

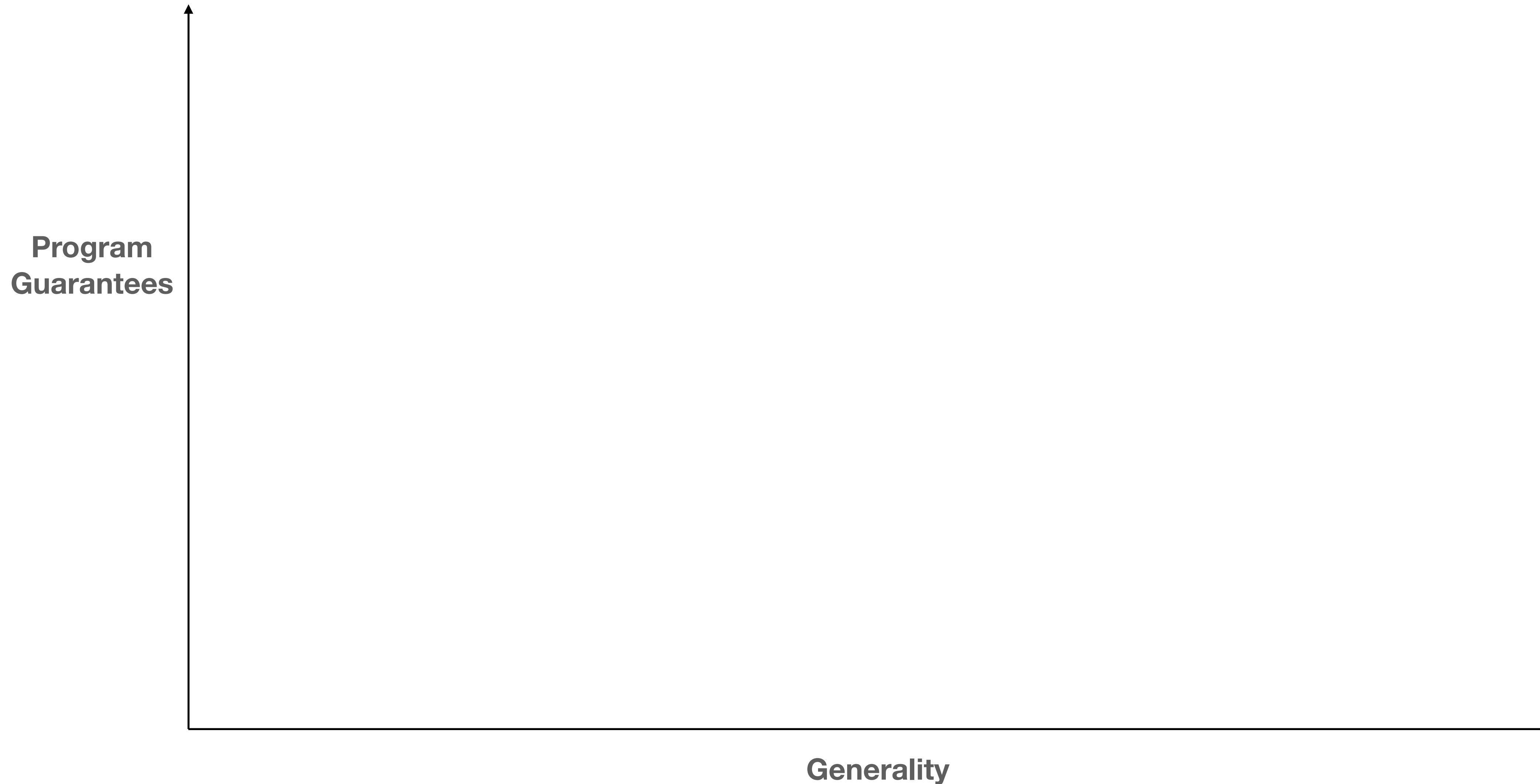
LLM-Assisted Synthesis of High-Assurance C Programs

Prasita Mukherjee, Minghai Lu, Benjamin Delaware

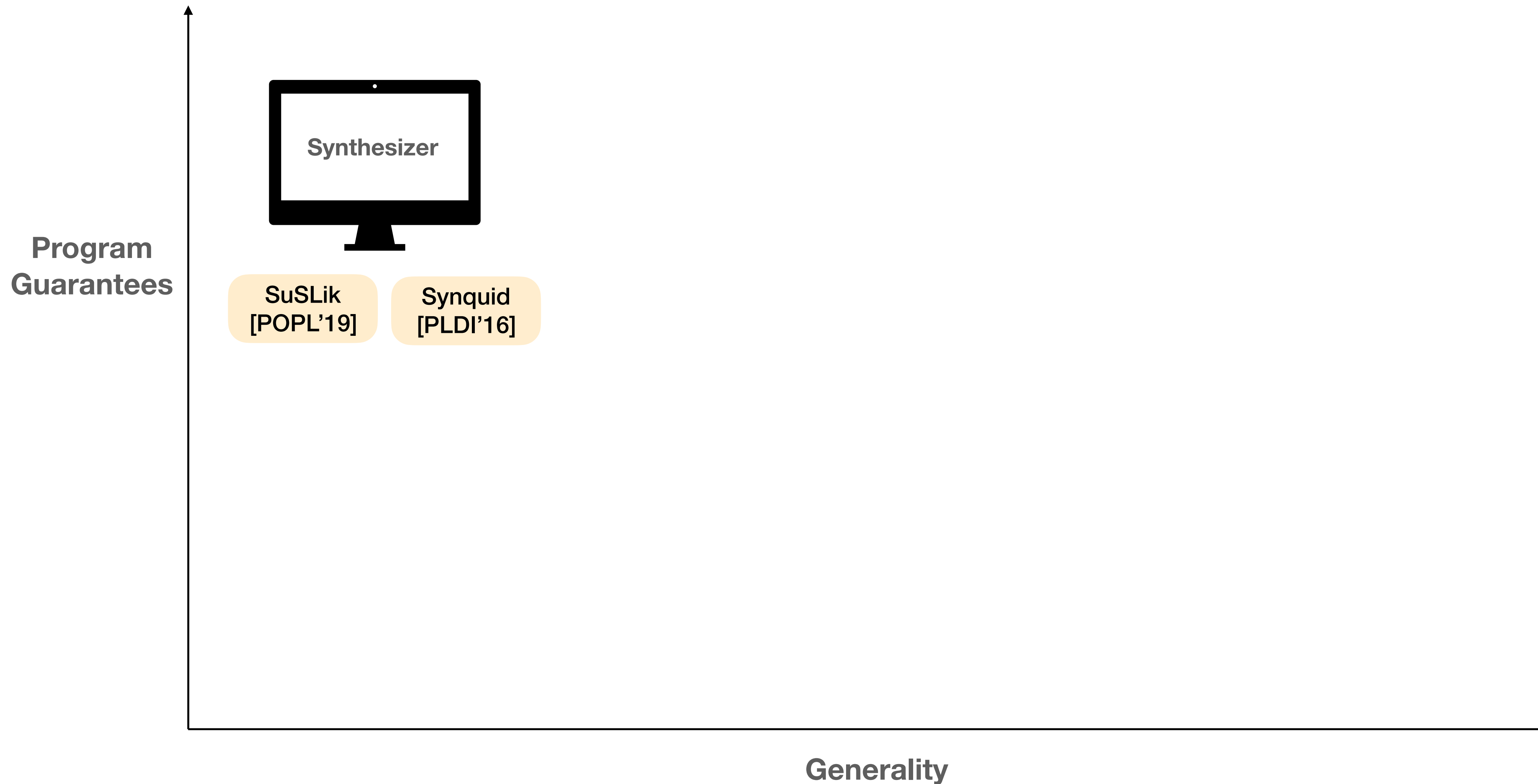


Goal of Program Synthesis : Automatically generate a **correct** program from an **arbitrary** specification of its behavior

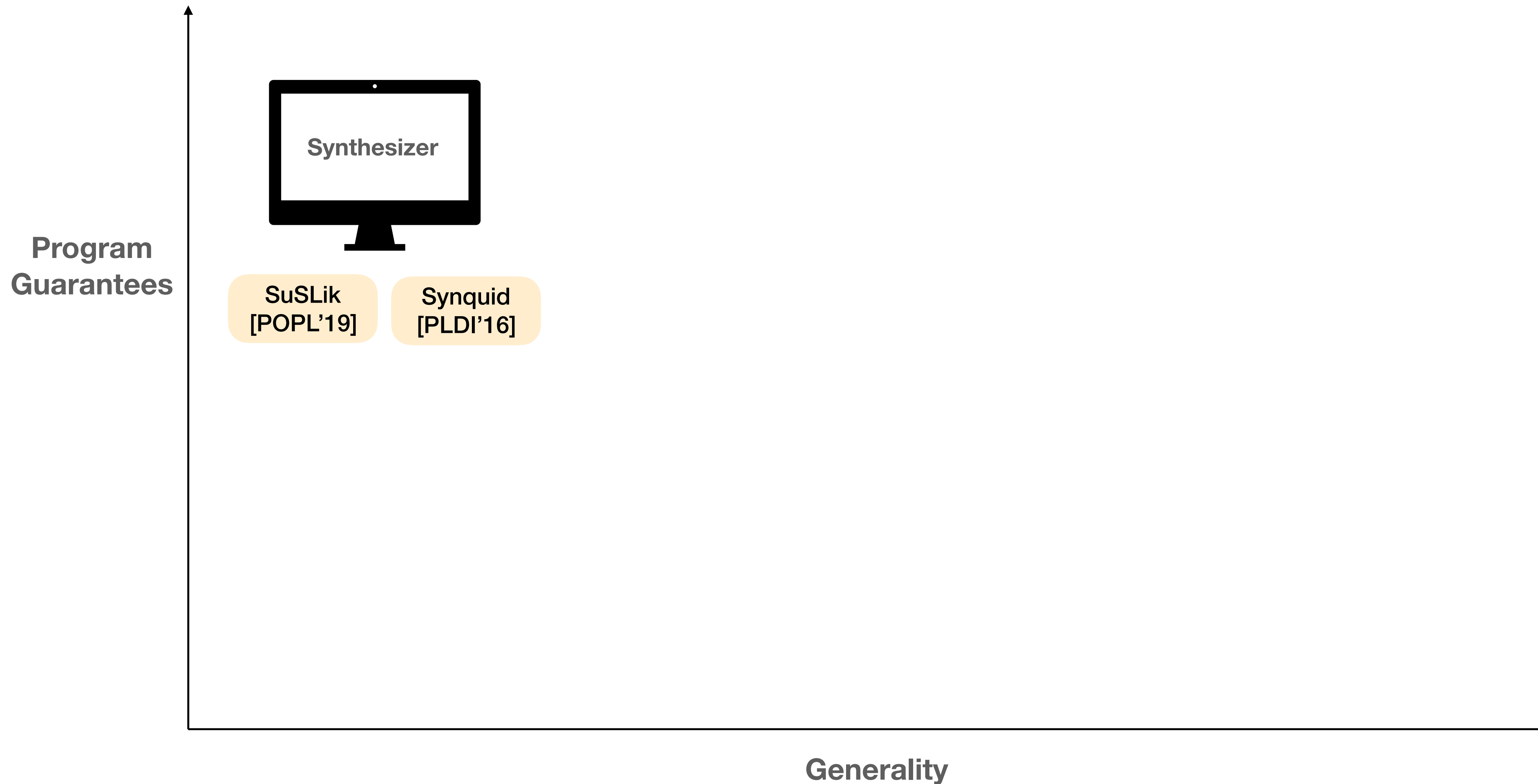
Goal of Program Synthesis : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



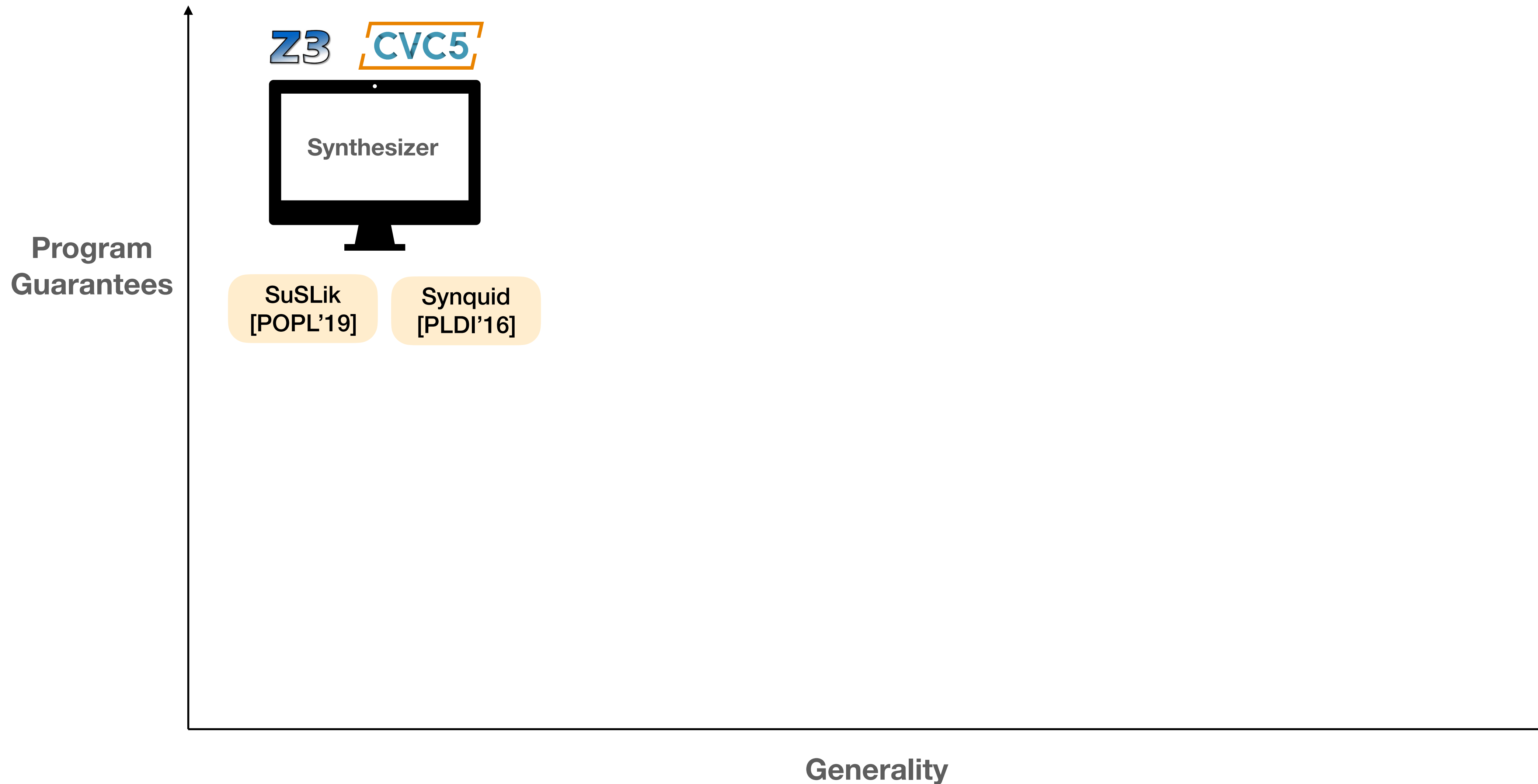
Goal of Program Synthesis : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



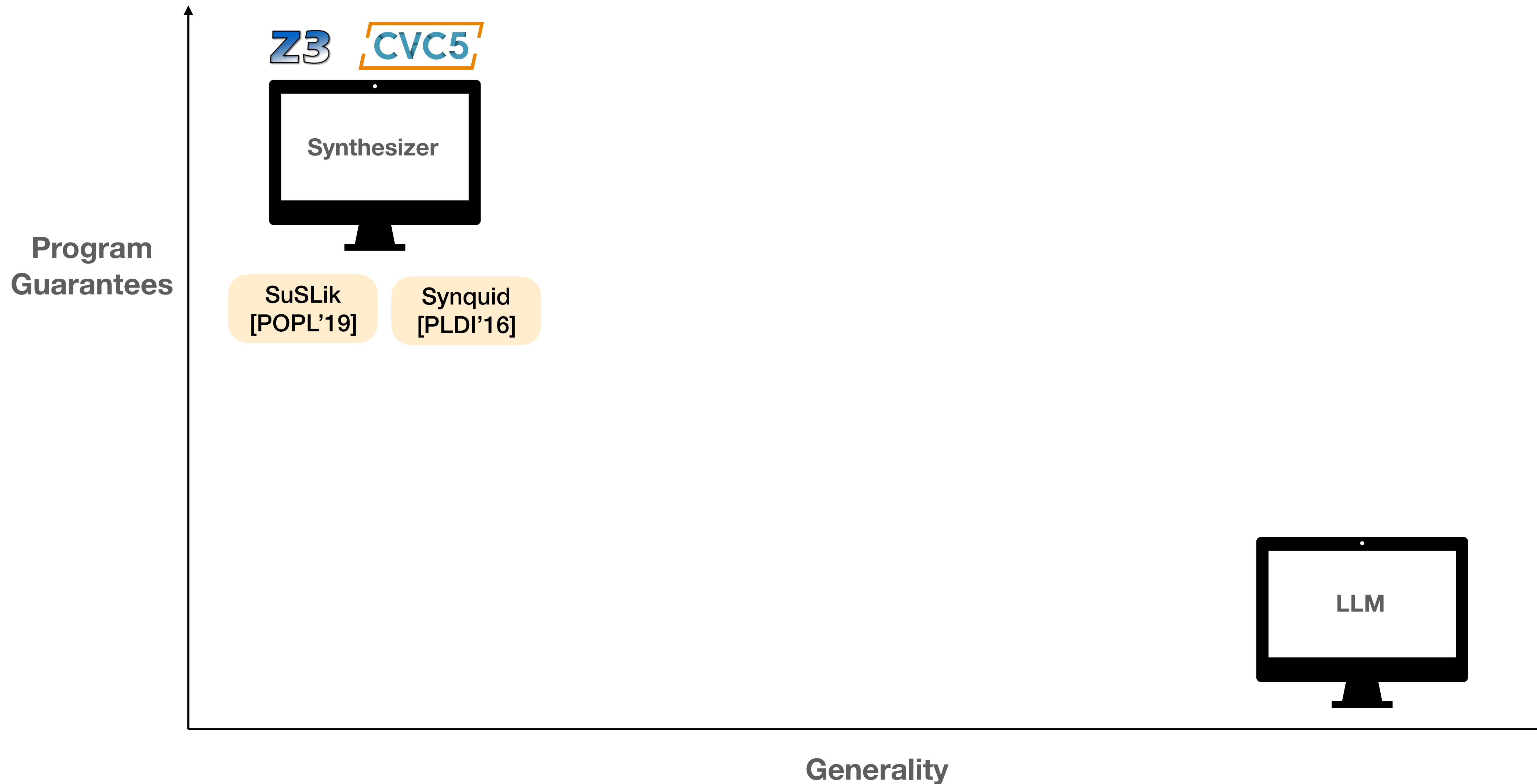
Goal of Program Synthesis : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



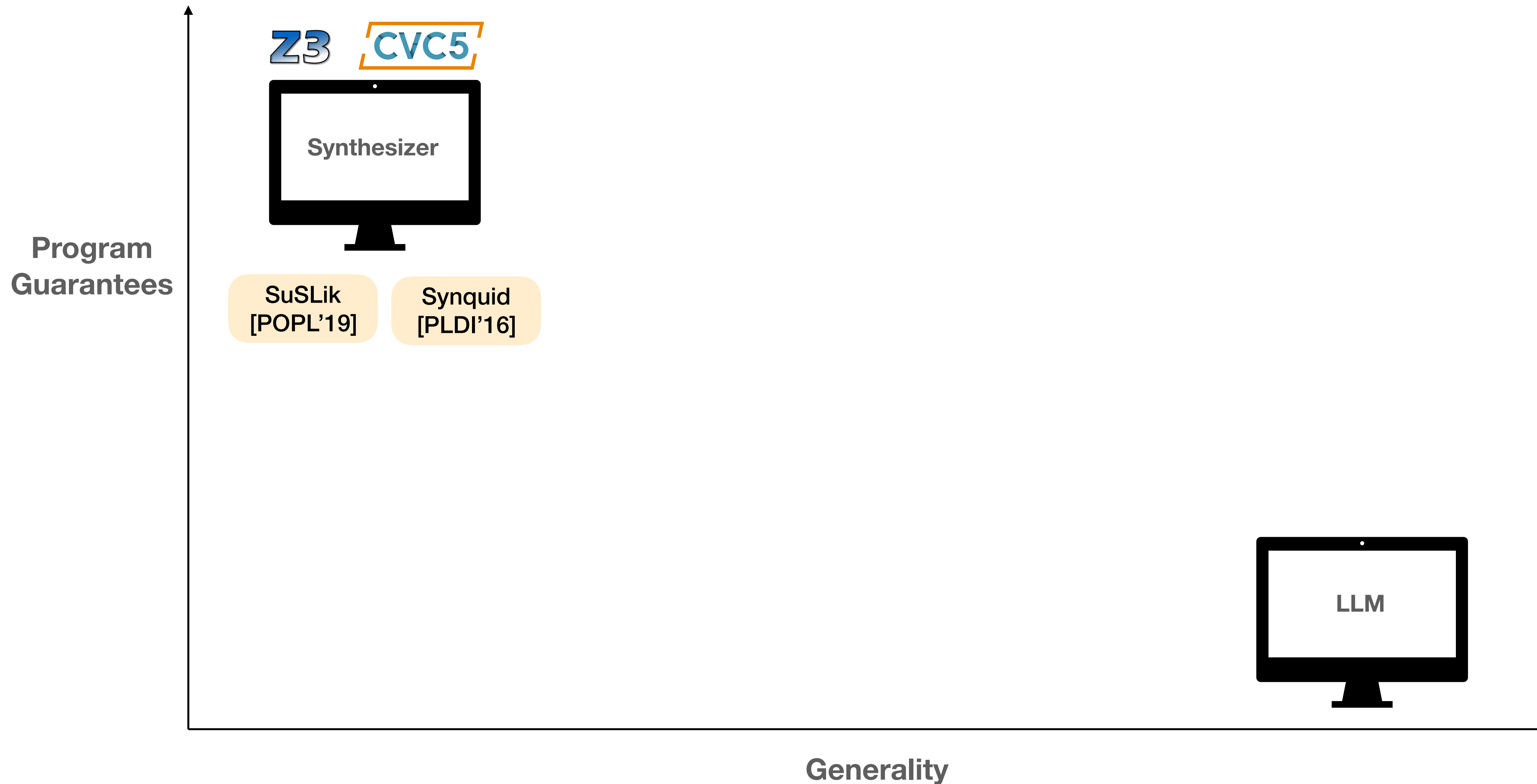
Goal of Program Synthesis : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



