Linear Types for Large-Scale Systems Verification

JIALIN LI, University of Washington, USA
ANDREA LATTUADA, ETH Zurich, Switzerland
YI ZHOU, Carnegie Mellon University, USA
JONATHAN CAMERON, Carnegie Mellon University, USA
JON HOWELL, VMware Research, USA
BRYAN PARNO, Carnegie Mellon University, USA
CHRIS HAWBLITZEL, Microsoft Research, USA

Verifying large systems

◀ VeriBεtrKV
[Hance OSDI’20]

on-disk crash-safe KV-store

44K lines code+proof in Dafny

31K lines of generated C++

Verifying large systems

VeriBεtrKV
[Hance OSDI’20]

on-disk crash-safe KV-store

44K lines code+proof in Dafny

31K lines of generated C++

performance focus

goal: scale verification techniques

Verifying large systems

VeriBεtrKV
[Hance OSDI’20]

on-disk crash-safe KV-store

44K lines code+proof in Dafny

31K lines of generated C++

performance focus

goal: scale verification techniques

success with Dafny, but

Challenges in SMT memory reasoning with dynamic frames (Dafny)

proof burden	developer effort
verification time	developer iteration time
diagnostics	

Challenges in SMT memory reasoning with dynamic frames (Dafny)

proof burden — developer effort

verification time

diagnostics

developer iteration time

compound at scale

Challenges in SMT memory reasoning with dynamic frames (Dafny)

proof burden — developer effort

verification time

diagnostics

developer iteration time

compound at scale

we measured them and set out to improve them

```
class Account {
  var balance: nat;
}


method Transfer(source: Account, dest: Account, amount: nat)
  requires source.balance >= amount

  ensures source.balance == old(source.balance) - amount
  ensures dest.balance == old(dest.balance) + amount
  modifies source, dest
{
  source.balance := source.balance - amount;
  dest.balance := dest.balance + amount;
}


method Main(acct: Account)
requires acct.balance >= 100
{
  Transfer(acct, acct, 100)
}
```

```
class Account {
  var balance: nat;
}


method Transfer(source: Account, dest: Account, amount: nat)
  requires source.balance >= amount

  ensures source.balance == old(source.balance) - amount
  ensures dest.balance == old(dest.balance) + amount    postcondition might not hold
  modifies source, dest
{
  source.balance := source.balance - amount;
  dest.balance := dest.balance + amount;
}


method Main(acct: Account)
requires acct.balance >= 100
{
  Transfer(acct, acct, 100)
}
```

```
class Account {
  var balance: nat;
}


method Transfer(source: Account, dest: Account, amount: nat)
  requires source.balance >= amount
  requires source != dest
  ensures source.balance == old(source.balance) - amount
  ensures dest.balance == old(dest.balance) + amount     postcondition might not hold
  modifies source, dest
{
  source.balance := source.balance - amount;
  dest.balance := dest.balance + amount;
}


method Main(acct: Account)
requires acct.balance >= 100
{
  Transfer(acct, acct, 100)
}
```

```
class Account {
  var balance: nat;
}


method Transfer(source: Account, dest: Account, amount: nat)
  requires source.balance >= amount
  requires source != dest
  ensures source.balance == old(source.balance) - amount
  ensures dest.balance == old(dest.balance) + amount
  modifies source, dest
{
  source.balance := source.balance - amount;
  dest.balance := dest.balance + amount;
}


method Main(acct: Account)
requires acct.balance >= 100
{
  Transfer(acct, acct, 100)
}
```

a precondition might not hold

```
class Account {
  var balance: nat;
}


method Transfer(source: Account, dest: Account, amount: nat)
  requires source.balance >= amount

  ensures source.balance == old(source.balance) - amount
  ensures dest.balance == old(dest.balance) + amount
  modifies source, dest
{
  source.balance := source.balance - amount;
  dest.balance := dest.balance + amount;
}


method Main(acct: Account)
requires acct.balance >= 100
{
  Transfer(acct, acct, 100)
}
```

postcondition might not hold

logic error or
missing framing condition?

