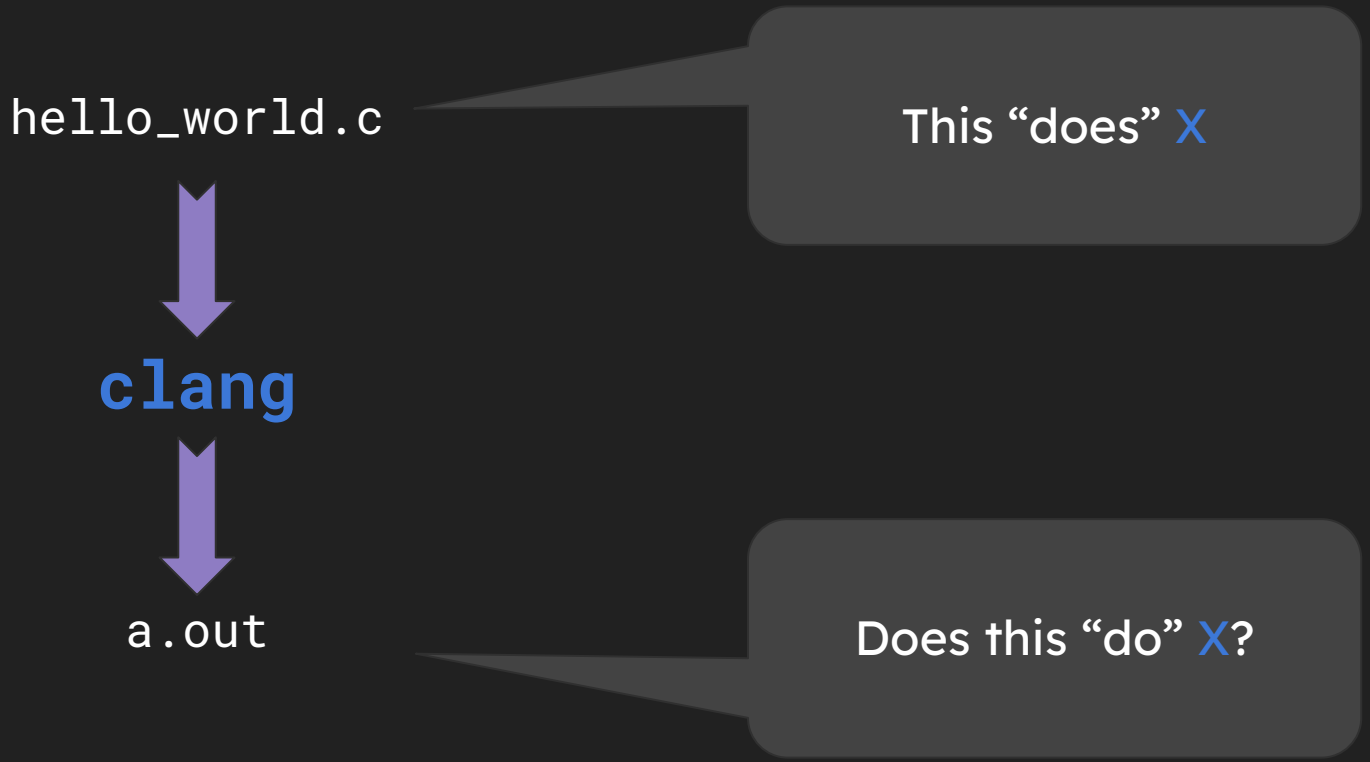
Mechanized Semantics *for the Clight Subset of the C Language*