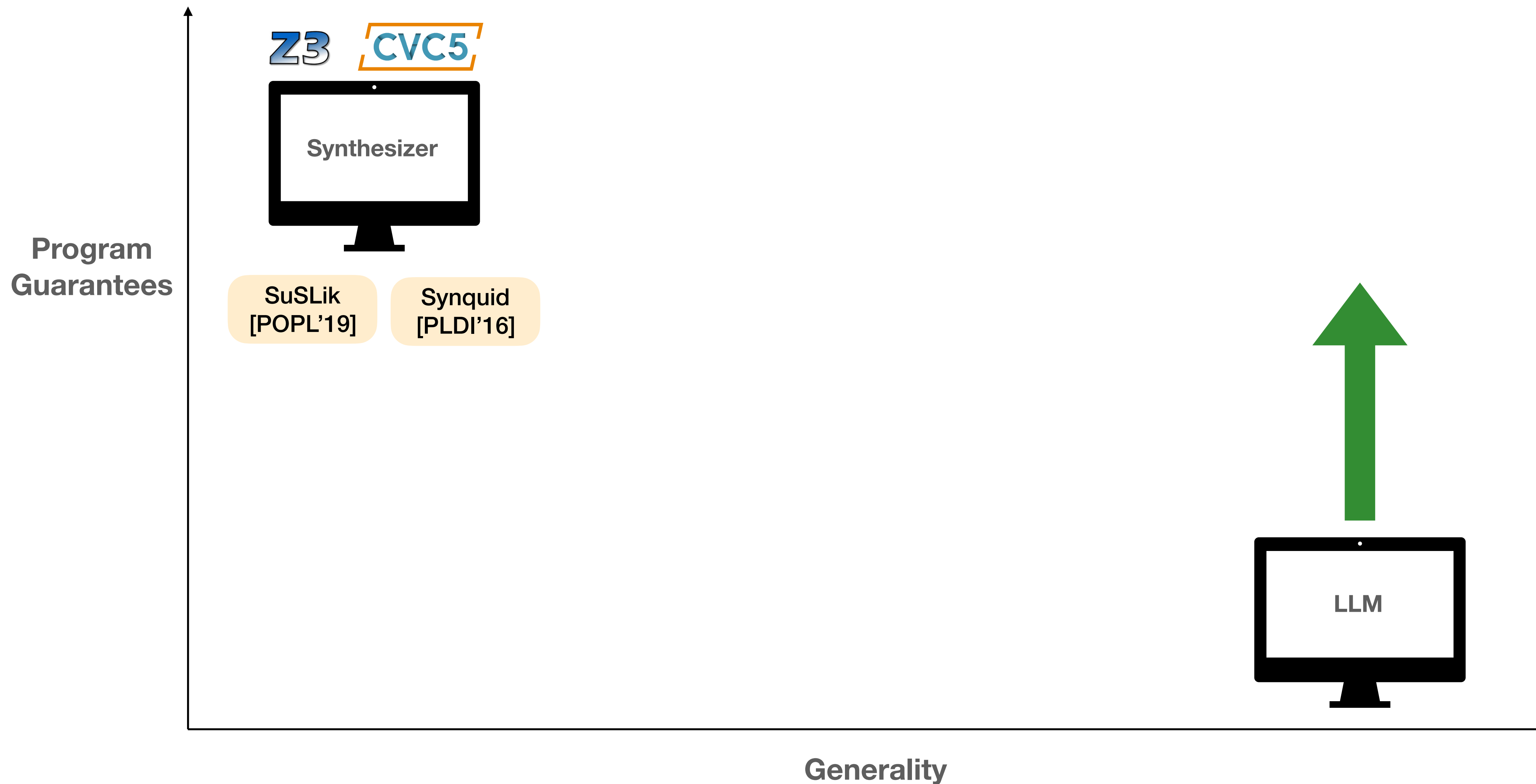
Goal of Program Synthesis : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



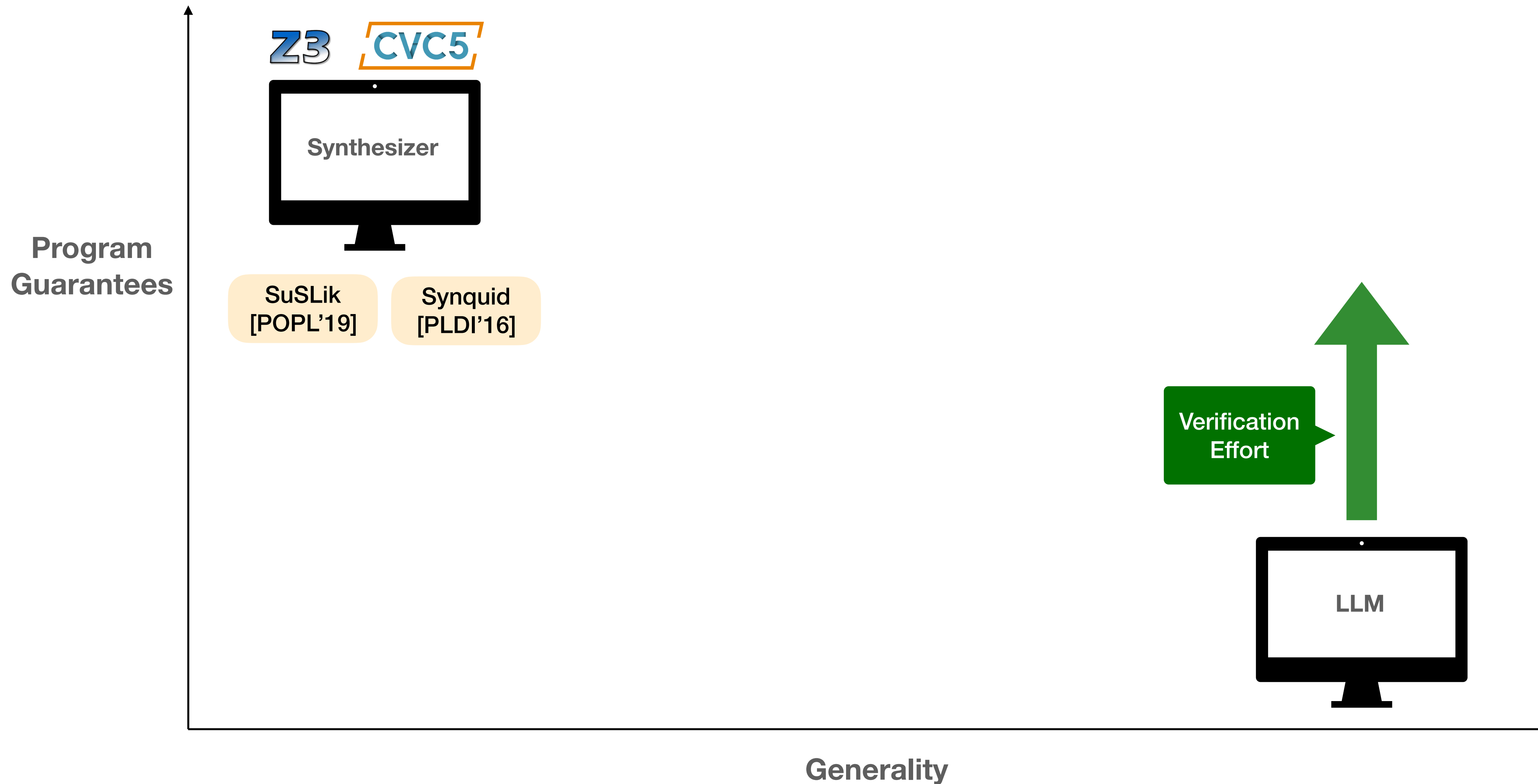
Goal of Program Synthesis : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



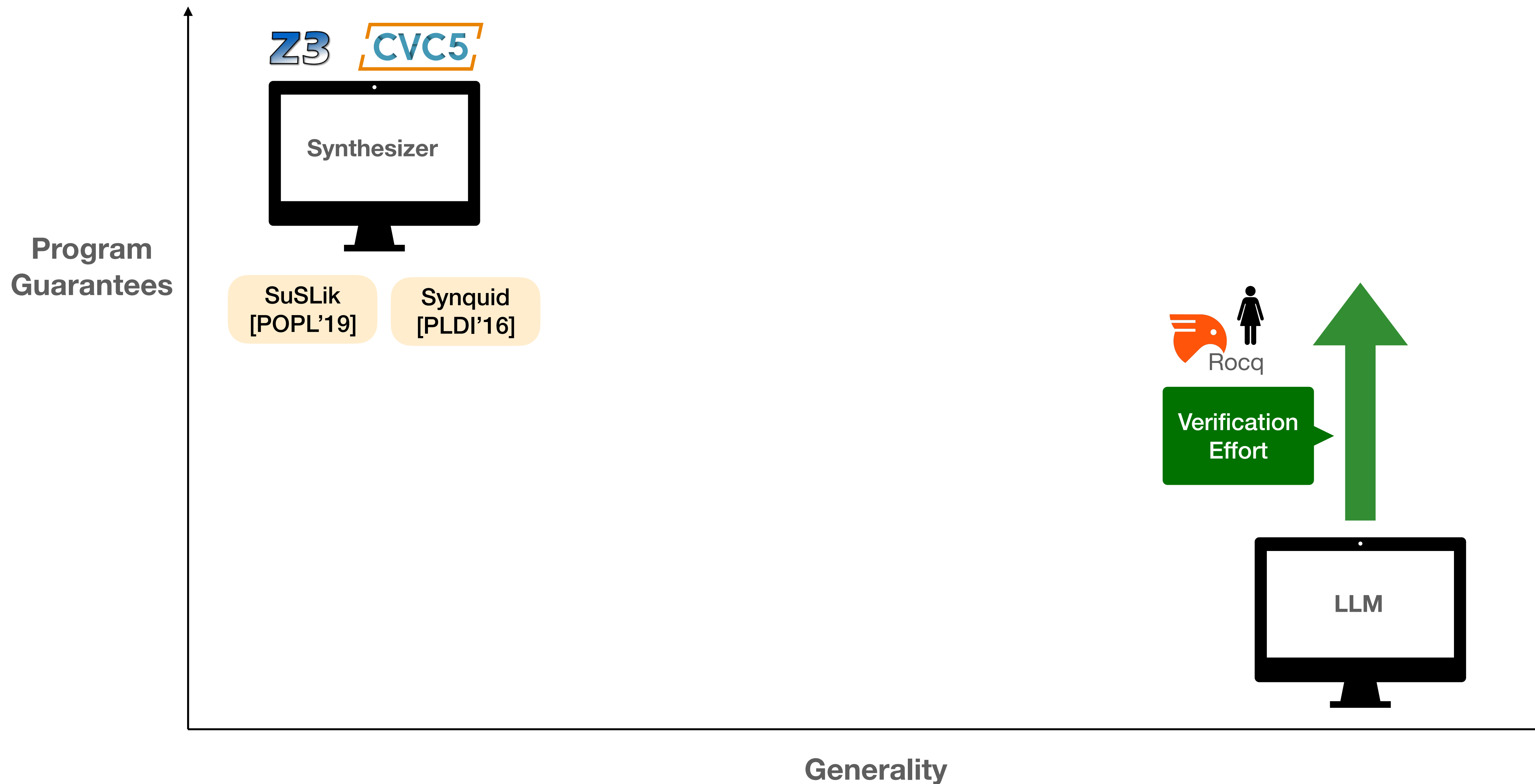
Goal of Program Synthesis : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



Goal of Program Synthesis : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



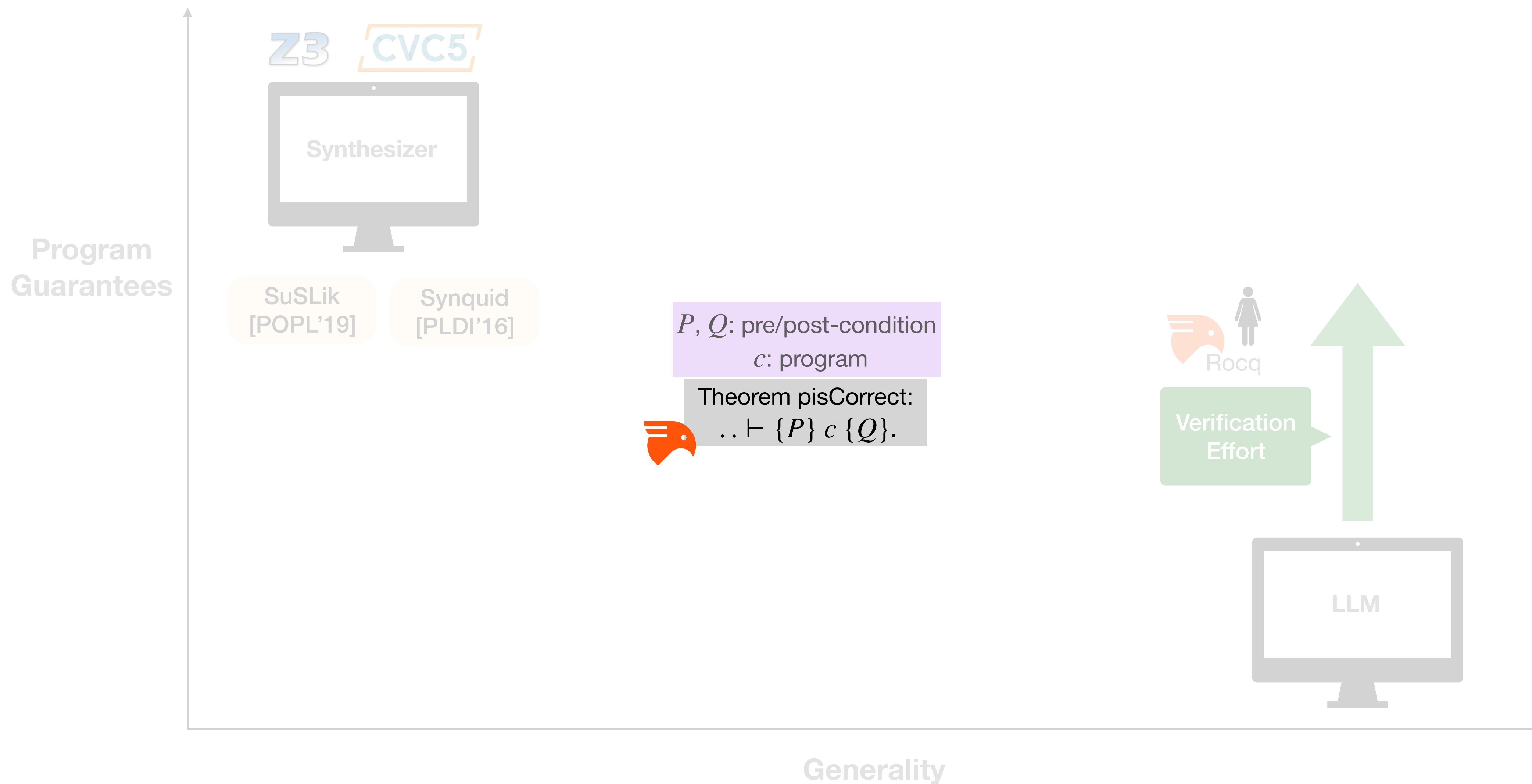
Goal of Program Synthesis : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



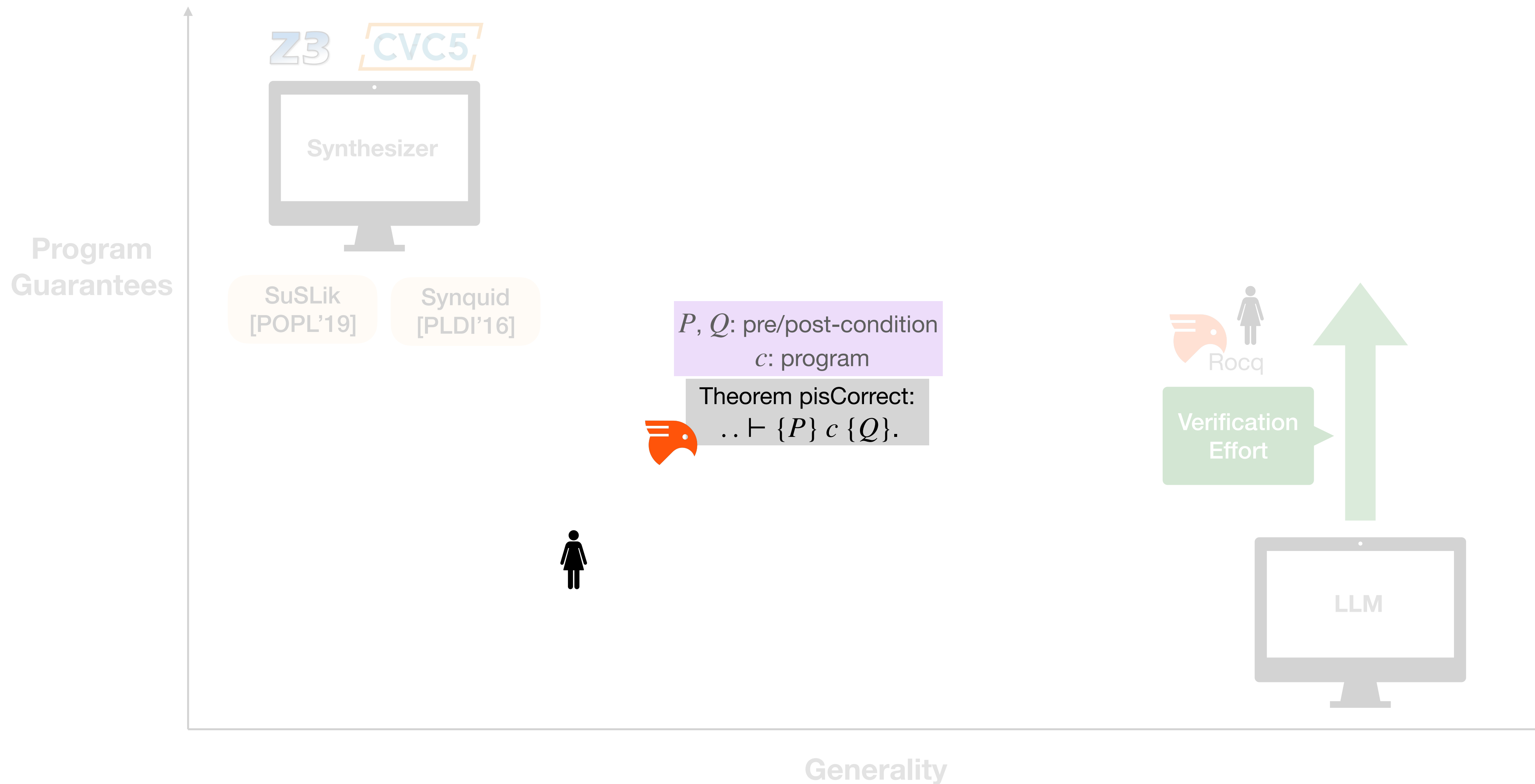
Goal of Program Synthesis : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



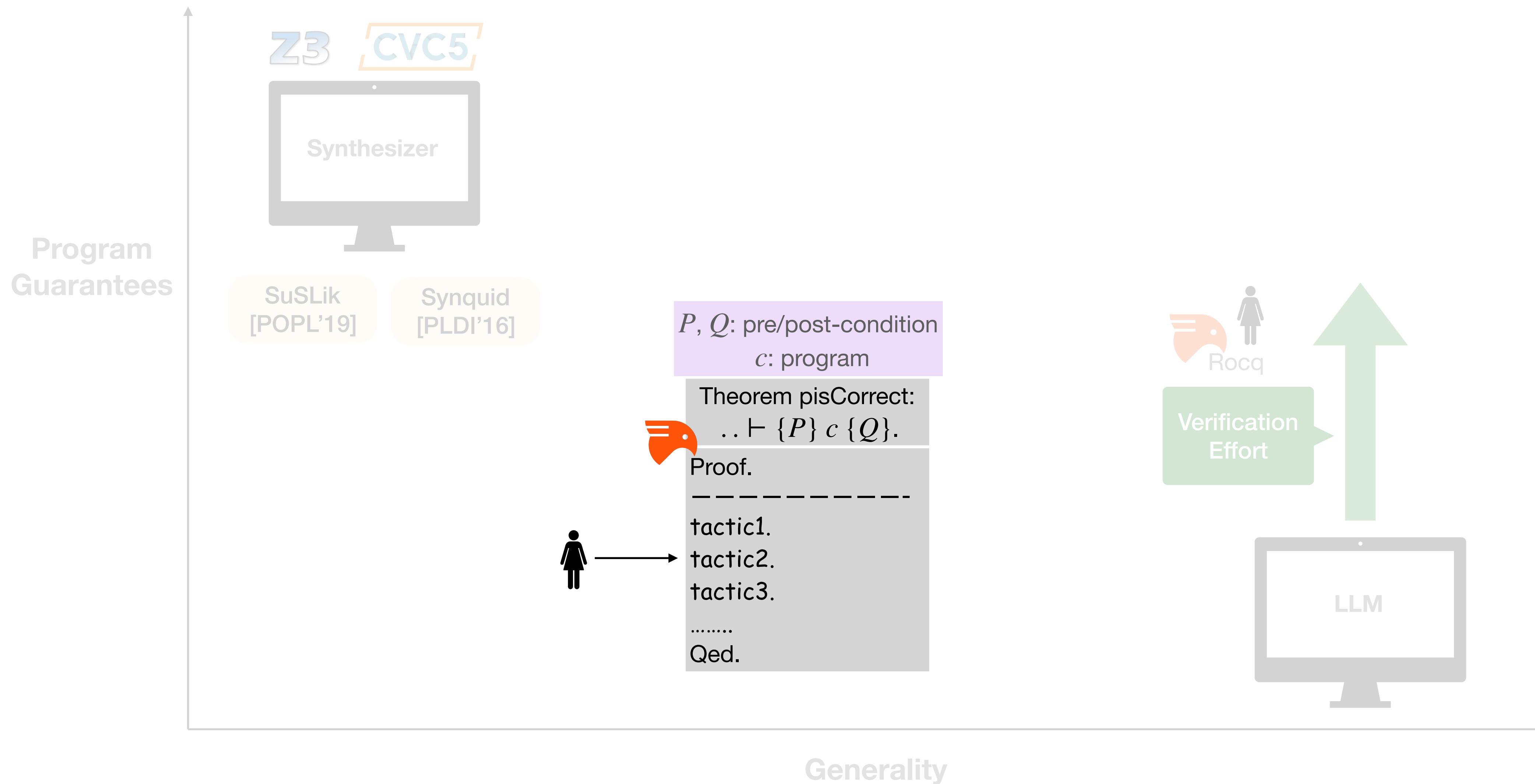
Goal of Program Synthesis : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