Dynamic frames address potential aliasing

general, but costly

Dynamic frames address potential aliasing

general, but costly

- vague error messages

Dynamic frames address potential aliasing

general, but costly

- vague error messages
- framing invariants grow with system size
  - → more proof text

Dynamic frames address potential aliasing

general, but costly

vague error messages

framing invariants grow with system size
→ more proof text

more difficult for the solver to discharge framing VCs
→ longer verification time

Dynamic frames address potential aliasing

general, but costly

Dynamic frames address potential aliasing

general, but costly

aliasing isn’t the common case

Dynamic frames address potential aliasing

general, but costly

aliasing isn’t the common case

demonstrated by Rust’s success

Dynamic frames address potential aliasing

general, but costly

aliasing isn’t the common case

demonstrated by Rust’s success

hypothesis: we can lower development effort by making the non-aliasing code cheaper to reason about

Dynamic frames address potential aliasing

general, but costly

aliasing isn’t the common case

demonstrated by Rust’s success

hypothesis: we can lower development effort by making the non-aliasing code cheaper to reason about

▶ Linear type system

Linear Dafny

linear type system for SMT-based verification

Linear Dafny

linear type system for SMT-based verification

type system + SMT solver

extend expressivity of linear types leveraging the solver

Linear Dafny

linear type system for SMT-based verification

type system + SMT solver

extend expressivity of linear types leveraging the solver

1. memory reasoning with linear types

Linear Dafny

linear type system for SMT-based verification

type system + SMT solver

extend expressivity of linear types leveraging the solver

1. memory reasoning with linear types
2. regions to address non-linear data

Linear Dafny

linear type system for SMT-based verification

type system + SMT solver

extend expressivity of linear types leveraging the solver

1. memory reasoning with linear types
2. regions to address non-linear data
3. quantitative and qualitative evaluation
   on a large system (VeriBεtrKV)

Variable usages

		duplicate	compiled
dafny	ordinary	yes	yes
	ghost	yes	

Variable usages

		duplicate	compiled
dafny	ordinary	yes	yes
	ghost	yes	
linear dafny	shared	yes*	yes
	linear		yes

linear usage

```
linear datatype Account = Account(balance: nat)

method Transfer(linear source: Account, linear dest: Account, amount: nat)
returns (linear source': Account, linear dest': Account)
  requires source.balance >= amount
  ensures source'.balance == source.balance - amount
  ensures dest'.balance == dest.balance + amount
{
  source’ := source;
  dest’ := dest;
  var new_source_balance := source’.balance - amount;
  var new_dest_balance := dest’.balance + amount;
  AccountSetBalance(inout source’, new_source_balance);
  AccountSetBalance(inout dest’, new_dest_balance);
}

method AccountSetBalance(linear inout a: Account, balance: nat)
ensures a.balance == balance
{
  inout a.balance := balance;
}
```

in-place update

linear usage

```
linear datatype Account = Account(balance: nat)

method Transfer(linear source: Account, linear dest: Account, amount: nat)
returns (linear source': Account, linear dest': Account)
  requires source.balance >= amount
  ensures source'.balance == source.balance - amount
  ensures dest'.balance == dest.balance + amount
{
  source’ := source;
  dest’ := dest;
  var new_source_balance := source’.balance - amount;
  var new_dest_balance := dest’.balance + amount;
  AccountSetBalance(inout source’, new_source_balance);
  AccountSetBalance(inout dest’, new_dest_balance);
}
```

shared usage

```
linear datatype Account = Account(balance: nat)

method Transfer(linear source: Account, linear dest: Account, amount: nat)
returns (linear source': Account, linear dest': Account)
  requires source.balance >= amount
  ensures source'.balance == source.balance - amount
  ensures dest'.balance == dest.balance + amount
{
  source’ := source;
  dest’ := dest;
  var new_source_balance := (
                                  borrow source’
    source’.balance - amount;
                                  end of borrow source’
  );
  var new_dest_balance := dest’.balance + amount;
  AccountSetBalance(inout source’, new_source_balance);
  AccountSetBalance(inout source’, new_dest_balance);
}
```

