An Executable Formal Semantics of C with Applications

Chucky Ellison

Department of Computer Science
University of Illinois

MVD'11 September 30, 2011

1. Introduction
   - Introduction
   - Motivation

2. Current Work
   - Semantics of C
   - Semantics-Based Analysis Tools

There is no formal semantics for C.

There ~~is~~ *was* no formal semantics for C *(until now)*.

There are partial semantics

- *Gurevich and Huggins* (1993) [ASM]
- *Cook, Cohen, and Redmond* (1994) [Denotational]
- *Cook and Subramanian* (1994) [Denotational]
- *Norrish* (1998) [Small- and big-step SOS]
- *Black* (1998) [Axiomatic]
- *Papaspyrou* (2001) [Denotational]
- *Blazy and Leroy* (2009) [Big-step SOS]

But, they simplify or leave out large parts of the language: Nondeterminism, casts, bitfields, unions, struct values, variadic functions, memory alignment, `goto`, dynamic memory allocation (`malloc()`), ...

But, Previous Definitions Leave out Features

			Definition			
Feature	GH	CCR	CR	No	Pa	BL
Bitfields	●	◐	○	○	◐	○
Enums	◐	●	○	○	●	○
Floats	○	○	○	○	◐	●
Struct/Union	●	●	●	◐	●	●
Struct as Value	○	○	○	●	○	○
Arithmetic	◐	●	●	○	●	●
Bitwise	○	●	○	○	●	●
Casts	◐	◐	○	◐	◐	●
Functions	●	●	◐	●	●	●
Exp. Side Effects	●	●	○	●	●	○
Variadic Funcs.	○	○	○	○	○	○
Eval. Strategies	○	◐	○	●	●	○
Overflow	○	○	○	○	○	○
Volatile	○	○	○	○	○	◐
Concurrency	○	○	○	○	○	○
Break/Continue	◐	●	◐	●	●	●
Goto	◐	○	○	○	●	○
Switch	◐	●	○	○	●	◐
Longjmp	○	○	○	○	○	○
Malloc	○	○	○	○	○	○

●: Fully Described
◐: Partially Described
○: Not Described

GH denotes *Gurevich and Huggins* (1993),
CCR is *Cook, Cohen, and Redmond* (1994),
CR is *Cook and Subramanian* (1994),
No is *Norrish* (1998),
Pa is *Papaspyrou* (2001), and
BL is *Blazy and Leroy* (2009).

No Semantics-Based Tools Either

There are many useful C analysis/verification tools, including:

- Lint/Purify/Coverity/Valgrind
- Blast
- Havoc
- Slam
- VCC
- Frama-C/Caduceus
- ...

These tools are based on approximative models of C.

The Need for Semantics Based Tools

Despite all this work on analyzing C programs...

There is still no formal semantics for C.

- Most tools are not even based on an *incomplete* semantics.
- Hard to argue for the soundness of the tools

Our Contribution

1. A complete formal semantics for C;

Our Contribution

1. A complete formal semantics for C;
2. Semantics-based analysis tools for C;

Our Contribution

1. A complete formal semantics for C;
2. Semantics-based analysis tools for C;
3. Constructive evidence that rewriting-based semantics scale.

Outline

1 Introduction
- Introduction
- Motivation

2 Current Work
- Semantics of C
- Semantics-Based Analysis Tools

C Specifications

- ANSI C (1989)
- ISO/IEC 9899:1990 “C90”
- ISO/IEC 9899:1999 “C99”
  - 540 pp.
  - 62 person-years of work (from 1995–1999)
  - Work continued until 2007
  - About 50 new features over C90, and many fixes
- ISO/IEC 9899:201x “C1X”
  - Adds first support for concurrency

Do We Really Need Formal Analysis Tools?

Question.

What happens when the approximative models of C fall short?

Answer.

Bad programs get proved correct, or behaviors go missing.

Two Unsequenced Writes to ‘x’

```
int main(void) {
   int x = 0;
   return (x = 1) + (x = 2);
}
```

Undefined according to C standard

GCC4, MSVC:	returns 4
GCC3, ICC, Clang:	returns 3

Both Frama-C and Havoc “prove” it returns 4

Undefined Behaviors are Fundamental to C

- That was just one kind of undefined behavior.
- There are over 200 explicitly undefined categories of behaviors in C. (Division by zero, referring to an object outside its lifetime, . . .)

