rightsquigarrow$ Assembly

Gallina assumes infinite memory
but CompCert C has a finite heap

This Talk

Gallina ⇝ CompCert C ⇝ Assembly

Gallina assumes infinite memory
    but CompCert C has a finite heap

Solution: garbage collect the CompCert C code

This Talk

Gallina ⇝ CompCert C ⇝ Assembly

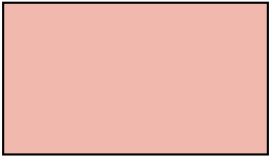
Gallina assumes infinite memory
but CompCert C has a finite heap

Solution: garbage collect the CompCert C code

New problem: verify the garbage collector

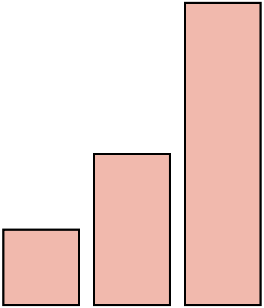
Garbage Collection: an Introduction

GC has jurisdiction over the heap



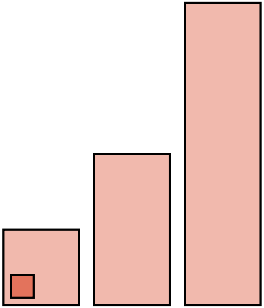
Garbage Collection: an Introduction

GC has jurisdiction over the heap



GC has jurisdiction over the heap

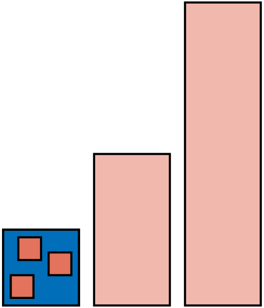
Mutator `mallocs` in special subheap



Garbage Collection: an Introduction

GC has jurisdiction over the heap

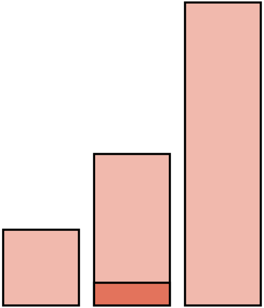
- Mutator `mallocs` in special subheap
  - If subheap is full



Garbage Collection: an Introduction

GC has jurisdiction over the heap

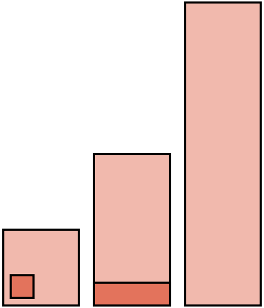
- Mutator `mallocs` in special subheap
  - If subheap is full call GC



Garbage Collection: an Introduction

GC has jurisdiction over the heap

- Mutator `mallocs` in special subheap
  - If subheap is full call GC and try again

