Group discussion agenda

- our UNSOUND future
  - would someone in the audience want to be the organizer next year?
  - every year? every 2 years?

Group discussion agenda

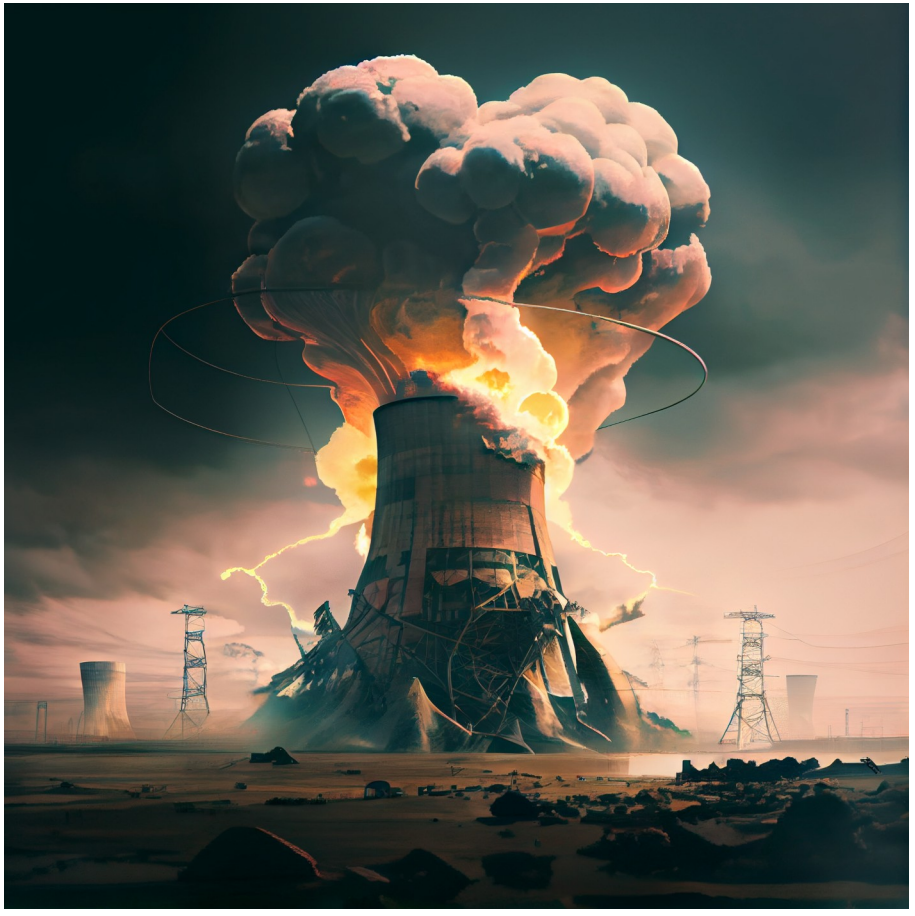
- What is soundest UNSOUND venue?

- Reach:
  - how should we reach out to the OO verification community
  - how should we reach out to the
    - verified compilers community?
    - verified tool chains community?
    - deep verification community?
    - ...community?

General implications for safety in computing

- How do we convince anyone here in person to board their airplane home?