Valid Nondeterminism

```
int r;

int f(int x) {
   return (r = x);
}

int main(void) {
   return f(1) + f(2), r;
}
```

Defined (Could return 1 or 2)

GCC, ICC, MSVC, Clang: returns 2

Both Frama-C and Havoc “prove” it can only return 2

Motivation Summary

When the models of C used by analysis tools are too simplistic

- Tools can draw incorrect conclusions about programs
- Hard to argue for soundness without a semantics to compare against

Outline

1. Introduction
   - Introduction
   - Motivation
2. Current Work
   - Semantics of C
   - Semantics-Based Analysis Tools

Outline

1. Introduction
   - Introduction
   - Motivation
2. Current Work
   - Semantics of C
   - Semantics-Based Analysis Tools

A Complete Definition of C

We have the first arguably complete formal definition of a conforming freestanding implementation of C.

A Complete Definition of C

We have the first arguably complete formal definition of a conforming freestanding implementation of C.

Conforming
: Must accept all portable programs, but can also accept non-portable programs.

A Complete Definition of C

We have the first arguably complete formal definition of a conforming freestanding implementation of C.

Conforming Must accept all portable programs, but can also accept non-portable programs.

Freestanding A precisely defined subset of all possible C features. This is the subset of C used when writing the kernel of an operating system.

A Complete Definition of C

We have the first arguably complete formal definition of a conforming freestanding implementation of C.

Conforming Must accept all portable programs, but can also accept non-portable programs.

Freestanding A precisely defined subset of all possible C features. This is the subset of C used when writing the kernel of an operating system. It includes only `<float.h>` `<iso646.h>`, `<limits.h>`, `<stdalign.h>`, `<stdarg.h>`, `<stdbool.h>`, `<stddef.h>`, and `<stdint.h>`.

Our Current Work on C

- Tested against the GCC torture tests:
  - Of 1093 tests, 776 tests appear to be standards compliant. Of those, we pass 770 (>99%).

Our Work is More Complete

Feature	Definition						
	GH	CCR	CR	No	Pa	BL	ER
Bitfields	●	◐	○	○	◐	○	●
Enums	◐	●	○	○	●	○	●
Floats	○	○	○	○	◐	●	◐
Struct/Union	●	●	●	◐	●	●	●
Struct as Value	○	○	○	●	○	○	●
Arithmetic	◐	●	●	○	●	●	●
Bitwise	○	●	○	○	●	●	●
Casts	◐	◐	○	◐	◐	●	●
Functions	●	●	◐	●	●	●	●
Exp. Side Effects	●	●	○	●	●	○	●
Variadic Funcs.	○	○	○	○	○	○	●
Eval. Strategies	○	◐	○	●	●	○	●
Overflow	○	○	○	○	○	○	●
Volatile	○	○	○	○	○	◐	○
Concurrency	○	○	○	○	○	○	◐
Break/Continue	◐	●	◐	●	●	●	●
Goto	◐	○	○	○	●	○	●
Switch	◐	●	○	○	●	◐	●
Longjmp	○	○	○	○	○	○	●
Malloc	○	○	○	○	○	○	●

●: Fully Described
◐: Partially Described
○: Not Described

GH denotes *Gurevich and Huggins* (1993),
CCR is *Cook, Cohen, and Redmond* (1994),
CR is *Cook and Subramanian* (1994),
No is *Norrish* (1998),
Pa is *Papaspyrou* (2001),
BL is *Blazy and Leroy* (2009), and
ER is *Ellison and Roșu* (our current work).

Some Information about Our Semantics

- Mechanized in $\mathbb{K}$ Framework
- 150 syntactic operators
- 5900 source lines of code
- 1200 different $\mathbb{K}$ rules
- Only 80 rules for statements
- Only 160 for expressions
- 500 rules for declarations and types!

Outline

1. Introduction
   - Introduction
   - Motivation
2. Current Work
   - Semantics of C
   - Semantics-Based Analysis Tools

Semantics-Based Analysis Tools

These tools are provided “for free” by rewriting logic and $\mathbb{K}$:

- Interpreter
- Debugger
- State-space search
- LTL Model-checker

Our tests have shown these tools work just as well with C as with tools based on definitions of smaller languages.