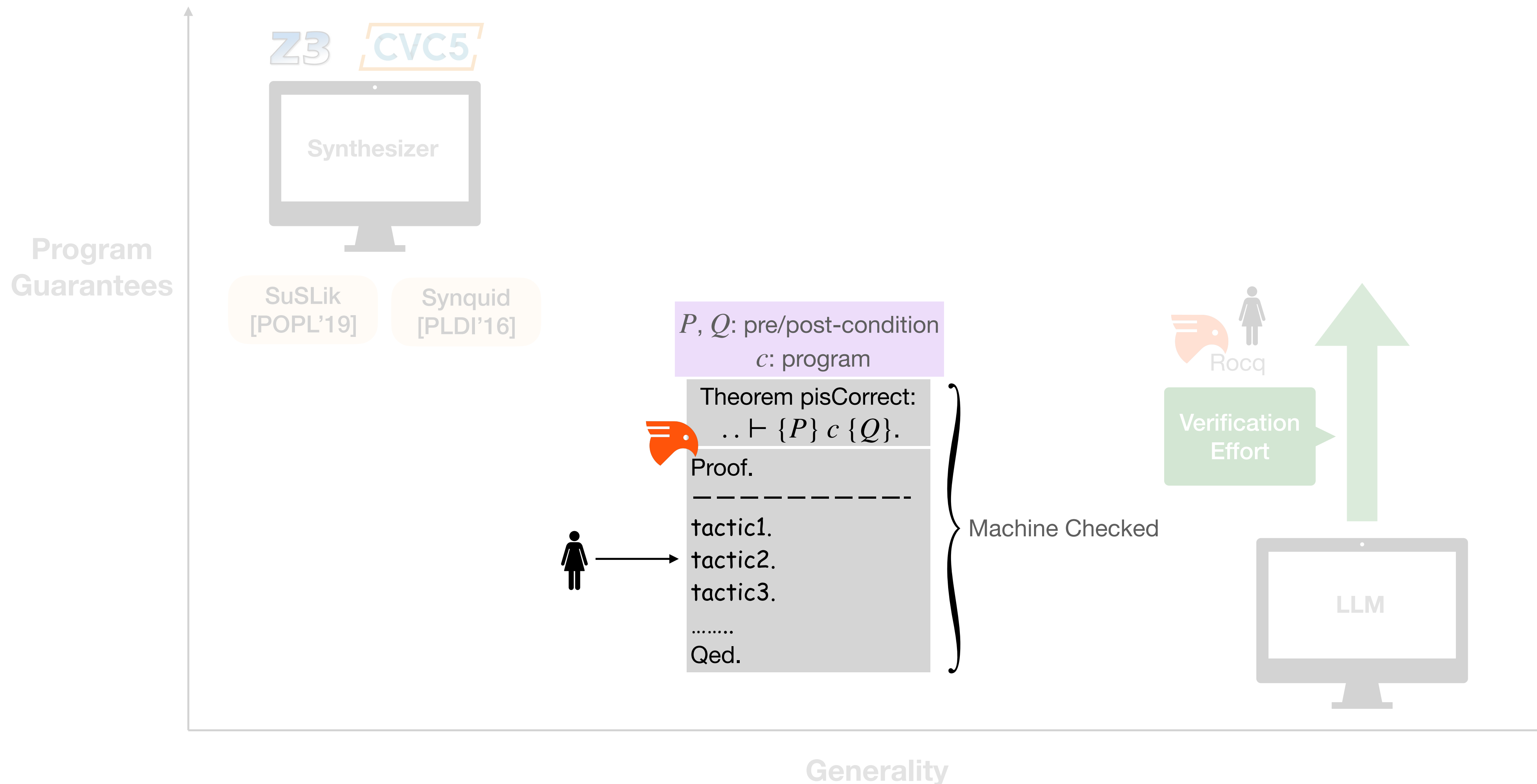
Goal of Program Synthesis : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



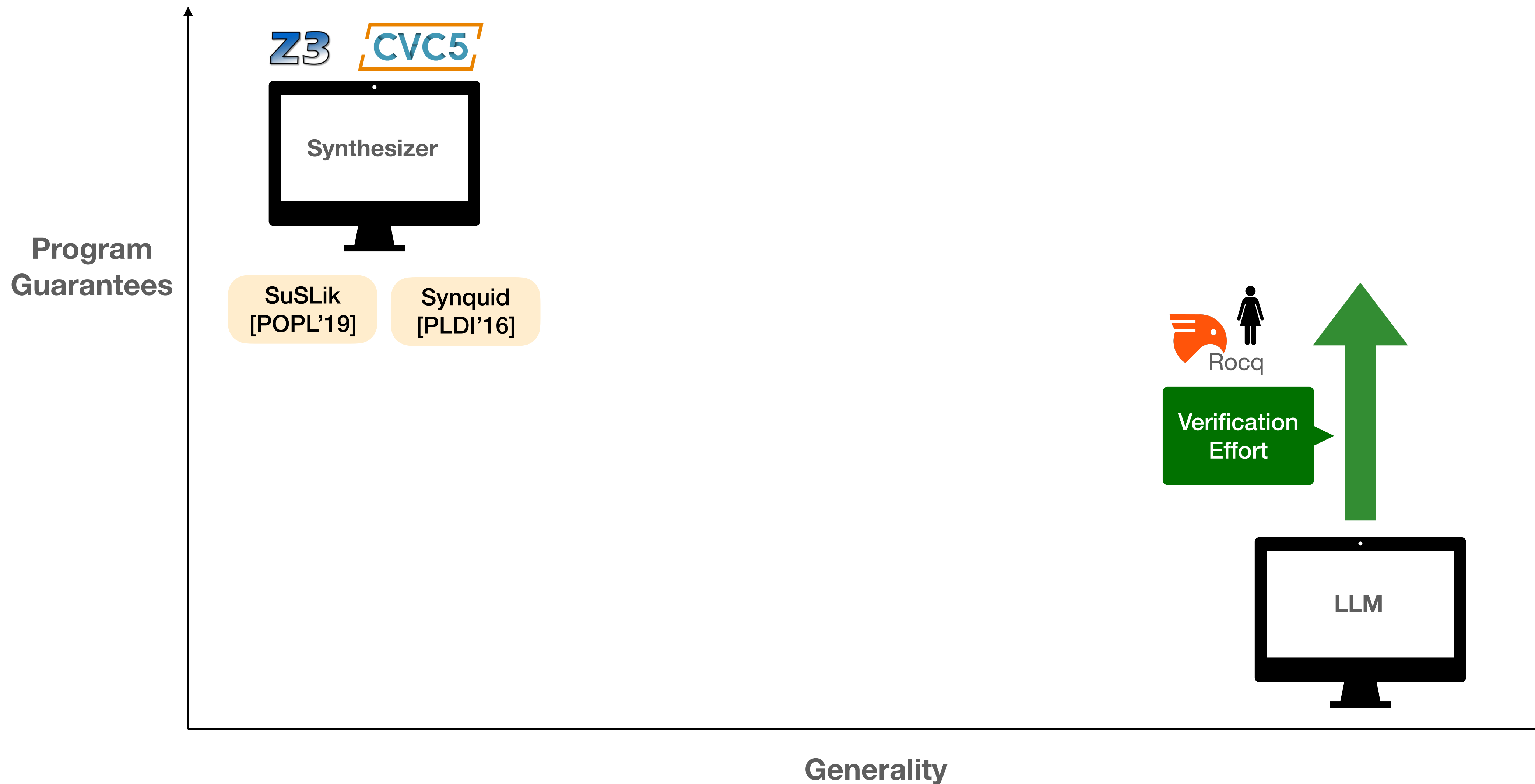
Goal of Program Synthesis : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



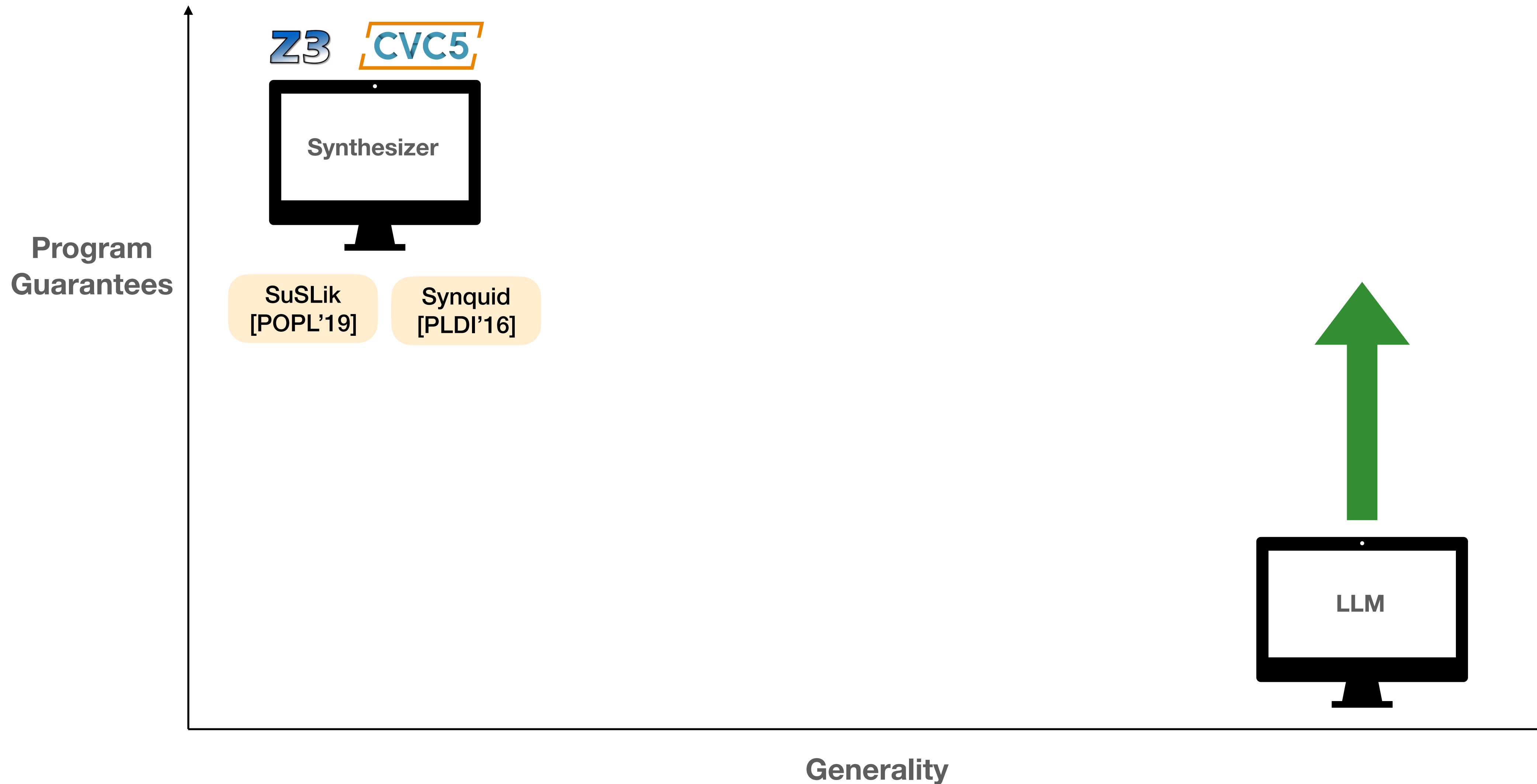
Goal of Program Synthesis : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



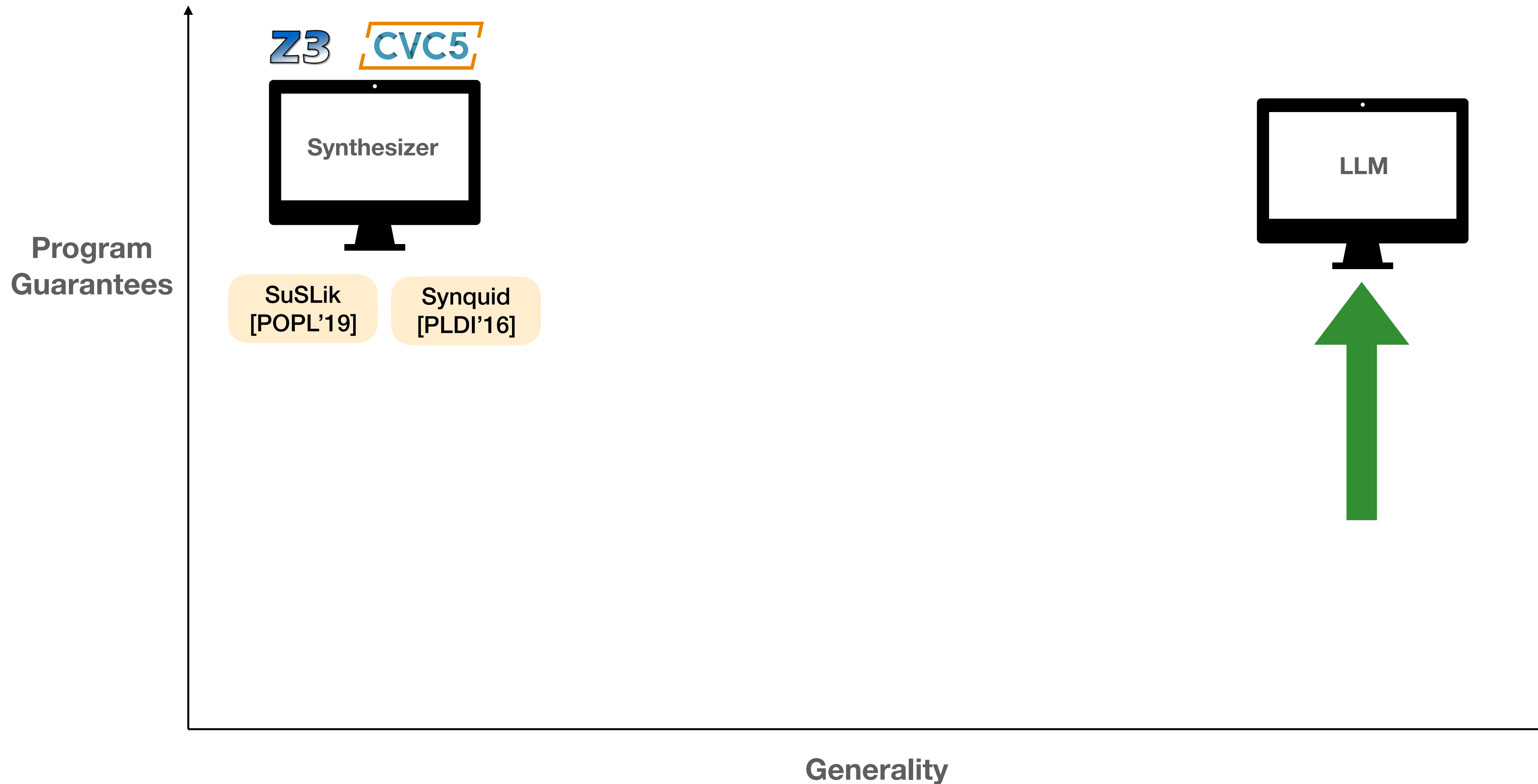
Goal of Program Synthesis : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



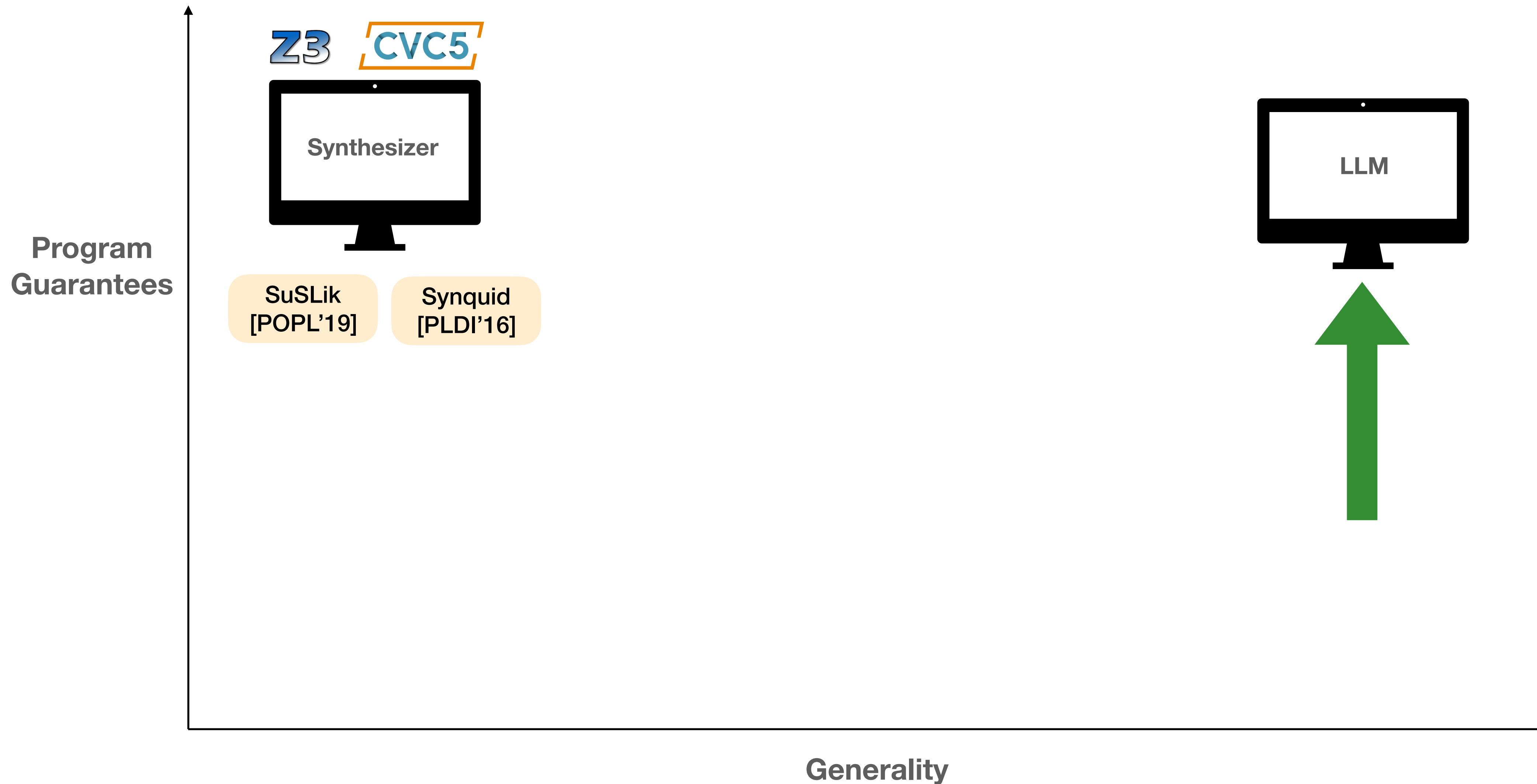
Goal of Program Synthesis : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



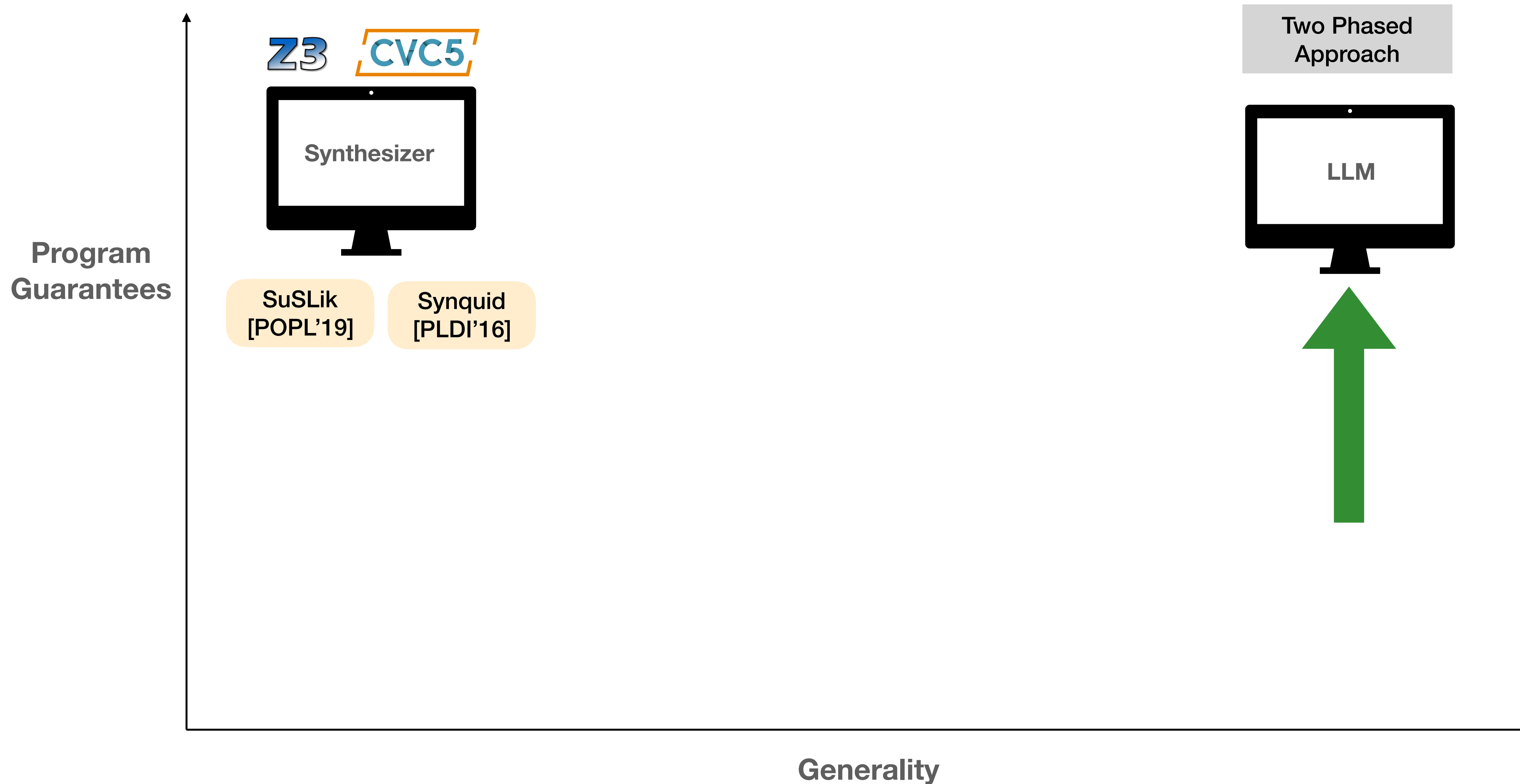
Goal of Program Synthesis : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



