Formalizing the C99 standard

Robbert Krebbers
Joint work with Freek Wiedijk

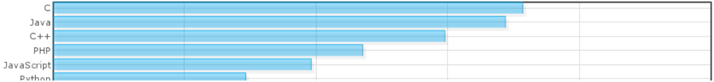
Radboud University Nijmegen

November 15, 2011 @ ICT.OPEN, Veldhoven

The C programming language

Among the two currently most used languages:

- LangPop.com - Programming Language Popularity



- TIOBE Software - Programming Community index

Position Jun 2011	Position Jun 2010	Delta in Position	Programming Language	Ratings Jun 2011	Delta Jun 2010	Status
1	2	⬆	Java	18.580%	+0.62%	A
2	1	⬇	C	16.278%	-1.91%	A
3	3	=	C++	9.830%	-0.55%	A
4	6	⬆⬆	C#	6.844%	+2.06%	A

Used for the smallest microcontroller to the largest supercomputer.

C programs can be very dangerous!

It is very easy to have programs that contain bugs

- `NULL`-pointers can be dereferenced
- arrays can be accessed outside their bounds
- memory can be used after it is freed
- ... or can be forgotten to be freed

C programs can be very dangerous!

It is very easy to have programs that contain bugs

- `NULL`-pointers can be dereferenced
- arrays can be accessed outside their bounds
- memory can be used after it is freed
- . . . or can be forgotten to be freed

A major cause of security vulnerabilities, viruses, crashes. . .

How to improve this situation? (1)

Use a more modern language, *e.g.* Haskell

Advantages:

- high level of abstraction
- strong type system
- easy to reason about such programs

Disadvantages:

- efficiency
- programs have to be rewritten
- small body of programmers

How to improve this situation? (2)

Use C together with tools, *e.g.* static analyzers or model checkers

Advantages:

- all the advantages of using C
- original programs can be used

Disadvantages:

- such tools rely on an ad-hoc C semantics
- neither sound nor complete
- behavior is unpredictable

How to improve this situation? (3)

Use C together with *formal proofs*

Advantages:

- all the advantages of using C
- original programs can be used
- highest level of confidence
- verification is fully transparent and coherent

Disadvantages:

- can be very costly
- the C standard is not suitable for a proof assistant

C together with formal proofs

The C99 standard is not in a shape that is usable in a *proof assistant*

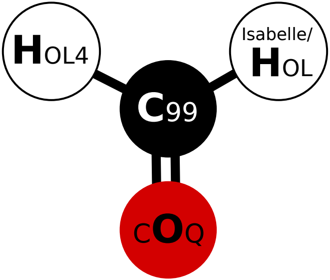
- written in English
- no mathematically precise formalism
- inherently incomplete and ambiguous

Related projects

- **Michael Norrish**
  C and C++ semantics (L4.verified)
- **Xavier Leroy et al.**
  Verified C compiler in Coq (Compcert)
- **Chucky Ellison and Grigore Rosu**
  Executable C semantics in Maude (KCC)
- **Peter Sewell et al.**
  Relaxed-Memory concurrency for C/C++

The Formalin project

- Formalize the full C99 standard in Coq, Isabelle and HOL4.
- Include features that are commonly left out:
  - aliasing rules,
  - alignment,
  - `volatile`, `const`, `restrict`,
  - non local control flow,
  - *etc*. . .

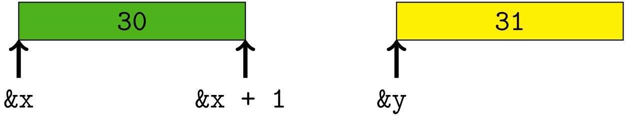


Example: continuously allocated objects

```
int x = 30, y = 31;
```

Example: continuously allocated objects

```
int x = 30, y = 31;
```



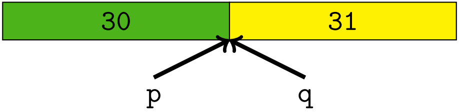
Example: continuously allocated objects

```
int x = 30, y = 31;
int *p = &x + 1, *q = &y;
```

Example: continuously allocated objects

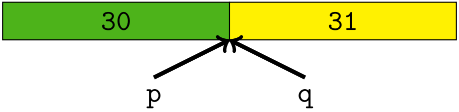
```
int x = 30, y = 31;
int *p = &x + 1, *q = &y;
if (memcmp(&p, &q, sizeof(p)) == 0) {

}
```



Example: continuously allocated objects

```
int x = 30, y = 31;
int *p = &x + 1, *q = &y;
if (memcmp(&p, &q, sizeof(p)) == 0) {
  printf("%d\n", *p);
}
```



Example: continuously allocated objects

```
int x = 30, y = 31;
int *p = &x + 1, *q = &y;
if (memcmp(&p, &q, sizeof(p)) == 0) {
  printf("%d\n", *p);
}
```

Defect report #260:

> *The implementation is permitted to use the derivation of a pointer value in determining whether or not access through that pointer is undefined behaviour, ...*

Why not just ignore defect report #260?

Defect report #260

- allows many optimizations,
- is extremely unclear,
- is not yet part of the official standard.

Why not just ignore defect report #260?

Defect report #260

- allows many optimizations,
- is extremely unclear,
- is not yet part of the official standard.

But compilers really perform optimizations based on DR #260

```
int x = 30, y = 31;
int *p = &x + 1, *q = &y;
if (memcmp(&p, &q, sizeof(p)) == 0) {
  *q = 34;
  printf("%d\n", *p);
}
```

prints 31 instead of 34 in gcc -O2

In case of doubt

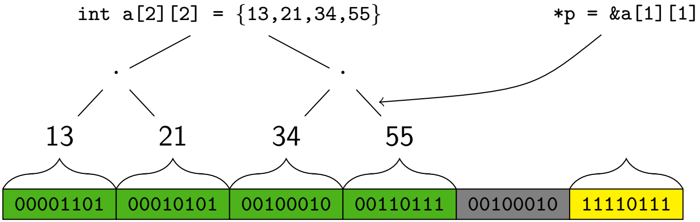
- *Soundness* is more important than *completeness*.
  - When a program that is proved correct with respect to our semantics is compiled with an optimizing compiler, it should not crash.

In case of doubt

- *Soundness* is more important than *completeness*.
  - When a program that is proved correct with respect to our semantics is compiled with an optimizing compiler, it should not crash.
- If the standard is unclear, we should make it undefined.
  - That means, our semantics does not guarantee anything about such programs.

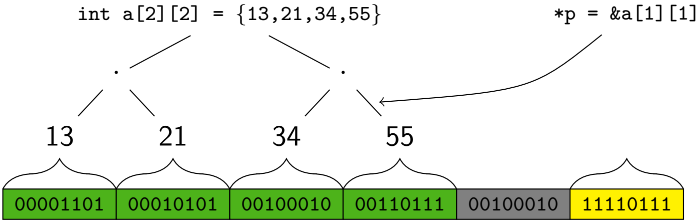
Stages of the Formalin project

1. The memory: *abstract* and *bit* level



Stages of the Formalin project

1. The memory: *abstract* and *bit* level



2. The control flow
3. The syntax and preprocessor
4. The standard library

Conclusions

- C programs are potentially dangerous
- Formal proofs can improve this situation
- Requires a mathematically precise C semantics
- The current C semantics is inconsistent
- Formalizing the standard has many uses!

Questions