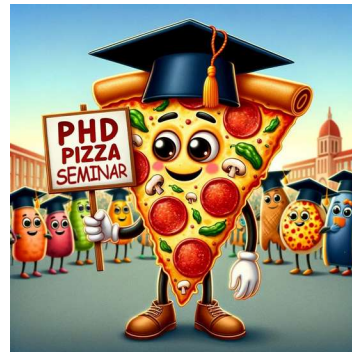


# Will my Table be Large Enough for so Many Ingredients?

Or: Formal Verification of Heap Space Bounds

Alexandre Moine (Cambium)  
PhD Pizza Seminar  
14/05/24



# If We Haven't Met

I'm a 3rd year PhD student working with:

- François Pottier
  - Arthur Charguéraud (remotely)
- 
- I work at office C332 (third floor, building C)
  - In front of the coffee machine
- 
- I co-organize the seminar with Jakob.
  - I will defend in September,
  - we need someone to take over: talk to me if you are interested!



# Computer Science

cs.AI (Artificial Intelligence)  
cs.AR (Hardware Architecture)  
cs.CC (Computational Complexity)  
cs.CE (Computational Engineering, Finance, and Science)  
cs.CG (Computational Geometry)  
cs.CL (Computation and Language)  
cs.CR (Cryptography and Security)  
cs.CV (Computer Vision and Pattern Recognition)  
cs.CY (Computers and Society)  
cs.DB (Databases)  
cs.DC (Distributed, Parallel, and Cluster Computing)  
cs.DL (Digital Libraries)  
cs.DM (Discrete Mathematics)  
cs.DS (Data Structures and Algorithms)  
cs.ET (Emerging Technologies)  
cs.FL (Formal Languages and Automata Theory)  
cs.GL (General Literature)

cs.GT (Computer Science and Game Theory)  
cs.HC (Human-Computer Interaction)  
cs.IR (Information Retrieval)  
cs.IT (Information Theory)  
cs.LG (Machine Learning)  
cs.LO (Logic in Computer Science)  
cs.MA (Multiagent Systems)  
cs.MM (Multimedia)  
cs.MS (Mathematical Software)  
cs.NA (Numerical Analysis)  
cs.NE (Neural and Evolutionary Computing)  
cs.NI (Networking and Internet Architecture)  
cs.OH (Other Computer Science)  
cs.OS (Operating Systems)  
cs.PF (Performance)  
cs.PL (Programming Languages)  
cs.RO (Robotics)

cs.SC (Symbolic Computation)  
cs.SD (Sound)  
cs.SE (Software Engineering)  
cs.SI (Social and Information Networks)  
cs.SY (Systems and Control)

**arXiv**  
**Category Taxonomy**

# Programs are Pizza Recipes



We study the art of writing pizza recipes!



1. In which language do we even write our recipes?  
Can we imagine a language that does not have the word “pineapple”?
2. How to guess if this recipe will make a good pizza?  
Are we sure that the absence of the word “pineapple” is enough?
3. Can we prove that our guess is correct?  
Are we sure that we did not miss a “pineapple” in this 27M LOC recipe (Linux)?

Cambium study these questions!

# Cambium 1: Designing and Implementing PL



- A functional programming language,
- with concurrency,
- strong types,
- a cool module system,
- and a garbage collector (more later).

Adopted by the industry!

facebook

 Microsoft

 docker

 Jane  
Street

Bloomberg



ahrefs

## Cambium 2: Reasoning about Programs

- Therac 25 (1981)  
A bug in radiation therapy machine gave radiation overdose
- Ariane 5 (1996)  
A bug in the autopilot of the rocket led to its destruction. Loss: 300M\$
- “The DAO” (2016)  
A bug in a smart contract on the Ethereum blockchain led to a 50M\$ robbery

How to ensure that a program has no bug?

- Testing gives confidence, **but it is not enough.**
- We want to **prove** that programs have no bugs!