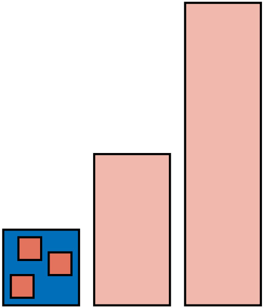


Intuitive Specification

Primum non nocere: first, do no harm

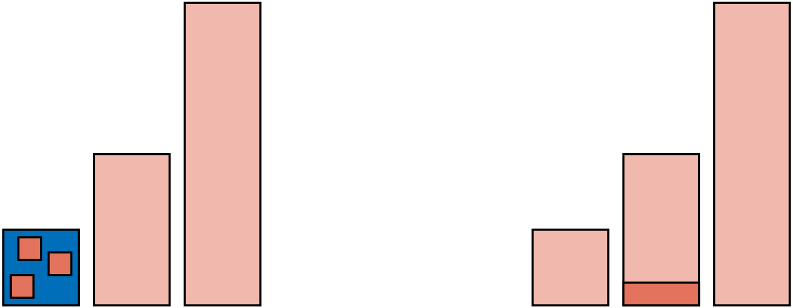
Intuitive Specification

Primum non nocere: first, do no harm



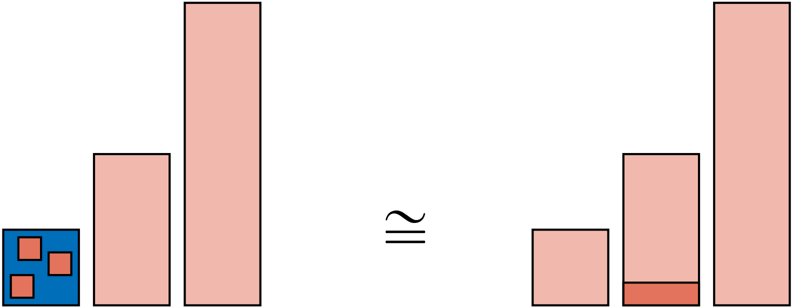
Intuitive Specification

Primum non nocere: first, do no harm



Intuitive Specification

Primum non nocere: first, do no harm



Our Garbage Collector

- 12 generations, doubling in size
- Functional mutator: no back pointers

Our Garbage Collector

- 12 generations, doubling in size
- Functional mutator: no back pointers
- Cheney's mark-and-copy collects gen to next
- Potentially triggers cascade of pairwise collections

Our Garbage Collector

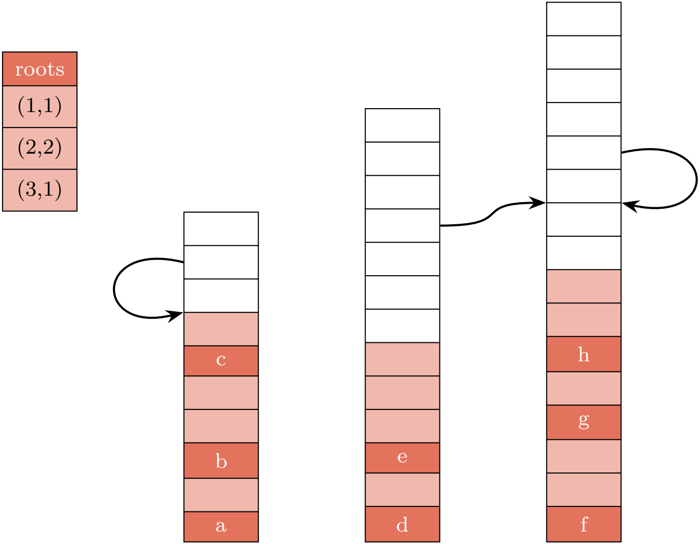
- 12 generations, doubling in size
- Functional mutator: no back pointers
- Cheney's mark-and-copy collects gen to next
- Potentially triggers cascade of pairwise collections
- Three key functions:

  `forward` copies individual objects

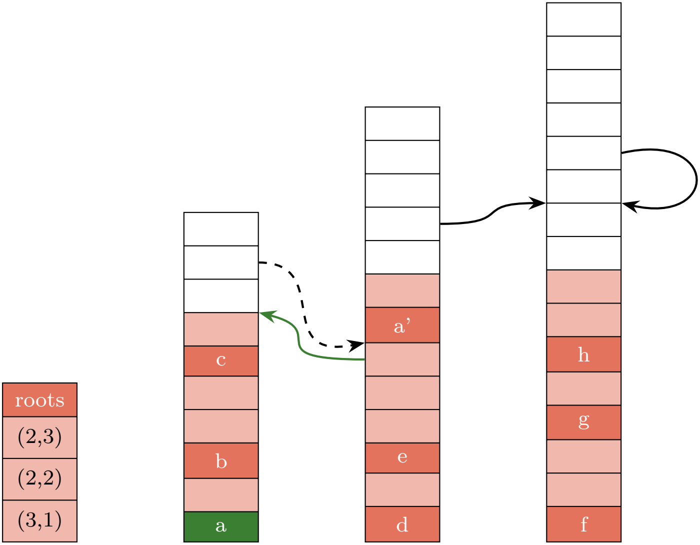
  `do_scan` repairs copied objects

  `forward_roots` kick-starts the collection

Overview of Operation’

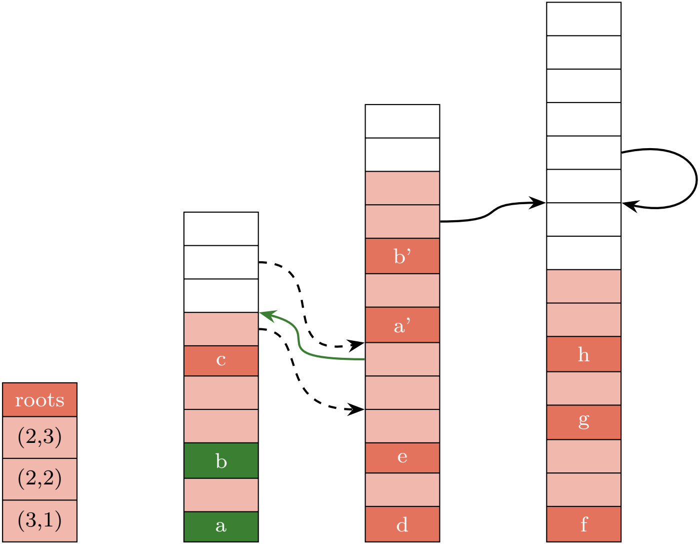


Overview of Operation’



