Agentic Specification Generation for Move Programs

Yu-Fu Fu, Meng Xu*, Taesoo Kim

Georgia Institute of Technology, *University of Waterloo

ASE 2025

Formal Verification Adoption

Growing adoption. From cloud, browser to your cars.

Automated Reasoning checks is now available in Amazon Bedrock Guardrails

Posted on: Aug 6, 2025

Verified cryptography for Firefox 57

HACL*

Benjamin Beurdouche | September 13, 2017

Ada and SPARK enter the automotive ISO-26262 market with NVIDIA.

Ada
In Strong Typing We Trust

SPARK 2014
Expanding the boundaries of safe and secure programming.

NVIDIA

Georgia Tech

Verify systems against formal specifications

High Assurance System

Rocket firmware 🚀

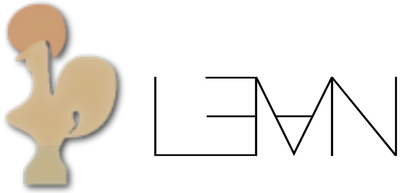
Smart contracts 💰💰

High Assurance System

Rocket firmware 🚀

Smart contracts 💰💰

Verifier



High Assurance System

Rocket firmware 🚀

Smart contracts 💰💰

Verifier

Dafny

MOVE

Formal Specification

High Assurance System

Rocket firmware 🚀

Smart contracts 💰💰

Verifier

LEAN

Dafny

MOVE

Automatic deductive verifier

Formal Specification

High Assurance System

Rocket firmware 🚀

Smart contracts 💰💰

Verifier

LEAN

Dafny

MOVE

Automatic deductive verifier

Formal Specification

Verification Expert Time-consuming

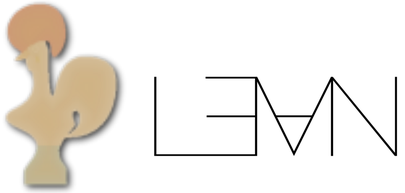
Georgia Tech

High Assurance System

Rocket firmware 🚀

Smart contracts 💰💰

Verifier



Automatic deductive verifier

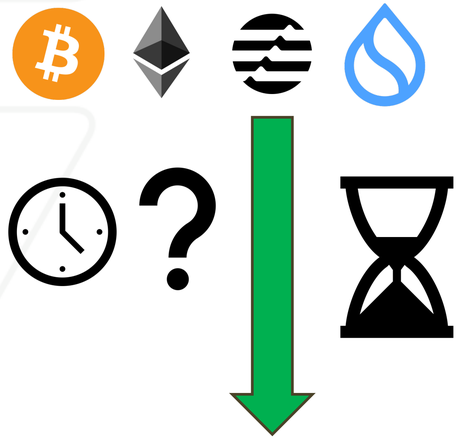
Formal Specification

Verification Expert
Time-consuming

Georgia Tech

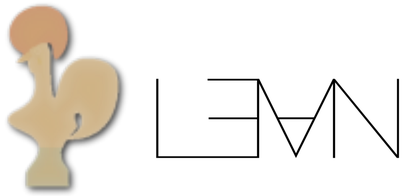
High Assurance System

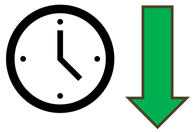
Rocket firmware 🚀
Smart contracts 💰💰



Time to market
Product release

Verifier





Automatic deductive verifier

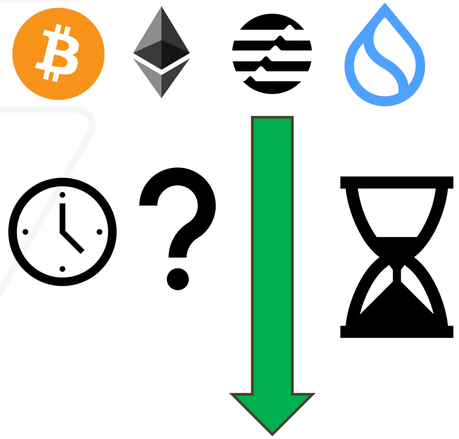
Formal Specification

Verification Expert
Time-consuming

Code first with **loose** documentation / NL Spec.
Specification & Verification is an **after-thought**!
Bugs lead to **billions of $$$ loss**.
To minimize **gap**,