Evaluation

improvement in proof burden at scale
verification time
diagnostics

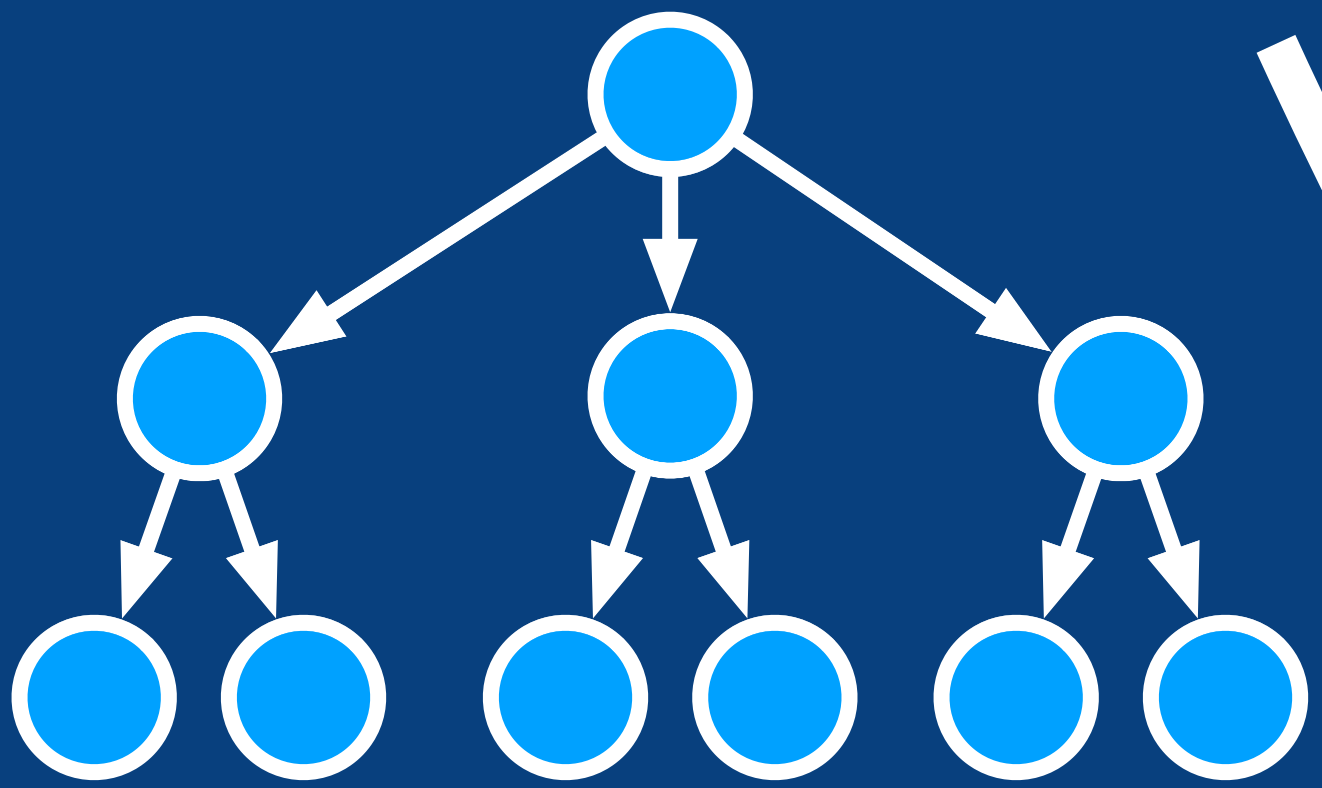
VeribetrKV — 24K lines code+proof
of imperative code

