C2ABS

or: How I Learned to Stop Worrying and Love the C Standard's Unspecified Evaluation Order

ABS Workshop 2021

Nathan Wasser

TECHNISCHE
UNIVERSITÄT
DARMSTADT

SF 4.0

Motivation

Analysis of sequential C programs, in particular:

- legacy code compiled with & run on older compilers & hardware
- with goal to locate parallelization opportunities
- in order to parallelize, compile and run with state-of-the-art

Re-use existing tools wherever possible

Problem

The C standard leaves some semantics less than fully specified.

Some nice advantages:

- Compilers can better take advantage of hardware
- Optimizations are more efficient

One big disadvantage:

- Program semantics might differ with compiler/environment

In particular in our case of legacy code being made state-of-the-art.

Types of Underspecified Behavior

Undefined behavior should be avoided, as the program is allowed to do anything at all in such a case. Examples:

- integer division by zero
- dereferencing a null pointer

Implementation defined behavior allows the compiler to make a public, global decision about how it will treat something. Ex.:

- negative ints (sign-magnitude, 1s- or 2s-complement)

Unspecified behavior gives a choice of semantics, one of which must be chosen, but the choice need not always be the same.

Unspecified Evaluation Order

It is unspecified what the evaluation order is for:

- function arguments
- subexpressions of (most) operators
- side effects of (most) expressions

Focus on this type of unspecified behavior for multiple reasons:

- In virtually all C programs (though it seldom changes the result)
- Poorly understood by even veteran C programmers[1]
- Severely neglected in almost all C analysis tools
- Can mask undefined behavior

[1] "Into the depths of C: elaborating the de facto standards" by Memarian et al. https://doi.org/10.1145/2908080.2908081

What Does This Program Return?

```
int x = 0;

int set(void) {
  x = 1;
  return 1;
}

int main(void) {
  return x + set();
}
```

Figure: `two_unspec.c`

Compiled with clang: 1

Compiled with gcc: 2

Other possibilities?!?

Unspec. Eval. Order Explosion

```
int x;

int set_x(const int ret_val) {
  x = 1;
  return ret_val;
}

int two_unspec(void) {
  x = 0;
  return x + set_x(1);
}

int add_zero(const int val) {
  x = 0;
  return val - x + set_x(0);
}
```

```
int fib(const int fib_num) {
  if (fib_num > 3)
    return fib(fib_num - 2) + add_zero(fib(fib_num - 1));
  else if (fib_num == 3)
    return two_unspec();
  else
    return 1;
}


int main(void) {
  return fib(n);
}
```

Figure: `one_to_fib_n.c`[2]

[2] https://shape-of-code.coding-guidelines.com/2011/06/18/fibonacci-and-jit-compilers/

Introducing Undefined Behavior

```
int x = 0;

int set(void) {
  x = 1;
  return 0;
}

int main(void) {
  return
    1 / (x + set());
}
```

Figure: `undefined.c`

Undefined behavior on one execution

Therefore, this program is undefined

But most tools don't recognize it!

Analysis with State-of-the-Art

CompCert C Compiler *"is formally verified, using machine-assisted mathematical proofs, to be exempt from miscompilation issues"* [3]

Frama-C *"is a suite of tools dedicated to the analysis of the source code of software written in C"* [4]

RV-Match *"provides strong correctness guarantees, leveraging a formally defined semantics of the target language to simulate execution symbolically"* [5]

Cerberus *"precisely defines the range of allowed behaviour, not just that of some specific implementation"* [6]

[3] https://compcert.org/compcert-C.html
[4] https://frama-c.com/
[5] https://runtimeverification.com/match/
[6] https://www.cl.cam.ac.uk/ pes20/cerberus/

CompCert C Compiler

Compilation of `two_unspec.c` with CompCert C compiler:

- Running executable returns 2
- Compiler is formally verified to adhere to C standard
- $\Rightarrow$ Guarantee that 2 is *one of* the valid results of program

However, CompCert website makes claim that could be misleading:

> *"the executable code it produces is proved to behave exactly as specified by the semantics of the source C program"* [7]

Semantics of `two_unspec.c` allow executables with different results!

Therefore, a guarantee about the compiled executable based on an input program is not a guarantee about the input program itself!

[7] https://compcert.org/compcert-C.html

CompCert C Compiler

Compilation of `undefined.c` with CompCert C compiler:

- Running executable returns 1
- Compiler is formally verified to adhere to C standard
- ⇒ Guarantee that 1 is *one of* the valid results of program

But, semantics of `undefined.c` allow executables to do anything!

Absence of dangerous undefined behavior is not guaranteed by knowing that the executable *may* return 1. However, this still holds:

> *"[T]he correctness proof of CompCert C guarantees that all safety properties verified on the source code automatically hold as well for the generated executable."* [8]

[8] https://compcert.org/compcert-C.html

Frama-C on `two_unspec.c`

Frama-C framework has plugins, which allow, e.g., *value analysis*[9]:

- `2 functions analyzed (out of 2): 100% coverage`
- `7 statements reached (out of 7): 100% coverage`
- `No errors or warnings raised during the analysis`
- `0 alarms generated by the analysis`
- `[eva:final-states] Values at end of function main:`
  `__retres` $\in \{2\}$

> *"At this time, Eva [...] silently chooses an order [...] These limitations will be addressed in a future version."* [10]

This promise (in version 23.1) has been in each user manual...