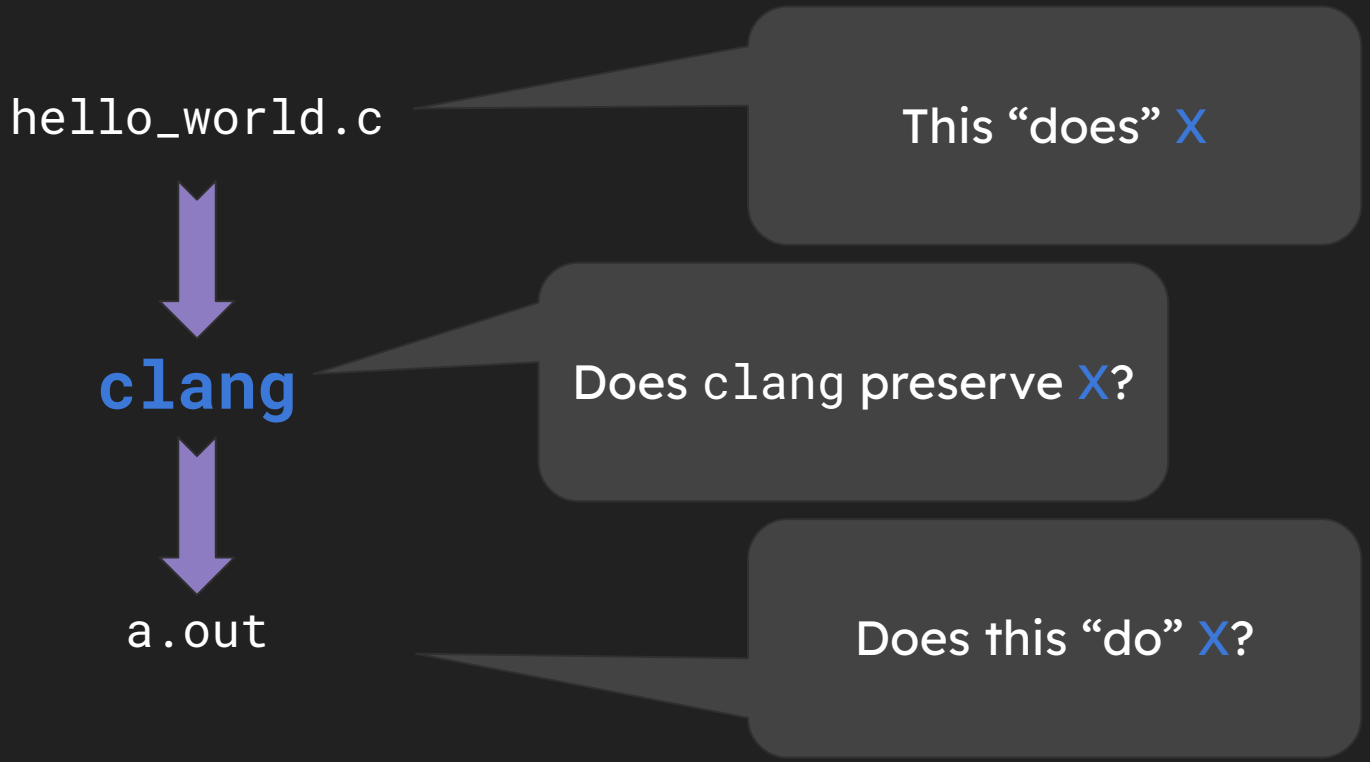
J Autom Reasoning (2009) 43:263–288
DOI 10.1007/s10817-009-9148-3

Mechanized Semantics for the Clight Subset of the C Language

Sandrine Blazy · Xavier Leroy


hello_world.c
clang
a.out


hello_world.c
clang
a.out
This “does” X
Does this “do” X?


hello_world.c
clang
a.out
This “does” X
Does clang preserve X?
Does this “do” X?