proven equivalent to high-level spec
via state-machine refinement

Conversion To Linear Dafny

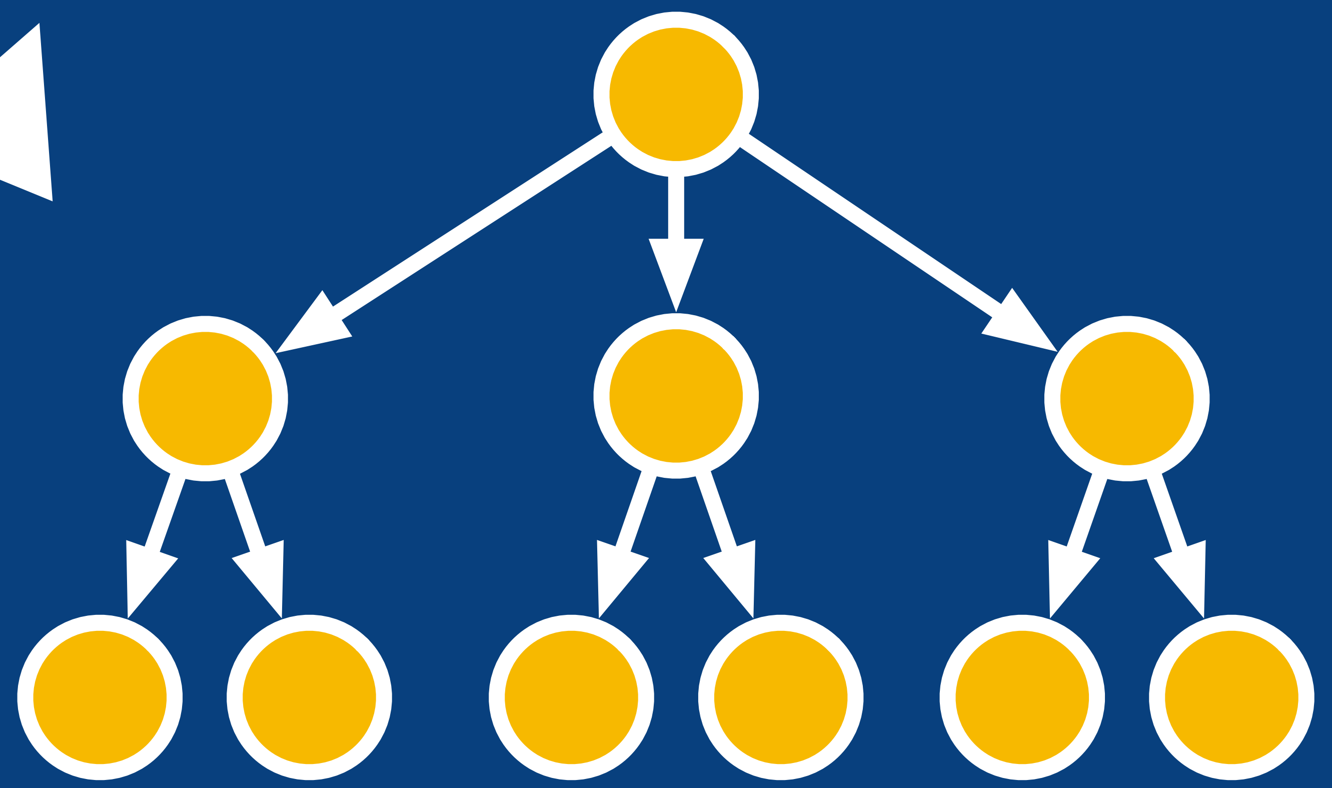
Dynamic frames

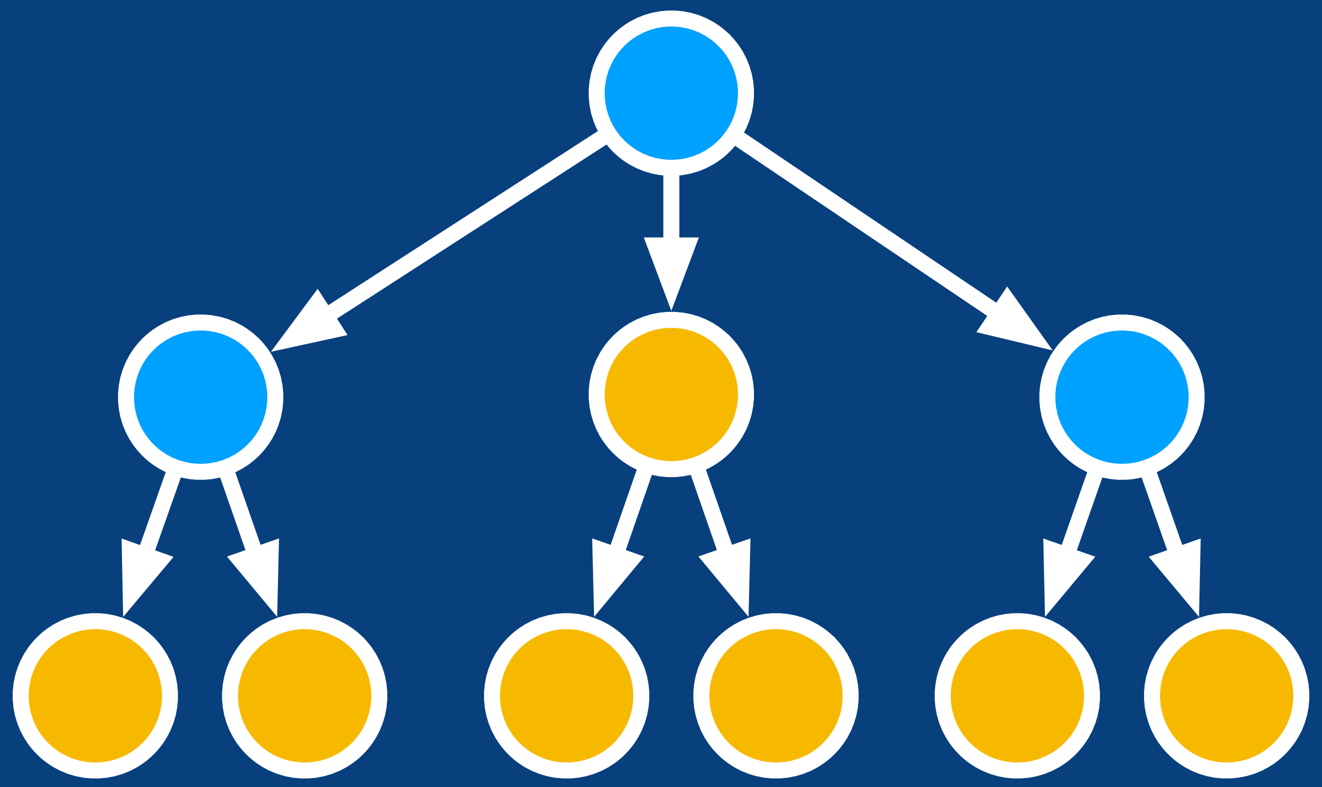
VeribetrKV-DF



Linear types

VeribetrKV-LT





linear data inside non-linear data

🟦 nonlinear component

🟨 linear component

Dynamic frames

