

From Answers to Contracts

LLM Agents as Untrusted Program Components

Haonan Li · University of California, Riverside

Why does static analysis need LLMs?

C provides limited abstractions for user intent.

From intent to implementation,
information is lost.

LLMs try to recover it from names, docs, and context.

Case 1: An array search.

```
int find_in_array(int *arr, int len, int target) {  
    for (int i = 0; i < len; ++i) {  
        if (arr[i] == target)  
            return i;  
    }  
    return -1;  
}
```

Could `arr[i]` go out of bounds?

To say “no bug”, we must add a premise.

A possible LLM answer

“No out-of-bounds access.”

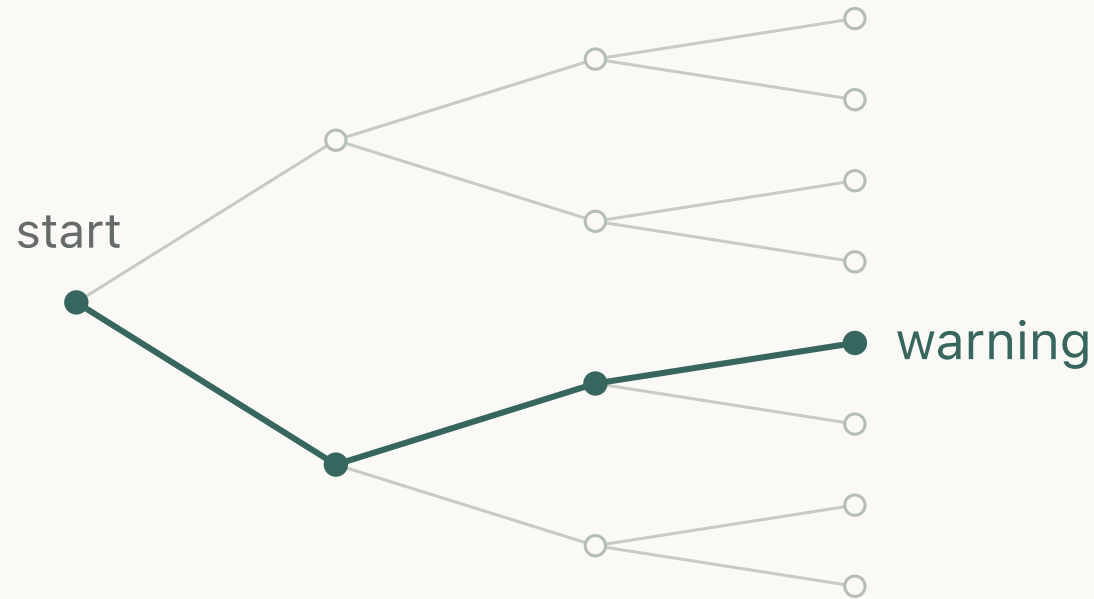
Guessed from names, docs,
conventions, and context.

The added premise

`arr` provides at least
`len` readable elements.

Propose-and-verify still needs a whole-program proof.
Every relevant call must provide at least `len` readable elements.

Case 0: Hard to find. Easy to check.



One proposed path through the branches.

One witness

The LLM proposes a path or a concrete input.

A fixed claim

Verify that the path is feasible and reaches the warning.

Propose and prove works here.

Case 2: The specification is missing.

```
uint32_t r = x + delta;
```

Hashing

Wraparound may be deliberate.

Allocation size

Wraparound may cause an undersized allocation.

Intermediate arithmetic

Wraparound may have no relevant downstream effect.

Whether this is dangerous depends on the intended specification.

Same implementation. Different verdicts.

User intent

Cyclic sequence number

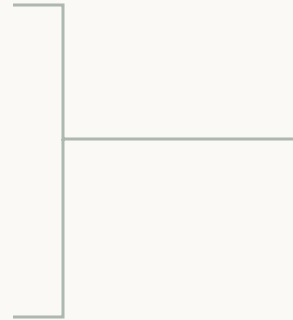
Allow wraparound.

Not a bug

Exact byte count

Preserve the full sum.

Bug



Same implementation

```
uint32_t r = x + delta;
```

Wraparound to zero

With a spec, check mechanically.

We lack one. The LLM must infer user intent.

From answers to contracts.

Bare answer

“SAFE.”

Assumption added by the LLM

“This value is intended to wrap.”

Which assumptions did the LLM add?

What is a contract?

A claim and the assumptions
the LLM introduced.

Claim

What conclusion
is being asserted?

Imported assumptions

What new premises
did the LLM introduce?

An inferred assumption is still a hypothesis.

Three verification cases

Case	LLM contribution	Verification
Case 0 · Search	Find constraints and a witness for a fixed claim.	Cheap to check. Classic propose-and-verify.
Case 1 · Analysis cost	<code>find_in_array</code> : the array-extent constraint.	Expensive to establish. Prove every relevant call.
Case 2 · Specification	Recover intended behavior or a safety policy.	Needs an independent specification.

Checking a witness and proving a global premise are different tasks.

Proposed direction 1: find contradictions.

LLM assumption

“seq is intended to wrap.”

Check its uses

Indexes and allocation sizes

Ordering assumptions

Derivable conflict

Show the conflicting use
and a reachable state.

No conflict found does not establish intent.

A conflicting assumption

```
uint32_t old_seq = seq;  
uint32_t new_seq = ++seq;  
assert(new_seq > old_seq);
```

At rollover: `old_seq = UINT32_MAX, new_seq = 0.`

Still unresolved

No conflict found
in the checked scope.

Proposed direction 2: ask for missing intent.

Program

```
uint32_t seq;  
seq++;
```

What remains unknown

Is wrapping part of the design?

Ask for the missing semantic choice

“Is this counter
intended to wrap?”

Wrapping counter

Allow modulo- 2^{32} updates.

Monotonic counter

Prevent rollover; preserve increasing values.

Use the human-supplied intent to refine the contract and recheck.

Verify what you can.
Make the rest explicit.

Check assumptions against the program.
Ask people for the information still missing.

Thank you.