Normal Interpretation

```
$ cat hello_world.c

int main(void) {
   printf("Hello world!\n");
}
```

Normal Interpretation

```
$ cat hello_world.c

int main(void) {
   printf("Hello world!\n");
}

$ kcc hello_world.c
$ ./a.out

Hello world!
```

Interpretation to Find Bugs

```
$ cat buggy_strcpy.c

int main(void) {
   char dest[5], src[5] = "hello";
   strcpy(dest, src);
}
```

Interpretation to Find Bugs

```
$ cat buggy_strcpy.c

int main(void) {
   char dest[5], src[5] = "hello";
   strcpy(dest, src);
}

$ kcc buggy_strcpy.c
$ ./a.out

ERROR! KCC encountered an error while executing this program.
Description: Reading outside the bounds of an object.
File: buggy_strcpy.c
Function: strcpy
Line: 4
```

Search to Find Bugs

```
$ cat eval_order.c

int main(void) {
   int x = 0, y = 0;
   return x++ + (y, x);
}
```

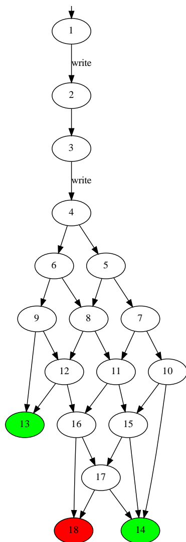
Search to Find Bugs

```
$ cat eval_order.c

int main(void) {
   int x = 0, y = 0;
   return x++ + (y, x);
}

$ kcc eval_order.c
$ SEARCH=1 ./a.out
```

Search to Find Bugs (Cont.)

```
3 solutions found
--------------------------------------------------------------
Solution 1
Program completed successfully
Return value: 1
--------------------------------------------------------------
Solution 2
Program completed successfully
Return value: 0
--------------------------------------------------------------
Solution 3
Program got stuck
File: eval_order.c
Line: 3
Description: Unsequenced side effect on scalar object with
value computation of same object.
```

Search to Find Bugs (Cont.)



LTL-Based Model Checking

```
$ cat lights.c

typedef enum {green, yellow, red} state;
state lightNS = green; state lightEW = red;
int changeNS() {
  switch (lightNS) {
    case(green): lightNS = yellow; return 0;
    case(yellow): lightNS = red; return 0;
    case(red):
      if (lightEW == red) { lightNS = green; } return 0;
  }
}
...
int main(void) { while(1) { changeNS() + changeEW(); } }
#pragma __ltl safety: [] (lightNS == red \/ lightEW == red)
#pragma __ltl progressNS: [] <> (lightNS == green)
```

LTL-Based Model Checking (Cont.)

```
$ kcc lights.c
$ MODELCHECK=safety ./a.out
```

LTL-Based Model Checking (Cont.)

```
$ kcc lights.c
$ MODELCHECK=safety ./a.out

False! The safety property does not hold.
```

LTL-Based Model Checking (Cont.)

```
$ kcc lights.c
$ MODELCHECK=safety ./a.out

False! The safety property does not hold.

# change "changeNS() + changeEW()" to "changeNS(); changeEW()"
```

LTL-Based Model Checking (Cont.)

```
$ kcc lights.c
$ MODELCHECK=safety ./a.out

False! The safety property does not hold.

# change "changeNS() + changeEW()" to "changeNS(); changeEW()"

$ kcc lights.c
$ MODELCHECK=safety ./a.out
```

LTL-Based Model Checking (Cont.)

```
$ kcc lights.c
$ MODELCHECK=safety ./a.out

False! The safety property does not hold.

# change "changeNS() + changeEW()" to "changeNS(); changeEW()"

$ kcc lights.c
$ MODELCHECK=safety ./a.out

True! The safety property holds.
```

LTL-Based Model Checking (Cont.)

```
$ kcc lights.c
$ MODELCHECK=safety ./a.out

False! The safety property does not hold.

# change "changeNS() + changeEW()" to "changeNS(); changeEW()"

$ kcc lights.c
$ MODELCHECK=safety ./a.out

True! The safety property holds.

$ MODELCHECK=progressNS ./a.out
```

LTL-Based Model Checking (Cont.)

```
$ kcc lights.c
$ MODELCHECK=safety ./a.out

False! The safety property does not hold.

# change "changeNS() + changeEW()" to "changeNS(); changeEW()"

$ kcc lights.c
$ MODELCHECK=safety ./a.out

True! The safety property holds.

$ MODELCHECK=progressNS ./a.out

True! The progressNS property holds.
```

Summary

We have the first arguably complete formal semantics of C

- Is executable, and has been thoroughly tested against the GCC torture test suite
- Can be used to generate analysis tools for finding program bugs
- Demonstrates that rewriting-based semantics can handle large languages and all their gritty details

What is Undefined Behavior?

undefined behavior Behavior, upon use of a non-portable or erroneous program construct or of erroneous data, [with] no requirements.

- In essence, this refers to problematic situations that are hard to identify statically or expensive to identify dynamically
- Implementations can do *anything* for undefined behavior, including failing to compile, crashing, or appearing to work
- Examples: division by zero, referring to an object outside its lifetime, `(x = 1) + (x = 2)`