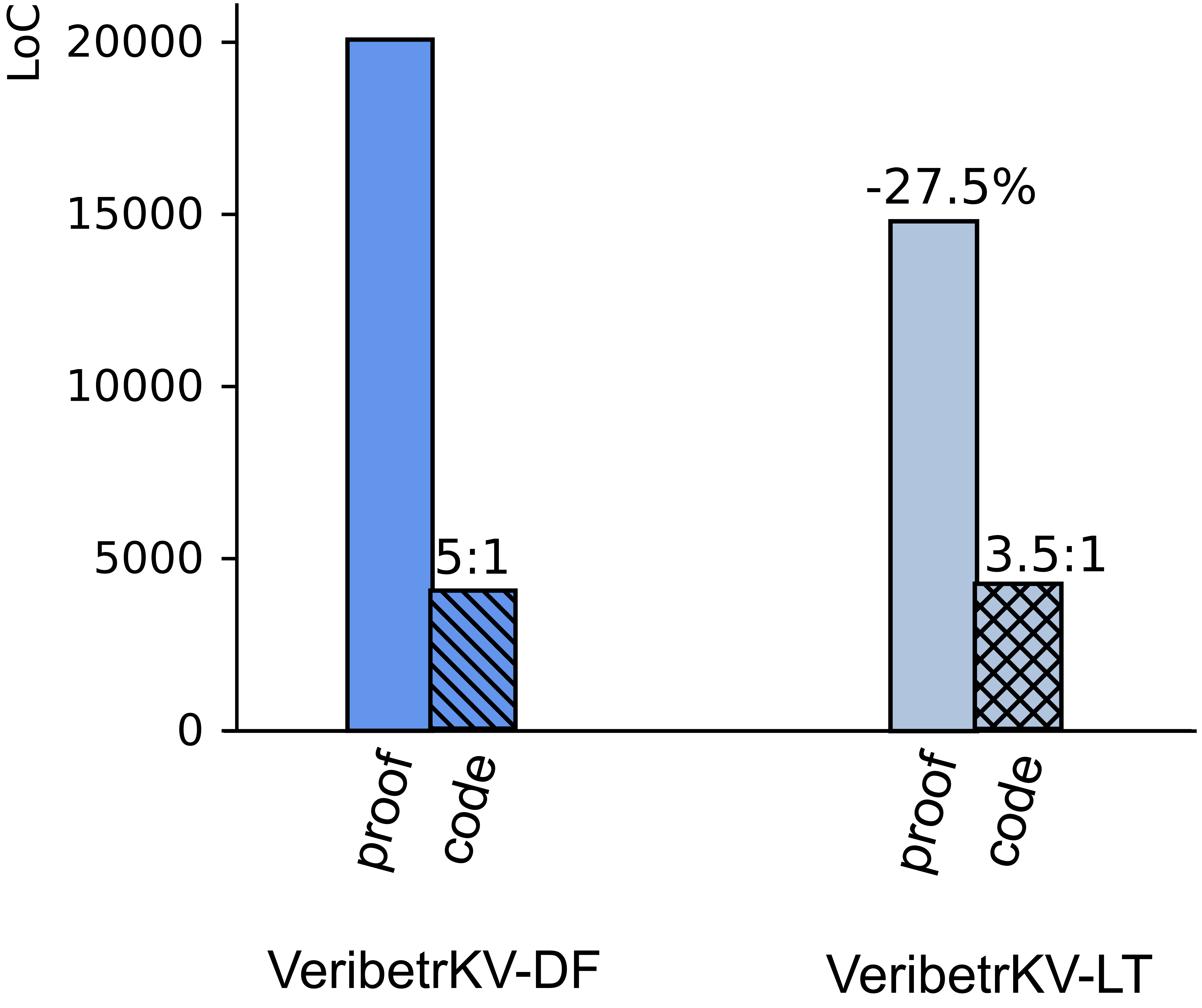
VeribetrKV-DF <> VeribetrKV-LT

Linear types

Proof burden

lines of proof

proof:code ratio



Dynamic frames

VeribetrKV-DF

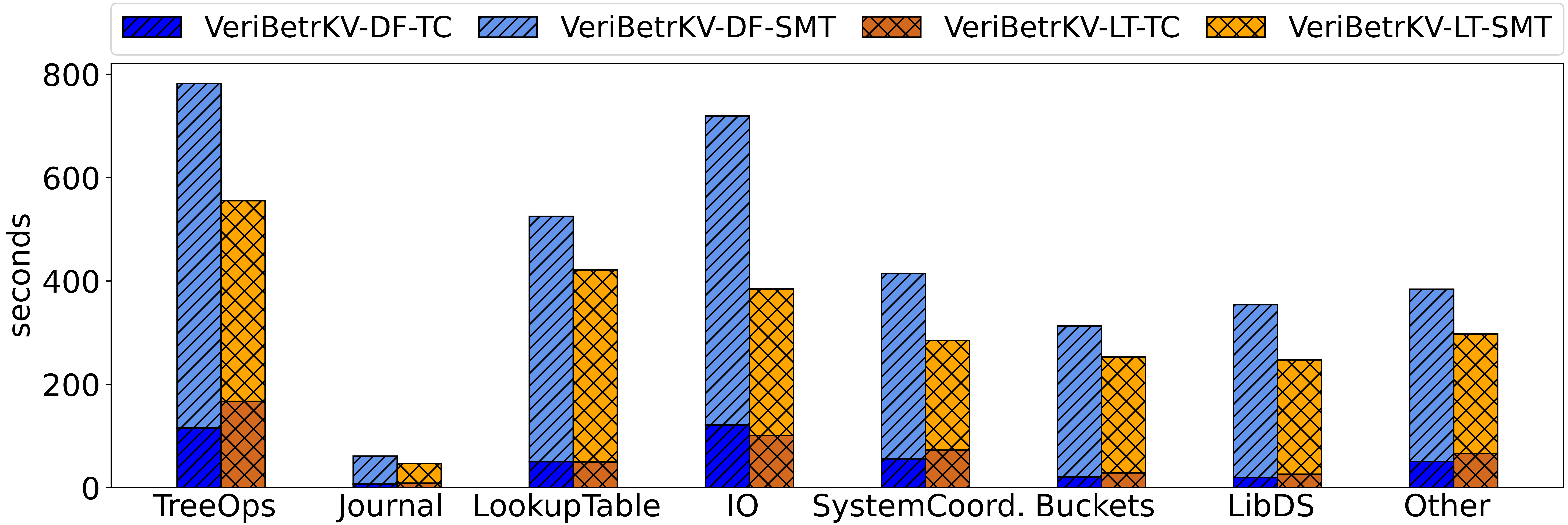
< >

Linear types

VeribetrKV-LT

Verification time proxy for developer iteration time

Type checking (TC), and SMT solving



Dynamic frames

VeribetrKV-DF