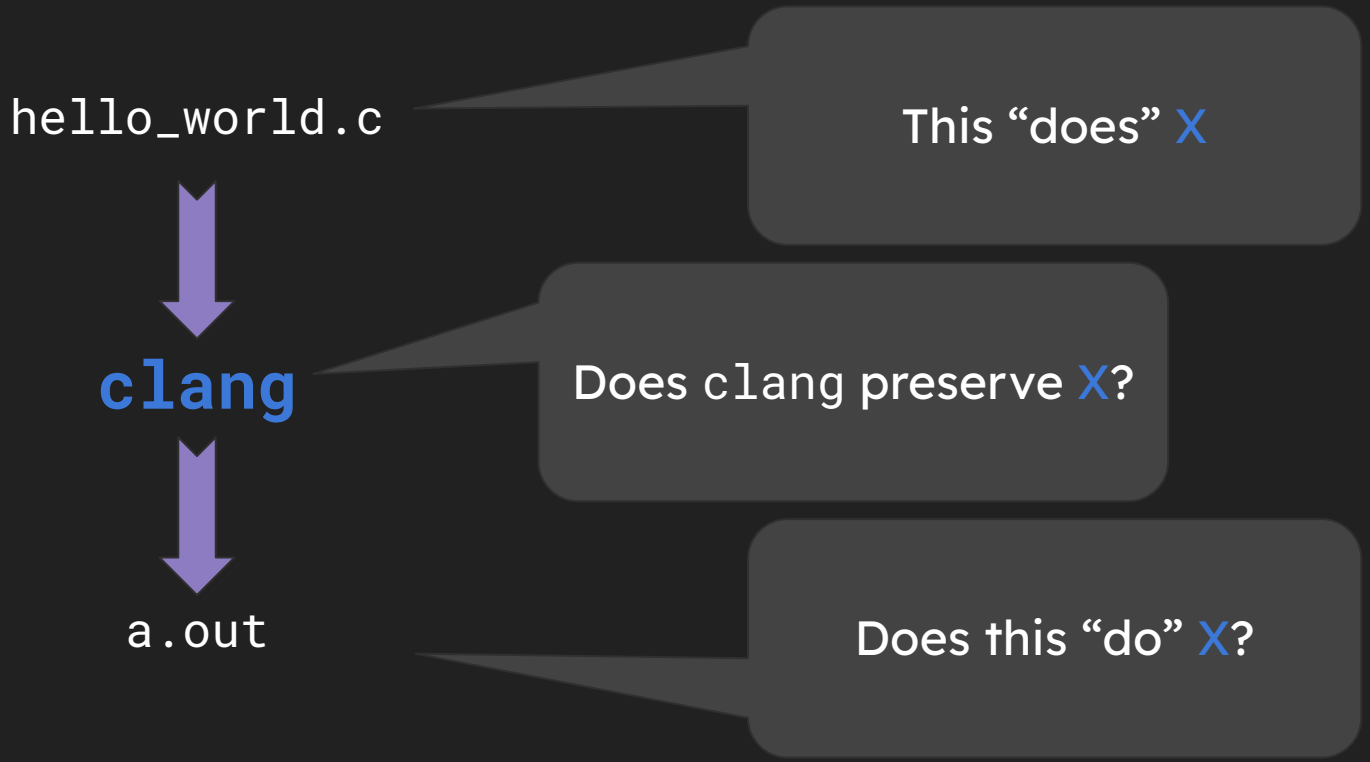
X ≈ “observable behaviors” of the program

