A <u>Formal</u>ization of the C99 Standard <u>in</u> HOL, Isabelle and Coq

Robbert Krebbers and Freek Wiedijk

Institute for Computing and Information Sciences
Radboud University Nijmegen
Heyendaalseweg 135, 6525 AJ Nijmegen, The Netherlands

Abstract. We recently started the Formalin project to create a formal version of the C99 standard for the C programming language. We are writing three matching formalizations for the interactive theorem provers HOL4, Isabelle/HOL and Coq, that all closely follow the existing C99 standard text. The project runs from 2011 to 2015, and involves a full time PhD student, a half time researcher and several scientific advisors. The project differs from existing work in that our aim is to formalize the *full* C99 standard. This means that we treat the C preprocessor, the C standard library, floating point arithmetic, and 'dirty' C features like signal handling and volatile variables. Importantly, this means we also treat embedded C programs without explicit input/output.

The Formalin project [14], with website `http://ch2o.cs.ru.nl/`, runs from May 2011 to May 2015. The research team consists of the following people:

Robbert Krebbers	*PhD student*	
Freek Wiedijk	*project leader*	
Herman Geuvers	*promotor*	
James McKinna		
Erik Poll		
Michael Norrish	*HOL advisor*	NICTA, Australia
Andreas Lochbihler	*Isabelle advisor*	KIT, Germany
Jean-Christophe Filliâtre	*Coq advisor*	CNRS, France

The first five people are from the Radboud University in The Netherlands. The first two people are the developers, while the other six are advising.

The C programming language [8] is one of the most popular in the world. It is among the two currently most popular languages [9,13], and is a dominant language from the smallest microcontroller with only a few hundred bytes of RAM to the largest supercomputer that runs at petaflops speeds. C is especially used for embedded software, but it is also the native language of most modern operating systems due to its tight connection to Unix.

The current official description of the syntax and semantics of the C language – the C99 standard [5], issued by ANSI and ISO together – is written in English and does not use a mathematically precise formalism. This makes it inherently incomplete and ambiguous. Our project is to create a mathematically precise version of the C99 standard. We *formalize* the standard using interactive theorem provers (proof assistants) and develop the C99 formalization in three matching versions, for the interactive theorem provers HOL4 [7], Isabelle/HOL [11] and Coq [4]. These formalizations will all be derived from a common master formalization, which will either be written for one of the three systems or using a fourth system like for instance the Ott tool [12]. Ott is a tool designed for defining language definitions, which can generate formalizations for all three of our systems. The whole formalization suite will be published as open source, under a BSD-style license. For dissemination, we will also use MKM tools like the ones that currently are being developed in the MathWiki project [6].

The formalizations that we create closely follow the existing C99 standard text. Specifically we treat the C preprocessor, the C standard library, and features that in a formal treatment are often left out: unspecified and undefined behavior due to unknown evaluation order, casts between pointers and integers, alignment requirements, floating point arithmetic, non-local control flow (`goto` statements, `setjmp`/`longjmp` and signal handling) and volatile variables. Most importantly, to make our work relevant for verification of embedded software, we include the part of the standard referring to C programs that run in a 'freestanding environment' (Section 5.1.2.1 of [5]).

A formal version of the full C standard is an important artifact. When establishing a property of a C program, it is very attractive to be able to claim that it has been proved with respect to the *full* official standard. This kind of 'knowledge' about the C semantics in the current state of the art is mostly implicit in various tools.

A formalization of the C99 standard has three main applications:

– The C99 formalization makes the C99 standard utterly precise. This is useful for compiler writers, who will get the means to establish how the standard needs to be understood without having to deal with the ambiguities of the English language. Programmers writing C programs get the same benefit.
– There already are various projects to prove C compilers correct, like the Compcert project of Xavier Leroy [2]. These projects need a semantics of a version of C. These currently are subsets of full C, with names like Clight or C0. With a formal version of the C99 semantics, the correctness of the compiler becomes provable with respect to the full *official* standard.
– Currently people proving C programs correct with proof assistants use tools like VCC [3] and Frama-C [10] which generate *verification conditions* from C source annotated in the style of Hoare logic. These tools implicitly 'know' about the semantics of C, but this knowledge is not explicit. A more thorough approach is to have such a tool not just generate the verification conditions, but to also have it synthesize formal *proofs* about the properties of the program.

Our three formalizations each consist of two parts. The first part defines a space of all possible C semantics as a type `C_semantics`. ('Semanticses' is not correct English, but we mean the plural of semantics here.) Points in this space correspond to various variants of C like the C99 standard, the upcoming C1X standard, and the behavior of specific C compilers on specific machines. The definition of this space will be as short as we can make it, to have it as clear as possible what our formalized C99 standard amounts to. This space also addresses the observation from page 70 of [1] that 'the C language does not exist'. The second part is a small step structured operational semantics of C. It corresponds to a *point* in the space of C semantics, which means that the second and main part of our formalizations will be a formal definition of an element

```
C99 : C_semantics
```

References

1. Al Bessey et al. A few billion lines of code later: using static analysis to find bugs in the real world. *Communications of the ACM*, 53(2):66–75, 2010.
2. Sandrine Blazy and Xavier Leroy. Mechanized semantics for the Clight subset of the C language. *Journal of Automated Reasoning*, 43(3):263–288, 2009.
3. Ernie Cohen, Markus Dahlweid, Mark A. Hillebrand, Dirk Leinenbach, Michal Moskal, Thomas Santen, Wolfram Schulte, and Stephan Tobies. VCC: A Practical System for Verifying Concurrent C. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *TPHOLs*, volume 5674 of *LNCS*, pages 23–42, 2009.
4. Coq Development Team. *The Coq Proof Assistant Reference Manual*, 2010.
5. International Organization for Standardization. *ISO/IEC 9899:1999: Programming languages – C*. ISO Working Group 14, 1999. Draft standard WG14/N1256, the combined C99 + TC1 + TC2 + TC3, dated September 7, 2007.
6. Foundations Group of the ICIS. Research/MathWiki. `http://www.fnds.cs.ru.nl/fndswiki/Research/MathWiki`.
7. Mike Gordon and Tom Melham, editors. *Introduction to HOL*. Cambridge University Press, 1993.
8. Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, 2nd edition, 1988.
9. LangPop.com. Programming Language Popularity. `http://langpop.com/`.
10. Yannick Moy and Claude Marché. *Jessie Plugin Tutorial,* Beryllium *version.* INRIA, 2009.
11. Tobias Nipkow, Larry Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
12. Peter Sewell et al. Ott: Effective tool support for the working semanticist. *Journal of Functional Programming*, 20(1):70–122, 2010.
13. TIOBE Software. TIOBE Programming Community index. `http://www.tiobe.com/content/paperinfo/tpci/`.
14. Freek Wiedijk. Formalizing the C99 standard in HOL, Isabelle and Coq. `http://www.cs.ru.nl/~freek/notes/ch2o.pdf`, 2010.