NICTA

COMP 4161

NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein, June Andronick, Toby Murray, Rafal Kolanski,
+ Thomas Sewell

$$\frac{\{P'\} \ \ldots \ \{Q'\}}{\{P\} \ \ldots \ \{Q\}}$$

Before the Break

- Program verification, Hoare logic and invariants.
- Real C programs
  - Side effects.
  - Types (fixed-width words, arrays, structs)
  - C Memory (pointers, heap representation)
  - Control flow (for, break, continue, return, etc)
  - Undefined execution (null pointers etc, Simpl Guard)
  - VCG

This time

Short summary of verification on C code: **it gets ugly.**

This week we consider alternatives:

- Why was C verification difficult?
- Kinds of alternatives
- Monads
- “AutoCorres”

Recap

Recap of C Verification

DEMO

(in which we aren't going to get anywhere)

The Challenge

Are we doing it right?

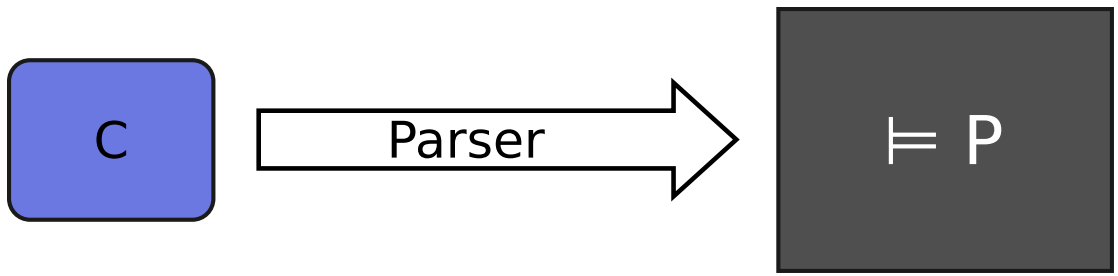
The Challenge

Are we doing it right?

This is a NICTA question especially, since all the C-related features in Isabelle were developed for a NICTA project.

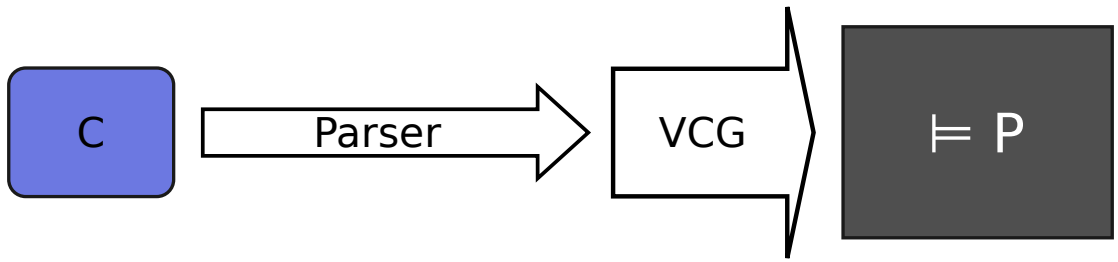
Approach

NICTA

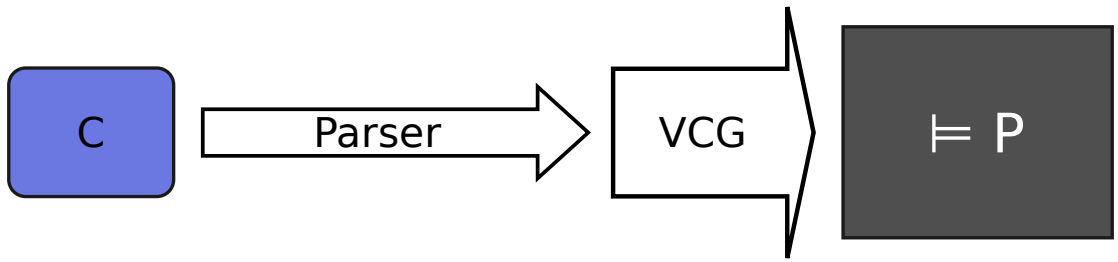


Approach

NICTA



Approach



We could try to change this diagram. There isn't necessarily a single

good way to approach this problem. This differs to the pre/post condition logic we've seen before.

The Alternatives

What else could we do? What are the alternatives?

The Alternatives

What else could we do? What are the alternatives?

- Assume a simpler dialect of C
- Use a higher level language, like Haskell, Java, ML or C#
- Cheat, focus on a simpler representation
- Do all the proof closer to the C program
- Generate the code from a simpler representation
- Simplify the program

The Alternatives

What else could we do? What are the alternatives?

- Assume a simpler dialect of C (Spark ADA?)
- Use a higher level language, like Haskell, Java, ML or C# (Haskell House kernel)
- Cheat, focus on a simpler representation (everyone)
- Do all the proof closer to the C program (Verve, Verisoft XT)
- Generate the code from a simpler representation (4 colour theorem)
- Simplify the program (AutoCorres)

Functional Programming Plus

We can try implementing `union` and `find` directly in Isabelle's logic language.

DEMO

Functional Programming Plus

We can try implementing `union` and `find` directly in Isabelle's logic language.

DEMO

Passing globals (such as the array) around makes sense for now, but not as our program grows. We can use the state monad to make this implicit.

DEMO

State Monads

All monads come with a `return` and `>>=` function, and the state monad also has a `get` and `set`.

Monads have a handy `do` notation. We'll talk more about monads later this week.

The state monad comes with some useful rewrite rules, for instance return_bind:

$$\text{do } x \leftarrow \text{return } y; \text{ f } x \text{ od} = \text{f } y$$

VCG on Monads

The state monad package also comes with a VCG equivalent called WP (for Weakest Precondition).

The WP tool works like the wp calculation on imperative programs we have seen, and the SIMPL VCG.

We'll see an example in a moment.

AutoCorres

One way to relate C/SIMPL programs to monadic programs is AutoCorres.

AutoCorres is an experimental tool developed at NICTA by David Greenaway. AutoCorres simplifies C/SIMPL programs into equivalent monadic programs. The monadic programs are sometimes much simpler.

DEMO
(from AutoCorres tests/examples/simple.c)

AutCorres plus

We can put everything together and try to prove that a C/SIMPL program is equivalent to a hand-written monadic program.

DEMO (in which we aren't going to get very far)

Next time . . . we'll talk about the theory behind these tools.