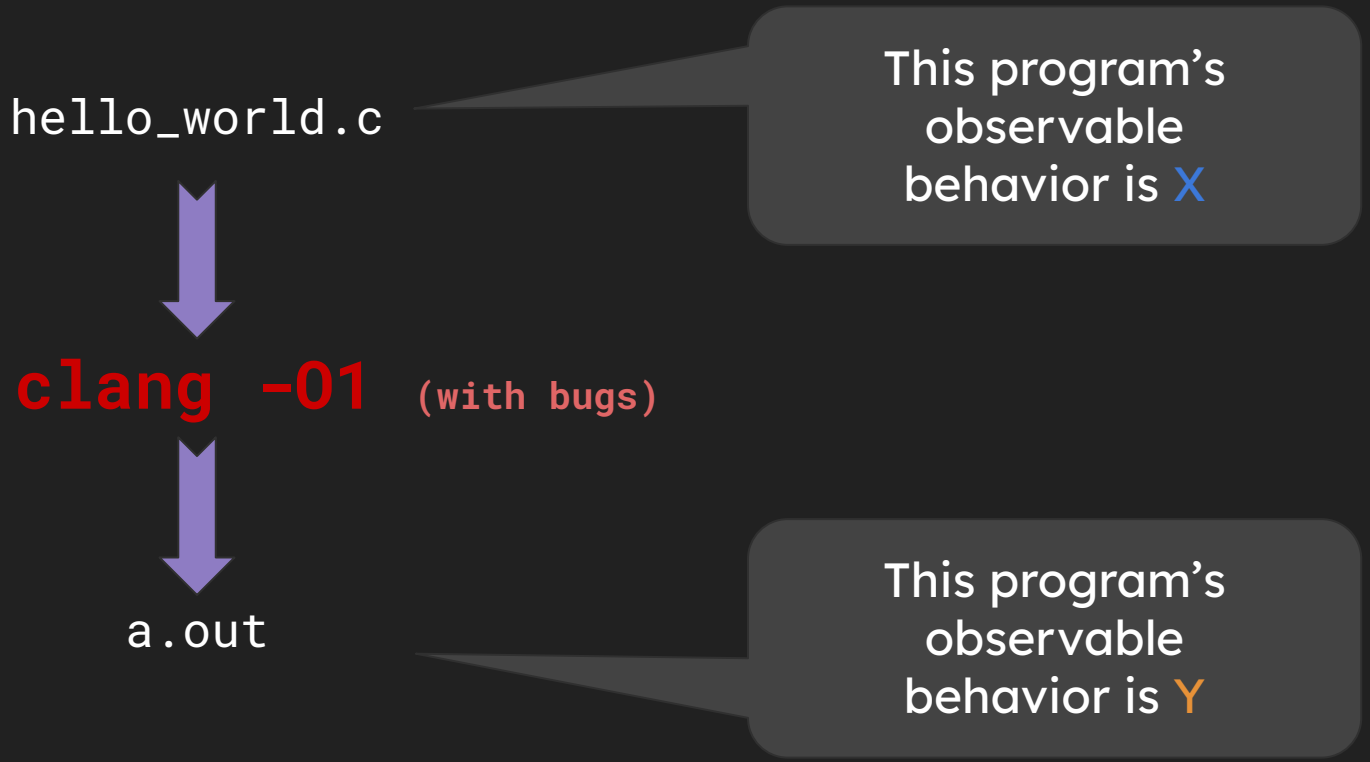
I/O Events

(e.g. system calls)

Result

(e.g. non-termination)


hello_world.c
clang
a.out
This “does” X
Does clang preserve X?
Does this “do” X?


hello_world.c
clang -O1
(with bugs)
a.out
This program's observable behavior is X
This program's observable behavior is Y

Exit Status of This Program?

```
int x = 0;
int test(int* restrict ptr) {
  *ptr = 1;
  if (ptr == &x)
    *ptr = 2;
  return *ptr;
}
int main() { return test(&x); }
```

Exit Status of This Program?

```c
int x = 0;

int test(int* restrict ptr) {

  *ptr = 1;

  if (ptr == &x)

    *ptr = 2;

  return *ptr;

}

int main() { return test(&x); }
```

```
❯ clang -O1 err3.c && ./a.out
Clang 17.0.6
test(&x) = 1
```

```
❯ gcc -O3 err3.c && ./a.out
13.2.1 20240417
test(&x) = 2
```

```
❯ ccomp -O3 err3.c && ./a.out
314
test(&x) = 2
```

https://github.com/llvm/llvm-project/issues/59679

miscompilation

How do we know if our compiler preserves semantics?

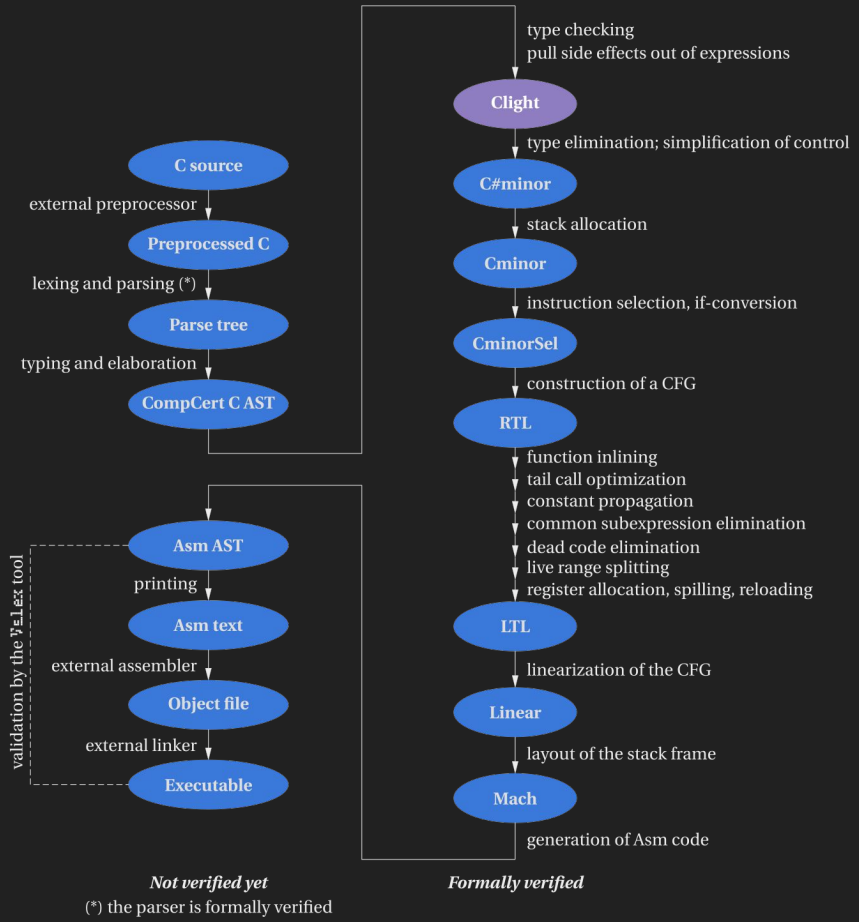
That is, preserves a program's observable behaviors

