Rethinking Automated Theorem Provers?

David J. Pearce

School of Engineering and Computer Science
Victoria University of Wellington

@WhileyDave

http://whiley.org

http://github.com/Whiley

Background

Verification: A Challenge for Computer Science

> *"A* **verifying compiler** *uses automated mathematical and logical reasoning methods to check the correctness of the programs that it compiles"*
>
> *–Hoare'03*

Verification: A Long History

"In order that the man who checks may not have too difficult a task the programmer should make a number of definite **assertions** *which can be checked individually, and from which the correctness of the whole program easily follows."*

–Turing'49

Whiley

Overview: What is Whiley?

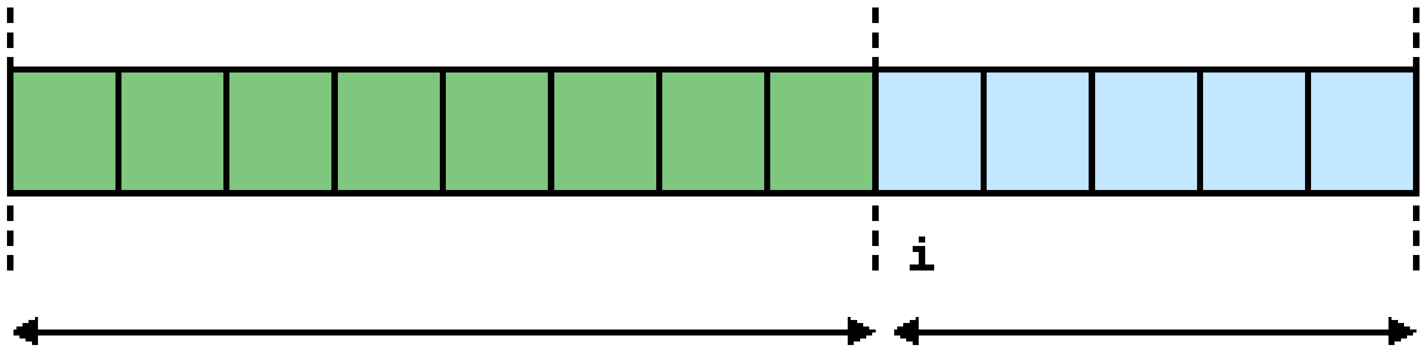
```
function max(int x, int y) -> (int z)
// result must be one of the arguments
ensures x == z || y == z
// result must be greater-or-equal than arguments
ensures x <= z && y <= z:
    ...
```

- A language designed specifically to simplify **verifying software**
- Several trade offs e.g. **performance for verifiability**
  - *Unbounded Arithmetic, value semantics, etc*
- **Goal**: to statically verify functions meet their specifications

Example: `max(int[])`

```
// Returns index of largest item in array
function max(int[] items) -> (int r)
```

Diagram!



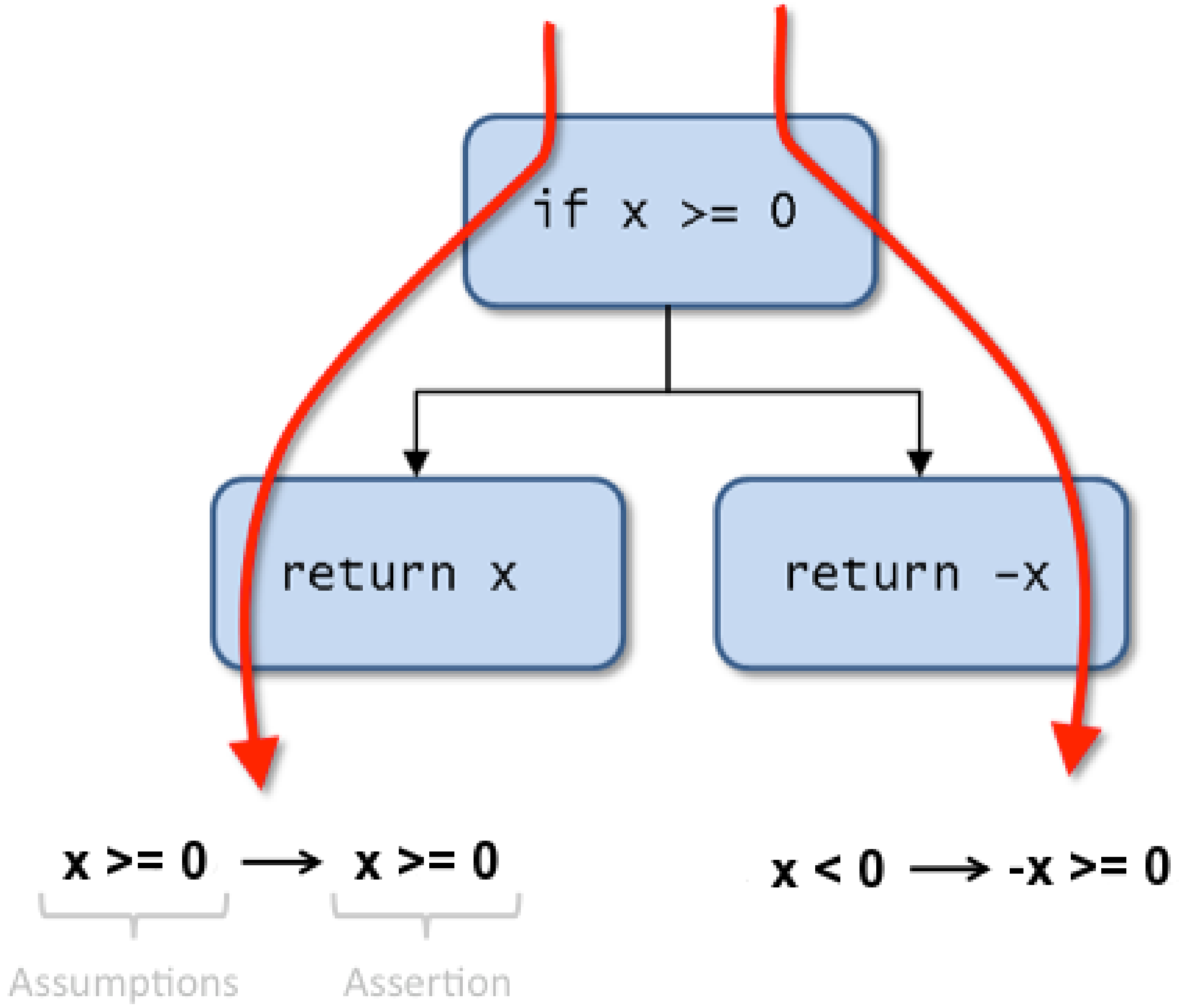
Verification Condition Generation

Verification Condition Generation

```
function abs(int x) -> (int r)
// Either x or its negation returned
ensures (r == x) || (r == -x)
// return value cannot be negative
ensures r >= 0:
     //
     if x >= 0:
          return x
     else:
          return -x
```