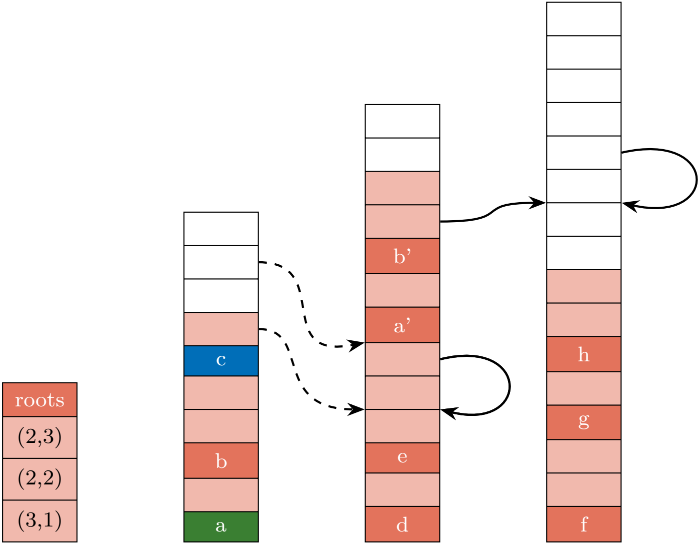
forward ✓

Overview of Operation’



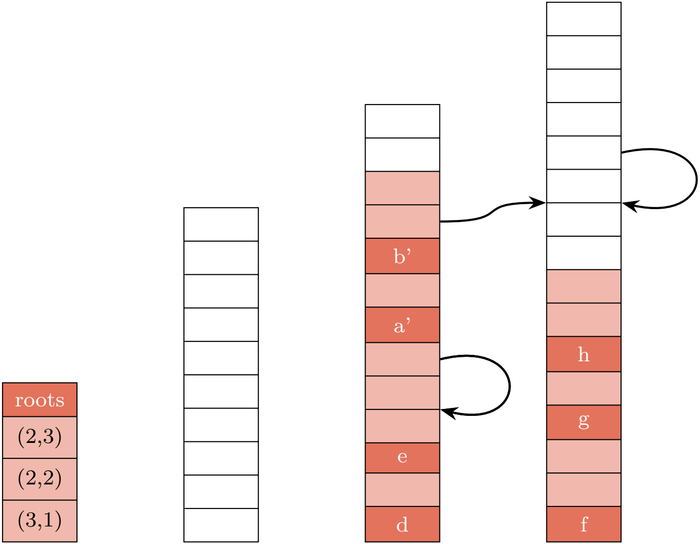
forward ✓

Overview of Operation’



forward ✓ do_scan ✓

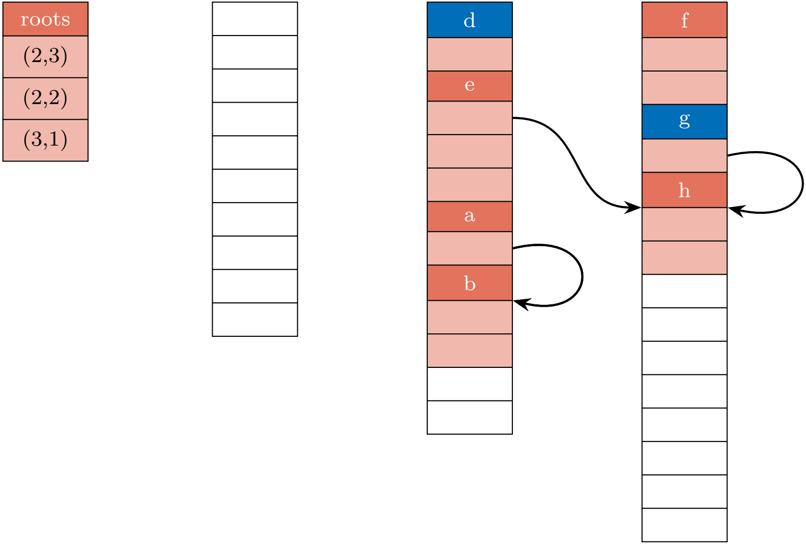
Overview of Operation’



roots
(2,3)
(2,2)
(3,1)