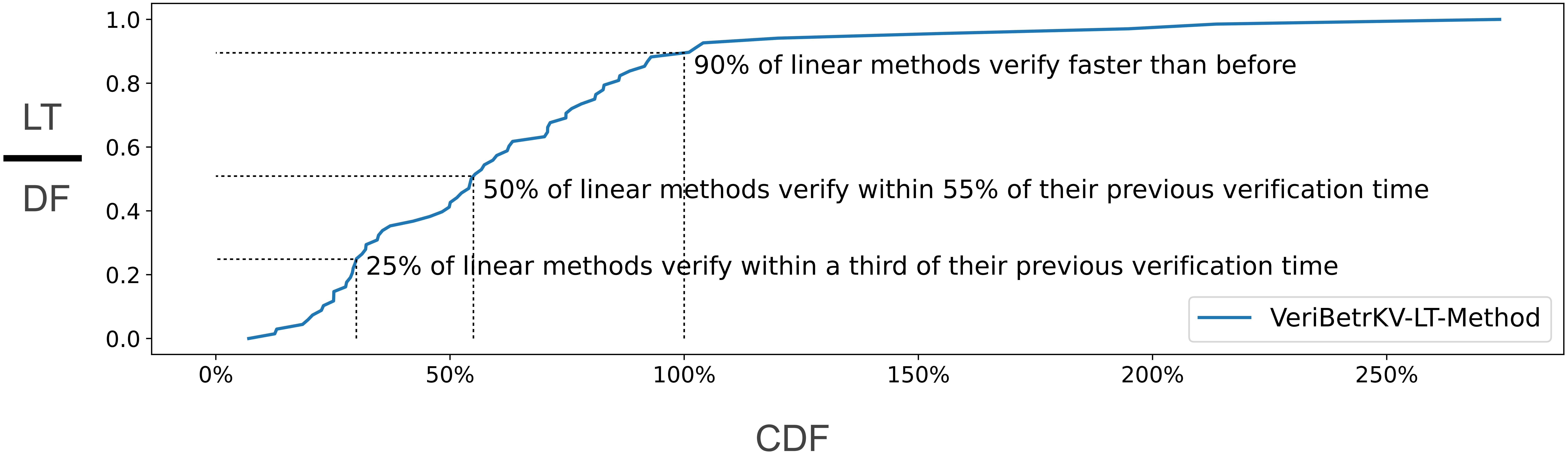
<>

Linear types

VeribetrKV-LT

Verification time improvement

for functions ≥ 5sec



90% of linear methods verify faster than before

50% of linear methods verify within 55% of their previous verification time

25% of linear methods verify within a third of their previous verification time

Linear Dafny

+ immutable+mutable borrowing

+ non-linear inside linear and viceversa

linear type system + SMT-based verification

→ lower developer effort

→ faster developer iteration time

evaluation likely underestimates potential benefits

incremental conversion enabled evaluation

→ improved diagnostics

SMT-based (semi-automated) verification at scale