- To verify above function, compiler generates **verification conditions** (roughly, **first-order logic formulas**)

Verification Condition Generation



Automated Theorem Proving

Automated Theorem Proving

"These [decision] procedures have to be **highly efficient***, since the problems they solve are* **inherently hard***."*

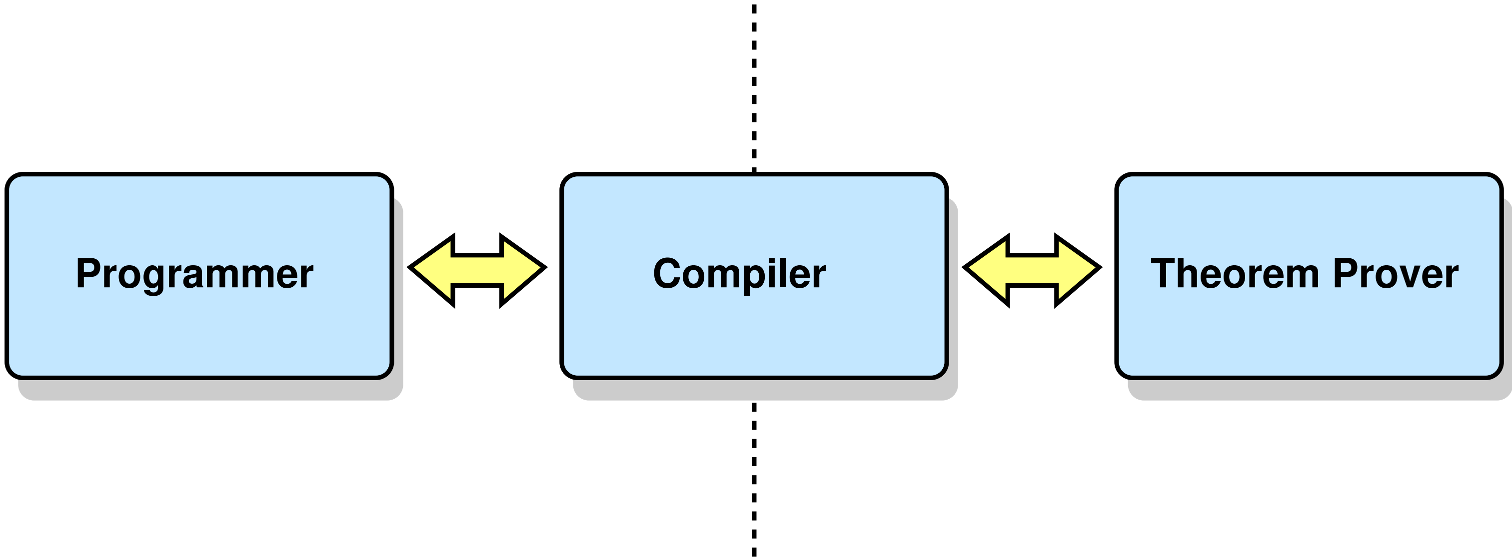
– Kroenig and Strichman

"Automatic theorem provers (ATPs) based on the resolution principle ... have reached a **high degree of sophistication***. They can often find long proofs even for problems having* **thousands of axioms***"*

–Benzmuller et al.

- "Automated Theorem Provers are a **dark art** — just use Z3!"

Theorem Prover Design Decisions

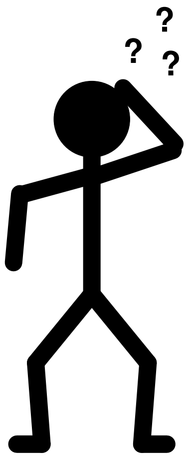


- Theorem prover typically viewed as **hidden** “out the back”

- But, theorem prover is **inevitably** part of user interface ...

  ... and should be given **first-class** status

Theorem Prover as a User Interface?



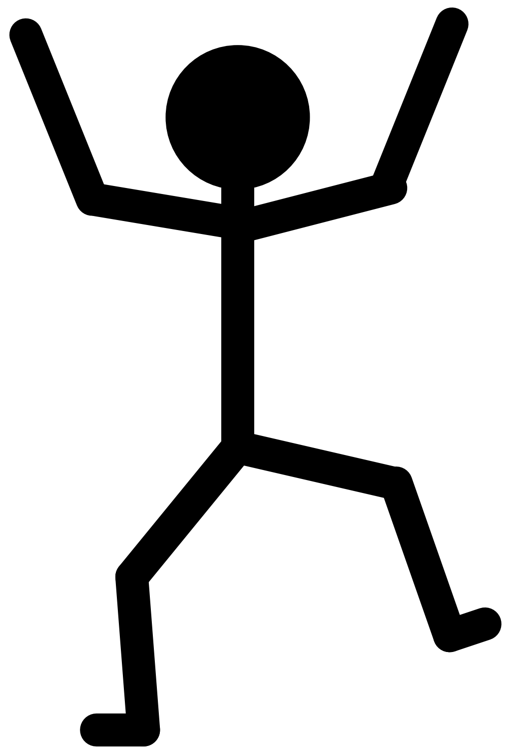
- When **verification succeeds**, all is well!
- But, when verification **inevitably fails** ...

  ... programmer must figure out **what happened**

Implications

What are the implications of this?