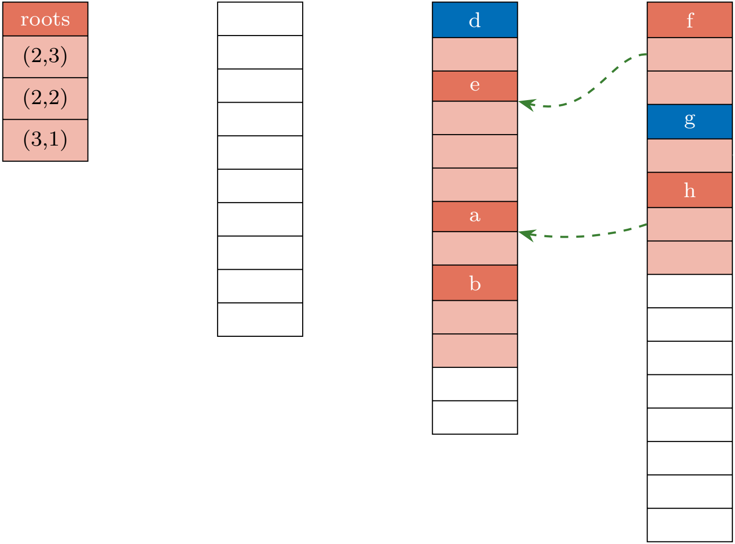
forward ✓ do_scan ✓ do_gen ✓

Non-Concerns



more garbage? ✗

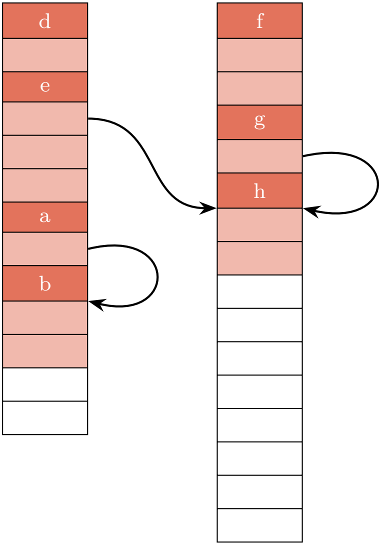
Non-Concerns



more garbage? ✗ back pointers? ✗

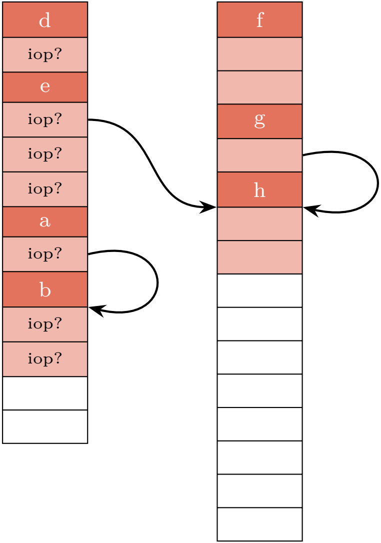
Sources of Complexity

variable-length objects



Sources of Complexity

variable-length objects
on-the-fly int/ptr disambiguation



Forward: a Deep Dive

forward is robust

`Forward`: a Deep Dive

`forward` is robust

  pointer?

Forward: a Deep Dive

forward is robust

pointer?

in from space?

`Forward`: a Deep Dive

`forward` is robust

pointer?

in `from` space?

already forwarded?

`Forward`: a Deep Dive

`forward` is robust and versatile

pointer?

in `from` space?

already forwarded?

`Forward`: a Deep Dive

`forward` is robust and versatile

pointer? called on root set

in `from` space?

already forwarded?

`Forward`: a Deep Dive

`forward` is robust and versatile

- pointer?
- in `from` space?
- already forwarded?

- called on root set
- called on heap

`Forward`: a Deep Dive

`forward` is robust and versatile

- pointer?
- in `from` space?
- already forwarded?

- called on root set
- called on heap

Specifying `forward` *functionally* is too hard

`Forward`: a Deep Dive

`forward` is robust and versatile

pointer?	called on root set
in `from` space?	called on heap
already forwarded?	

