Tutorial

# Basics

* Open “tutorial\basics.c”.
* Explain code.
  + //@ are specifications (ignored by C compiler).
* Verify the Program.
* Make an error (replace 1 by 2).
  + Explain parts of the symbolic state.
  + Explain the symbolic debugger.
  + Point out that Z3 is used to check
* Add a function “inc”.
  + Symbolically execute inc.
  + Point out that the value for the parameter “x” is a fresh variable.
* Use “inc” in “main”.
  + Symbolic debugger to explain how calls are handled.
* Add a function “abs”.
  + Make an error in its specification.
  + Explain how if-statements are handled.
  + Show use of “abs” in main.

# Heap

* Open “tutorial\counter.c”.
* Verify the program.
  + Ask audience why verification fails.
  + Abort if c is 0.
  + Ask audience again why verification fails (leak).
  + Free c at the end.
  + Walk the audience through the symbolic execution.
    - Explain what malloc\_block is.
  + Explain how verifier detected the leak and the faulty memory access.
  + Show double free and ask audience why it does not go through.
* Add a method “inc”.
  + Use empty contract and explain why it fails.
  + Add precondition.
  + Add postcondition.
  + Walk the user through the symbolic execution of “inc”.
  + Call “inc” in main and explain permission transfer.
* Add a method “swap”.
  + Explain separating conjunction.
  + Use swap in main.
* Add a methods “create\_counter”, “counter\_get” and “dispose”.
  + Forget to initialize c->x in create\_counter and ask audience why it fails.
  + Use these methods in main.
* Add a predicate named “counter”.
  + Refactor the specifications of “create\_counter”, “inc”, “swap” and “dispose”.
  + Explain the use of open and close.
  + Forget close at some point and ask why it fails.
* Show that you can put the specification in a .h file and verify main against counter.h.
  + Explain that the verifier complain if you implement the .h file wrong.

# List (recursive predicates, inductive data types, fixpoints and lemmas)

* Open “tutorial\list.c”.
  + Memory safety
    - Show loop invariant in dispose.
  + Length
  + Inductive + fixpoint
  + Lemma to prove that 0 <= list\_length(l);

# Concurrency

* Idea 1: Take a non-concurrent Program and show how to make it concurrent?
  + E.g. http server?
  + Version 1: only 1 connection at a time.
  + Version 2: handle connection in new thread.
    - Explain predicate families indexed by function pointers.
      * thread\_run\_data
    - Explain predicate constructors.
  + Version 3: have some shared data structure that needs to be locked?
    - E.g. add number of times webpage has been accessed?
    - Can we store number of times inside a global field and protect it with a lock?
    - Add fields for shared object + lock to request?
* Idea 2: Show a program that performs two big operations sequentially.
  + Fork a new thread to perform one of the operations for faster performance.
* Idea 3: Just do the chat server. (but complex!)
* Idea 4: Owicki-Gries.
* Idea 5: Producer-Consumer
* Open “tutorial\fork-join.c”
  + Explain fork/join.
* Explain fractions as read-only permissions.
* Explain locks.
* (Explain owicki-gries? Complicated though.)
* (Explain producer-consumer?)

# Fine-grained concurrency