- *Represent formulae in CNF?* **NO!**
- *Use DPLL as main loop?* **NO!**
- *Use SMT-LIB as input language?* **NO!**



... *Am I Mad?*

Whiley Automated Theorem Prover (WyTP)

Theorem Proving: Assertion Language

- Whiley compiler emits verification conditions in **assertion language**

```
define abs_ensures_0(int x, int r) is:
    (r == x) || (r == -x)

assert "postcondition␣not␣satisfied":
    forall(int x):
        if:
            x >= 0
        then:
            abs_ensures_0(x, x)
```

- Verification conditions from `abs()` example shown above

- In principle, can hook up different **automatic theorem provers**

Theorem Proving: More Assertion Language

- Assertions are **paths** through function ... and should reflect that

```
...
int i = 0
while i < |xs|
   where all { k in 0 .. i | xs[k] != x }:
   ...
```

- **Example** assertion generated from above:

```
define indexOf_loopinvariant_50(int[] xs, int x, int i) is:
    forall(int k).(((0 <= k) && (k < i)) ==> (xs[k] != x))

assert "loop␣invariant␣does␣not␣hold␣on␣entry":
    forall(int x, int i, int[] xs):
        if:
            i == 0
        then:
            indexOf_loopinvariant_50(xs, x, i)
```

Theorem Proving: Proofs

(1)	$\exists(\texttt{int x}).(\texttt{x} \geq 0 \wedge \texttt{x} < 0)$	
(2)	$x_1 < 0 \wedge x_1 \geq 0$	($\exists$-*elimination*, 1)
(3)	$x_1 \geq 0$	($\wedge$-*elimination*, 2)
(4)	$x_1 < 0$	($\wedge$-*elimination*, 2)
(5)	$0 < 0$	($\leq$-*closure*, 3 + 4)
(6)	$\perp$	(*simplification*, 5)

- Purpose-built **Automated Theorem Prover** developed
- Focus on **simplicity** rather than **scale**
- For example, not based on DPLL

Theorem Proving: Simplification

Largest group of proof rules are for *simplification*:

- **Logical**. For example, `f ∨ f ⟹ f`, `true ∧ f ⟹ f`, etc.
- **Arithmetic**. For example, `1+1 ⟹ 2`, `(2*x)-x ⟹ x`, etc.

  Also, `x < x ⟹ 0 < 0`, etc.
- **Arrays**. For example, `[x,y,z][0] ⟹ x`,

  `|[x,y,z]| ⟹ 3`, and `|[e;n]| ⟹ n`, etc.
- **Records**. For example, `{x:1,y:2}.y ⟹ 2`,

  `{x:1,y:2}[x:=3] ⟹{x:3,y:2}`, etc.

Theorem Proving: $\vee$-Elimination

(1) $\exists(\texttt{int x}).((\texttt{x} = 0 \vee \texttt{x} > 0) \wedge \texttt{x} < 0)$

---

(2)	$(\texttt{x}_1 = 0 \vee \texttt{x}_1 > 0) \wedge \texttt{x}_1 < 0$	($\exists$-*elimination*, 1)
(3)	$(\texttt{x}_1 = 0 \vee \texttt{x}_1 > 0)$	($\wedge$-*elimination*, 2)
(4)	$\texttt{x}_1 < 0$	($\wedge$-*elimination*, 2)

(5)	$\texttt{x}_1 = 0$	($\vee$-*elimination*, 2)
(6)	$\texttt{x}_1 < \texttt{x}_1$	(*congruence*, 4 + 5)
(7)	$\bot$	(*simplification*, 6)

(8)	$\texttt{x}_1 > 0$	($\vee$-*elimination*, 2)
(9)	$0 < 0$	($\leq$-*closure*, 4 + 8)
(10)	$\bot$	(*simplification*, 5)

Theorem Proving: Axioms

- Should the following **hold** or not?

```
assert:
  forall(int i, int[] xs):
    if:
      xs[i] > 0
    then:
      (i >= 0)
```

- **Arithmetic**. If x/y is defined, then $y \neq 0$.
- **Arrays (a)**. If $xs[i]$ is defined, then $0 \leq i < |xs|$
- **Arrays (b)**. If $xs = [e; n]$ is defined, then $0 \leq n$ and $|xs| = n$.
- **Arrays (b)**. If xs has array type, then $|xs| > 0$.
- **Functions**. If $f(x) \neq f(y)$ then $x \neq y$.

Theorem Proving: Case Analysis

- How to **prove** the following?

```
assert:
    forall(int i, int[] xs):
        if:
            xs == [1,2,3]
        then:
            xs[i] >= 0
```

- Above reduces to showing `[1,2,3][i] < 0` is **contradiction**
- Apply **case analysis** with $(i{=}{=}0) \vee (i{=}{=}1) \vee (i{=}{=}2)$

Theorem Proving: Quantifier Instantiation

- How to **prove** the following?

```
assert:
    forall(int[] xs):
        if:
            forall(int i).(xs[i] > 0)
        then:
            forall(int j).(xs[j] >= 0)
```

- What is **meaning** of `∀(int i).(xs[i] > 0)`?

- Equivalent to **infinite conjunction**!

- We just need to pick the **right conjunct** ...

Theorem Proving: Proof Optimisation

(1) $\exists(\texttt{int i}).((\texttt{i} \leq 0) \wedge (\texttt{i} == 0) \wedge (\texttt{i} \geq 0))$

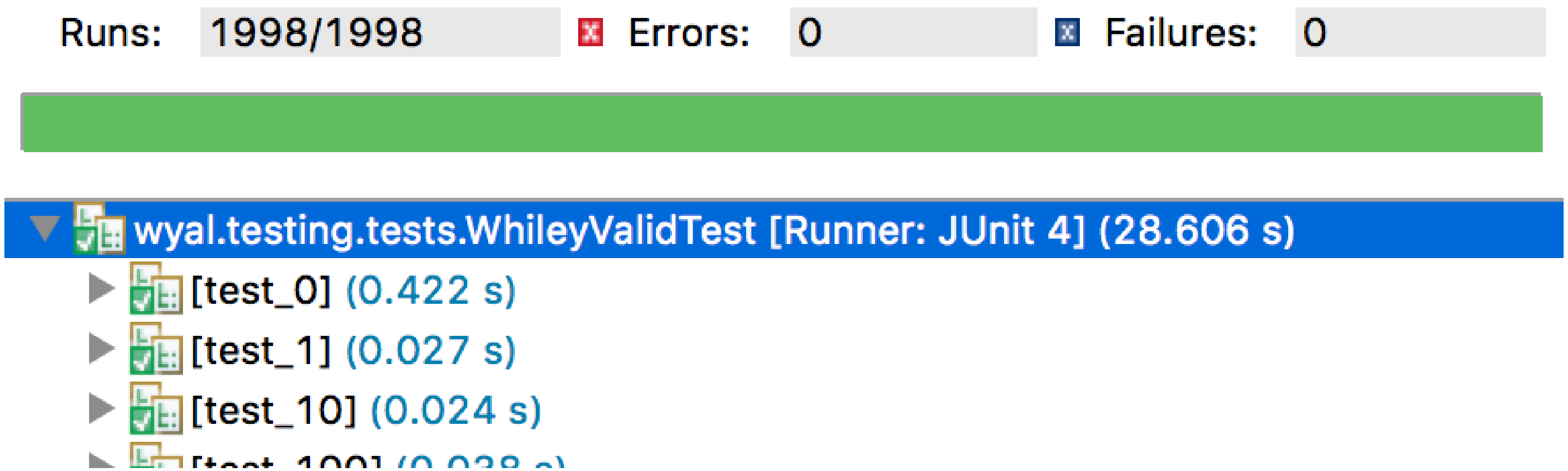
---

(2)	$(\texttt{i}_1 < 0) \wedge (\texttt{i}_1 == 0) \wedge (\texttt{i}_1 > 0)$	($\exists$-*elimination*, 1)
(3)	$\texttt{i}_1 < 0$	($\wedge$-*elimination*, 2)
(4)	$\texttt{i}_1 == 0$	($\wedge$-*elimination*, 2)
(5)	$\texttt{i}_1 > 0$	($\wedge$-*elimination*, 2)
(6)	$0 < 0$	(*congruence*, 3+4)
(7)	$\perp$	(*simplification*, 6)

- **Full Proof.** Reflects work done searching proof space by automated theorem prover.
- **Pruned Proof.** For easier reading, should eliminate unused facts which were explored.

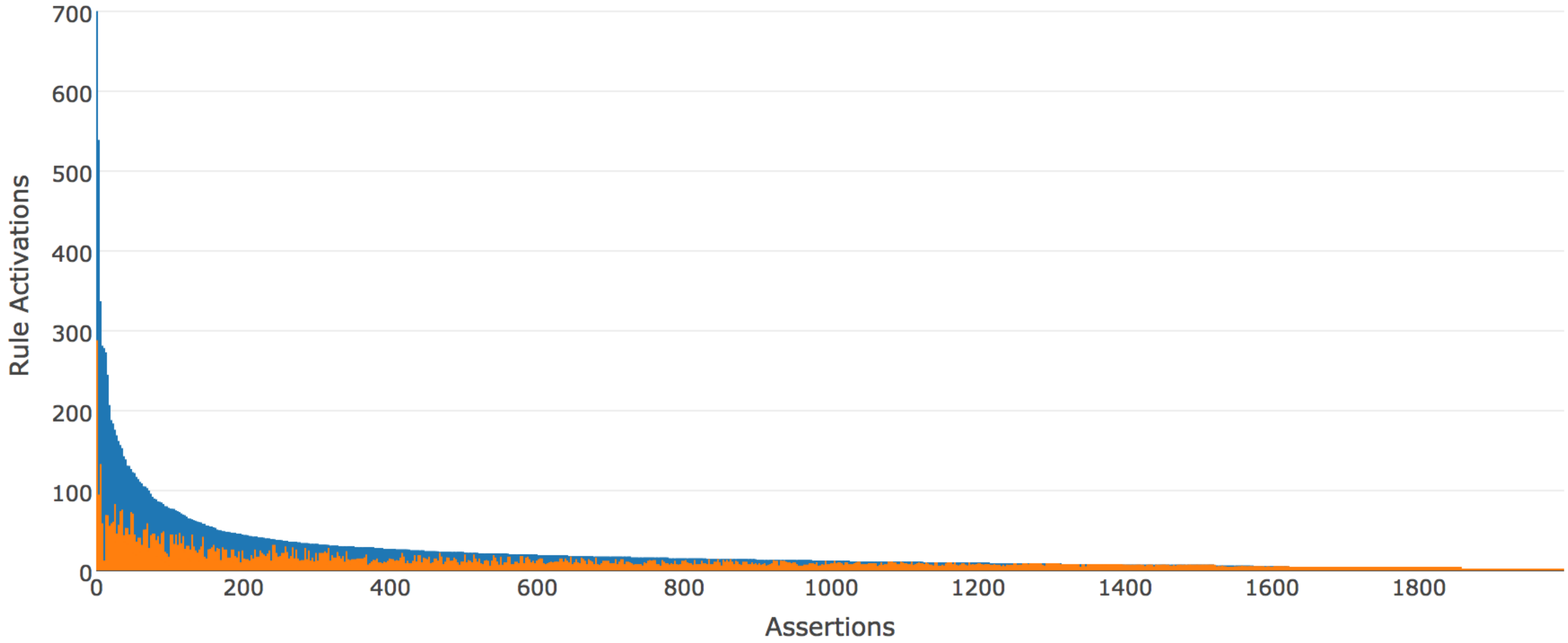
Q) *how big are these proofs?*

Theorem Proving: Data Set

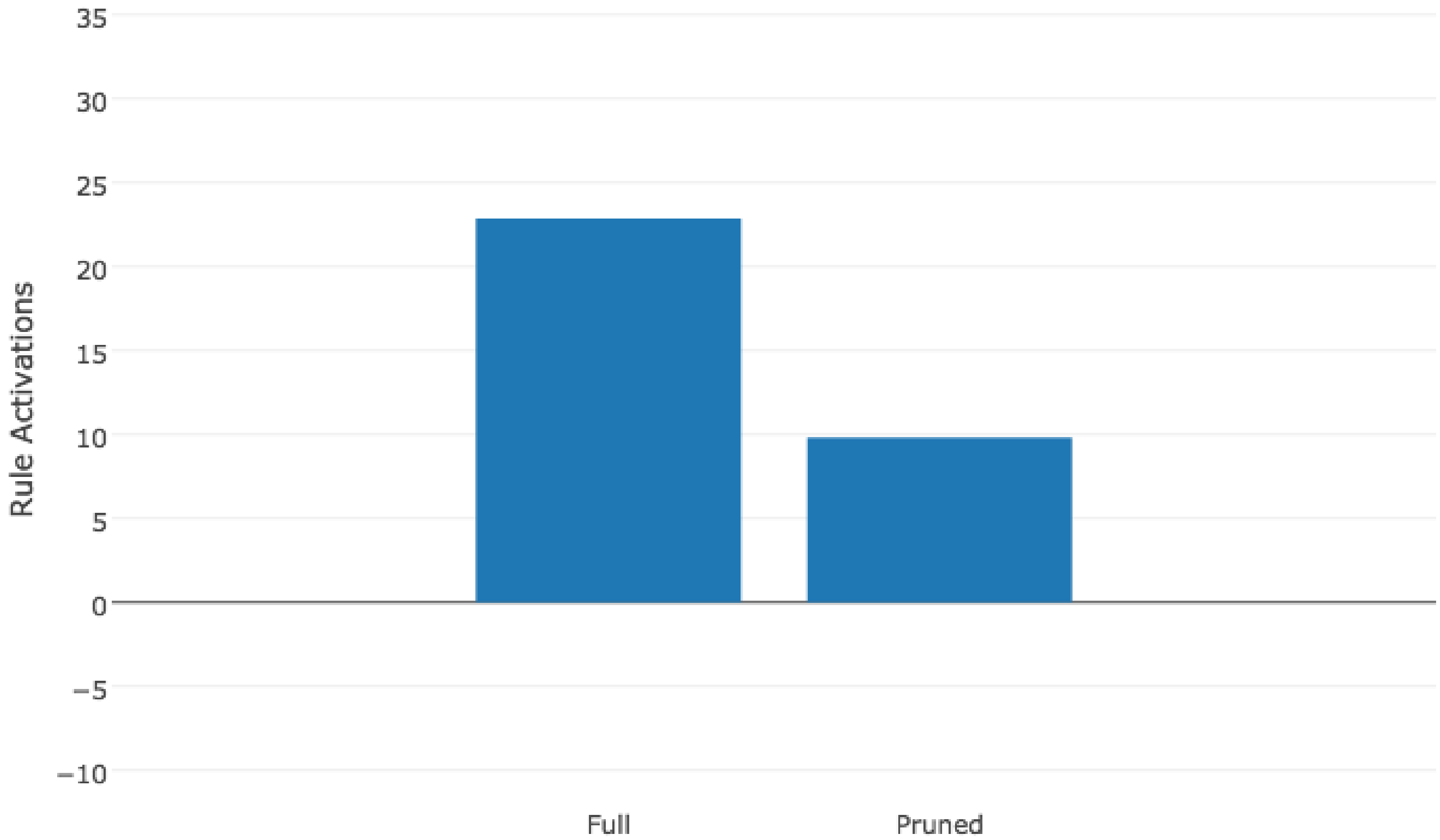


- Whiley Compiler has (approx) 540 valid and 287 invalid **test cases**
- Each test case is **single Whiley file** (either correct or not)
- From this, generated **1998 valid assertions** and **91 invalid assertions**

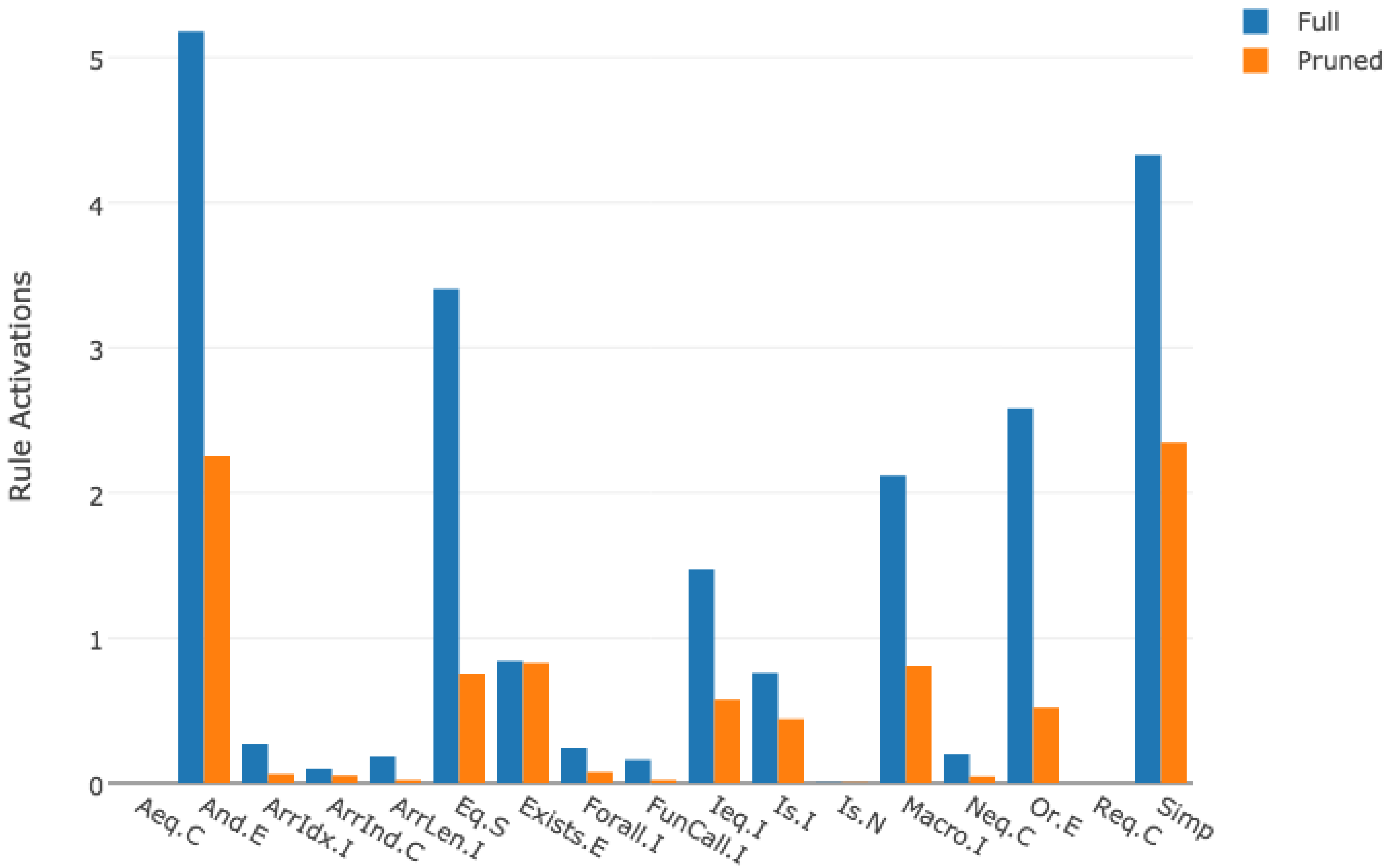
Theorem Proving: Experimental Results I



Theorem Proving: Experimental Results II

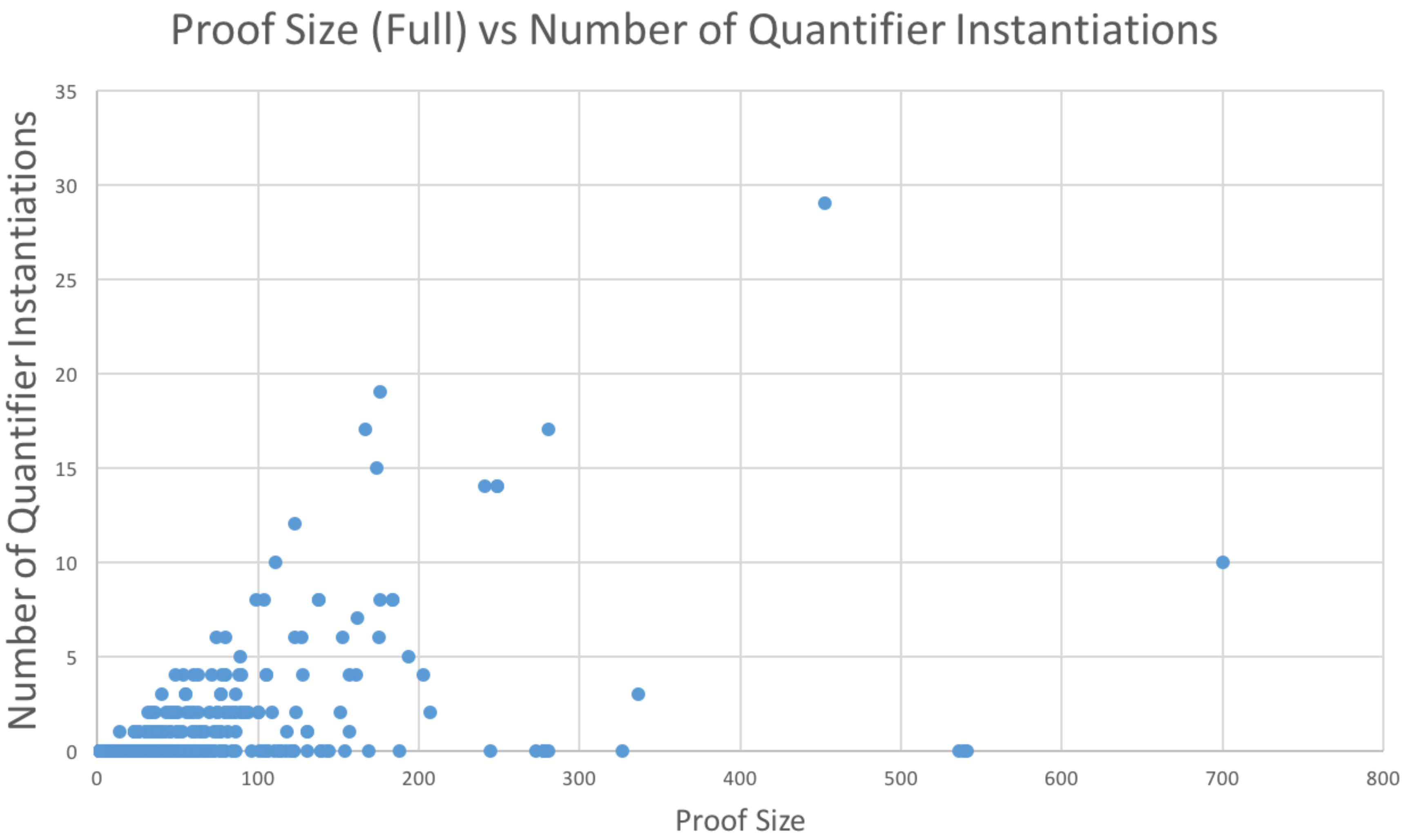


Theorem Proving: Experimental Results III

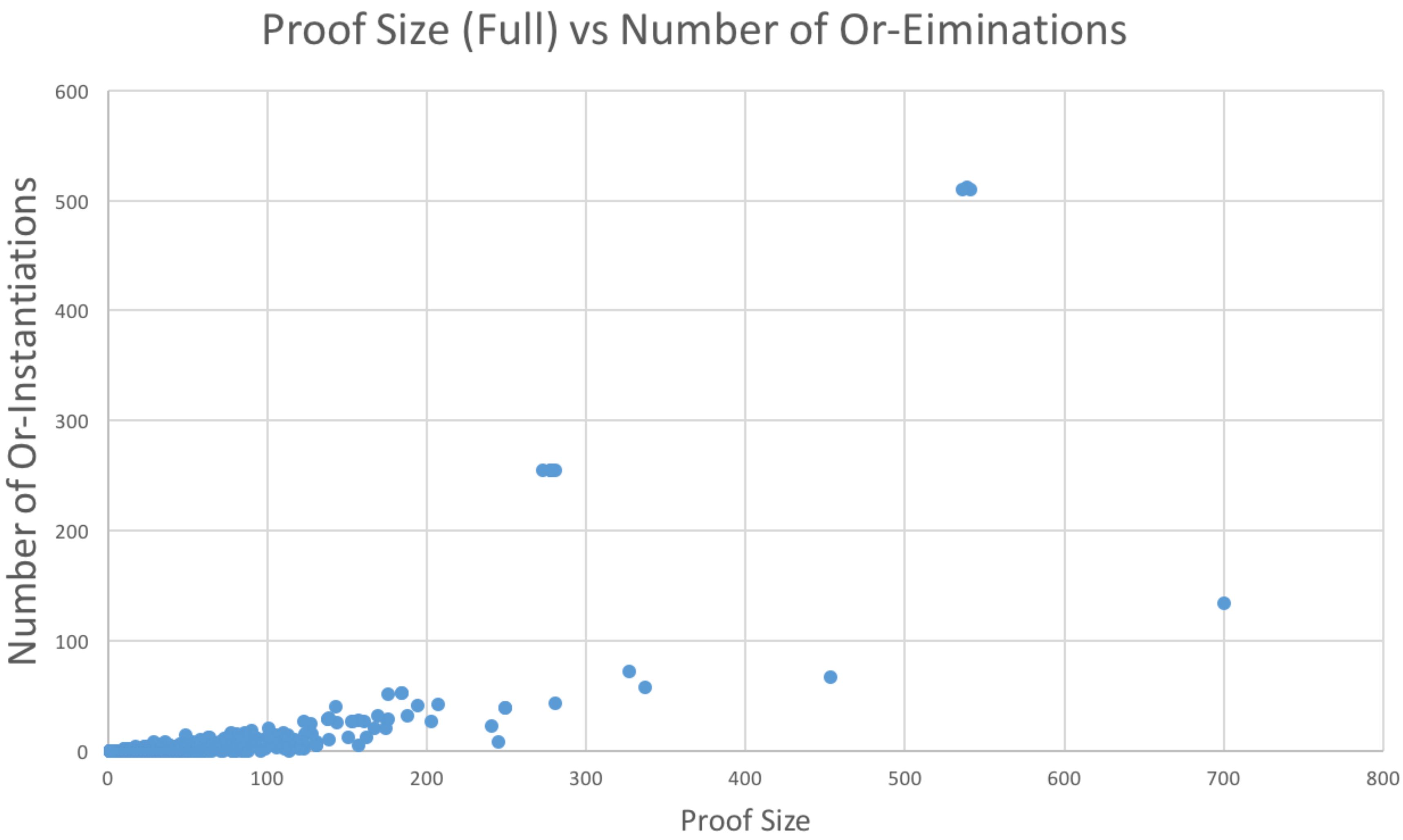


Q) *What Causes a Large Proof?*

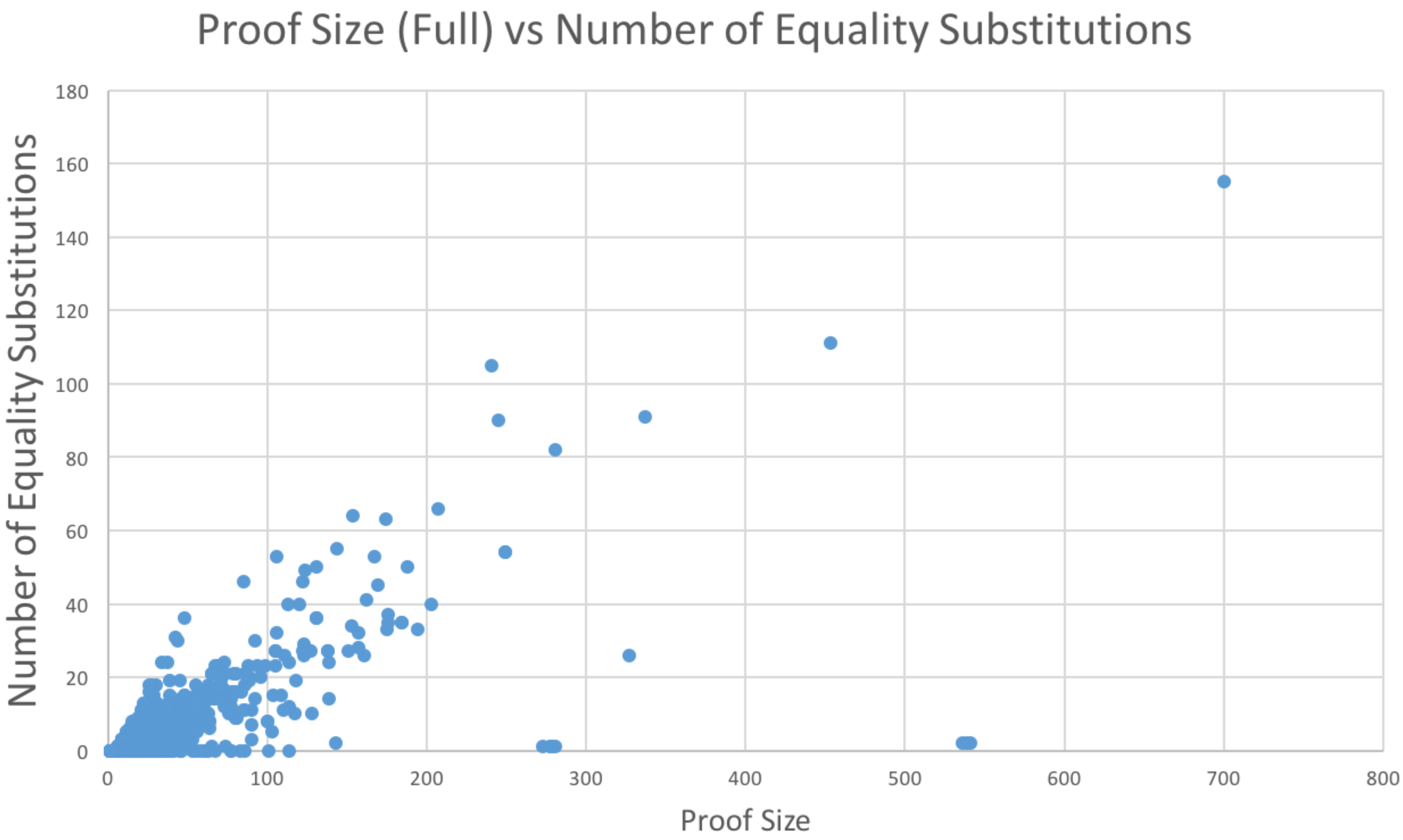
Theorem Proving: Experimental Results IV



Theorem Proving: Experimental Results V



Theorem Proving: Experimental Results VI



Theorem Proving: Counterexample Generation?

“Most bugs have small counter examples”

-Jackson’06

Theorem Proving: Counterexample Generation

- **Approach.** Use brute force generation with a "small world" (e.g. integers in range $\langle -5 \ldots 5 \rangle$, array lengths $\langle 0 \ldots 2 \rangle$, etc).

```
forall(int i, int[] arr):
    (arr[i] >= 0) ==> (i == |arr|)
```

- **Example.** For above, generate models `i=0,arr=[]`, `i=0,arr=[0]`, `i=1,arr=[0]`, etc.

- **Problems.** E.g. *uninterpreted functions* and `any`, `!int`, etc. And, what about *undefined behaviour*?

```
forall(int[] xs):
    xs[0] > 0
```

Theorem Proving: Counterexample Generation

Test	**Counterexample**
test_11	`i=1, x=[0],` i_1`=1,` i_2`=1`
test_102	`xs=[0], y=0, x=1`
test_129	x_1`={f:-1}, x={f:1}`
test_198	r_1`=[0], r=[0,0], i=0,` i_1 `=1, ls=[0,0]`

- Generated counterexamples for **75** / **91** invalid assertions!

Conclusion

Should we optimise theorem provers for BIG problems?

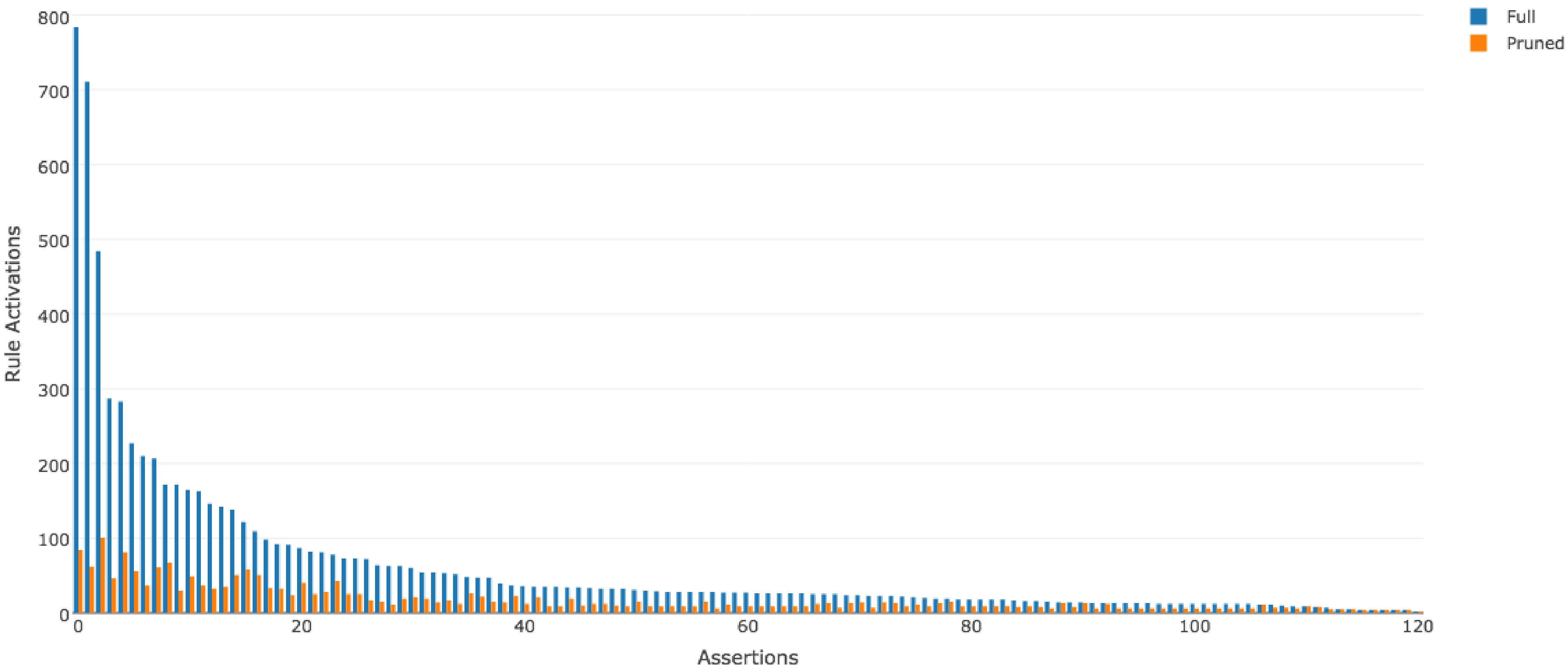
OR

Should we make them more user friendly?

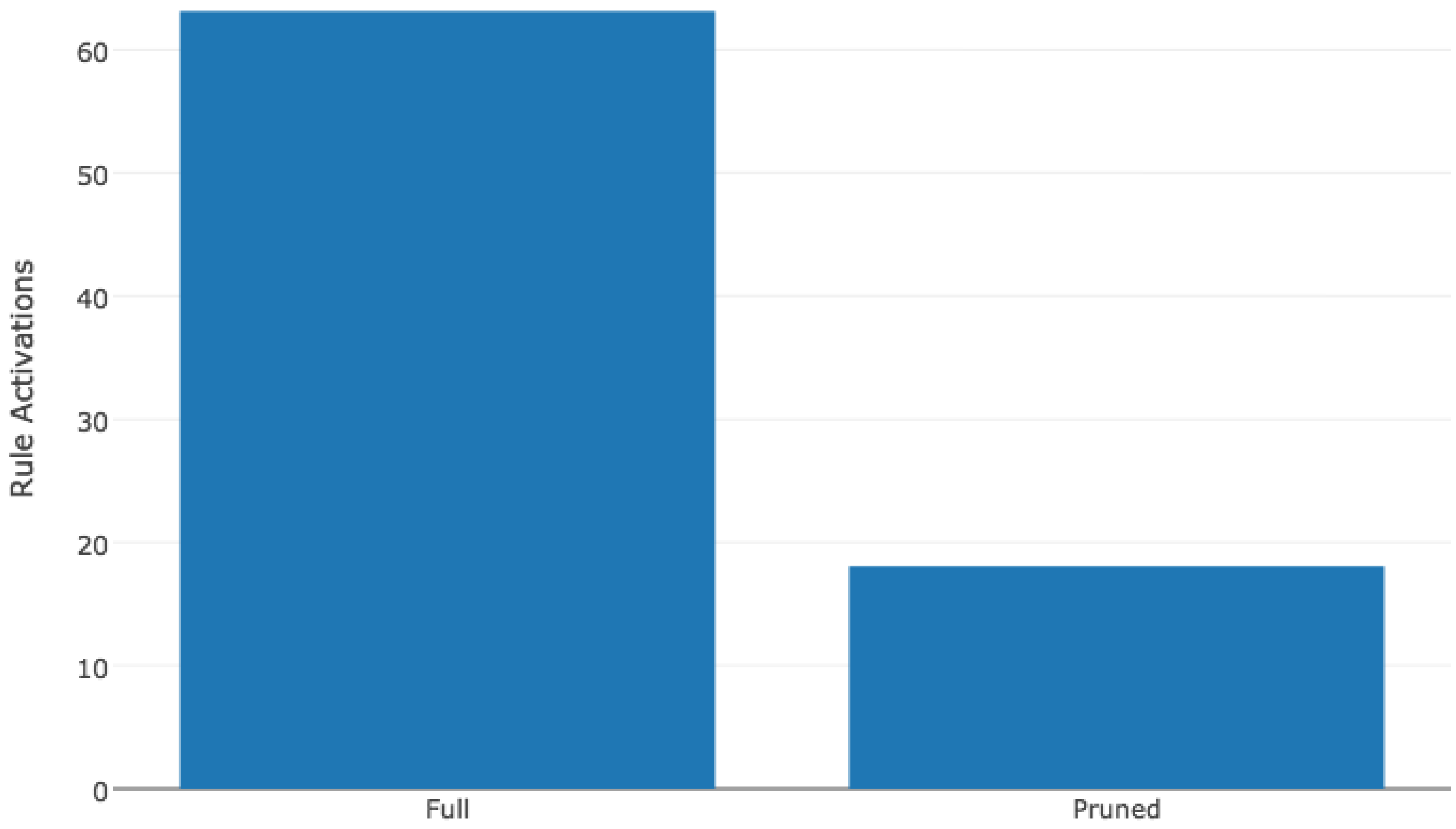
http://whiley.org

@WhileyDave
http://github.com/Whiley

Theorem Proving: (SWEN224 Dataset) Experimental Results I



Theorem Proving: (SWEN224 Dataset) Experimental Results II



Theorem Proving: (SWEN224 Dataset) Experimental Results III