How do we know if our compiler preserves semantics?

That is, preserves a program's observable behaviors

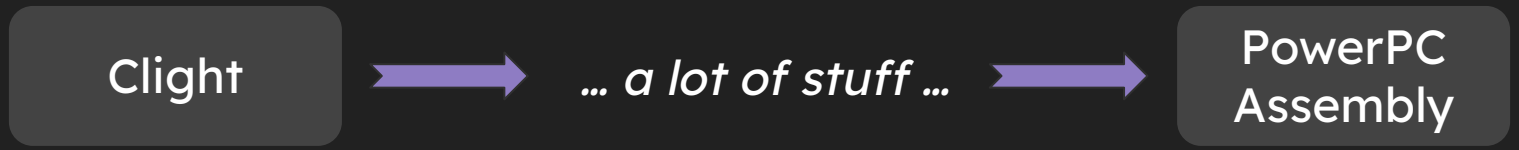
Formal verification.

CompCert ≈ 90% verified compiler

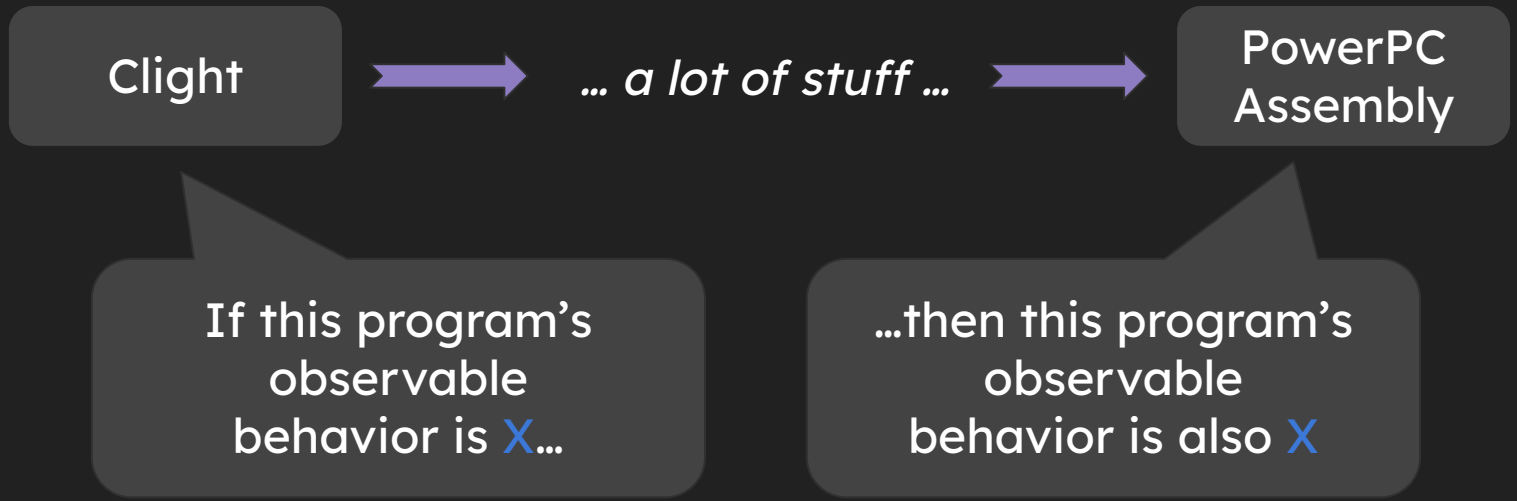


(*) the parser is formally verified

CompCert ≈ 90% verified compiler



CompCert ≈ 90% verified compiler



How do we know CompCert preserves semantics?