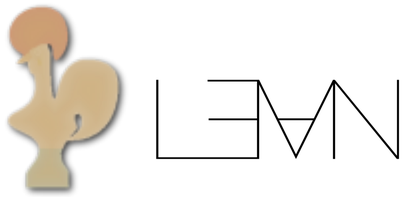
High Assurance System

Rocket firmware 🚀
Smart contracts 💰💰

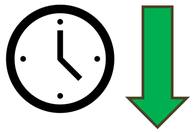


Time to market
Product release

Verifier



MOVE



Automatic deductive verifier

Formal Specification

Verification Expert
Time-consuming

Code first with **loose** documentation / NL Spec.
Specification & Verification is an **after-thought**!
Bugs lead to **billions of $$$ loss**.
To minimize **gap**,

Specification generation tool from implementation

MSG: Move Specification Generator

Georgia Tech

Move

- **New** languages for smart contracts; originally for Diem from Facebook.
- Sui & Aptos: evolved to two different Move dialects
  - Total **over $10B** market cap combined. Gain fast attention.
- Rust-like syntax/semantic & built-in safety feature
  - borrow checking / resource safety / overflow check
- Move Specification Language & Move Prover
  - **automatic** verifier powered by SMT solvers like Z3/CVC5.

Georgia Tech

Move Specification Example

```
struct Coin has store {
  value: u64
}
```

```
struct Balance has key {
  coin: Coin
}
```

```
/// withdraw `amount` to get a Coin
fun withdraw(addr: address, amount: u64): Coin
            acquires Balance {
    let balance = balance_of(addr);
    assert!(balance >= amount, EINSUFFICIENT_BALANCE);
    let balance_ref = &mut
      borrow_global_mut<Balance>(addr).coin.value;
    *balance_ref = balance - amount;
    Coin { value: amount }
  }
```

```
spec withdraw {
  let balance = global<Balance>(addr).coin.value;
  modifies global<Balance>(addr);

  aborts_if !exists<Balance>(addr);
  aborts_if balance < amount;

  let post balance_post = global<Balance>(addr).coin.value;
  ensures result == Coin { value: amount };
  ensures balance_post == balance - amount;}
```

modifies	whether global resources are changed
aborts_if	whether function aborts for the condition (like pre-condition)*
ensures	post-condition
N/A	loop invariant (inlined in function body)

let: before state
let post: after state

*: Move also has “requires” clauses for pre-condition but developers prefers runtime validation

Move Specification Example

```
struct Coin has store {
  value: u64
}
```

```
struct Balance has key {
  coin: Coin
}
```

```
/// withdraw `amount` to get a Coin
fun withdraw(addr: address, amount: u64): Coin
            acquires Balance {
    let balance = balance_of(addr);
    assert!(balance >= amount, EINSUFFICIENT_BALANCE);
    let balance_ref = &mut
      borrow_global_mut<Balance>(addr).coin.value;
    *balance_ref = balance - amount;
    Coin { value: amount }
  }
```

```
spec withdraw {
  let balance = global<Balance>(addr).coin.value;
  modifies global<Balance>(addr);

  aborts_if !exists<Balance>(addr);
  aborts_if balance < amount;

  let post balance_post = global<Balance>(addr).coin.value;
  ensures result == Coin { value: amount };
  ensures balance_post == balance - amount;}
```

modifies	whether global resources are changed
aborts_if	whether function aborts for the condition (like pre-condition)*
ensures	post-condition
N/A	loop invariant (inlined in function body)

let: before state
let post: after state

*: Move also has “requires” clauses for pre-condition but developers prefers runtime validation