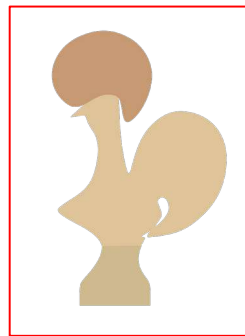
# Cambium 3: Reasoning About Proofs

How to ensure that a proof has no bug?

How to prove that a proof is correct?

- Pen-and-paper proofs are not satisfactory.
- There exists **proofs assistants**.

We then have to trust the proof assistant...



Agda

LEMN  
THEOREM PROVER

# Focus on: Reasoning about Programs

How to **prove** that a program has no bug?

How to **prove** that a program satisfies its **specification**?

Mostly automatic approach

- Type systems
- Abstract Interpretation
- Model Checking

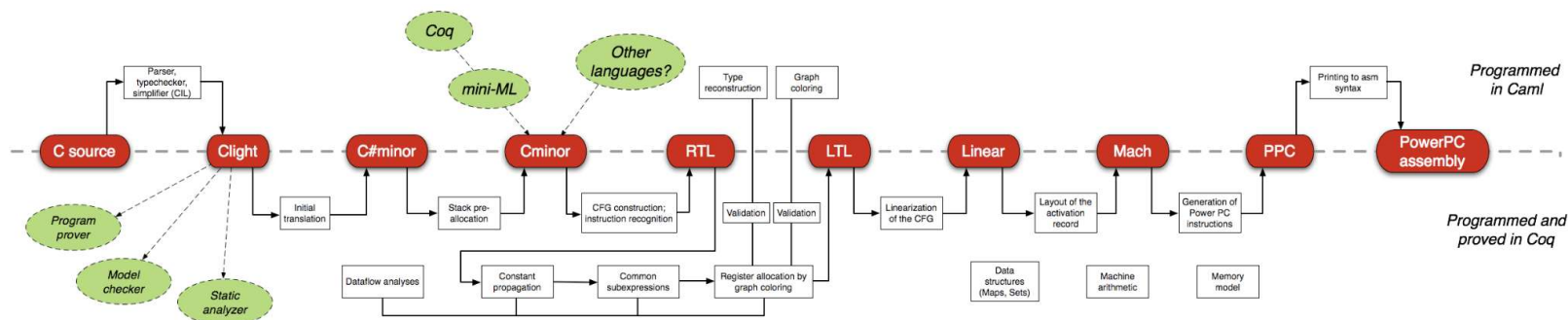
Mostly manual approach

- Proof assistant as PL
- Program logics!



# The CompCert Compiler

- A compiler for (a large subset) of C, written inside Coq
- and entirely verified!



## Main Theorem:

The compiler preserves the semantics of the program.

$$\{P\} e \{ \lambda v. Q v \}$$

- $e$  is the verified program
- $P$  describes the state before executing  $e$  (its precondition)
- $v$  is a name for returned value
- $Q v$  describes the state after executing  $e$  (its postcondition)

Soundness Theorem:

If  $\{P\} e \{ \lambda v. Q v \}$  holds, then if the initial state satisfies  $P$ , the execution of  $e$  is safe and, if it terminates, the end state satisfies  $Q v$ .

# Separation Logic: Basics

- Ground assertion: the points-to

$$\ell \mapsto v$$

- New connector: the separating conjunction

$$P_1 * P_2$$

$$\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2$$

The two locations must be different!

# Separation Logic: Reasoning Rules

ALLOC

$$\{ \text{True} \} \text{ref } v \{ \lambda \ell. \ell \mapsto v \}$$

LOAD

$$\{ \ell \mapsto v \} !\ell \{ \lambda v'. \lceil v' = v \rceil * \ell \mapsto v \}$$

STORE

$$\{ \ell \mapsto v \} \ell := v' \{ \lambda \_. \ell \mapsto v' \}$$

FREE

$$\{ \ell \mapsto v \} \text{free } \ell \{ \lambda \_. \text{True} \}$$


FRAME

$$\frac{\{ P \} e \{ \lambda v. Q v \}}{\{ P * R \} e \{ \lambda v. Q v * R \}}$$

# Beyond Functional Correctness

- Base Separation Logic guarantees *safety*:  
a verified program will not crash due to (correctness) bug
- We can extend Separation Logic to guarantee other properties!