[9] https://frama-c.com/fc-plugins/eva.html
[10] https://frama-c.com/download/frama-c-eva-manual.pdf (page 41)

Frama-C on `undefined.c`

> *"[T]he fact that Eva computes correct, over-approximated sets of possible values prevents it from remaining silent on a program that contains a run-time error. [...] Only if the set of possible values computed by Eva does not contain zero is the warning omitted, and that means that the divisor really cannot be null at run-time."* [11]

And yet:

- `[eva:final-states] Values at end of function main:`
  `__retres` $\in \{1\}$
- `2 functions analyzed (out of 2): 100% coverage`
- `8 statements reached (out of 8): 100% coverage`
- `No errors or warnings raised during the analysis`
- `0 alarms generated by the analysis`

[11] https://frama-c.com/download/frama-c-eva-manual.pdf (page 10)

RV-Match

RV-Match supplies `kcc` (or `kclang`), a C compiler with knowledge of the K-framework's C semantics[12] which provides:

- Static checks while compiling source code to executable
- Dynamic checks when executable is run
- Error messages pointing to violation of the C standard

However, for both `two_unspec.c` and `undefined.c` neither compilation nor execution report any warnings at all!

[12] https://github.com/kframework/c-semantics

Cerberus

Cerberus has semantic models for a substantial fragment of C:

- Following the C11 standard
- Precisely defining the range of allowed behaviour
- Not bound to a specific implementation
- Allows exploring all behaviours of small test programs

With the right flags set in the Cerberus web interface[13] it:

- Correctly identifies both 1 and 2 as valid results of `two_unspec.c`
- Correctly identifies undefined behavior in `undefined.c`
- Times out after 45 seconds on `one_to_fib_4.c`

However, unspecified evaluation order of side effects is not correct!

[13] https://cerberus.cl.cam.ac.uk/cerberus (currently down)

Unspec. Eval. Order: Side Effects

Cerberus finds only 4 of 8 possible results for `assign_chain.c`:

```
int x = 1, y = 2, z = 3;

int sum(void) {
  return x + y + z;
}

int main(void) {
  return
    (x = y = z = 5) + sum();
}
```

Figure: `assign_chain.c`

Result depends on order of:

- side effect x $\leftarrow$ 5
- side effect y $\leftarrow$ 5
- side effect z $\leftarrow$ 5
- function call sum()

Cerberus fixes side effect order:

1. side effect z $\leftarrow$ 5
2. side effect y $\leftarrow$ 5
3. side effect x $\leftarrow$ 5

Solution: ABS

ABS Model

- Automatically extracted from C source code (C2ABS tool)
- Abstracts away from a specific compiler
- Faithfully models unspecified evaluation order
  - with method calls on same ABS object
  - as non-deterministic scheduling choice
- Can be automatically analyzed by a suite of tools built for ABS

As an Example

- We can show that `two_unspec` only has 2 possible results

Modelling Unspec. Eval. Order

```
int main(void) {
  return x + set();
}
```

```
class F_main(...) implements I {
  T call(...) {
   Fut<T> fx   = this!read_x();
   Fut<T> fset = this!call_set();
   Fut<T> fadd = this!add(fx, fset);
   await fadd?;
   return fadd.get;
  }
  T add(Fut<T> fx, Fut<T> fy) {
   await fx? & fy?;
   ...
  }
  ...
}
```

C2ABS: From Humble Beginnings

- Proof-of-concept
- Toy language, types restricted to `int` and modelled as `Int`
- Utilized SYCO to exhaustively validate extracted models
- Outperformed Cerberus on `one_to_fib_4.c`
- Correctly found all 8 results for `assign_chain.c`

Publication

Wasser, N., Heydari Tabar, A., Hähnle, R.
Modeling Non-deterministic C Code with Active Objects.
FSEN'19, 8th Intl. Conf., Vol. 11761 of LNCS, Springer, pp. 213-227

C2ABS: Rising Up in the World

- Slight tweaks to extraction process
- Auto-generation of many correct-by-construction specifications
- Translation of ACSL[14]-specifications in source code
- Utilized Crowbar to prove correctness of extracted models
- Case study added ACSL specs to `one_to_fib_n.c` and verified that C function `fib(n)` produces a value between 1 and $Fib(n)$

Unpublished work with Eduard Kamburjan

[14] https://frama-c.com/html/acsl.html

C2ABS: To Academic Tool

- Large subset of C including pointers and arrays
- Data types `Val`, `IntVal`, `PtrVal` model values
- Class `Mem` models memory using `List<Val>`
- Data type `Loc` models locations as `Mem` object plus `Int` offset
- Well-defined Calculus for extracting ABS models (ca. 40 rules)
- C2ABS implementation validated with input program test-suite
- Snapshot of source code made open source[15]
- Utilized Erlang simulator to validate extracted models

Publication

Wasser, N., Heydari Tabar, A., Hähnle, R.
Automated model extraction: From non-deterministic C code to active objects.
Science of Computer Programming Journal, Volume 204, April 2021, 102597

[15] https://github.com/se-tud/C2ABS-for-SCP-Journal-Paper

C2ABS: And Beyond?

New features:

- structs (with data type `StructVal` and complete rewrite of offsets)
- enums, typedefs
- floats as `Int` (allows for general analysis of program structure)

Work-in-progress:

- floats as `Float` (with support in Erlang simulator)
- OpenMP[16]-parallel-for pragma modelled with distributed objects

Future work:

- floats as IEEE-754 bits (as done by most compilers)
- Sequence point tracking to enable undefined behavior check

[16] https://www.openmp.org/

Summary

- C standard leaves evaluation order unspecified
- C compilers differ in chosen evaluation order
- Optimizations might further change that order
- Yet most existing tools ignore some or all of this!

$\Rightarrow$ C2ABS closes this gap, by:

- Modelling the *program*, rather than a single compiled executable
- Extracting to ABS where semantics are clear
- Faithfully maintaining unspecified evaluation order
- Allowing analysis using any and all tools available for ABS
- Tools utilized so far include: SYCO, Erlang simulator & Crowbar