By proving that the observable behaviors
before and after compilation are the same.

How do we know CompCert preserves semantics?

By proving that the observable behaviors before and after compilation are the same.

Requires a formal semantics for observable behaviors

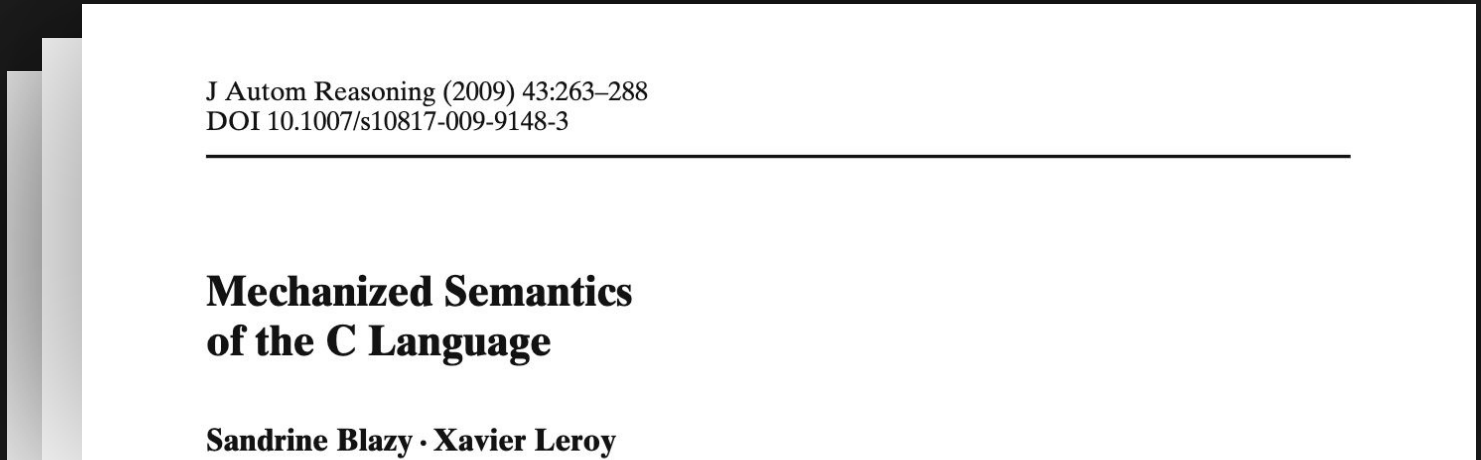
How do we know CompCert preserves semantics?

By proving that the observable behaviors
before and after compilation are the same.

Requires a formal semantics for observable behaviors

Which itself is going to require a formal semantics for the source language

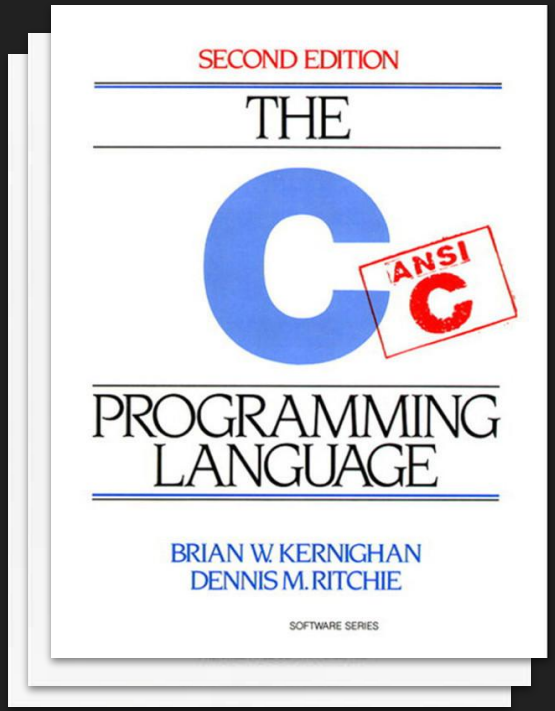
Mechanized Semantics *of the C* Language