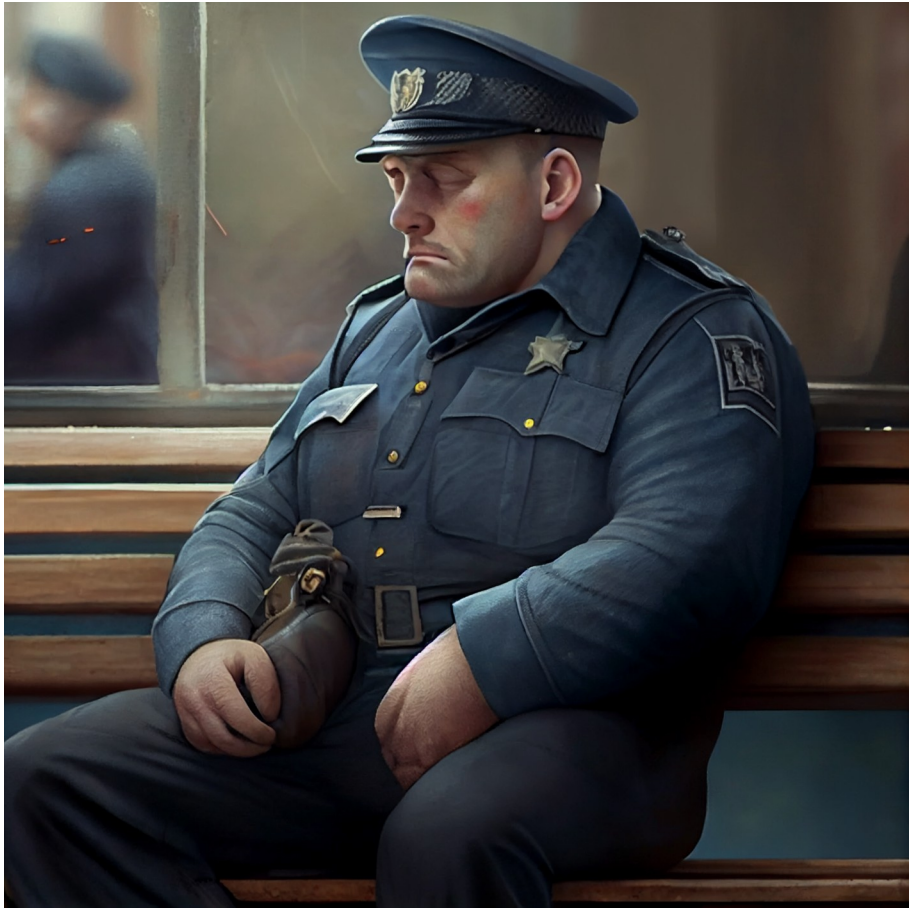
- No problem: even safety of nuclear power plants is modeled in eclipse-generated C++ monte-carlo simulations running without any formal proof, so you unsoundness might be your least concern when no-one is even trying to be formal in practice.



Perverse incentives in a world with unsound software verification

- Perverse incentives in a world with unsound software verification:
  - what if having “verified” code makes us relax the “testing/auditing” process of coding?

  - Implications for hackers submitting “verified” code.
  - Comparison with model checking



Consider the following:
@Post \result.size()>this.size()

Most sound traditional OO verification language would require the method size() to be total and pure: that is, it must always terminate, be deterministic and not mutate the state or do I/O.

Is totality overly restrictive?

We could, for example, conservatively say that such a postcondition is false if the call \result.size() would not terminate.

We experimented with this idea:
an OO setting where non termination means false.

We discovered that all of the sources of falseness we encountered in our experiments could be encoded by non termination:
we could simply make a function that loops if the parameter is not of the expected value.

This pushed us to attempt a verification system based on proving
termination alone,
and encode all of the other properties on top.

That is, we propose a specification paradigm that unifies the underlying object oriented language with the specification language, where termination is centred as the source of truth in the language.

This mindset is nicely aligned with the semantics of pure OO languages:

In a pure OO setting, everything should be an object, and the only meaningful operation should be the single dispatch method call.

That is, objects are black boxes and they can communicate with each other by message passing; where other black box objects will be parameters and results.
In this context, where there is no such a thing as a primitive boolean, numbers, reference equality, what does it mean to ‘specify a property’?

Objects are black boxes, and we can only observe them by calling methods, but the result of such methods are other black boxes.
The only observable event seems to be ‘termination’
(non termination, on the other hand, can not be observed).

The two main observations

(1)-We can use termination to prove arbitrary decidable properties.

(2)- We argue that the only thing we should be allowed to observe in a pure OO language is termination; that is: observing more breaks the OO abstraction/model.

The two main observations

(1)-We can use termination to prove arbitrary decidable properties.

In context Gamma = { a:Num,b:Num,c:Num } the term
if (((a+b)+c)==(a+(b+c)))
        then true
        else loop

In OO: just add to booleans a ‘checkTrue’ method that loops on false

Forall a,b,c:Num
    a.plus(b).plus(c).equals(a.plus(b.plus(c))).checkTrue()

The two main observations

(2)- We argue that the only thing we should be allowed to observe in a pure OO language is termination; that is: observing more breaks the OO abstraction/model.

The core of the OO abstraction is the method call (message passing).
An object is not a record of fields but a black box that can answer messages
In a pure OO setting, there are no primitive types, so every method just returns another black box object, that in turn can only be observed by calling methods on it.

Attempting to verify pure OO programs using traditional verification techniques violates the OO paradigm; the black boxes are open and interpreted as a composition of primitive data types. Instead, by simply observing termination, we can do proofs while preserving the pure OO abstraction: not only we can do the reasoning in the OO paradigm, but the property itself is specified as an OO expression.