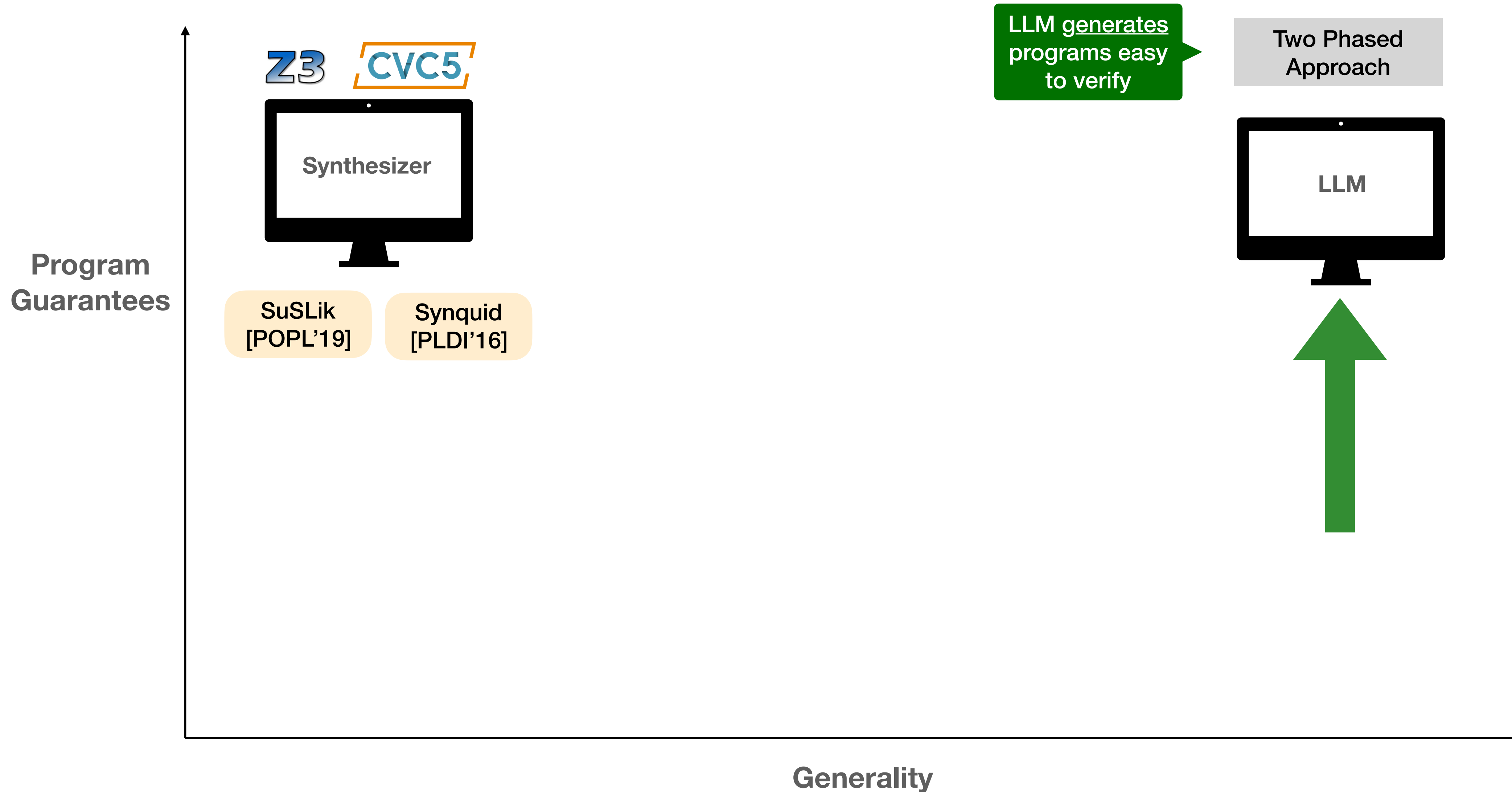
Goal of Program SYNVER : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



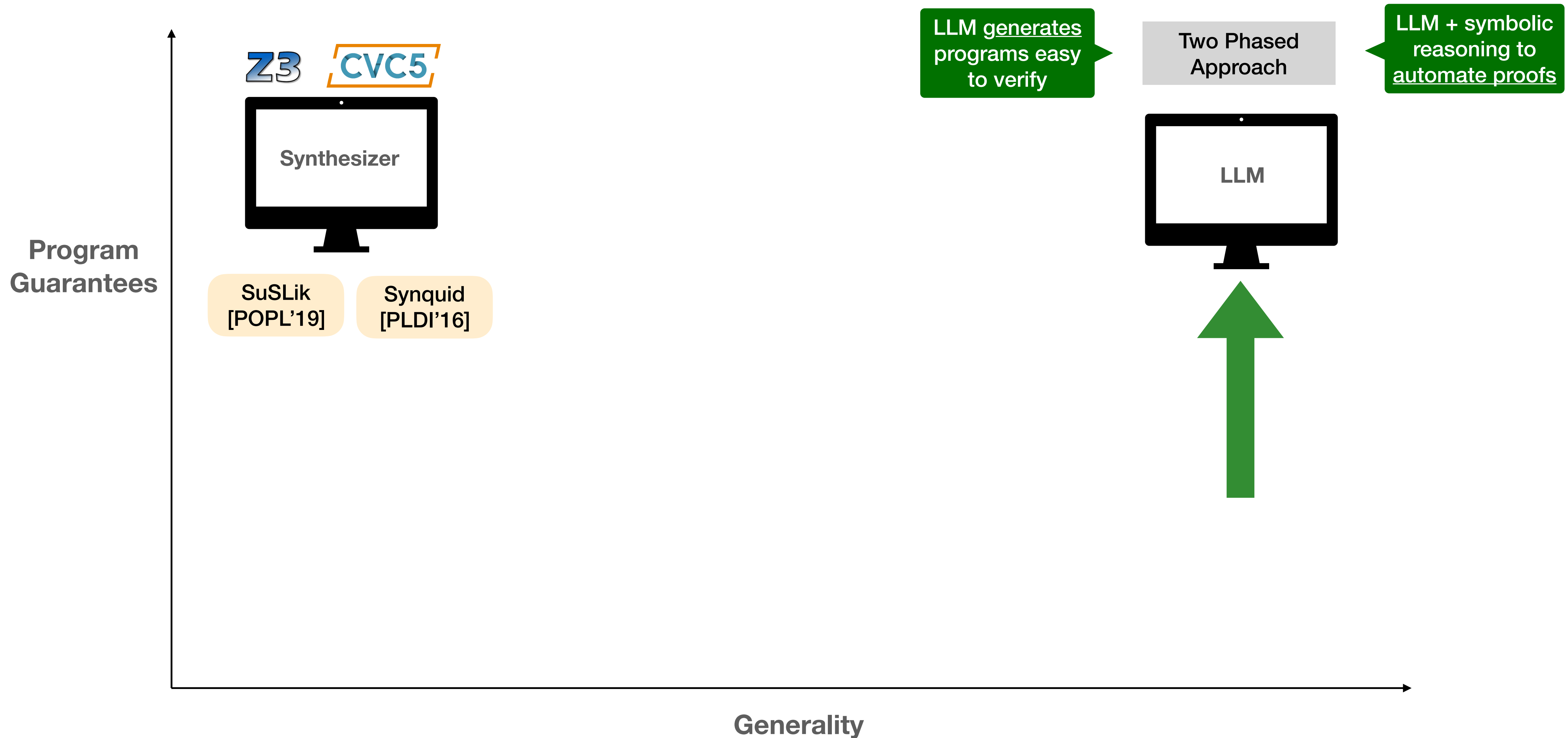
Goal of Program SYNVER : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



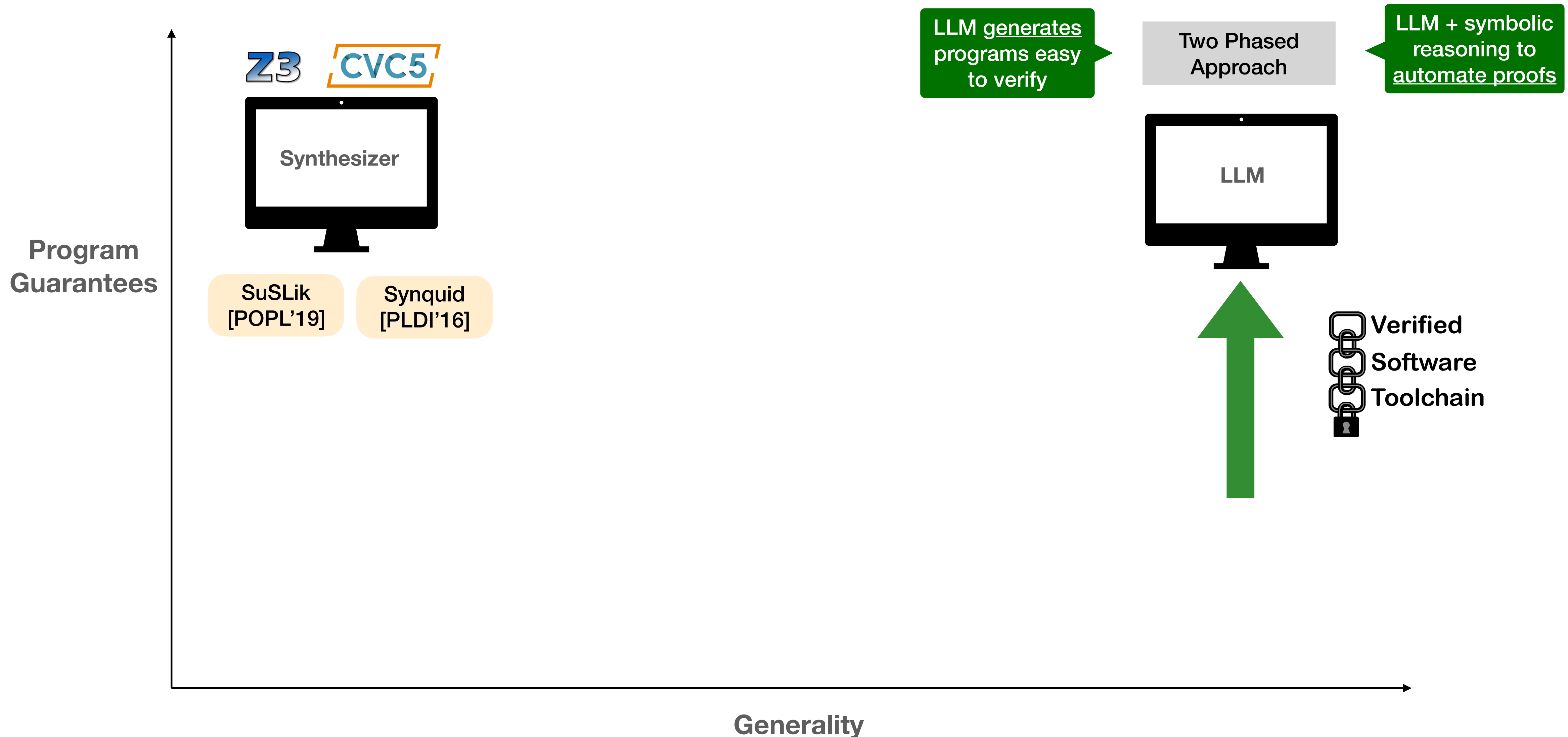
Goal of Program SYNVER : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



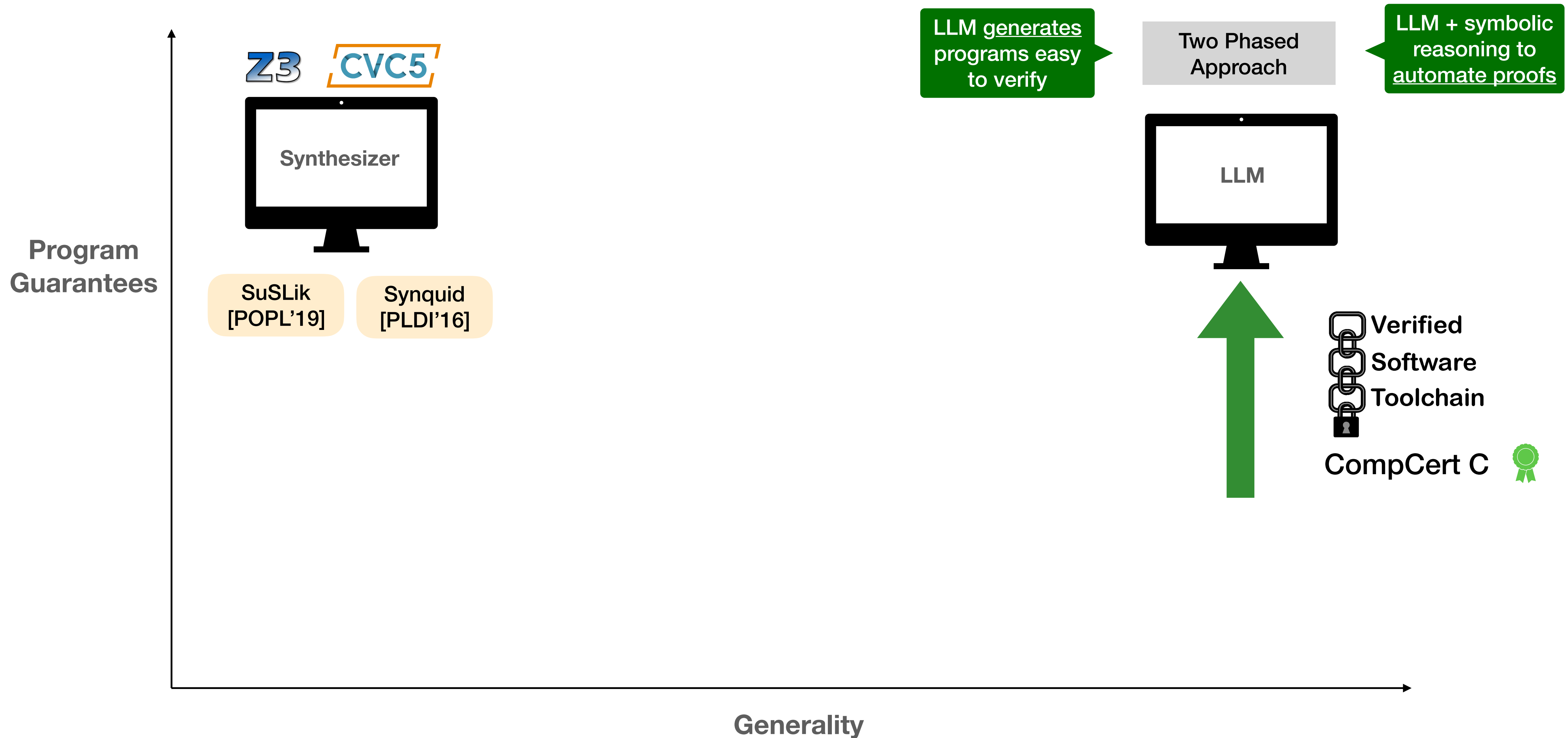
Goal of Program SYNVER : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



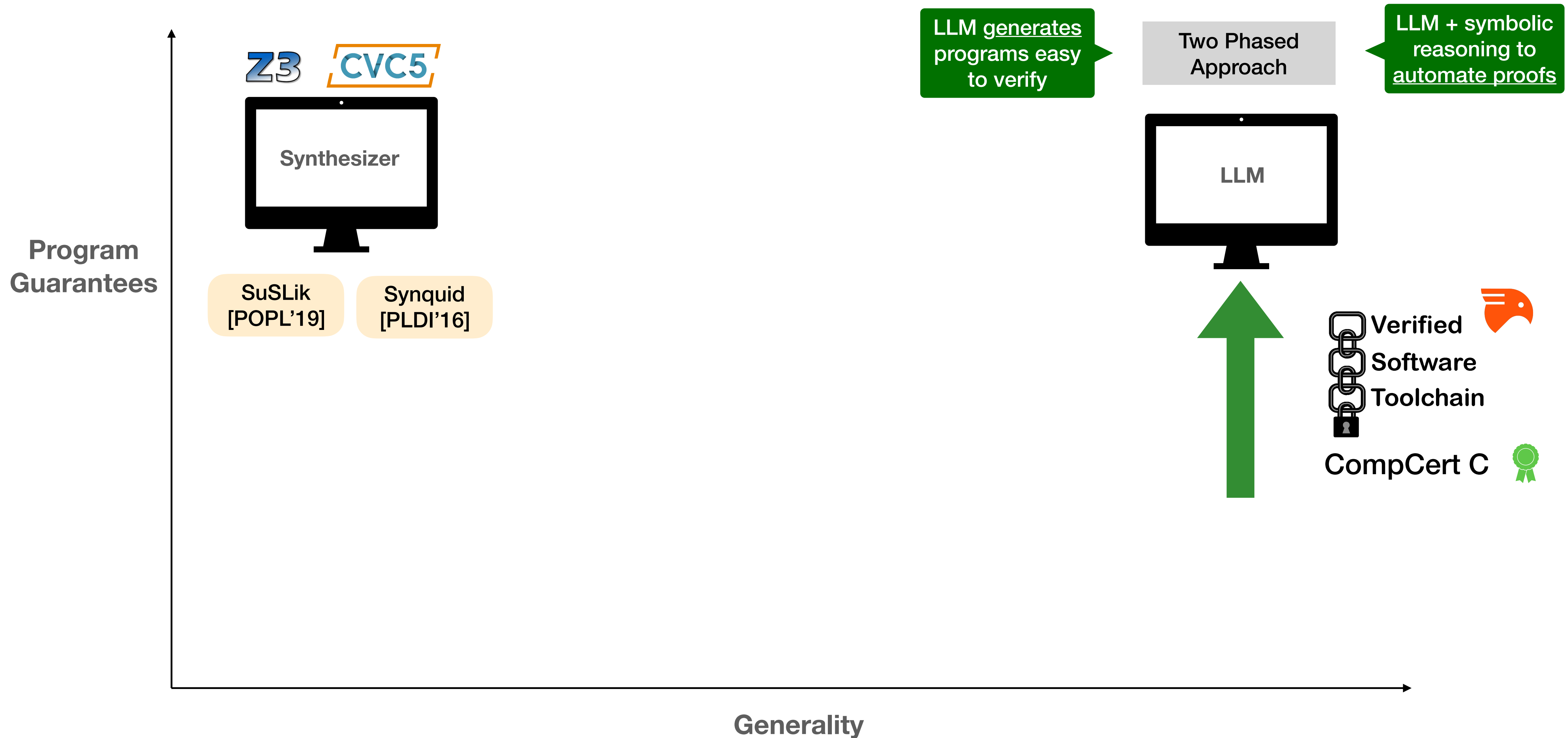
Goal of Program SYNVER : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



Goal of Program SYNVER : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



Goal of Program SYNVER : Automatically generate a **correct** program from an **arbitrary** specification of its behavior



Phase 1: Generate

Phase 1: Generate

- Generate programs that are easier to automatically verify

Phase 1: Generate

- Generate programs that are easier to automatically verify
 - Programs use recursion instead of loops

Phase 1: Generate

- Generate programs that are easier to automatically verify
- Programs use recursion instead of loops

```
{h1 ↦ l1 * h2 ↦ l2}
struct sll *append (struct sll *h1,
                    struct sll *h2) {
    if (h1 == NULL) {
        return h2;
    }
    h1->next = append(h1->next, h2);
    return h1;
}
{h ↦ (l1 ++ l2)}
```



Phase 1: Generate

- Generate programs that are easier to automatically verify
 - Programs use recursion instead of loops
 - Programs only call helper functions whose specifications are available


```
{h1 ↦ l1 * h2 ↦ l2}  
struct sll *append (struct sll *h1,  
                    struct sll *h2) {  
    if (h1 == NULL) {  
        return h2;  
    }  
    h1->next = append(h1->next, h2);  
    return h1;  
}  
{h ↦ (l1 ++ l2)}
```




Phase 1: Generate

- Generate programs that are easier to automatically verify
 - Programs use recursion instead of loops
 - Programs only call helper functions whose specifications are available

```
{h1 ↦ l1 * h2 ↦ l2}
struct sll *append (struct sll *h1,
                    struct sll *h2) {
    if (h1 == NULL) {
        return h2;
    }
    h1->next = append(h1->next, h2);
    return h1;
}
{h ↦ (l1 ++ l2)}
```



```
void add (int *s) {
    *s = *s + 1;
}
{x ↦ a * y ↦ b}
void swap (int *x, int *y) {
    int a = *x, b = *y;
    *x = b;
    *y = a;
    add(x);
    add(y);
}
{x ↦ (b + 1) * y ↦ (a + 1)}
```



Phase 1: Generate

- Generate programs that are easier to automatically verify
 - Programs use recursion instead of loops
 - Programs only call helper functions whose specifications are available

Phase 1: Generate

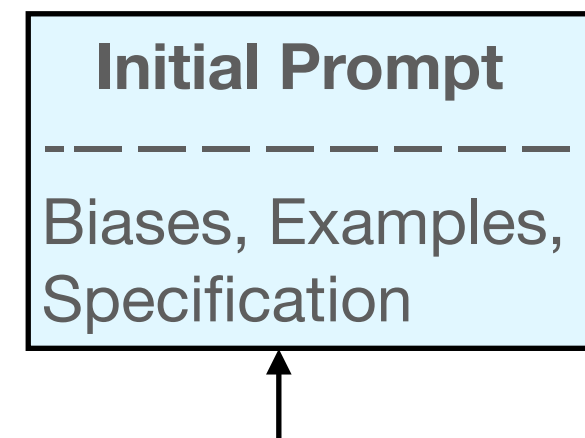
- Generate programs that are easier to automatically verify
 - Programs use recursion instead of loops
 - Programs only call helper functions whose specifications are available

Initial Prompt

Biases, Examples,
Specification

Phase 1: Generate

- Generate programs that are easier to automatically verify
 - Programs use recursion instead of loops
 - Programs only call helper functions whose specifications are available



Phase 1: Generate

- Generate programs that are easier to automatically verify
 - Programs use recursion instead of loops
 - Programs only call helper functions whose specifications are available

Initial Prompt

Biases, Examples,
Specification



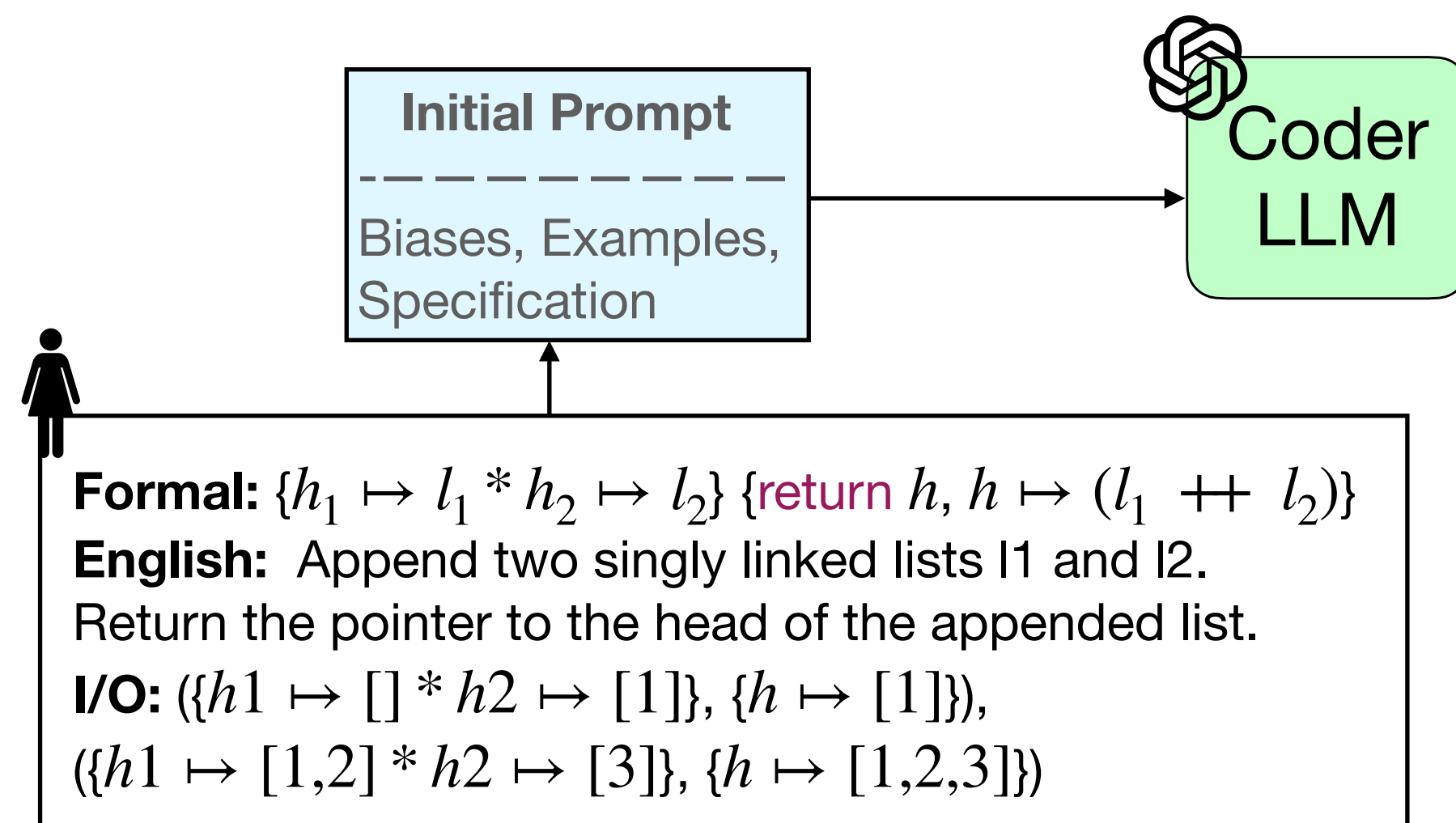
Formal: $\{h_1 \mapsto l_1 * h_2 \mapsto l_2\} \{\text{return } h, h \mapsto (l_1 \mathbin{++} l_2)\}$

English: Append two singly linked lists l1 and l2.
Return the pointer to the head of the appended list.