Specifying `forward` *functionally* is too hard

`forward_relation` explains how
the graph may change because of `forward`

forward_relation

```
Inductive forward_relation (from to : nat) :
    forward_t -> LGraph -> LGraph -> Prop :=
```

forward_relation

```
Inductive forward_relation (from to : nat) :
    forward_t -> LGraph -> LGraph -> Prop :=
| fr_v_not_in : forall (v : VType) (g : LGraph),
    vgeneration v <> from ->
    forward_relation from to (inl (inr v)) g g
```

forward_relation

```
Inductive forward_relation (from to : nat) :
    forward_t -> LGraph -> LGraph -> Prop :=
| fr_v_not_in : forall (v : VType) (g : LGraph),
    vgeneration v <> from ->
    forward_relation from to (inl (inr v)) g g
| fr_e_to_fwded : forall (e : EType) (g : LGraph),
    vgeneration (dst g e) = from ->
    raw_mark (vlabel g (dst g e)) = true ->
    let new_g := labeledgraph_gen_dst g e
      (copied_vertex (vlabel g (dst g e))) in
    forward_relation from to (inr e) g new_g
```

forward_relation, *cont.*

```
| fr_e_to_not_fwded_Sn : forall (e : EType) (g g' : LGraph),
    vgeneration (dst g e) = from ->
    raw_mark (vlabel g (dst g e)) = false ->
    let new_g :=
      labeledgraph_gen_dst (lgraph_copy1v g (dst g e) to)
      e (copy1v_new_v g to) in forward_loop from to
      (make_fields new_g (copy1v_new_v g to)) new_g g' ->
      forward_relation from to (inr e) g g'
```

forward_relation, *cont.*

```
| fr_e_to_not_fwded_Sn : forall (e : EType) (g g' : LGraph),
    vgeneration (dst g e) = from ->
    raw_mark (vlabel g (dst g e)) = false ->
    let new_g :=
      labeledgraph_gen_dst (lgraph_copy1v g (dst g e) to)
      e (copy1v_new_v g to) in forward_loop from to
      (make_fields new_g (copy1v_new_v g to)) new_g g' ->
      forward_relation from to (inr e) g g'
  with forward_loop (from to : nat) :
    list field_t -> LGraph -> LGraph -> Prop :=
| fl_nil : forall (g : LGraph), forward_loop from to  [] g g
| fl_cons : forall (g1 g2 g3 : LGraph)
                   (f : field_t) (fl : list field_t),
      forward_relation from to  (field2forward f) g1 g2 ->
      forward_loop from to  fl g2 g3 ->
      forward_loop from to  (f :: fl) g1 g3
```

Specification

Similar to `forward_relation`, we have

```
  forward_roots_relation
  do_scan_relation
  do_generation_relation
  garbage_collect_relation
```

Specification

Similar to `forward_relation`, we have

```
forward_roots_relation
do_scan_relation
do_generation_relation
garbage_collect_relation
```

A composition of these gives us our isomorphism

More Meat and Potatoes

Bugs in the source C code

- Cheney implemented too conservatively:
  only part of `to` space needs to be scanned

Bugs in the source C code

- Cheney implemented too conservatively:
  only part of `to` space needs to be scanned
  Performance doubled

Bugs in the source C code

- Cheney implemented too conservatively:
  only part of `to` space needs to be scanned
  Performance doubled

- Overflow in the following calculation:

  ```
  int space_size =
       h->spaces[i].limit - h->spaces[i].start;
  ```

Bugs in the source C code

- Cheney implemented too conservatively:
  only part of `to` space needs to be scanned
  Performance doubled

- Overflow in the following calculation:

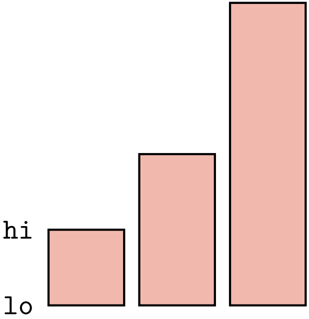
  ```
  int space_size =
      h->spaces[i].limit - h->spaces[i].start;
  ```

  Fixed by adjusting nursery size

Undefined behavior in C

Double-bounded pointer comparisons:

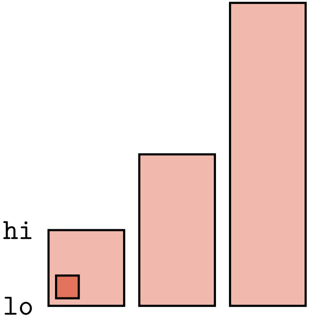
```
int Is_from(value * lo, value * hi, value * v) {
    return (lo <= v && v < hi); }
```