We want to prove that a program does not use too much resources.



Time  
(2011)

Space  
(2022)

Entropy  
(2024)

Energy  
(????)

# Space Credits for Heap Space

For heap space we use **space credits**.

- Let  $\diamond 1$  be one space credit.
- A space credit represents the right to allocate one memory word.
- Credits are splittable and joinable:

$$\diamond(n + m) \equiv \diamond n * \diamond m$$

# Heap Space Bounds, with Manual Memory Management

ALLOC

$$\{ \Diamond 1 \} \text{ref } v \{ \lambda \ell. \ell \mapsto v \}$$

LOAD

$$\{ \ell \mapsto v \} !\ell \{ \lambda v'. \ulcorner v' = v \urcorner * \ell \mapsto v \}$$

STORE

$$\{ \ell \mapsto v \} \ell := v' \{ \lambda \_ . \ell \mapsto v' \}$$

FREE

$$\{ \ell \mapsto v \} \text{free } \ell \{ \lambda \_ . \Diamond 1 \}$$

FRAME

$$\frac{\{ P \} e \{ \lambda v. Q v \}}{\{ P * R \} e \{ \lambda v. Q v * R \}}$$

What do we Prove in the End?

Soundness Theorem:

If  $\{\diamond S\} e \{\lambda\_. True\}$  holds, then running the program  $e$  is safe and cannot generate an “Out of Memory” error with a heap of size  $\geq S$ .



# The Garbage Collector Enters the Scene

- OCaml (and many other languages) comes with a Garbage Collector (GC).
- There is **no** free operation.
- Instead, if space is needed, the GC will free **unreachable locations**.
  
- The GC simplifies the life of the programmer:
  - No more “use after free” bug
  - No more “double free” bug
- But reasoning about space becomes difficult: **where space is recovered?**

# Going Back to Pizzas

- In standard recipes, there is no “put this ingredient away” instruction.
- Instead, if we ever lack space on our table,
- we have to lookup if an ingredient will be used again or not.

Formal Verification of Heap Space Bounds with GC

≈

Will my Table be Large Enough for so Many Ingredients?

# What is the Minimal Heap Space Usage of this Program?

```
let example () =  
  let a = Array.make 100 true in  
  let b = Array.copy a in  
  a.(42) <- false;  
  let c = Array.copy b in  
  b.(42) && c.(24)
```

- The allocation of a consumes 101 free memory words.
- The allocation of b consumes 101 free memory words.
- The allocation of c *does not need new space!*

The GC can deallocate a, and the allocator can reuse its space.

# Reachability and Unreachability

How does the GC compute the set of **reachable** locations?

- First it determines the set of **roots** ( $\approx$  local variables)
- Then the GC “**walks the heap**”
- Parts of the heap that cannot be reached by this process can be freed.

Hence, a location is **unreachable** if:

- It is not a root.
- It is not pointed-by any reachable heap block.

# Contribution: A Separation Logic for Heap Space with GC

Key idea:  $\diamond 1$  is one free *or freeable* memory word

FREE

$$\{ P * \diamond 1 \} e \{ \lambda v. Q v \}$$

---

$$\{ P * \ell \mapsto v * \text{"}\ell \text{ is not a root"} * \text{"}\ell \text{ is not pointed-by any block"} \} e \{ \lambda v. Q v \}$$

Soundness Theorem:

If  $\{ \diamond S \} e \{ \lambda \_. True \}$  holds, then running the program  $e$  with a GC is safe and cannot generate an “Out of Memory” error with a heap of size  $\geq S$ .

# Conclusion

Takeaway: we can reason about heap space in the presence of a GC.

Main Takeaway: we can reason about unreachability.

What I did not show (ask me details during the pizza break (at your own risk)):

- The various assertions of the logic
- The details on *concurrency*
- The reasoning on first-class functions (a.k.a *closures*)
- Case studies: the logic can be used on small programs
- Everything is mechanized in Coq

# Thank you for your attention!

Do not ask an AI for “a smiling pizza”:

