Tunable Automation

Alexander Bai, Chris Hawblitzel, Andrea Lattuada
VSTTE 25

Verus

- Verus is a semi-automated program verifier for Rust programs
- Lots of systems and security work using Verus
  - https://verus-lang.github.io/verus/publications-and-projects/
- Recent effort at AWS to adopt Verus
- *Intrinsic* Verification, proof and executable code are intermixed, no need for external tools to translate and verify

Spec/Proof/Exec Mode

```
spec fn divides(n: int, k: nat) -> bool { n % (k as int) == 0 }
spec fn is_prime(n: nat) -> bool { forall|k: nat| 2 <= k < n ==> !divides(n as int, k) }
spec fn is_even(i: int) -> bool { divides(i, 2) }

proof fn even_gt_2_isnt_prime(i: nat)
  requires i > 2 && is_even(i as int)
  ensures !is_prime(i) { }

fn is_prime_impl(n: u64) -> (result: bool)
  requires n >= 2,
  ensures result == is_prime(n as nat)
{ /* ... implementation and proof ... */}
```

Quantification

Quantifier Instantiation is the primary source of incompleteness, also the primary source of automation.

In auto-active theorem provers, we often use user-level *triggers* to guide quantifier instantiations.

Triggers

```
spec fn is_even(i: int) -> bool {
    i % 2 == 0
}

proof fn seq_trigger_example(s: Seq<int>)
    requires
        5 <= s.len(),
        forall|i: int| 0 <= i < s.len() ==> #[trigger] is_even(s[i]),
{
    assert(s[3] % 2 == 0); // FAILS
}
```

only instantiating for is_even(s[3]).

Triggers

```
spec fn is_even(i: int) -> bool {
    i % 2 == 0
}

proof fn seq_trigger_example(s: Seq<int>)
    requires
        5 <= s.len(),
        forall|i: int| 0 <= i < s.len() ==> #[trigger] is_even(s[i]),
{
    assert(s[3] % 2 == 0); // OK
}
```

~~#[trigger] is_even(s[i]),~~

```
is_even(#[trigger] s[i]),
```

then we can instantiate the forall:

```
0 <= 3 < s.len() ==> is_even(s[3]),
```

Triggers

```
spec fn is_even(i: int) -> bool {
    i % 2 == 0
}

proof fn seq_trigger_example(s: Seq<int>)
    requires
        5 <= s.len(),
        forall|i: int| 0 <= i < s.len() ==> #[trigger] is_even(s[i]),
{   assert(is_even(s[3])); // OK
    assert(s[3] % 2 == 0); // OK
}
```

then we can instantiate the forall:

```
0 <= 3 < s.len() ==> is_even(s[3]),
```

Implicit Context

In Dafny/Verus, there's also an prelude of axioms, for example

```
proof fn seq_axiom_usage(s1: Seq<nat>, s2: Seq<nat>)
    requires
        s1.len() > 10 && s2.len() > 20,
    ensures
        s1.add(s2).len() > 30,
{}
```

uses this axiom:

```
pub broadcast axiom fn axiom_seq_add_len<A>(s1: Seq<A>, s2: Seq<A>)
    ensures
        #[trigger] s1.add(s2).len() == s1.len() + s2.len(),
;
```

Verus

- Verus has a “conservative” design for verification performance, which leads to more manual proofs
  - No preconditions for spec functions (total math functions)
  - Limited Prelude
    (~280 axioms in `DafnyPrelude.bpl` vs ~96 in `vir/src/prelude.rs`, and ~111 of them in vstd)
  - No non-linear reasoning by default

All of this is because Verus wants more “native” encoding to SMT level

Automation - Performance spectrum

more facts in contexts?
freer triggers?

Quantified Facts à la Carte

A mechanism for fine-grained control of quantified facts.

You can import quantified facts at any level:

- after any `verus!` macro
- modules
- proof functions (`proof fn`)
- assert (expr) by { /* proof */ }
- calculational proofs

Quantified Facts à la Carte

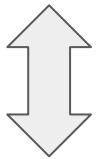
```
pub proof fn push_contains(a: Seq<int>)
{
    let b = a.push(3);
    assert(b.contains(3)); // FAILS
}
```

because Verus can’t infer the following from the default Verus context:

```
pub proof fn lemma_seq_contains_after_push<A>(s: Seq<A>, v: A, x: A)
    ensures
        s.push(v).contains(x) <==> v == x || s.contains(x),
{/* manual proof … */}
```

Quantified Facts à la Carte

```
pub broadcast proof fn lemma_seq_contains_after_push<A>(s: Seq<A>, v: A, x: A)
    ensures
        #[trigger] s.push(v).contains(x) <==> v == x || s.contains(x),
```



```
pub proof fn lemma_seq_contains_after_push<A>()
    ensures
        forall|s: Seq<A>, v: A, x: A|
            #[trigger] s.push(v).contains(x) <==> v == x || s.contains(x),
```

In VeriFast/Dafny, this kind of lemmas are named “auto” lemmas

Quantified Facts à la Carte

```
pub proof fn push_contains(a: Seq<int>)
{
    broadcast use vstd::seq_lib::lemma_seq_contains_after_push;
    let b = a.push(3);
    assert(b.contains(3)); // PASSES because of the increased context
}
```

Verus Standard Library

In 2023, we provide some sort of automation by explicitly universally quantifying the input parameters. Now it's just a broadcast group.

```
/// Properties of sequences from the Dafny prelude (which were axioms in Dafny, but proven here in Verus)
#[deprecated = "Use `broadcast use group_seq_properties` instead"]
pub proof fn lemma_seq_properties<A>()
    ensures
        forall|s: Seq<A>, x: A|
            s.contains(x) <==> exists|i: int| 0 <= i < s.len() && #[trigger] s[i] == x,  //from lemma_seq_contains(s, x),
        forall|x: A| !(#[trigger] Seq::<A>::empty().contains(x)),  //from lemma_seq_empty_contains_nothing(x),
        forall|s: Seq<A>| #[trigger] s.len() == 0 ==> s =~= Seq::<A>::empty(),  //from lemma_seq_empty_equality(s),
        forall|x: Seq<A>, y: Seq<A>, elt: A| #[trigger]
            (x + y).contains(elt) <==> x.contains(elt) || y.contains(elt),  //from lemma_seq_concat_contains_all_elements(x, y, elt),
        forall|s: Seq<A>, v: A, x: A| #[trigger] s.push(v).contains(x) <==> v == x || s.contains(x),  //from lemma_seq_contains_after_push(s, v, x)
        forall|s: Seq<A>, start: int, stop: int, x: A|
            (0 <= start <= stop <= s.len() && #[trigger] s.subrange(start, stop).contains(x)) <==> (
            exists|i: int| 0 <= start <= i < stop <= s.len() && #[trigger] s[i] == x),  //from lemma_seq_subrange_elements(s, start, stop, x),
        forall|s: Seq<A>, n: int| 0 <= n <= s.len() ==> #[trigger] s.take(n).len() == n,  //from lemma_seq_take_len(s, n)
        forall|s: Seq<A>, n: int, x: A|
            (#[trigger] s.take(n).contains(x) && 0 <= n <= s.len()) <==> (exists|i: int|
                0 <= i < n <= s.len() && #[trigger] s[i] == x),  //from lemma_seq_take_contains(s, n, x),
        forall|s: Seq<A>, n: int, j: int| 0 <= j < n <= s.len() ==> #[trigger] s.take(n)[j] == s[j],  //from lemma_seq_take_index(s, n, j),
        forall|s: Seq<A>, n: int| 0 <= n <= s.len() ==> #[trigger] s.skip(n).len() == s.len() - n,  //from lemma_seq_skip_len(s, n),
        forall|s: Seq<A>, n: int, x: A|
            (#[trigger] s.skip(n).contains(x) && 0 <= n <= s.len()) <==> (exists|i: int|
                0 <= n <= i < s.len() && #[trigger] s[i] == x),  //from lemma_seq_skip_contains(s, n, x),
```

```
// include all the Dafny prelude lemmas
pub broadcast group group_seq_properties {
    lemma_seq_contains,
    lemma_seq_empty_contains_nothing,
    lemma_seq_empty_equality,
    lemma_seq_concat_contains_all_elements,
    lemma_seq_contains_after_push,
    lemma_seq_subrange_elements,
    lemma_seq_take_len,
    lemma_seq_take_contains,
    lemma_seq_take_index,
    lemma_seq_skip_len,
    lemma_seq_skip_contains,
    lemma_seq_skip_index,
    lemma_seq_skip_index2,
    lemma_seq_append_take_skip,
    lemma_seq_take_update_commut1,
    lemma_seq_take_update_commut2,
    lemma_seq_skip_update_commut1,
    lemma_seq_skip_update_commut2,
    lemma_seq_skip_build_commut,
    lemma_seq_skip_nothing,
    lemma_seq_take_nothing,
    // Removed the following from group due to bad verification performance
    // for `lemma_merge_sorted_with_ensures`
    // lemma_seq_skip_of_skip,
    group_to_multiset_ensures,
}
```

Quantified Facts à la Carte

```
pub proof fn push_contains(a: Seq<int>)
{
    broadcast use vstd::seq_lib::group_seq_properties;
    let b = a.push(3);
    assert(b.contains(3));
}
```

Worried about verification performance for larger proofs?

Quantified Facts à la Carte

```
pub proof fn push_contains(a: Seq<int>)
{
                                    lemma_seq_contains_after_push
    broadcast use vstd::seq_lib::[redacted];
    let b = a.push(3);
    assert(b.contains(3));
}
```

```
> verus broadcast.rs -V axiom-usage-info
note: checking this function used these broadcasted lemmas and broadcast groups:
        - (group) vstd::seq_lib::group_seq_properties,
        - vstd::seq_lib::lemma_seq_contains_after_push
 --> broadcast.rs:5:1
  |
5 | pub proof fn push_contains(a: Seq<int>)
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

verification results:: 1 verified, 0 errors
```

Modularized Proof Libraries

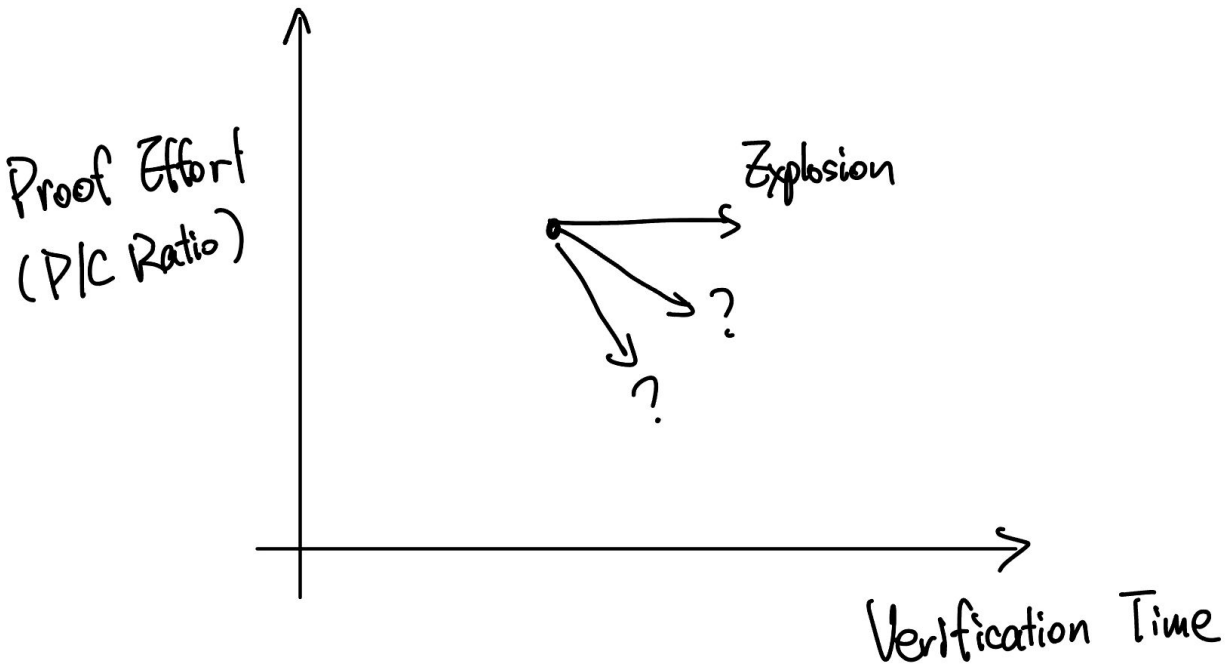
```
pub trait Key {
    ...
    proof fn key_obligations()
        ensures // ... conditions necessary for the type to be a valid key

    broadcast proof fn trans_lt_lt(a:Self, b:Self, c:Self)
        ensures a < b && b < c ==> a < c
    { /* justified thanks to the ensures of `key_obligations` */ }
    // ... additional properties ...
}

pub broadcast group group_key_cmp_properties {
    Key::trans_lt_lt,
    // ... additional `broadcast` proofs from the trait
}
```

Exploring the Automation Tradeoff

With all the Dafny Prelude lemma in Verus (with broadcast proofs), we can see what the impact is for increased implicit context:

Collection datatype	Seq	Map	Set	MultiSet	Total
# of group_<type>_properties lemmas	25	2	10	12	49

1. Does increasing the number of quantified facts in context result in more automation, i.e. fewer manual user-provided hints (in the form of asserts)?
2. Does increasing the number of quantified facts in context hinder verification performance or the verification experience?


Proof Effort
(P/C Ratio)
Explosion
?
?
Verification Time

Projects under Study

- IronKV (SOSP24) is a distributed key-value store
- Splinter is an ongoing work on a key-value store designed around a Bε-tree.
- Anvil (OSDI24) is a framework for building and formally verifying Kubernetes controllers.
- CapybaraKV (OSDI25) is a storage system targeting persistent memory devices

Minimization

- To quantify “automation”, we use the number of asserts as a metric, but most projects don’t have a minimized number of assertions.
- We ran VerusMinimizer to compare the number of asserts w/ and w/o ambient facts

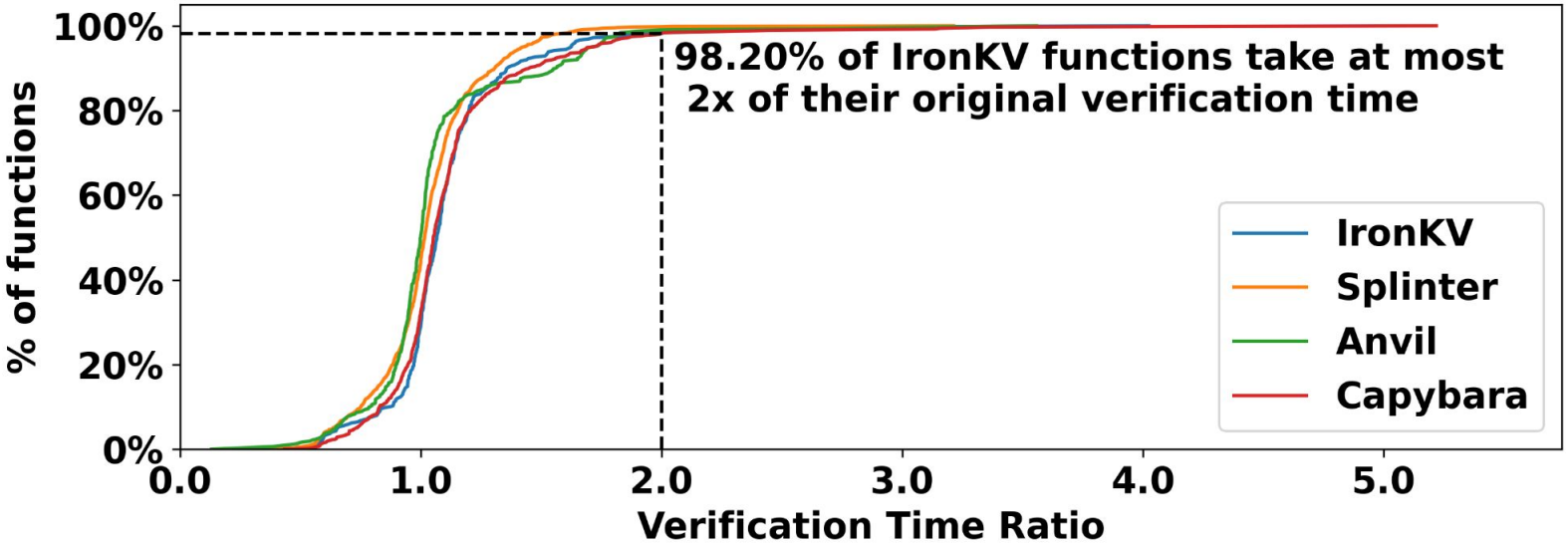
- Limitations:
  - Only a linear scan
  - Only target assertions (since detecting lemma calls can’t be done syntactically)

Effect of Ambient Facts

System	Original # of asserts	Minimized (baseline)	Minimized with Ambient Facts
IronKV	646	268	245(-23, -8.6%)
Splinter	2678	1158	1130(-28, -2.4%)
Anvil	701	343	336(-7, -2.0%)
CapybaraKV	941	449	415 (-34, -7.6%)

Table 1: Assertion counts for verification systems after minimization and importing ambient facts

Impact on Verification Failure Time

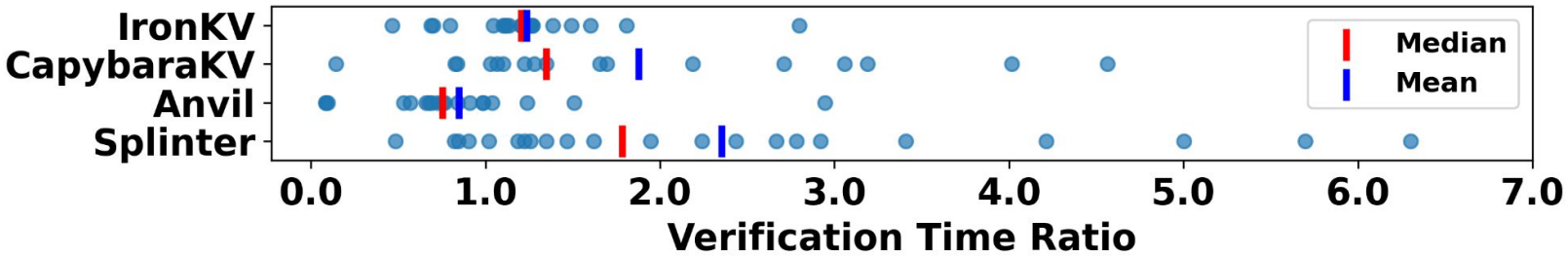


Verification Failure Time

- Once the solver finds a proof, it stops, which does not happen if the proof is incomplete or incorrect.

- As we dramatically increased search space, we might also have dramatic increase in verification failure time.

Impact on Verification Failure Time



Triggers

- default safe triggers, or more aggressive multiple trigger groups?

```
forall|x: int, y: int|       f(x) && g(y) && h(x,y)
              default: (0)                  ^^^^^^

       all candidates: (1)   ^^^^    ^^^^
                       (2)   ^^^^            ^^^^^^ X <= 5
                       (3)         ^^^^      ^^^^^^ X <= 5
                       (4)   ^^^^  ^^^^      ^^^^^^ X <= 5
                       (5)                   ^^^^^^

         all_triggers: (1)   ^^^^    ^^^^
                       (5)                   ^^^^^^
```

More Automation with More Triggers (Axioms?)

Is it possible to obtain a more extensive automation (i.e. a smaller number of required hints (asserts) to the solver) by changing how triggers are selected for the default set of quantified facts imported when using the Verus standard library?

You can’t really “free” the triggers without explosion

```
pub broadcast axiom fn axiom_seq_add_len<A>(s1: Seq<A>, s2: Seq<A>)
    ensures
        #[trigger] s1.add(s2).len() == s1.len() + s2.len(),
;
```