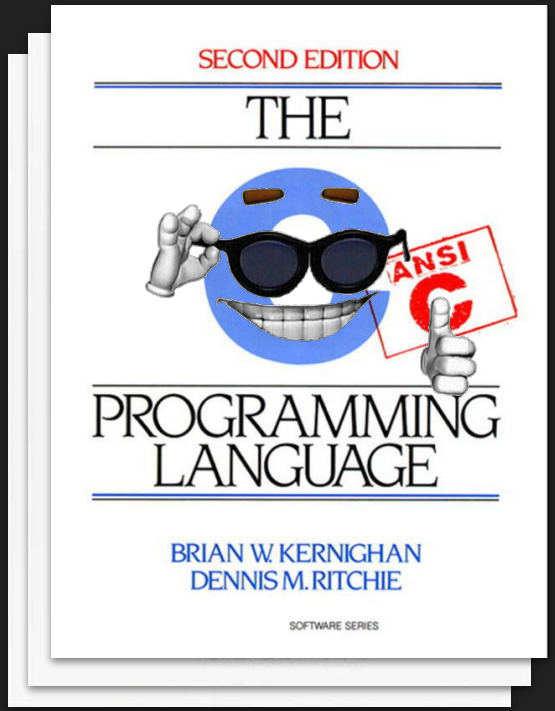
J Autom Reasoning (2009) 43:263–288
DOI 10.1007/s10817-009-9148-3

Mechanized Semantics of the C Language

Sandrine Blazy · Xavier Leroy



Undefined and unspecified behaviors...



Undefined and unspecified behaviors…

…don’t play nice with formal semantics

Mechanized Semantics *for the Clight Subset of the C Language*

J Autom Reasoning (2009) 43:263–288
DOI 10.1007/s10817-009-9148-3

Mechanized Semantics for the Clight Subset of the C Language

Sandrine Blazy · Xavier Leroy

This paper defines a formal semantics for the Clight language

Clight is a subset of C that seeks to reduce C's ambiguity

J Autom Reasoning (2009) 43:263–288
DOI 10.1007/s10817-009-9148-3

Mechanized Semantics for the Clight Subset of the C Language

Sandrine Blazy · Xavier Leroy

How does Clight formalize observable behaviors?

I/O Events

Traces of external function calls are tracked

Clight statements may update the trace

$$G \vdash F(v_{\text{args}}), M \xrightarrow{\text{T}} \infty$$

Results

Programs may terminate with a final value or diverge

$$\vdash P \Rightarrow \texttt{terminates}(t, n)$$

$$\vdash P \Rightarrow \texttt{diverges}(T)$$

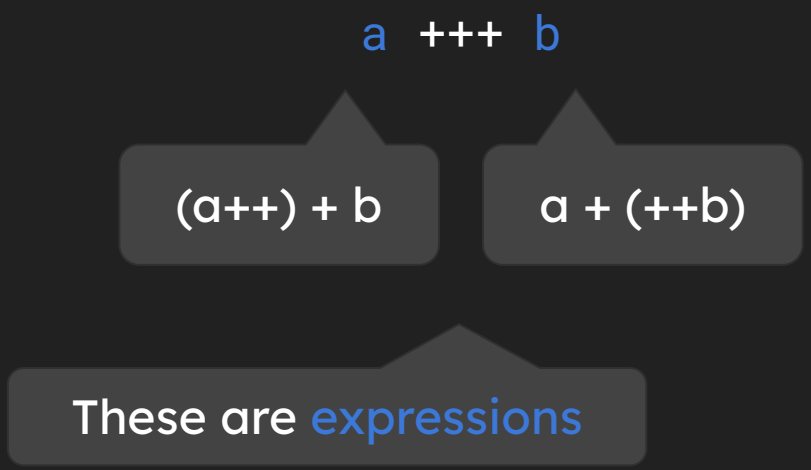
How does Clight handle ambiguous C features?