Georgia Tech

Move Specification Example

```
struct Coin has store {
  value: u64
}
```

```
struct Balance has key {
  coin: Coin
}
```

```
/// withdraw `amount` to get a Coin
fun withdraw(addr: address, amount: u64): Coin
            acquires Balance {
    let balance = balance_of(addr);
    assert!(balance >= amount, EINSUFFICIENT_BALANCE);
    let balance_ref = &mut
      borrow_global_mut<Balance>(addr).coin.value;
    *balance_ref = balance - amount;
    Coin { value: amount }
  }
```

```
spec withdraw {
  let balance = global<Balance>(addr).coin.value;
  modifies global<Balance>(addr);

  aborts_if !exists<Balance>(addr);
  aborts_if balance < amount;

  let post balance_post = global<Balance>(addr).coin.value;
  ensures result == Coin { value: amount };
  ensures balance_post == balance - amount;}
```

modifies	whether global resources are changed
aborts_if	whether function aborts for the condition (like pre-condition)*
ensures	post-condition
N/A	loop invariant (inlined in function body)

let: before state
let post: after state

*: Move also has “requires” clauses for pre-condition but developers prefers runtime validation

Georgia Tech

Move Specification Example

```
struct Coin has store {
  value: u64
}
```

```
struct Balance has key {
  coin: Coin
}
```

```
/// withdraw `amount` to get a Coin
fun withdraw(addr: address, amount: u64): Coin
            acquires Balance {
    let balance = balance_of(addr);
    assert!(balance >= amount, EINSUFFICIENT_BALANCE);
    let balance_ref = &mut
      borrow_global_mut<Balance>(addr).coin.value;
    *balance_ref = balance - amount;
    Coin { value: amount }
  }
```

```
spec withdraw {
  let balance = global<Balance>(addr).coin.value;
  modifies global<Balance>(addr);

  aborts_if !exists<Balance>(addr);
  aborts_if balance < amount;

  let post balance_post = global<Balance>(addr).coin.value;
  ensures result == Coin { value: amount };
  ensures balance_post == balance - amount;}
```

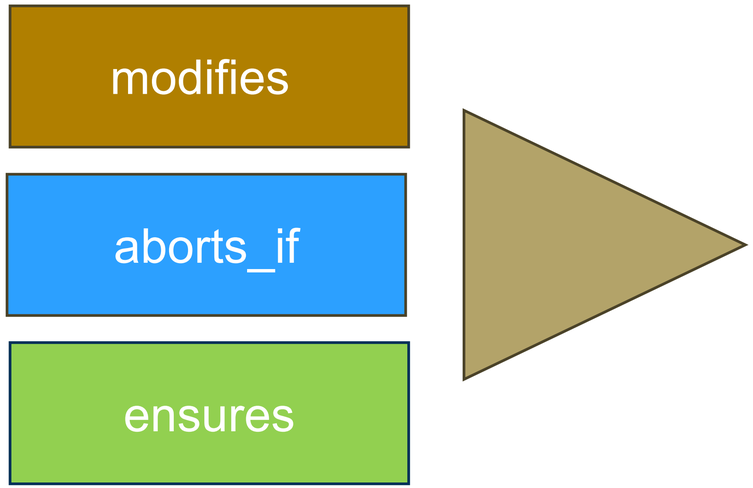
modifies	whether global resources are changed
aborts_if	whether function aborts for the condition (like pre-condition)*
ensures	post-condition
N/A	loop invariant (inlined in function body)

let: before state
let post: after state

*: Move also has “requires” clauses for pre-condition but developers prefers runtime validation

