An Executable Formal Semantics of C with Applications

Chucky Ellison

Department of Computer Science
University of Illinois

August 2011
1 Introduction
   - Introduction
   - Motivation

2 Current Work
   - Current Work on C
   - Work on Analysis Tools
There is no formal semantics for C.
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

<table>
<thead>
<tr>
<th>Feature</th>
<th>Definition</th>
<th>GH</th>
<th>CCR</th>
<th>CR</th>
<th>No</th>
<th>Pa</th>
<th>BL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitfields</td>
<td></td>
<td>●</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>●</td>
<td>☐</td>
</tr>
<tr>
<td>Enums</td>
<td></td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>☐</td>
<td>○</td>
<td>☐</td>
</tr>
<tr>
<td>Floats</td>
<td></td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>Struct/Union</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Struct as Value</td>
<td></td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Arithmetic</td>
<td></td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Bitwise</td>
<td></td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Casts</td>
<td></td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Functions</td>
<td></td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Exp. Side Effects</td>
<td></td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Variadic Funcs.</td>
<td></td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Eval. Strategies</td>
<td></td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Overflow</td>
<td></td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Volatile</td>
<td></td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Concurrency</td>
<td></td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Break/Continue</td>
<td></td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Goto</td>
<td></td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Switch</td>
<td></td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Longjmp</td>
<td></td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Malloc</td>
<td></td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

●: 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.
Despite all this work on analyzing C programs...
There is still no formal semantics for C.
Despite all this work on analyzing C programs... There is still no formal semantics for C.

- Hard to argue for the soundness of the tools
The Need for Semantics Based Tools

Despite all this work on analyzing C programs...

There is still no formal semantics for C.

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

2 Current Work
   - Current Work on C
   - Work on Analysis Tools
C Specifications

- ANSI C (1989)
  - 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'

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

Undefined according to C standard

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC4, MSVC</td>
<td>returns 4</td>
</tr>
<tr>
<td>GCC3, ICC, Clang</td>
<td>returns 3</td>
</tr>
</tbody>
</table>

Both Frama-C and Havoc “prove” it returns 4
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, \((x = 1) + (x = 2)\)
Left Shift of Negative Number

```c
int main(void){
    return -5 << 2;
}
```

**Undefined** according to C standard

- GCC, ICC, Clang: returns \(-20\)
- MSVC: returns 127

Both Frama-C and Havoc “prove” it returns \(-20\)
Write to String Literal

```c
int main(void) {
    "foo"[0] = 'x';
    return "foo"[0];
}
```

**Undefined according to C standard**

- GCC: doesn’t compile
- ICC, Clang: segmentation fault
- MSVC: returns 'f'

Frama-C “proves” it returns ’x’
Undefined Behaviors are Fundamental to C

This was just 3 undefined programs. There are over 190 explicitly undefined categories of behaviors in C.
Valid Nondeterminism

```c
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
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.
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>`. 
1 Introduction
   - Introduction
   - Motivation

2 Current Work
   - Current Work on C
   - Work on Analysis Tools
Outline

1. Introduction
   - Introduction
   - Motivation

2. Current Work
   - Current Work on C
   - Work on Analysis Tools
Our Current Work on C

- We currently have a preliminary semantics that is *more complete* than other semantics to date.
- Tested against the GCC torture tests:
  - Of 1093 tests, *776 tests* appear to be standards compliant. Of those, we pass 770 (>99%).

```c
int f(void){
    signed char c = -1;
    return c < 0;
}

int main(void){
    if (f() != 1) { abort(); }
    return 0;
}
```
Our Current Work on C

- We currently have a preliminary semantics that is *more complete* than other semantics to date.
- Tested against the GCC torture tests:
  - Of 1093 tests, *776 tests* appear to be standards compliant. Of those, we pass 770 (>99%).

```c
int f(void){
    signed char c = -1;
    return c < 0;
}

int main(void){
    if (f() != 1) { abort(); }
    return 0;
}
```
## Current Work on C

### Work on Analysis Tools

Our Current Work is Already More Complete

<table>
<thead>
<tr>
<th>Feature</th>
<th>GH</th>
<th>CCR</th>
<th>CR</th>
<th>No</th>
<th>Pa</th>
<th>BL</th>
<th>ER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitfields</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>Enums</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>Floats</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Struct/Union</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Struct as Value</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Bitwise</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Casts</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Functions</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Exp. Side Effects</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Variaedic Funcs.</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Eval. Strategies</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Overflow</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Volatile</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Concurrency</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Break/Continue</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Goto</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Switch</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Longjmp</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Malloc</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

- ●: 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 Rosu (our current work).
Some Statistics about Our Semantics

- Mechanized in \textit{K} Framework
- 150 syntactic operators
- 5900 source lines of code
- 1200 different \textit{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
   - Current Work on C
   - Work on Analysis Tools
These tools are provided “for free” by rewriting logic and Maude:

- Interpreter
- Debugger
- State-space search

Our tests have shown these tools work just as well with C as with tools based on definitions of smaller languages.
Interpretation to Find Bugs
Search to Find Bugs

Chucky Ellison
An Executable Formal Semantics of C with Applications
LTL-Based Model Checking
Test Case Reduction
Duff’s Device

- Unstructured control flow (goto, switches)

```c
int n = (count+7)/8;
switch(count%8) {
    case 0: do{ *dest++ = *src++; 
    case 7: *dest++ = *src++; 
    case 6: *dest++ = *src++; 
    case 5: *dest++ = *src++; 
    case 4: *dest++ = *src++; 
    case 3: *dest++ = *src++; 
    case 2: *dest++ = *src++; 
    case 1: *dest++ = *src++; 
} while(--n>0);
}
```