Undefined behavior in C

Double-bounded pointer comparisons:

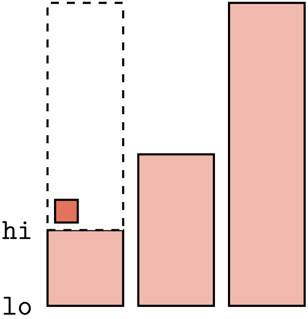
```
int Is_from(value * lo, value * hi, value * v) {
    return (lo <= v && v < hi); }
```



Undefined behavior in C

Double-bounded pointer comparisons:

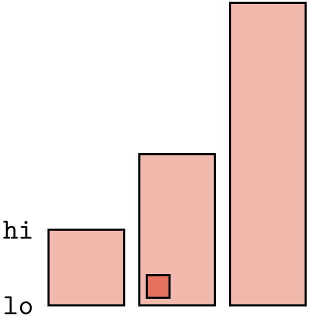
```
int Is_from(value * lo, value * hi, value * v) {
    return (lo <= v && v < hi); }
```



Undefined behavior in C

Double-bounded pointer comparisons:

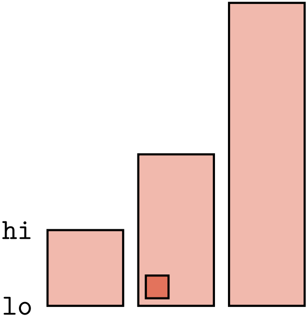
```
int Is_from(value * lo, value * hi, value * v) {
    return (lo <= v && v < hi); }
```



Undefined behavior in C

Double-bounded pointer comparisons:

```
int Is_from(value * lo, value * hi, value * v) {
    return (lo <= v && v < hi); }
```



Resolved using CompCert's `extcall_properties`

Undefined behavior in C, *cont.*

A classic OCaml trick:

```
int test_int_or_ptr (value x) {
    return (int)(((intnat)x)&1); }
```

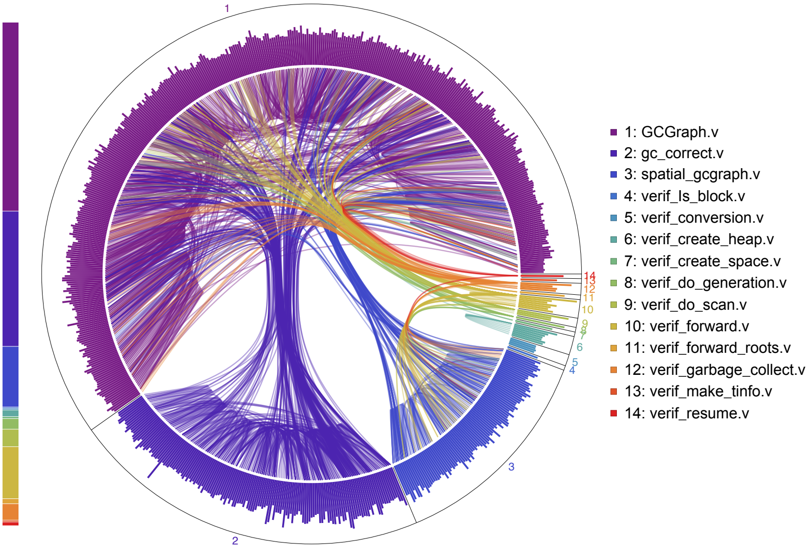
Undefined behavior in C, *cont.*

A classic OCaml trick:

```
int test_int_or_ptr (value x) {
    return (int)(((intnat)x)&1); }
```

Discussing `char` alignment issues with CompCert

Reusability: separation between pure and spatial reasoning



Future Work

Problems of a similar shape

Future Work

Problems of a similar shape

serialization

other collectors

Future Work

Problems of a similar shape

serialization

other collectors

Towards a verified GC for OCaml

Future Work

Problems of a similar shape
- serialization
- other collectors

Towards a verified GC for OCaml
- mutability
- calculate root set
- allow other datatypes

Future Work

Problems of a similar shape

- serialization
- other collectors

Towards a verified GC for OCaml

- mutability
- calculate root set
- allow other datatypes

Further refinements required in C semantics

- before we can specify and verify OCaml's GC?