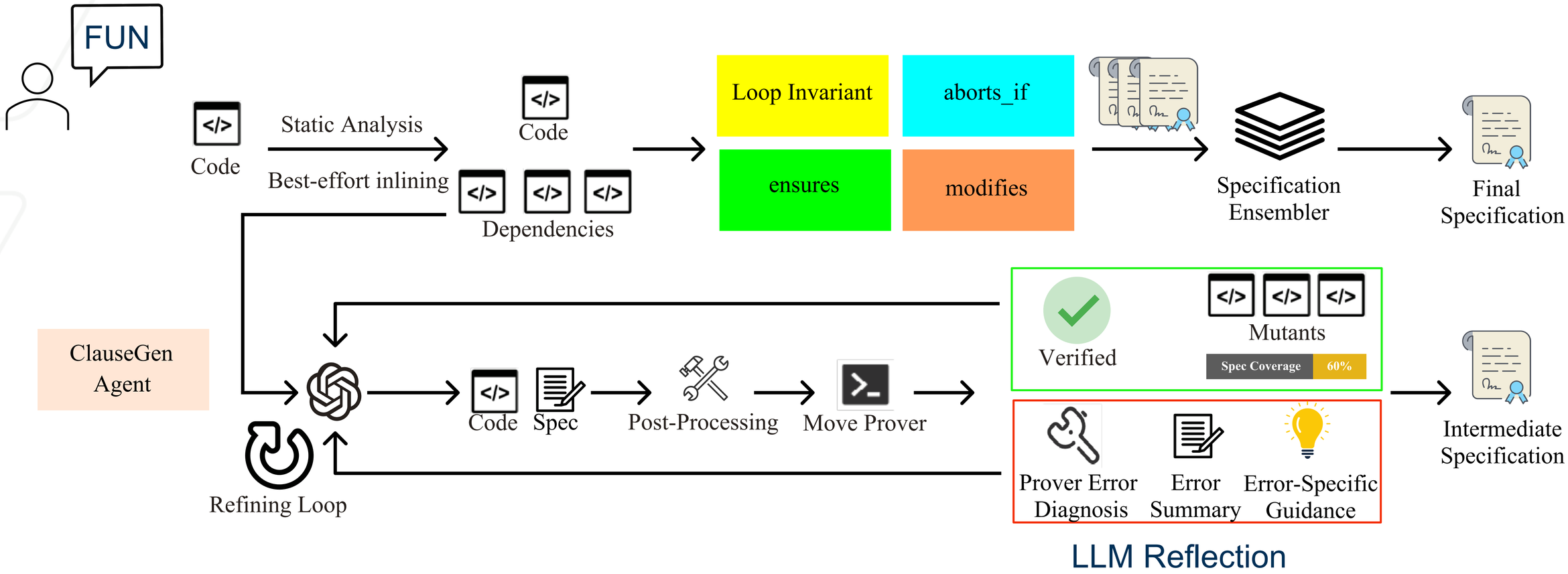
Observation: Compositional Specification

- Verify each clause type **independently** => generate **independently**
- **Ensemble** individual specifications into a **unified and idiomatic** one.
- **Not only Move**. We showed similar construct on Certora Prover for Solidity in Eth.