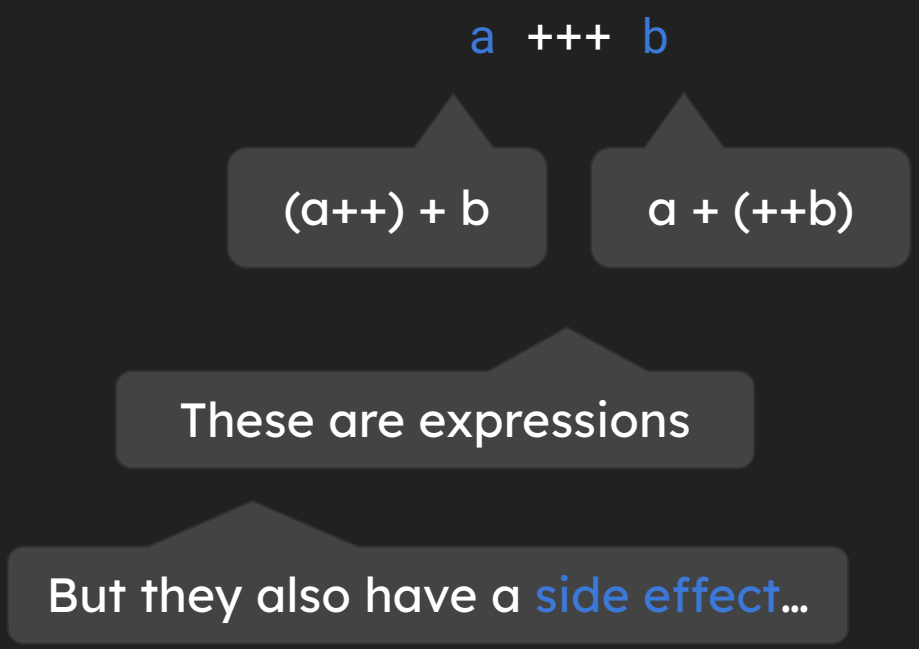
How do we evaluate the following code snippet?

```
a +++ b
```

(a++) + b

a + (++b)


a +++ b
(a++) + b
a + (++b)
These are expressions


a +++ b
(a++) + b
a + (++b)
These are expressions
But they also have a side effect...

In C, expressions can have side effects:

```
a * b++
```

...and the evaluation order of these expressions is sometimes ambiguous:

```
a +++ b
```

In Clight, expressions cannot have side effects

…which gets rid of ambiguous evaluation orders:

a + b

In Clight, expressions cannot have side effects

- No function calls
- No assignments (++, +=, etc.)

These features must appear in Clight statements

In Clight, expressions cannot have side effects

- Simplifies semantics for expressions
- Allows expressions to be evaluated in any order

Evaluation order partially unspecified in C

How do we evaluate the following code snippet?

```
a +++ b
```

(a++) + b

a + (++b)

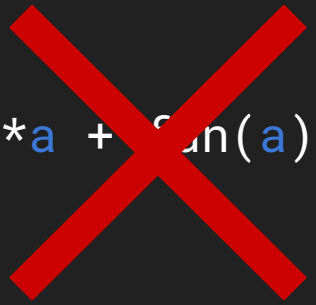
How do we evaluate the following code snippet?

Trick question. This program is incorrect.

How do we evaluate the following code snippet?

```
*a + fun(a)
```

How do we evaluate the following code snippet?



Trick question. This program is incorrect.

Other examples...

What is the size (bytes) of the following variable?

```
long a = 0;
```

What is the size (bytes) of the following variable?

```
long a = 0;
```

4? 8? It depends...

Sizes of types in C can be ambiguous

Clight uses exact, specified sizes for its types...

What is the size (bytes) of the following variable?

```
I32 a = 0;
```

4.

What about other interesting C features?

Pointers become locations

Locations come in the form (b, δ)

Block within memory

Offset within block

Pointers become locations

Locations come in the form (b, δ)

Pointer arithmetic simply adds to δ in (b, δ)

Pointers become locations

Locations come in the form (b, δ)

Pointer arithmetic simply adds to δ in (b, δ)

Functions and arrays become locations when passed as arguments

Variables can be in:

- Global scope
- Function scope

Clight does not have block scope

Variables can be in:

- Global scope
- Function scope

Clight does not have block scope

Simplifies the retrieval of a variable's address with &

How were Clight's semantics verified?

Manual review by C compiler experts

Manual review by C compiler experts

Perform tests using an interpreter *(in progress)*

Manual review by C compiler experts

Perform tests using an interpreter *(in progress)*

Define multiple semantics for Clight

and prove their equivalence *(in progress)*

We can use other verified parts of the CompCert compiler

If we verify properties about X…

…then we implicitly verify properties about Clight

Thank you for listening!