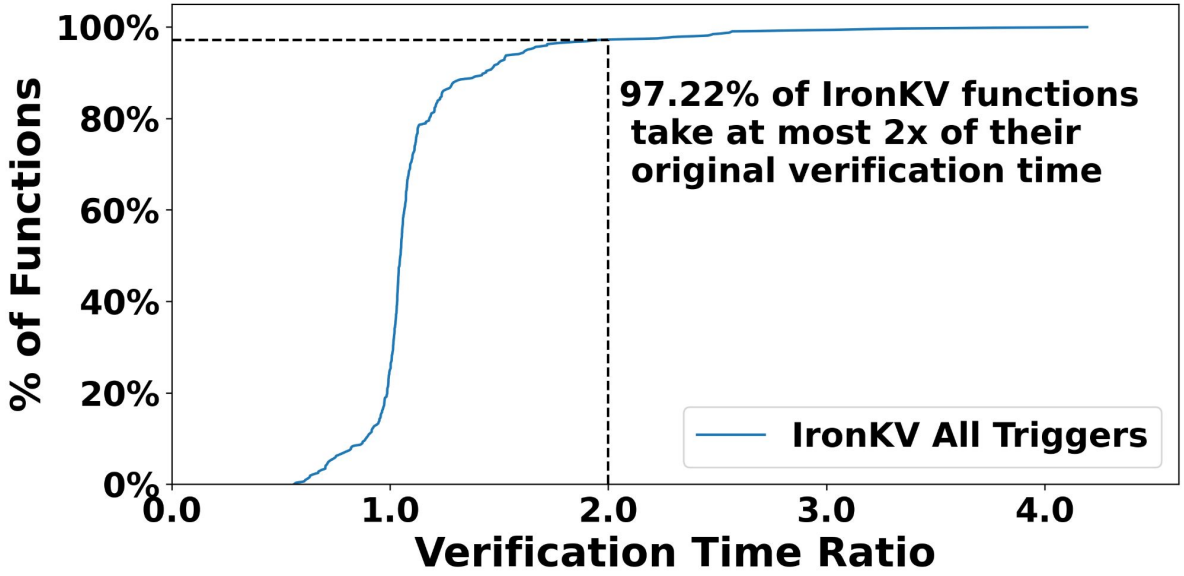
More Automation with More Triggers (User Code?)

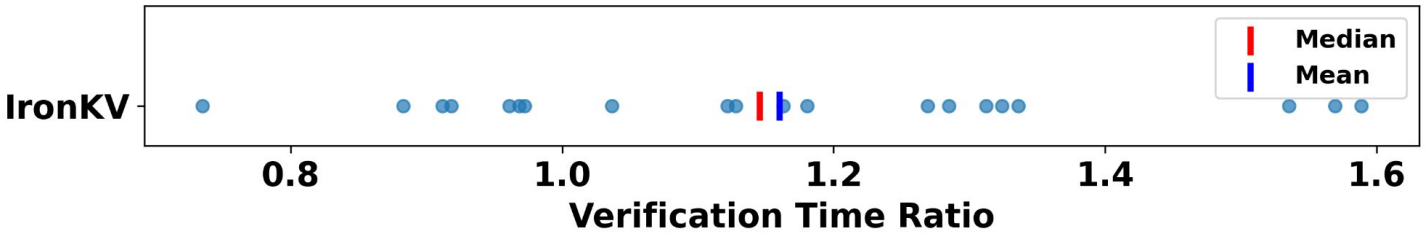
We added 206 #![all_triggers] annotations in the 261 quantifiers in IronKV.

One proof broke.

System	Original	Min	All Triggers	Ambient Facts
IronKV	646	268	235(-33, -12.3%)	245(-23, -8.6%)

Table 2: Assertion counts for IronKV after minimization with all_triggers


100%
80%
60%
40%
20%
0%
% of Functions
97.22% of IronKV functions
take at most 2x of their
original verification time
IronKV All Triggers
0.0
1.0
2.0
3.0
4.0
Verification Time Ratio


IronKV
Median
Mean
0.8
1.0
1.2
1.4
1.6
Verification Time Ratio

Lessons from all_triggers?

- all_triggers has a non-trivial impact, though it is more susceptible to verification failure
- Managing triggers are trickier, it’s hard to manage in bulk
- Experiment with multiple trigger groups for different level of automation?

Conclusion

Thank you!
ayb5065@nyu.edu

- Broadcast as a way of managing quantified facts
  - Customized axioms for user built abstractions
  - unsat-core for pinpointing useful lemmas
- VerusMinimizer to check “minimized” assertions
- More ambient facts in context:
  - reduced 2-9% assertions
  - only have impact on a handful of functions
- Turning on `all_triggers`
  - Preliminary result showed it’s more impactful than ambient facts
  - harder to automatically use all_triggers