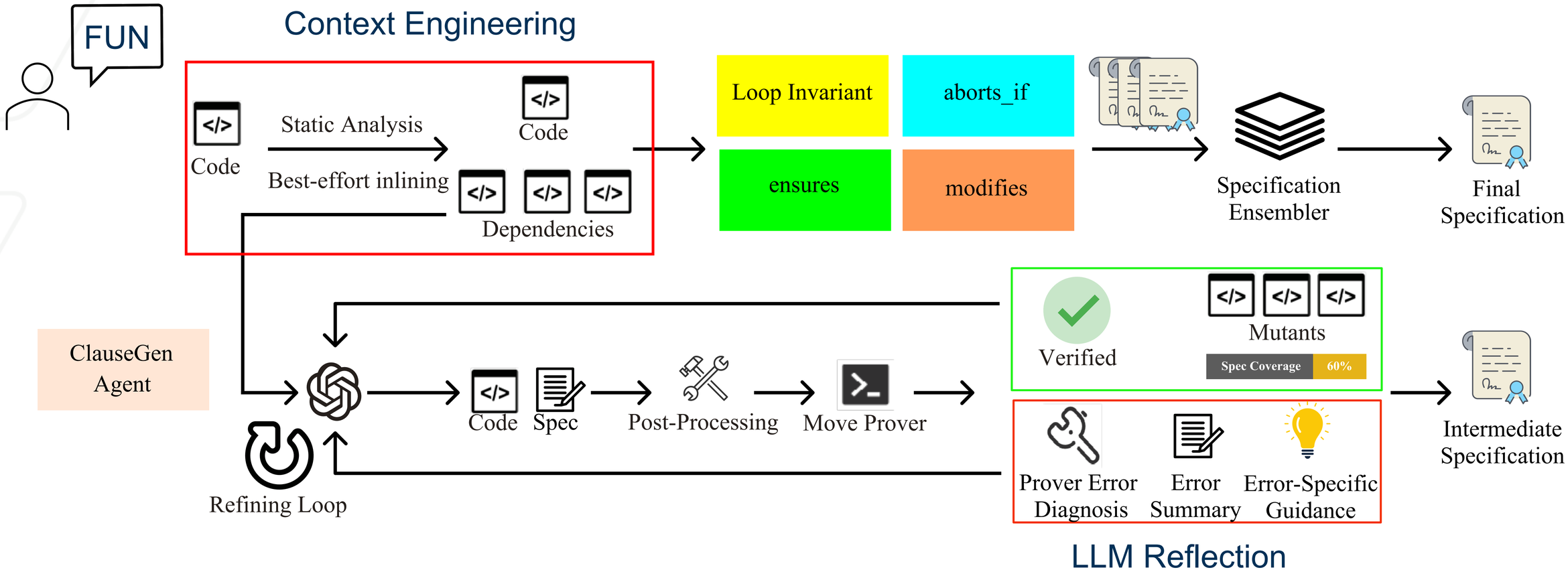

```
spec withdraw {
  let balance = global<Balance>(addr).coin.value;
  modifies global<Balance>(addr);

  aborts_if !exists<Balance>(addr);
  aborts_if balance < amount;

  let post balance_post = global<Balance>(addr).coin.value;
  ensures result == Coin { value: amount };
  ensures balance_post == balance - amount;}
```

MSG: Move Specification Generator



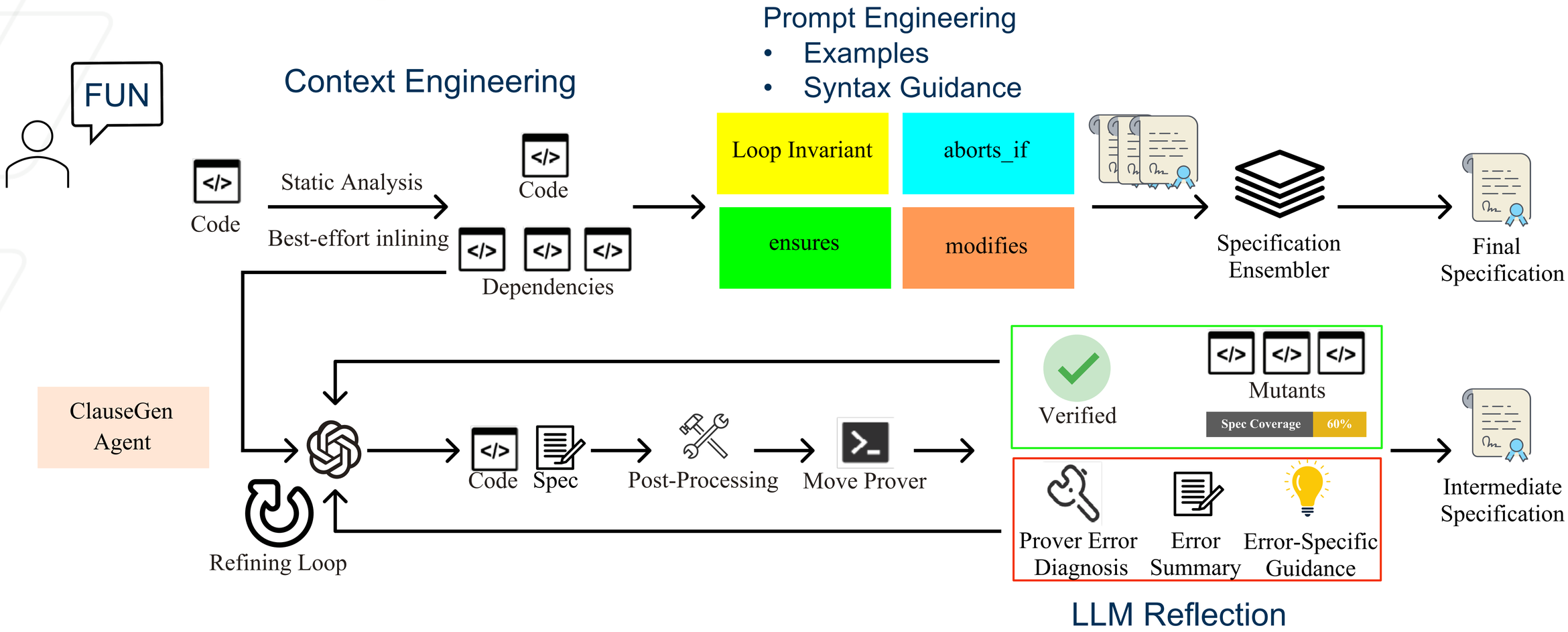
Integrated with Aptos Move Toolchain
~7.3k LoC Rust code

MSG: Move Specification Generator



Integrated with Aptos Move Toolchain
~7.3k LoC Rust code

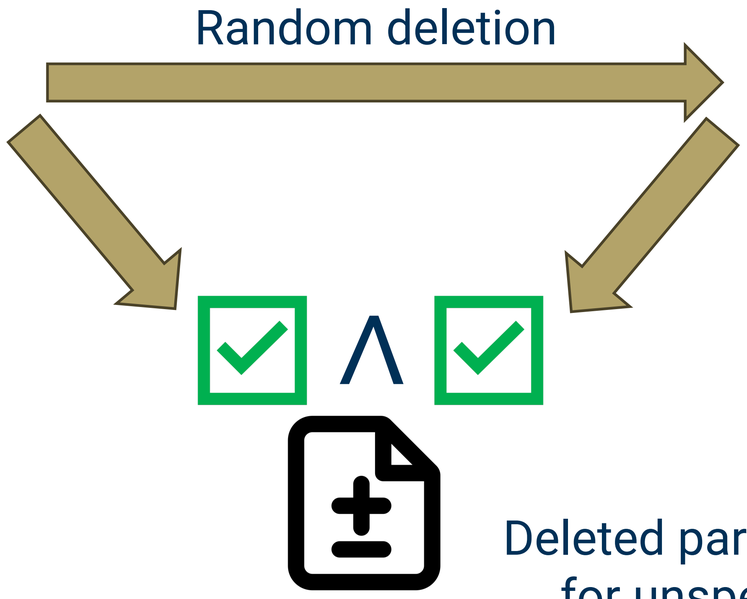
MSG: Move Specification Generator



Integrated with Aptos Move Toolchain
~7.3k LoC Rust code

Specification Coverage - Find Blind Spots

complete **spec verify** **only full** **implementation**
partial **spec verify** **both full and partial** **implementations**



language-agnostic approach
(e.g., by tree-sitter grammar)

Analogous to
code coverage
but for spec!

Deleted parts as feedback
for unspecified parts

Details & Remaining techniques: static analysis, best-effort inlining, etc… in paper

Main Research Questions

- Can LLM understand **a new and less-prominent** PL like Move for FM?
  - less training data
- How good is MSG? Verifiability and quality compared to experts.
- Whether leveraging unique language features is beneficial?
  - **compositional** generation
  - is it better than **monolithic** approach – generate all clauses in one-go?

Specification Verifiability

- Evaluated on Aptos Framework
  - core library (like glibc) for smart contracts on Aptos blockchain.
- Evaluated with OpenAI o3-mini with **5 refining loops** per ClauseGen agent
  - weaker models are also evaluated in ablation study; please check the paper!
- **84%** verifiability (300 out of 357 functions)
  - 876 clauses in total.
  - 14 loop invariants out of 17 loops.

Codebase	LOC	Public	Private	Args	Loops
move-stdlib	82	10	0	15	8
aptos-stdlib	397	58	5	122	1
aptos-framework	2,780	205	79	492	8
TOTAL	3,259	273	84	629	17

	Msg
Fail	0
Success	300
Abstract	57
`ensures`	466
`aborts_if`	252
`modifies`	144
`Loop invariants`	14
All clauses	876
$\frac{\text{Success}}{Total}$	**84%**

Specification Comprehensiveness

- Measure how many generated clauses are matched with expert-written ones.
  - implication checking & subsumption
- MSG generated **139%** of expert-written clauses
  - 82% matches, 57% differs.
- **291 unique** clauses are missed by experts.

	move-stdlib	aptos-stdlib	aptos-framework
Verified Functions	10	48	242
Expert-written Clauses	29	149	442
Generated Clauses	29	143	690
Matched Clauses	29	133	347
Loop Invariants	8	1	5
Unique `ensures`	0	4	126
Unique `aborts_if`	1	3	82
Unique `modifies`	0	3	72
$\frac{\text{Match}}{\text{Total Generated}}$	100%	89.3%	78.5%
$\frac{\text{Total Generated}}{\text{Total Uniques}}$	100%	96%	156.1%
$\frac{\text{Total Uniques}}{\text{Generated}}$	3.4%	7%	40.6%

Detailed method with **implication checking & subsumption** in paper

Better than Generation in One-go

- All-in-one variant: MSG_{AIO} that generated all clauses in one-go.
  - **53.5%** verifiability only vs. **84.0%** for MSG
- Most naïve design: $\text{MSG}_{\text{AIO}-\text{naive}}$ (give only function body)
  - **38.3%** with o3-mini; **18.4%** with GPT-4o-mini
  - Better model for better generation.
  - Techniques are still crucial for correct generations with current states of LLM.

Variant	A	P	S	I	V
MSG	✓	✓	✓	✗	84.0%
MSG^{-}	✓	✗	✓	✗	70.9%
$\text{MSG}_{\text{inline}}$	✓	✓	✓	✓	83.8%
MSG_{AIO}	✗	✓	✓	✗	53.5%
$\text{MSG}^{-}_{\text{AIO}}$	✗	✗	✓	✗	41.1%
$\text{MSG}_{\text{AIO-inline}}$	✗	✓	✓	✓	54.1%
$\text{MSG}_{\text{AIO-naive}}$	✗	✗	✗	✗	38.3%

A	Agentic design
P	Prover feedback
S	System Prompt & Static Analysis
I	Function Inlining

Detailed ablation study in the paper!

Takeaway

- LLMs can reason emerging PL like Move very well with less presence.
- MSG generates with high verifiability with expert-level quality.
  - 84% verifiability with 291 unique clauses missed by experts.
- Compositional Generation (Ensembling) by leveraging PL features.
  - addressing smaller independent problems and merge the results.
- Beyond Move. Methods are transferable to other PL.
  - compositional generation & specification coverage

sslab-gatech/MSG

paper

code

Thanks!

Poster session later!

Sui Research Gift from Sui

SSLab @GeorgiaTech

University of Waterloo

Georgia Tech