I/O: $(\{h1 \mapsto [] * h2 \mapsto [1]\}, \{h \mapsto [1]\}),$
 $(\{h1 \mapsto [1,2] * h2 \mapsto [3]\}, \{h \mapsto [1,2,3]\})$

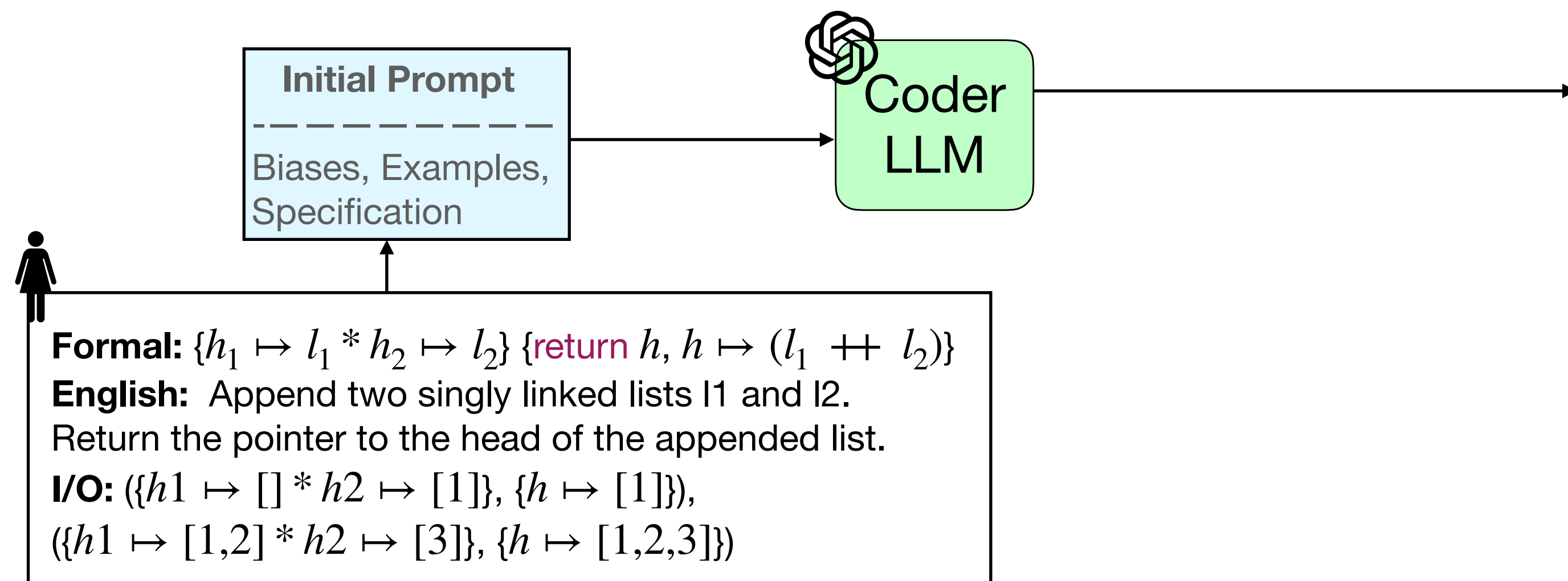
Phase 1: Generate

- Generate programs that are easier to automatically verify
 - Programs use recursion instead of loops
 - Programs only call helper functions whose specifications are available



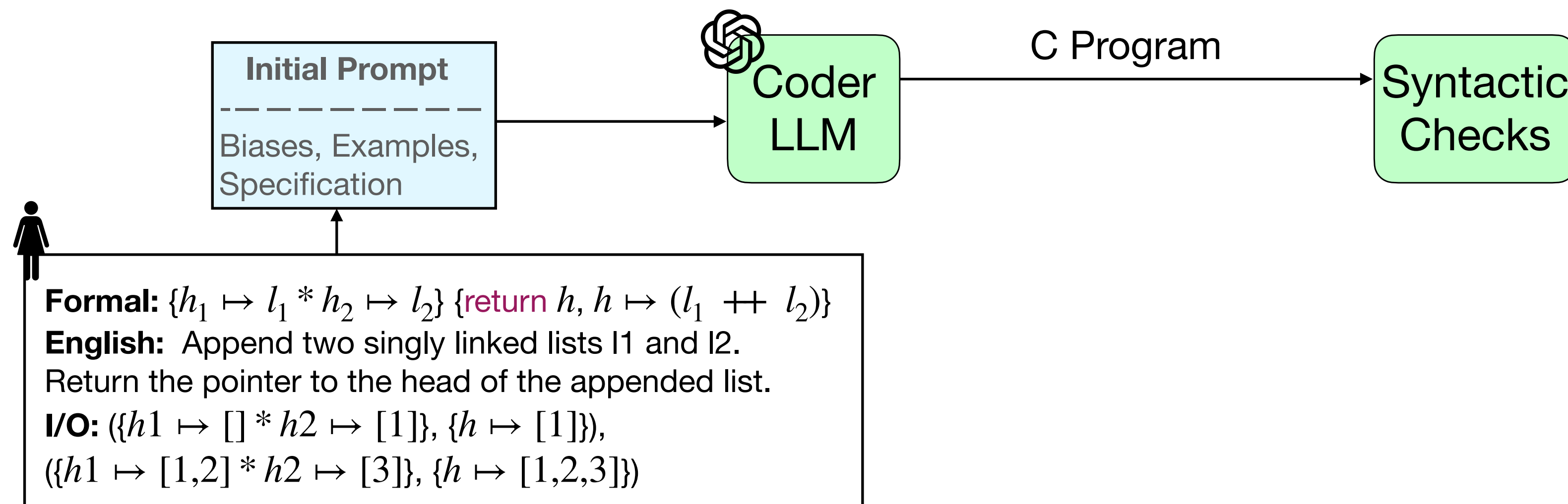
Phase 1: Generate

- Generate programs that are easier to automatically verify
 - Programs use recursion instead of loops
 - Programs only call helper functions whose specifications are available



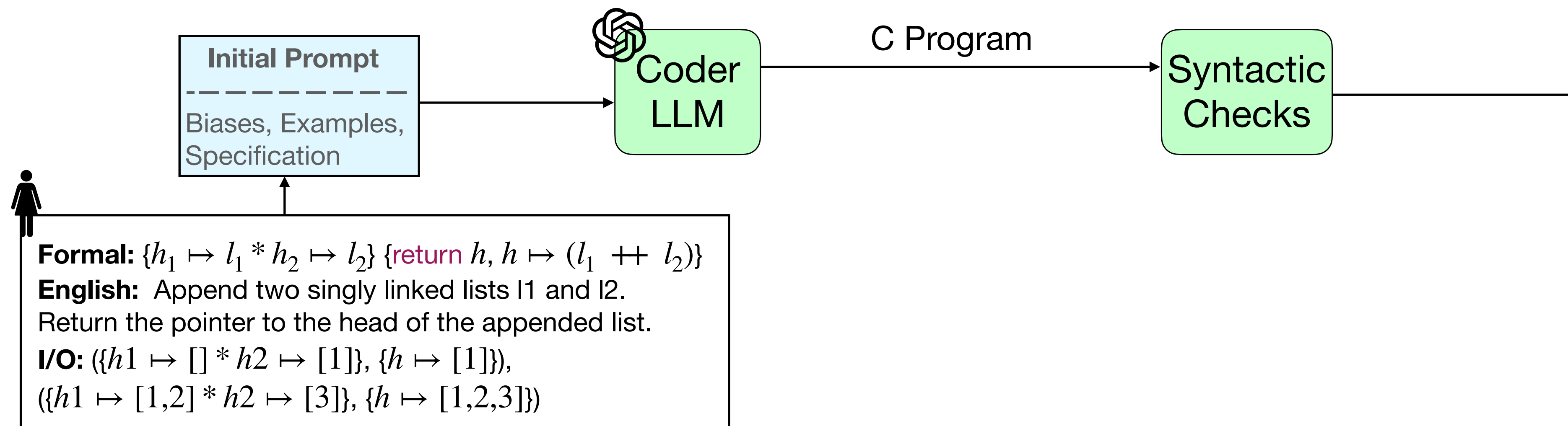
Phase 1: Generate

- Generate programs that are easier to automatically verify
 - Programs use recursion instead of loops
 - Programs only call helper functions whose specifications are available



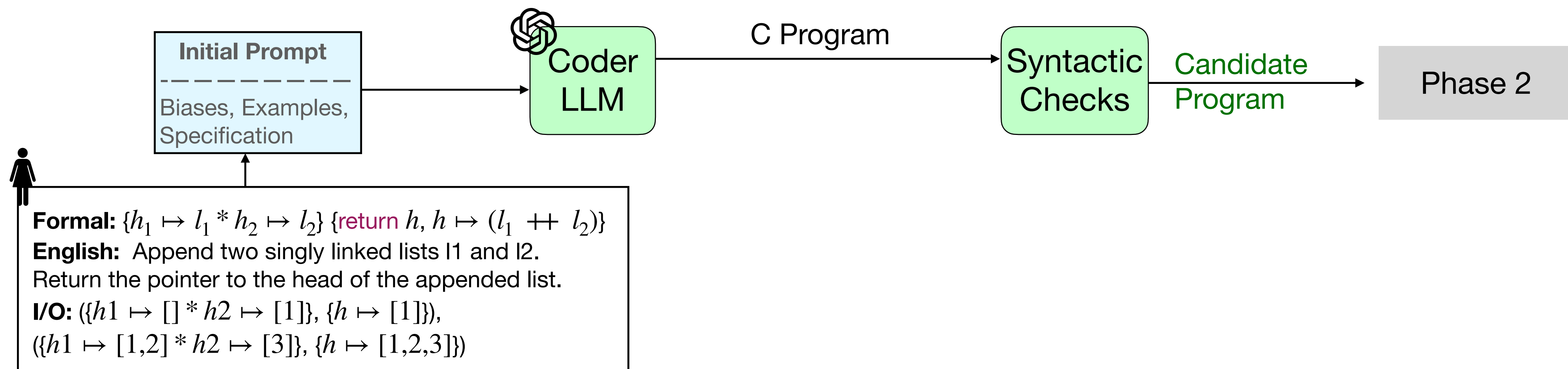
Phase 1: Generate

- Generate programs that are easier to automatically verify
 - Programs use recursion instead of loops
 - Programs only call helper functions whose specifications are available



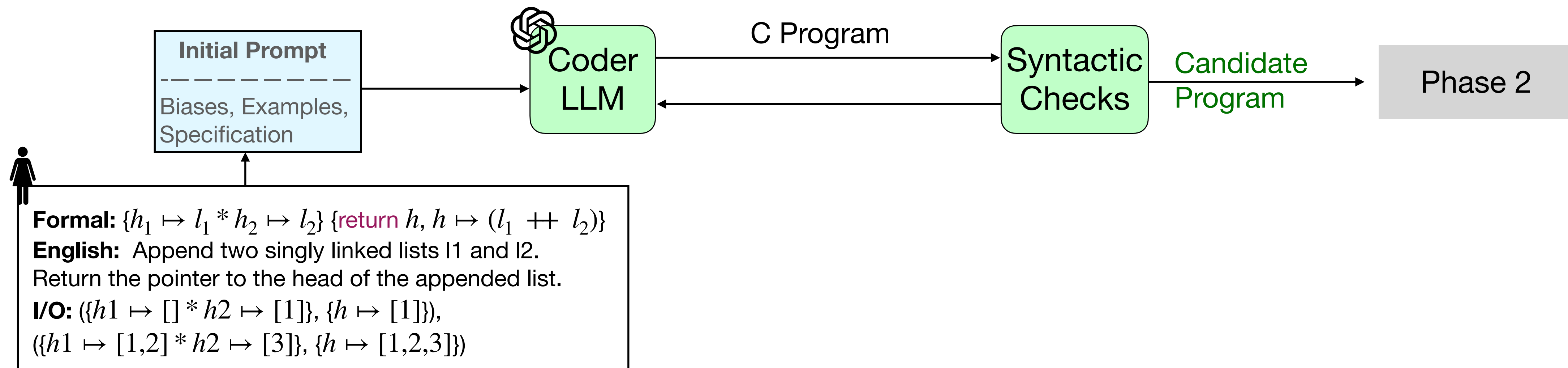
Phase 1: Generate

- Generate programs that are easier to automatically verify
 - Programs use recursion instead of loops
 - Programs only call helper functions whose specifications are available



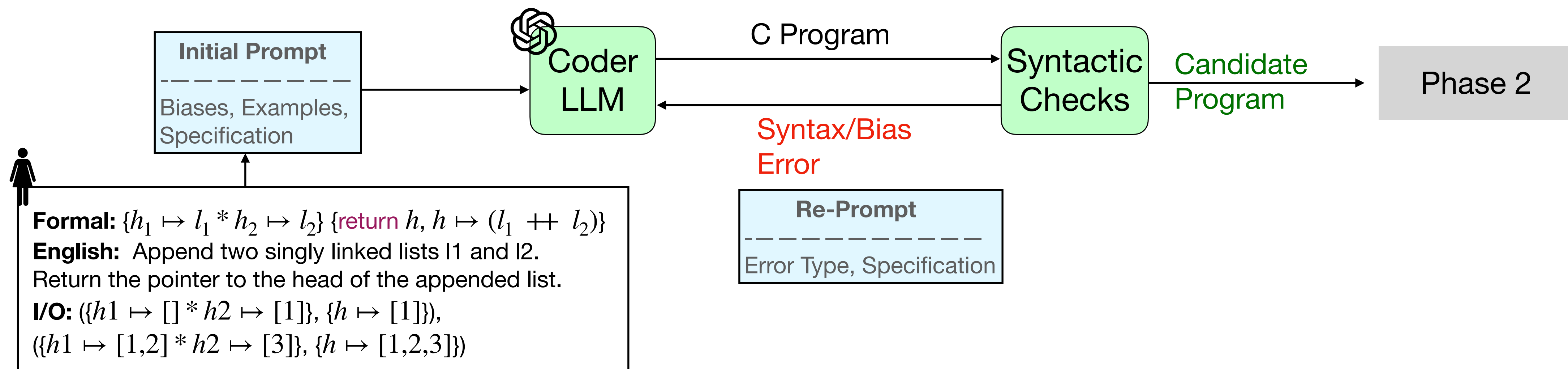
Phase 1: Generate

- Generate programs that are easier to automatically verify
 - Programs use recursion instead of loops
 - Programs only call helper functions whose specifications are available



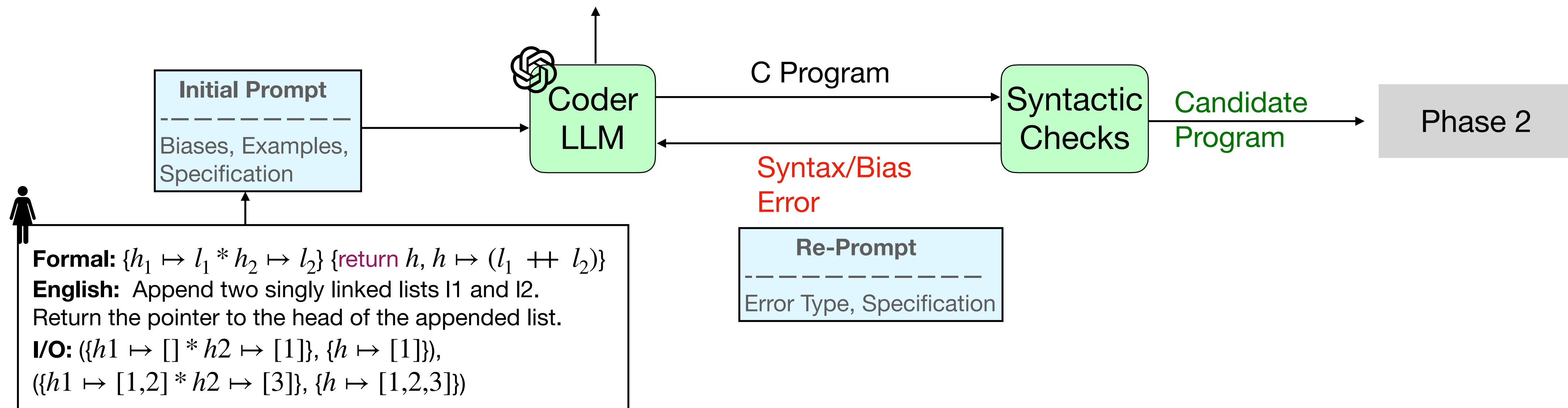
Phase 1: Generate

- Generate programs that are easier to automatically verify
 - Programs use recursion instead of loops
 - Programs only call helper functions whose specifications are available



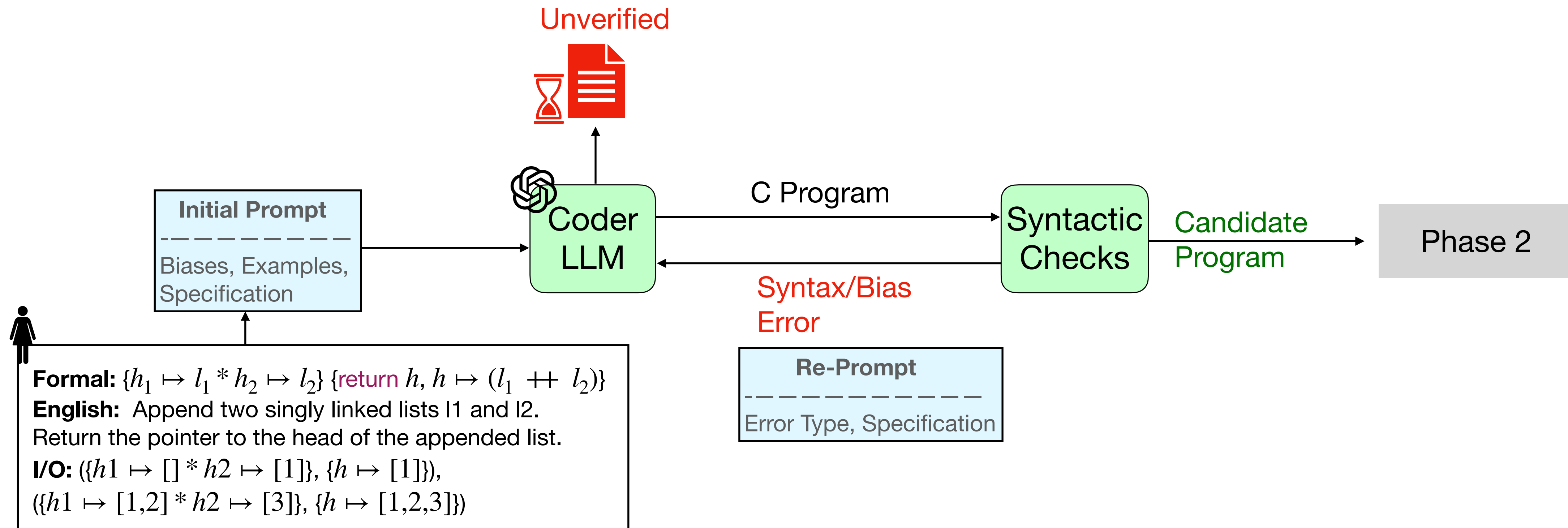
Phase 1: Generate

- Generate programs that are easier to automatically verify
 - Programs use recursion instead of loops
 - Programs only call helper functions whose specifications are available



Phase 1: Generate

- Generate programs that are easier to automatically verify
 - Programs use recursion instead of loops
 - Programs only call helper functions whose specifications are available



Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic

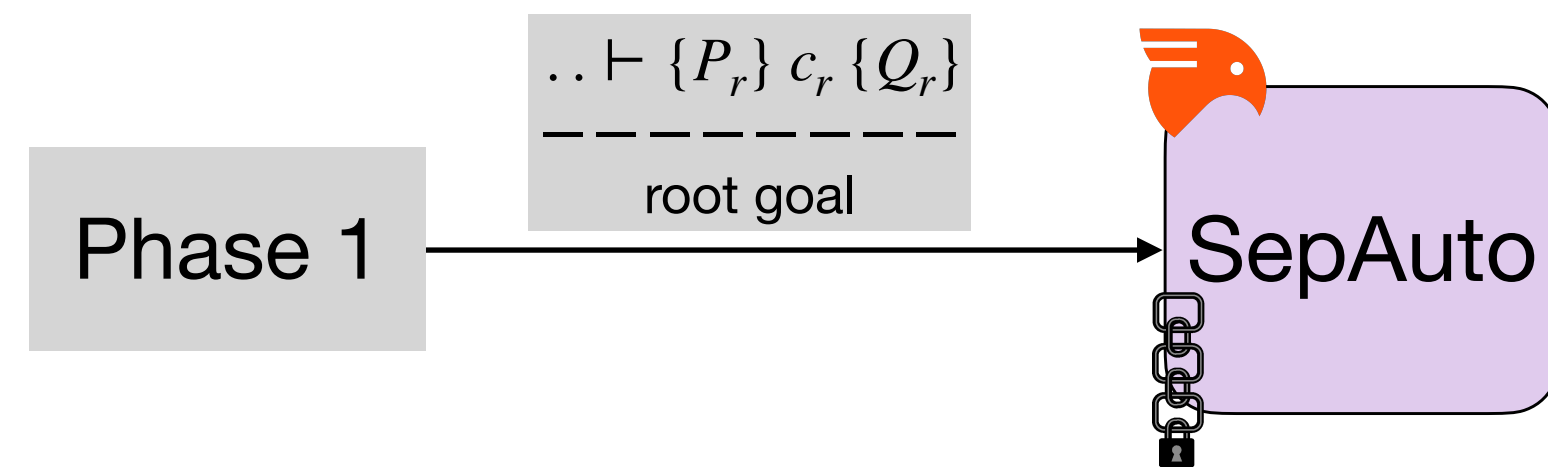
Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic

Phase 1

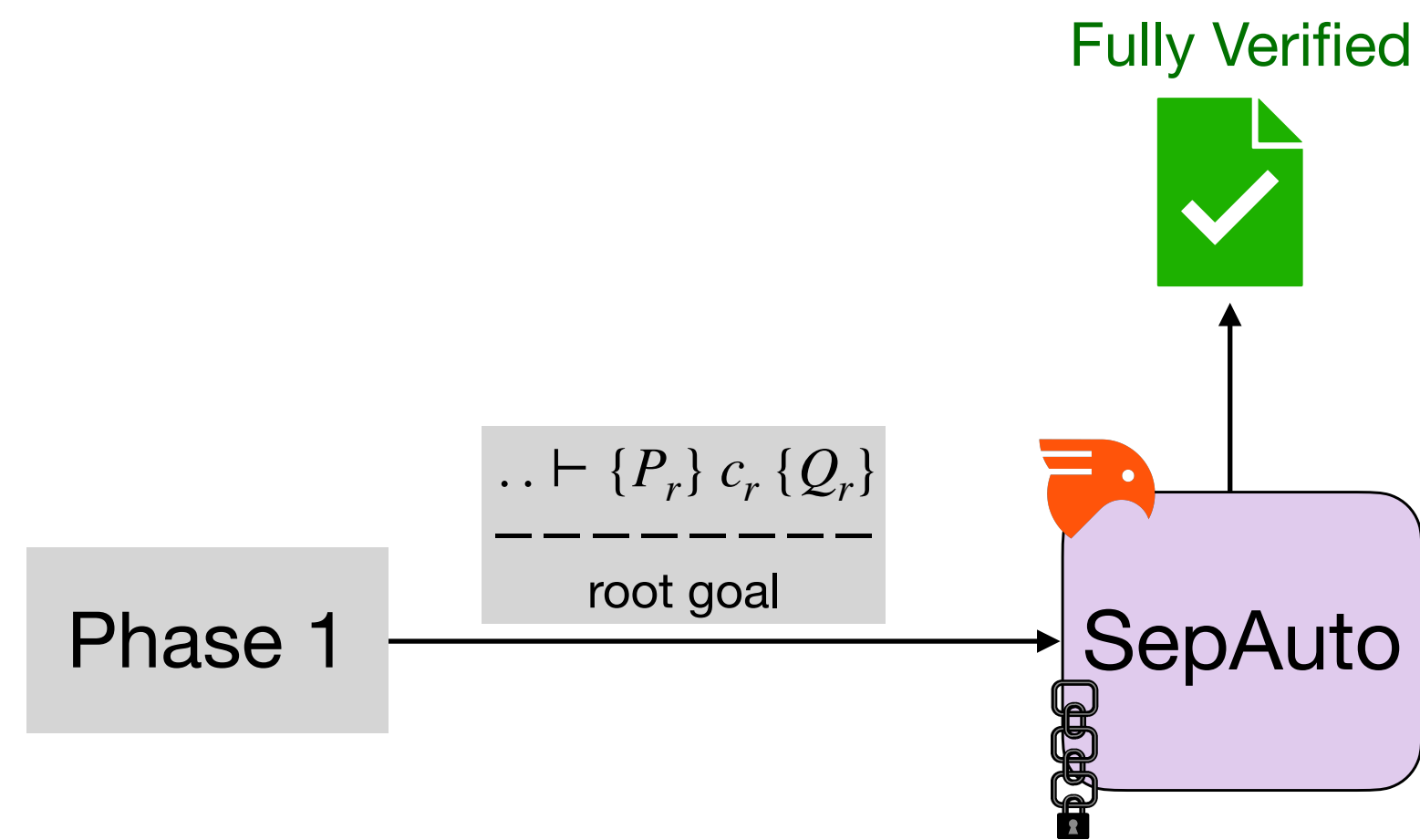
Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic



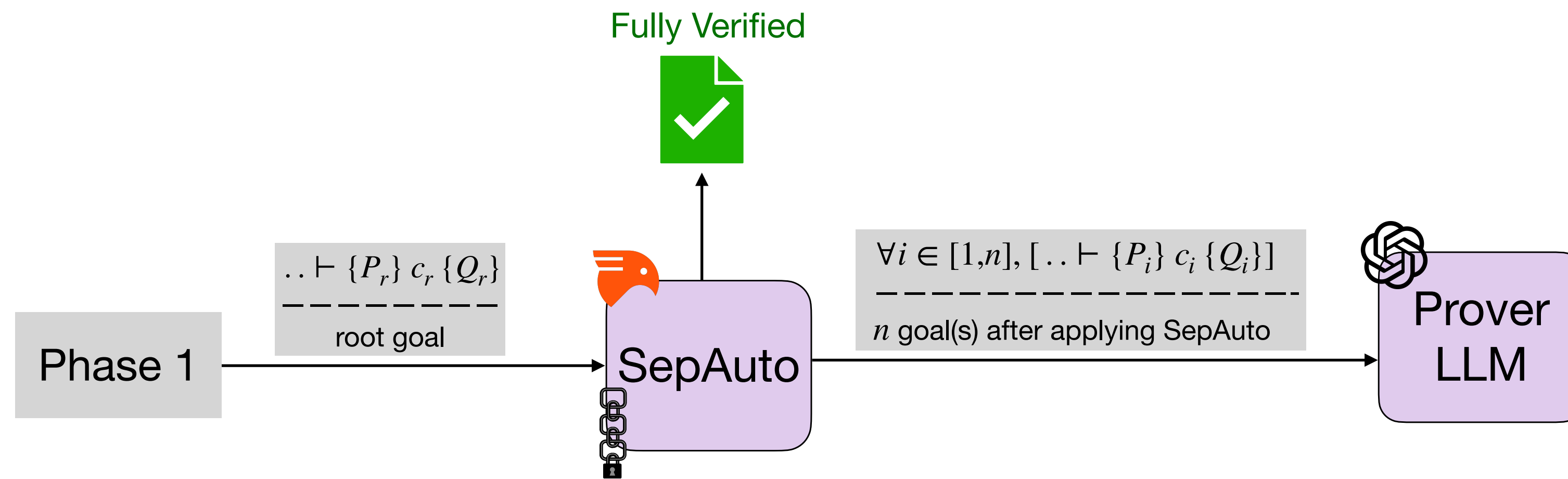
Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic



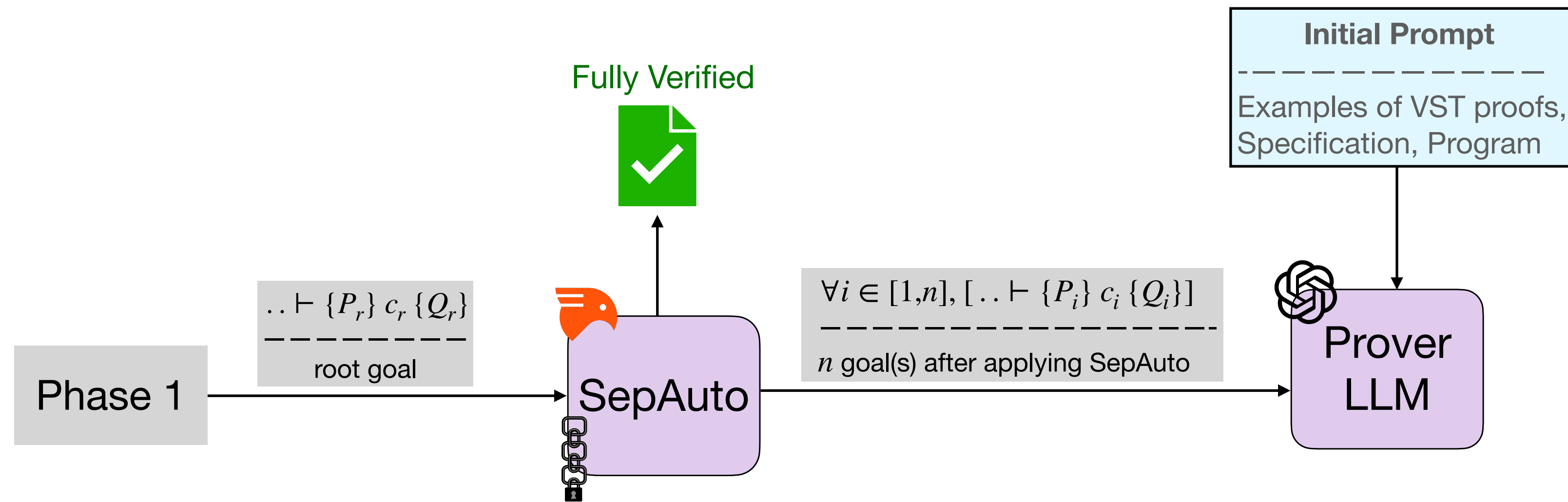
Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic



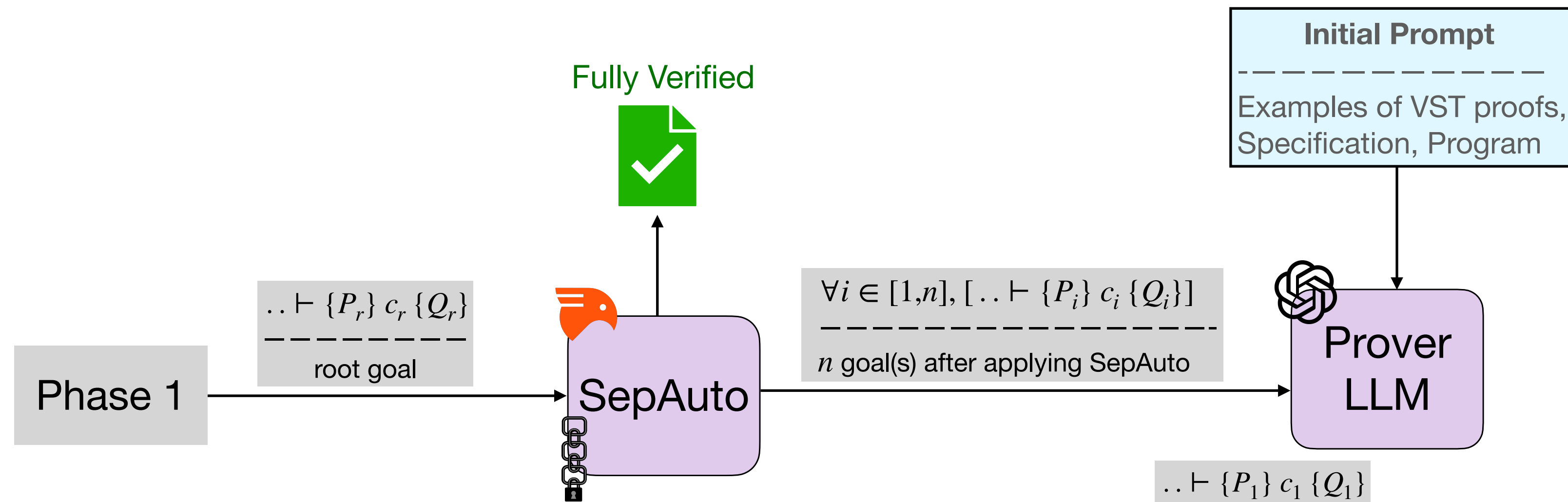
Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic



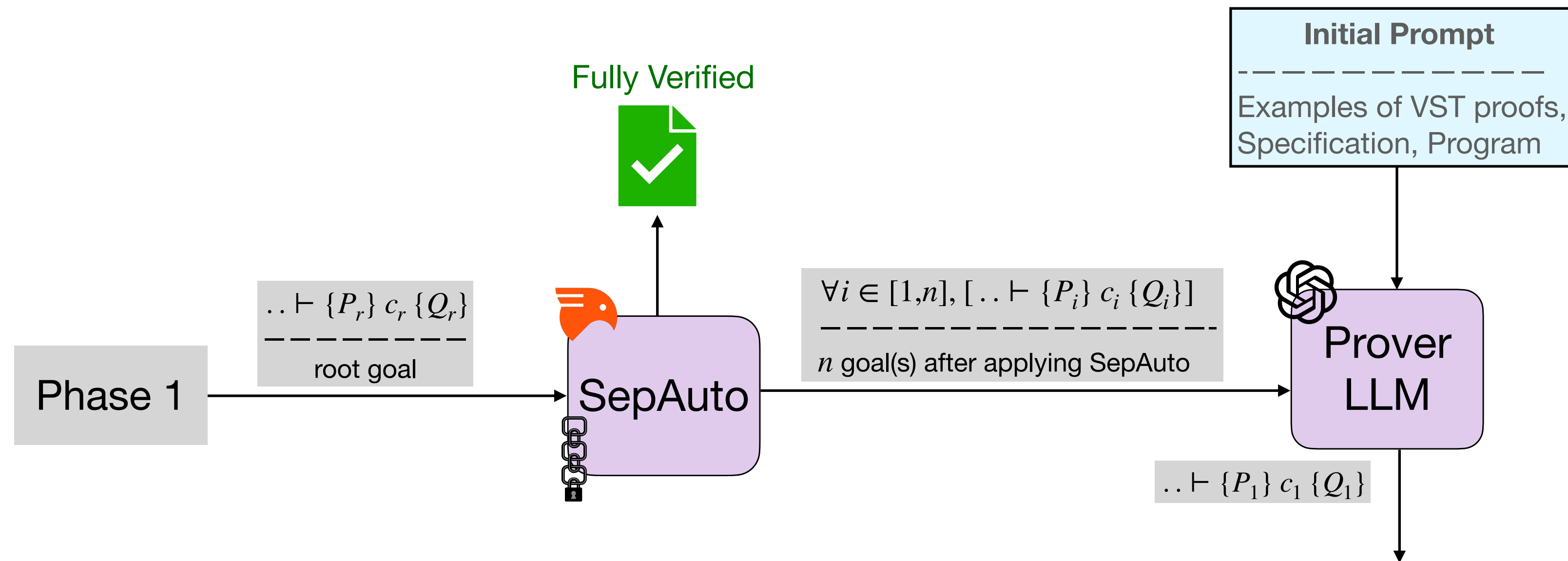
Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic



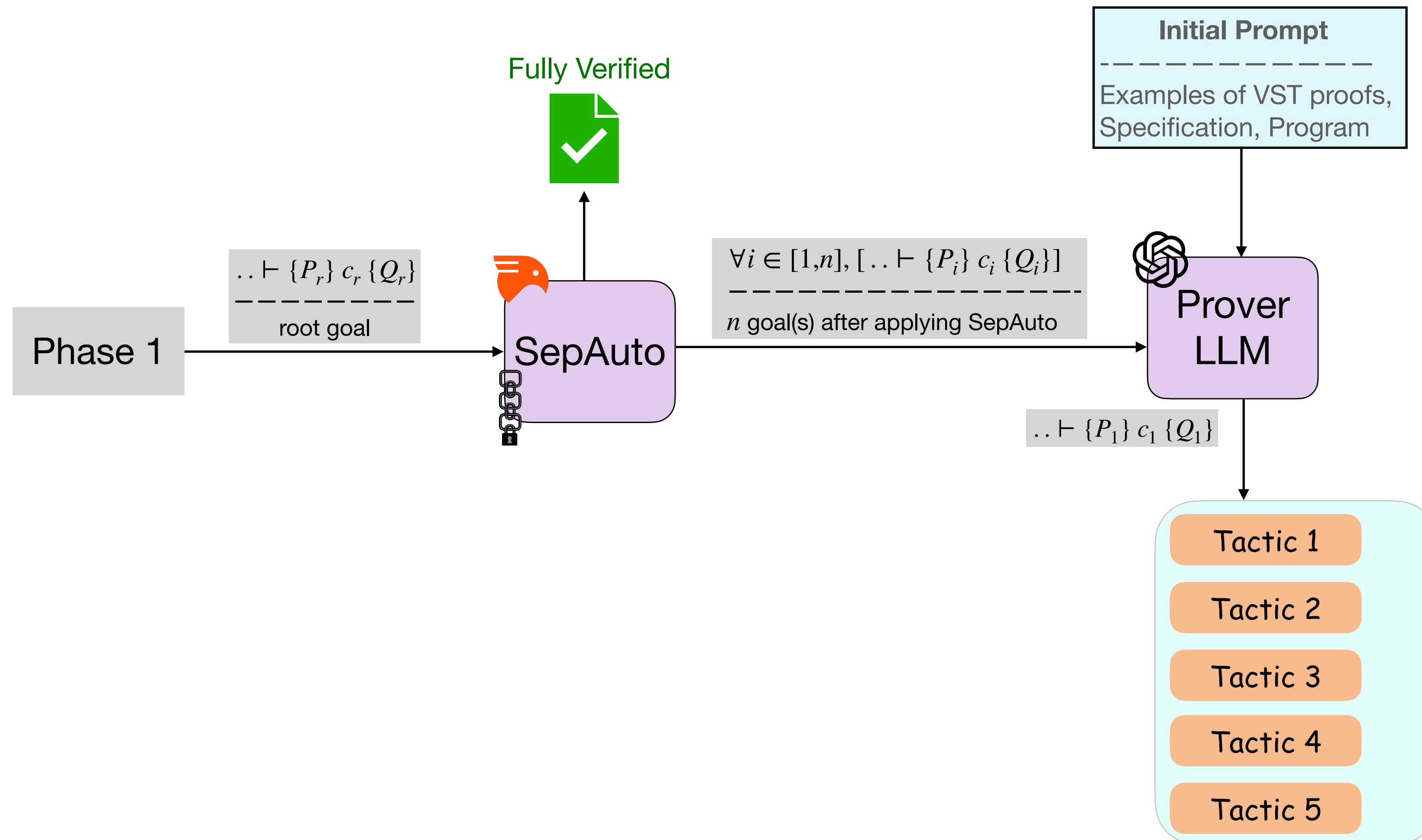
Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic



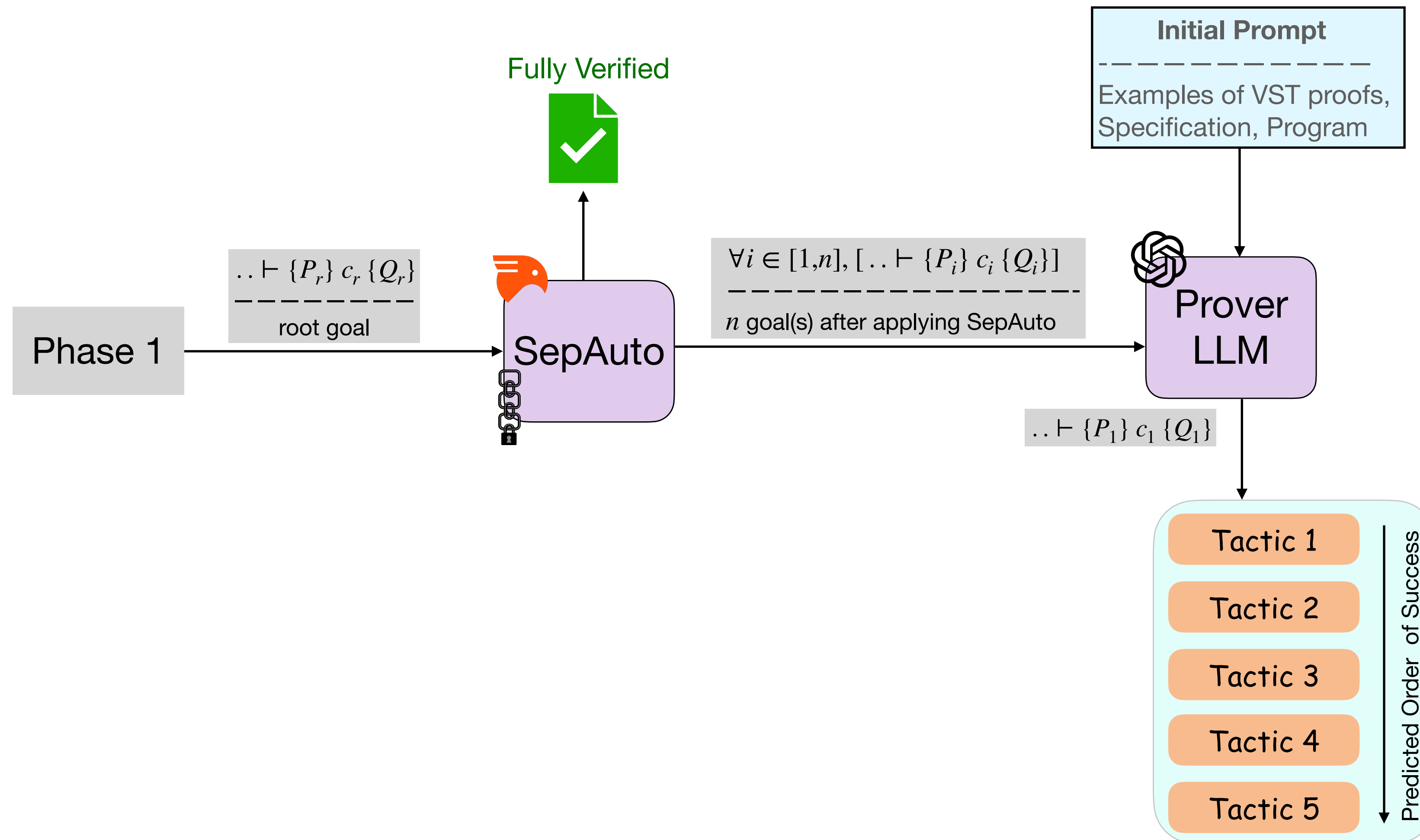
Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic



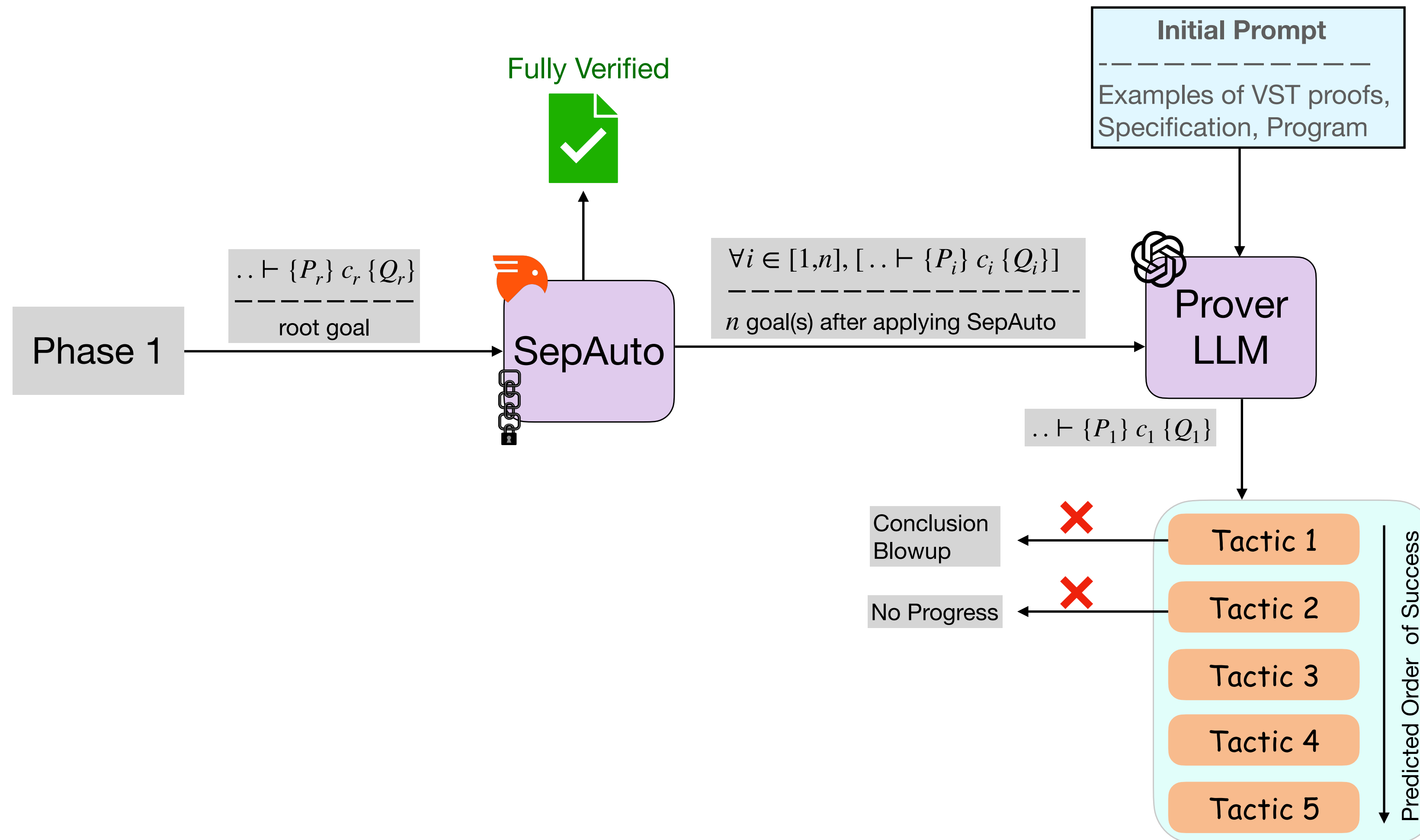
Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic



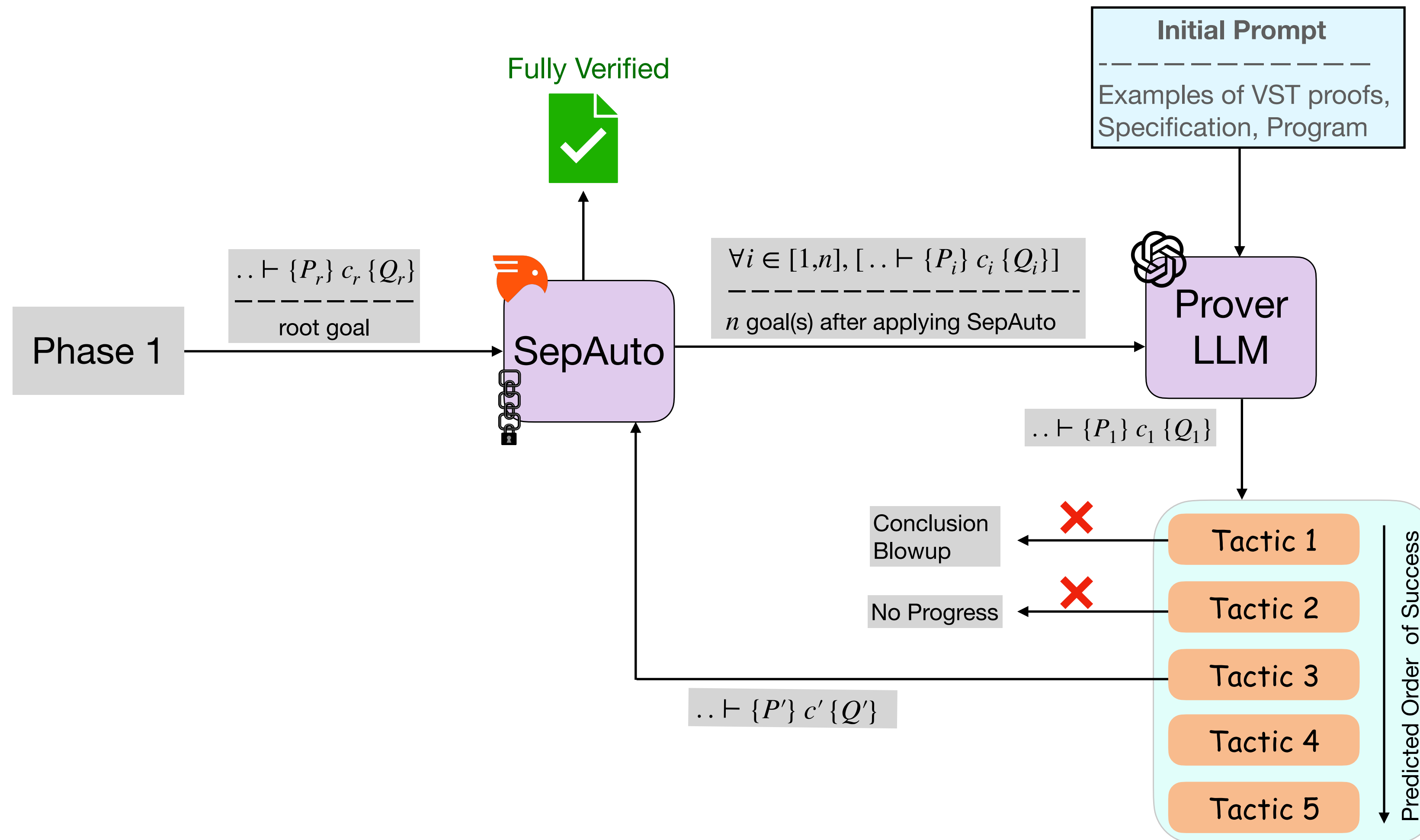
Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic



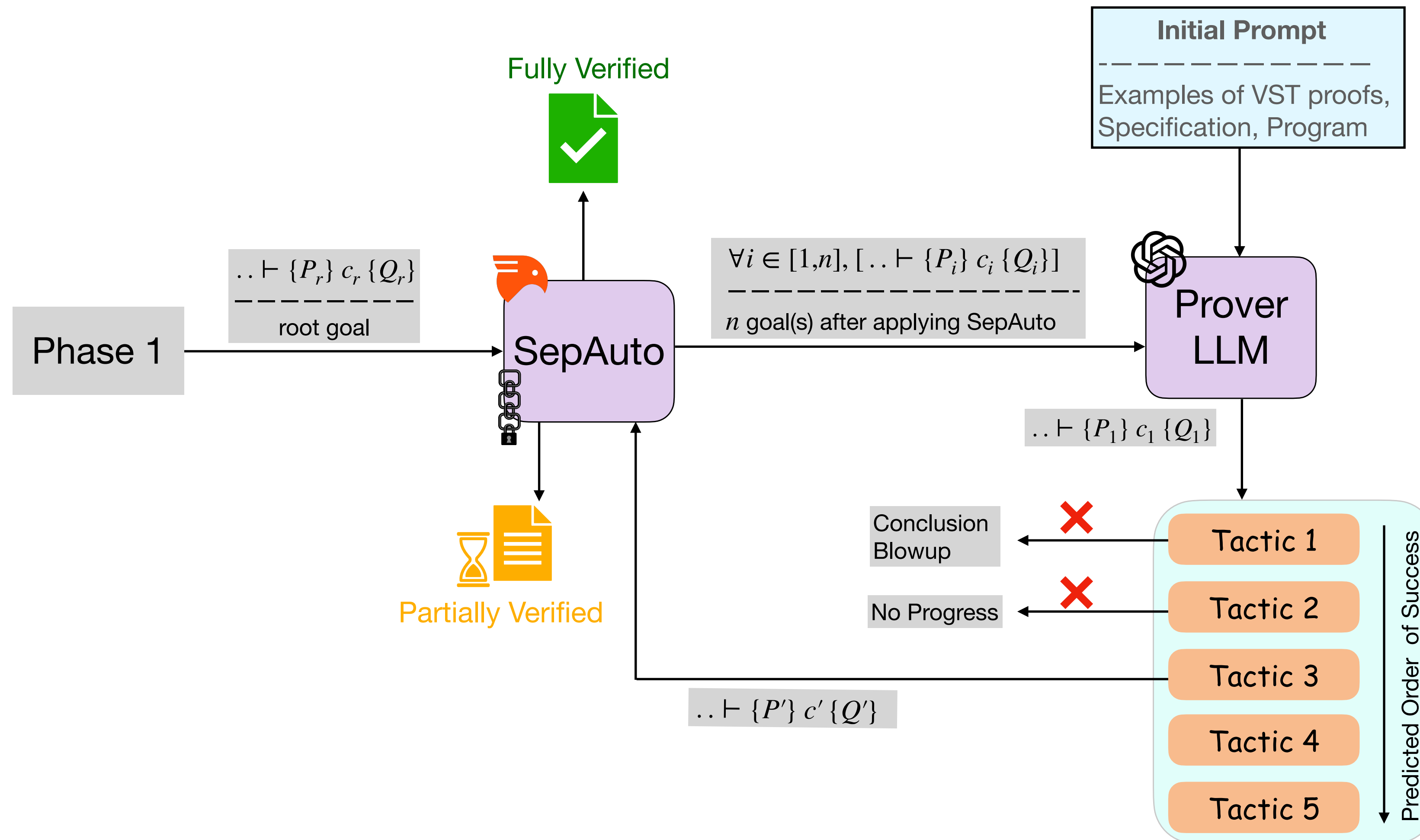
Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic



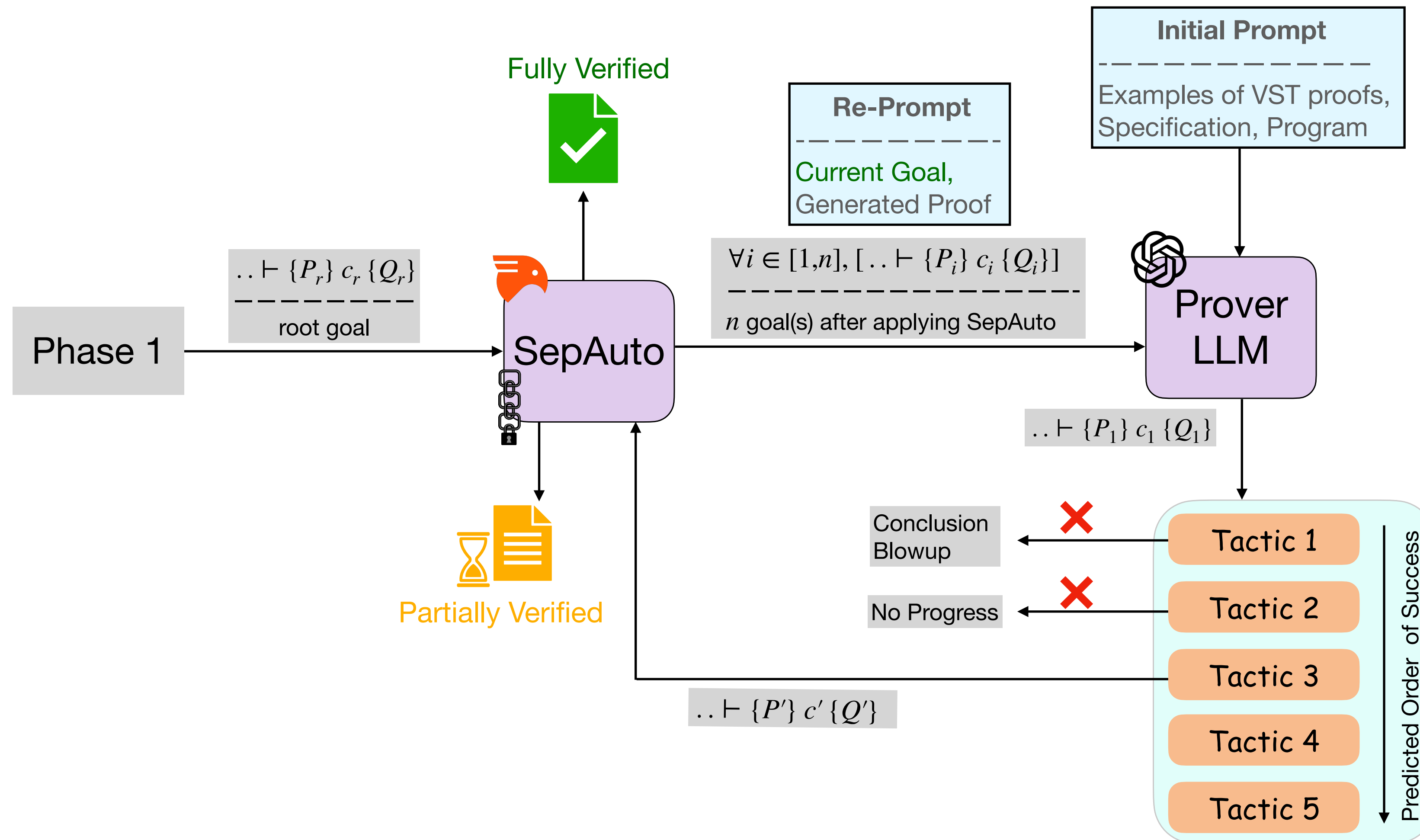
Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic



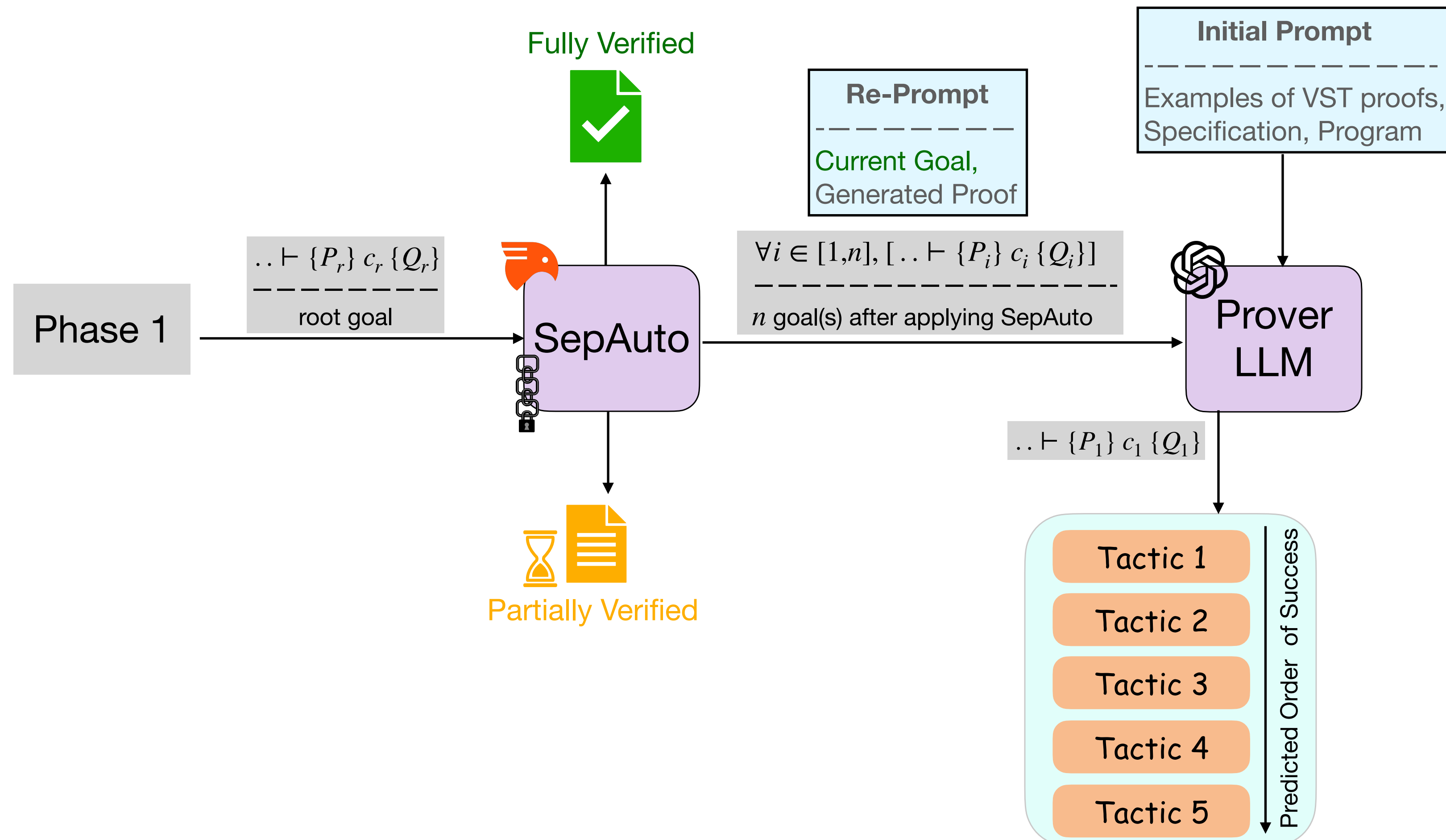
Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic



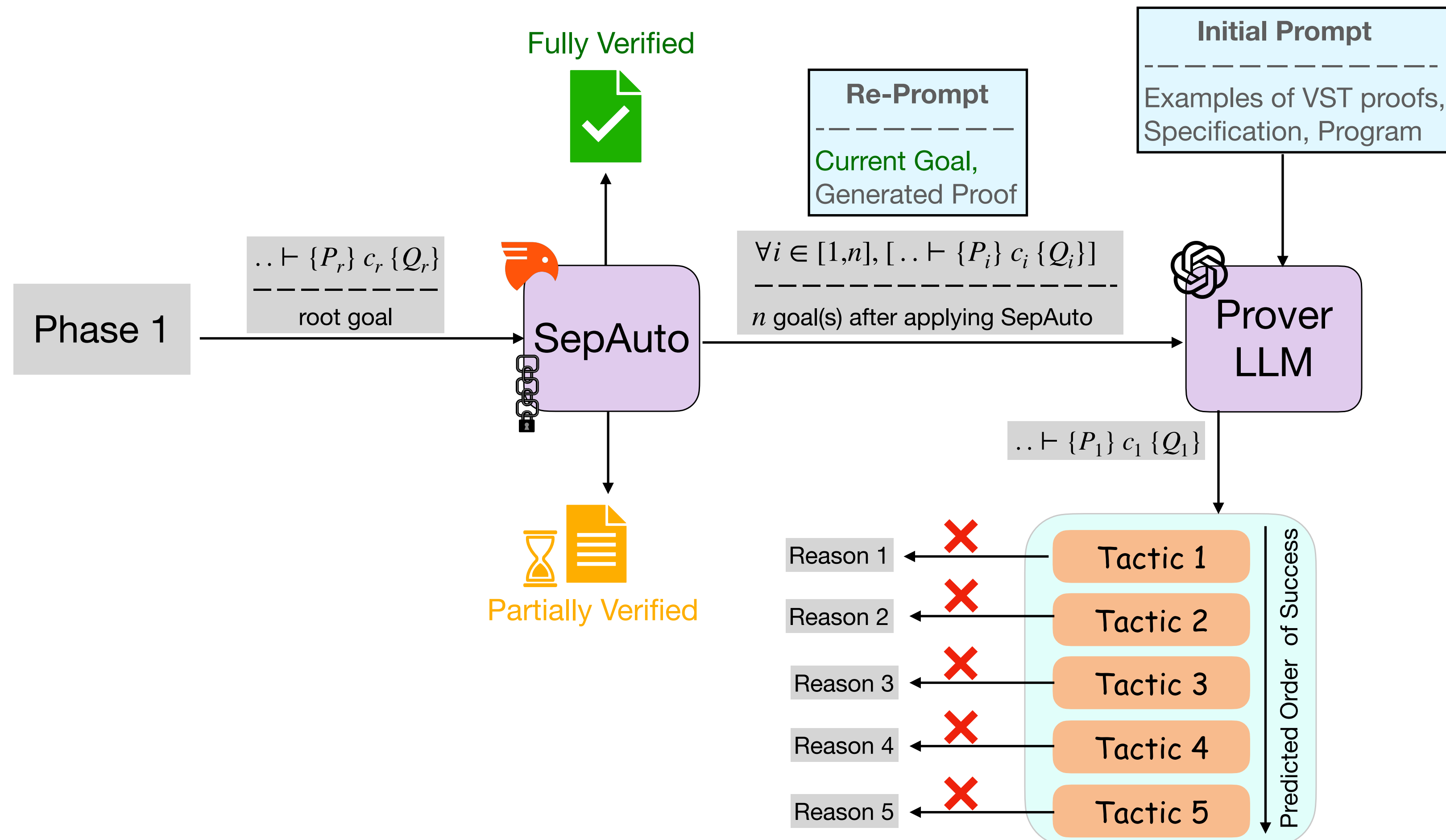
Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic



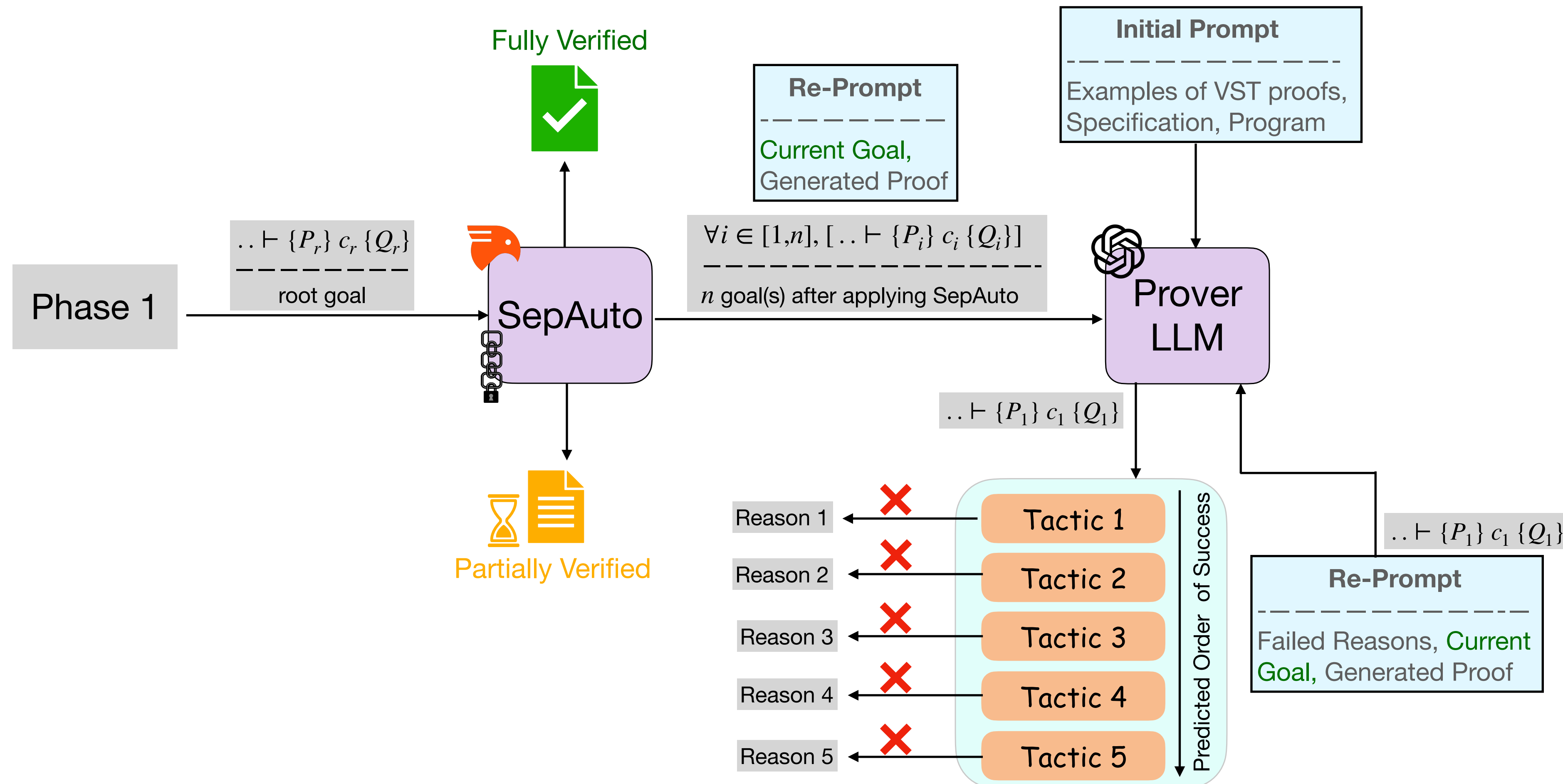
Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic



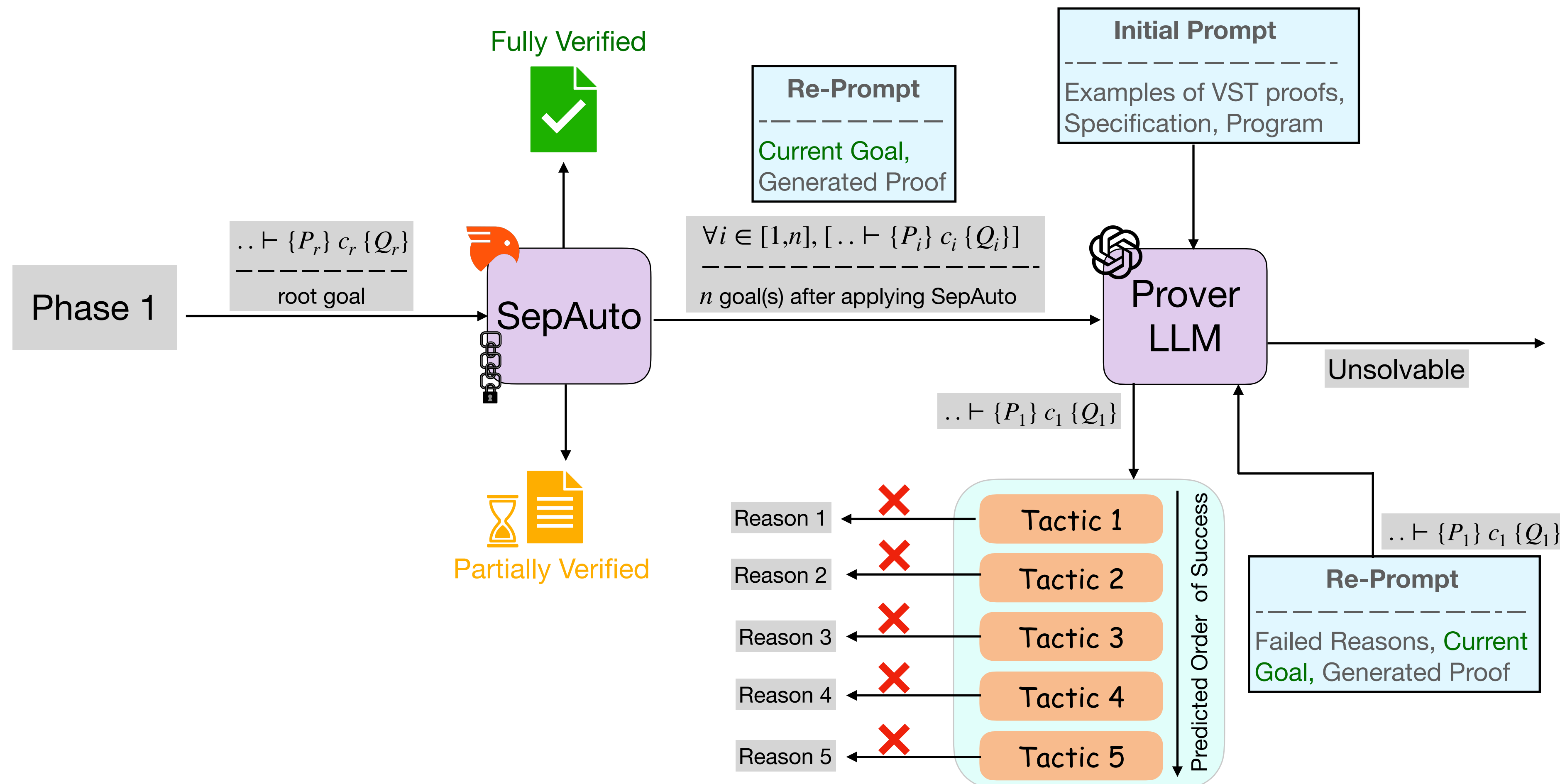
Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic



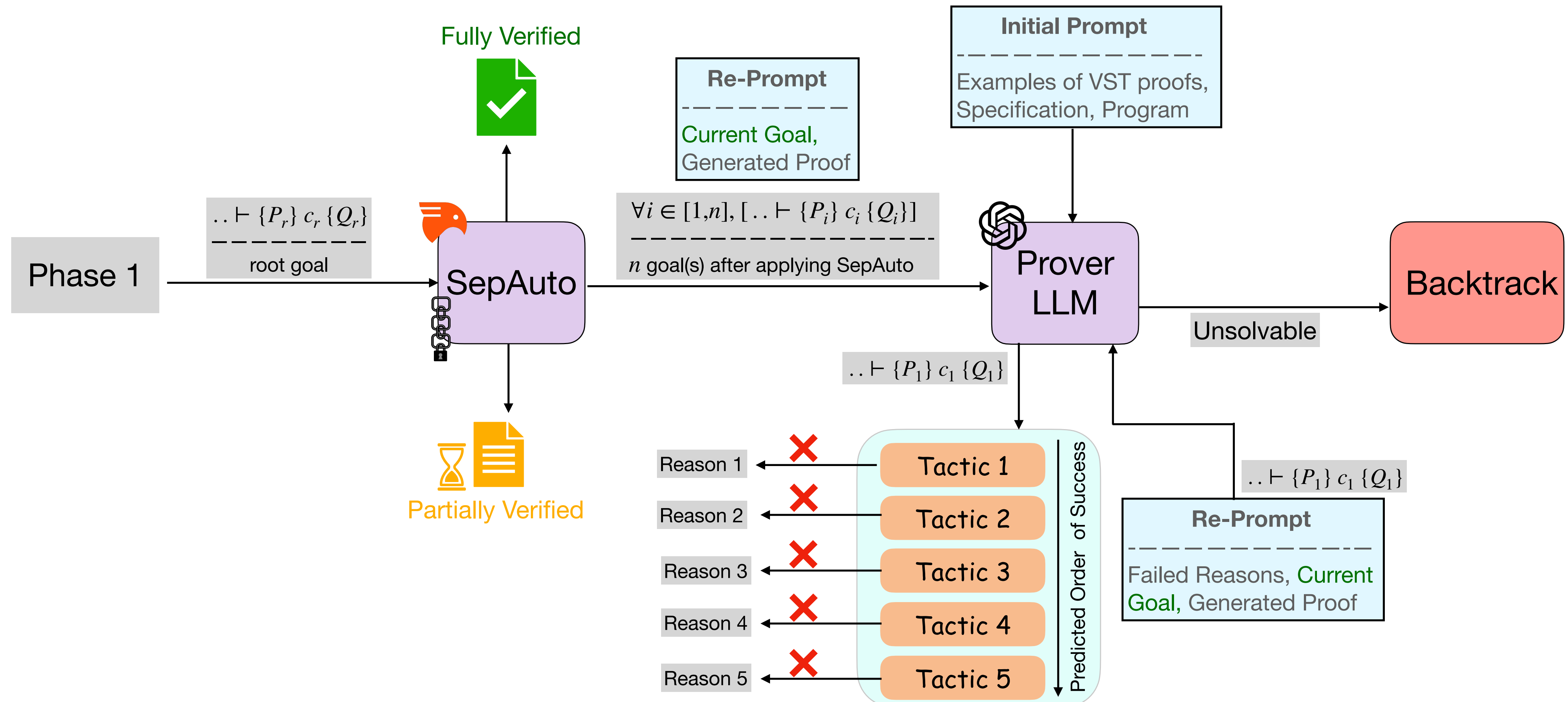
Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic



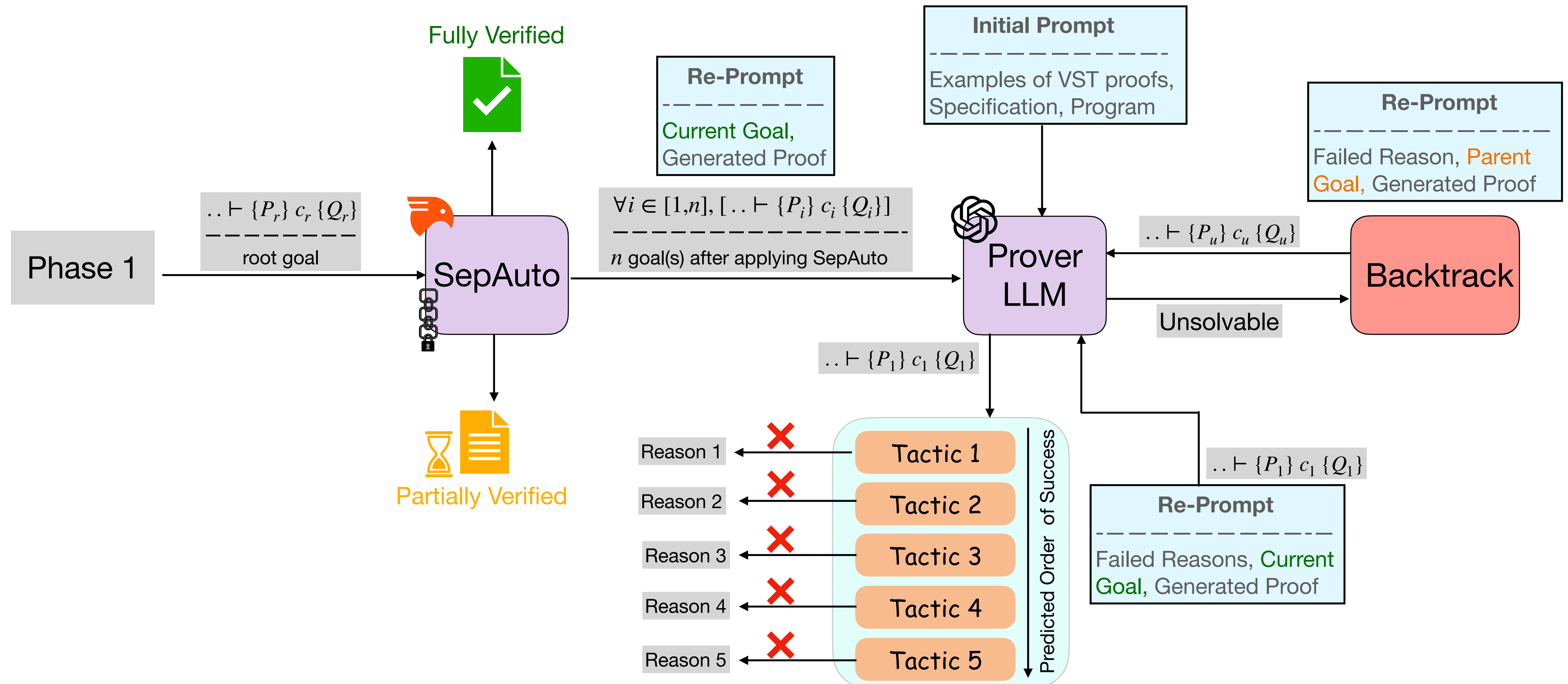
Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic



Phase 2: Verify

- Combination of symbolic reasoning (SepAuto) and LLM-based tactic prediction
- Heuristics to select or reject the LLM-proposed tactic



Evaluation

Evaluation

- We evaluate SYNVER on three domains of synthesis:

Evaluation

- We evaluate SYNVER on three domains of synthesis:
 - Basic [1] - 19 benchmarks. Use arrays and primitive types.

[1] M.R.H. Misu, C.V. Lopes, I. Ma, and J. Noble, “Towards AI-Assisted Synthesis of Verified Dafny Methods”, FSE 2024.

Evaluation

- We evaluate SYNVER on three domains of synthesis:
 - Basic [1] - 19 benchmarks. Use arrays and primitive types.
 - Heap [2] - 24 benchmarks. Use linked lists, trees and arrays.

[1] M.R.H. Misu, C.V. Lopes, I. Ma, and J. Noble, “Towards AI-Assisted Synthesis of Verified Dafny Methods”, FSE 2024.

[2] Y. Watanabe, K. Gopinathan, G. Pirlea, N. Polikarpova, and I. Sergey, “Certifying the synthesis of heap-manipulating programs”, ICFP 2021.

Evaluation

- We evaluate SYNVER on three domains of synthesis:
 - Basic [1] - 19 benchmarks. Use arrays and primitive types.
 - Heap [2] - 24 benchmarks. Use linked lists, trees and arrays.
 - API [3] - 5 benchmarks.

[1] M.R.H. Misu, C.V. Lopes, I. Ma, and J. Noble, “Towards AI-Assisted Synthesis of Verified Dafny Methods”, FSE 2024.

[2] Y. Watanabe, K. Gopinathan, G. Pirlea, N. Polikarpova, and I. Sergey, “Certifying the synthesis of heap-manipulating programs”, ICFP 2021.

[3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, “Introduction to Algorithms, Third Edition, 3rd ed.” The MIT Press, 2009.

Evaluation

- We evaluate SYNVER on three domains of synthesis:
 - Basic [1] - 19 benchmarks. Use arrays and primitive types.
 - Heap [2] - 24 benchmarks. Use linked lists, trees and arrays.
 - API [3] - 5 benchmarks.
- SYNVER generates **all** programs correctly in the first try.

[1] M.R.H. Misu, C.V. Lopes, I. Ma, and J. Noble, “Towards AI-Assisted Synthesis of Verified Dafny Methods”, FSE 2024.

[2] Y. Watanabe, K. Gopinathan, G. Pirlea, N. Polikarpova, and I. Sergey, “Certifying the synthesis of heap-manipulating programs”, ICFP 2021.

[3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, “Introduction to Algorithms, Third Edition, 3rd ed.” The MIT Press, 2009.

Evaluation

- We evaluate SYNVER on three domains of synthesis:
 - Basic [1] - 19 benchmarks. Use arrays and primitive types.
 - Heap [2] - 24 benchmarks. Use linked lists, trees and arrays.
 - API [3] - 5 benchmarks.
- SYNVER generates **all** programs correctly in the first try.
- SYNVER fully verifies **70%** of the programs.

[1] M.R.H. Misu, C.V. Lopes, I. Ma, and J. Noble, “Towards AI-Assisted Synthesis of Verified Dafny Methods”, FSE 2024.

[2] Y. Watanabe, K. Gopinathan, G. Pirlea, N. Polikarpova, and I. Sergey, “Certifying the synthesis of heap-manipulating programs”, ICFP 2021.

[3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, “Introduction to Algorithms, Third Edition, 3rd ed.” The MIT Press, 2009.

Evaluation

- We evaluate SYNVER on three domains of synthesis:
 - Basic [1] - 19 benchmarks. Use arrays and primitive types.
 - Heap [2] - 24 benchmarks. Use linked lists, trees and arrays.
 - API [3] - 5 benchmarks.
- SYNVER generates **all** programs correctly in the first try.
- SYNVER fully verifies **70%** of the programs.
- Detailed evaluation on wall clock time, prompt components, and prover comparisons are present in the paper!

[1] M.R.H. Misu, C.V. Lopes, I. Ma, and J. Noble, “Towards AI-Assisted Synthesis of Verified Dafny Methods”, FSE 2024.

[2] Y. Watanabe, K. Gopinathan, G. Pirlea, N. Polikarpova, and I. Sergey, “Certifying the synthesis of heap-manipulating programs”, ICFP 2021.

[3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, “Introduction to Algorithms, Third Edition, 3rd ed.” The MIT Press, 2009.

Conclusions

- SYNVER is the first general purpose synthesizer for high-assurance C programs
 - **Key idea 1:** Syntactic biases to reduce human effort
 - **Key idea 2:** Custom hybrid reasoning engine
- Evaluation: Automatically verifies majority of the programs
- Evaluation: Applicable to different domains of synthesis
- Check our poster and codebase out!

Artifact

<https://zenodo.org/records/17230953>

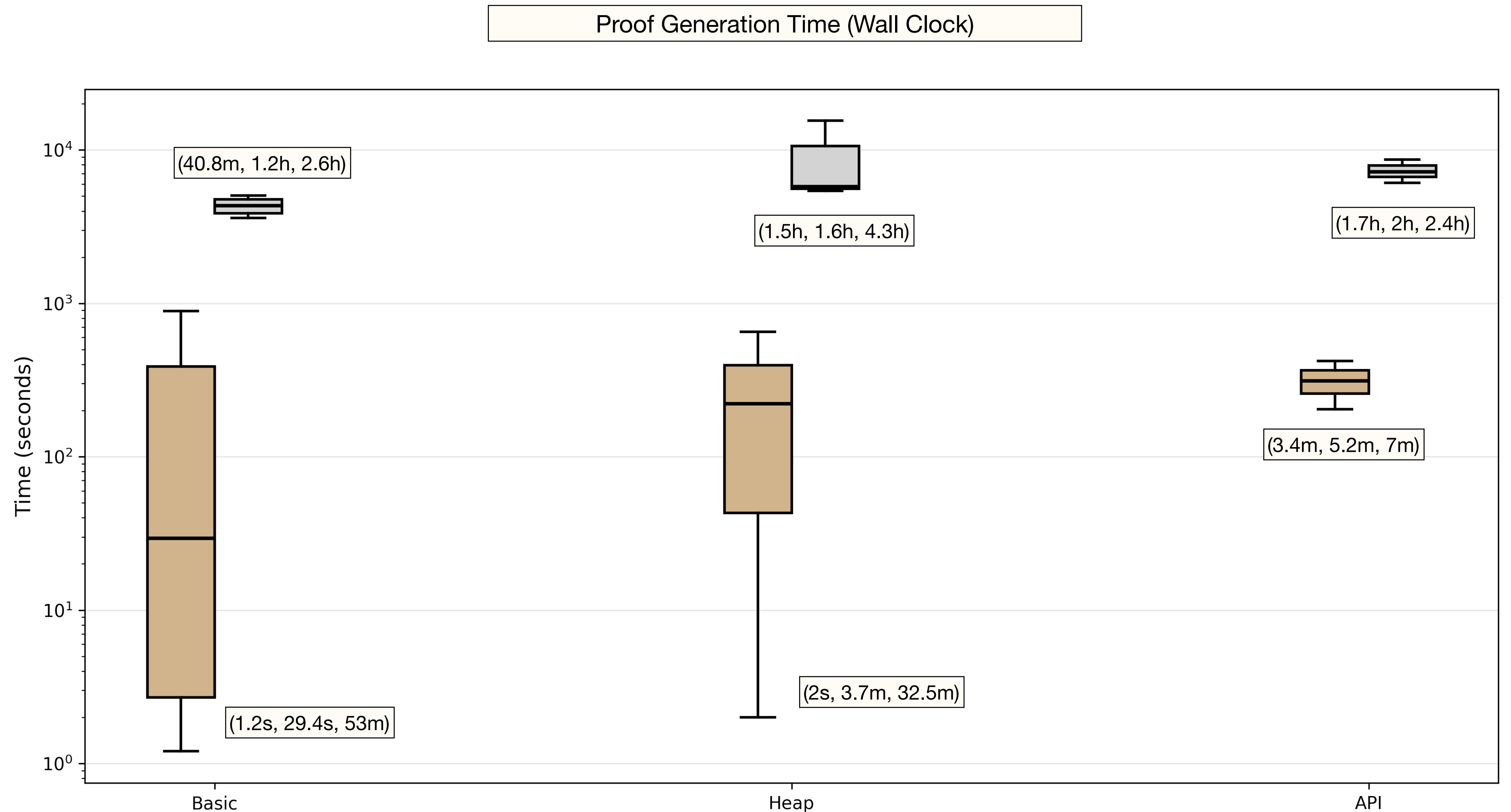
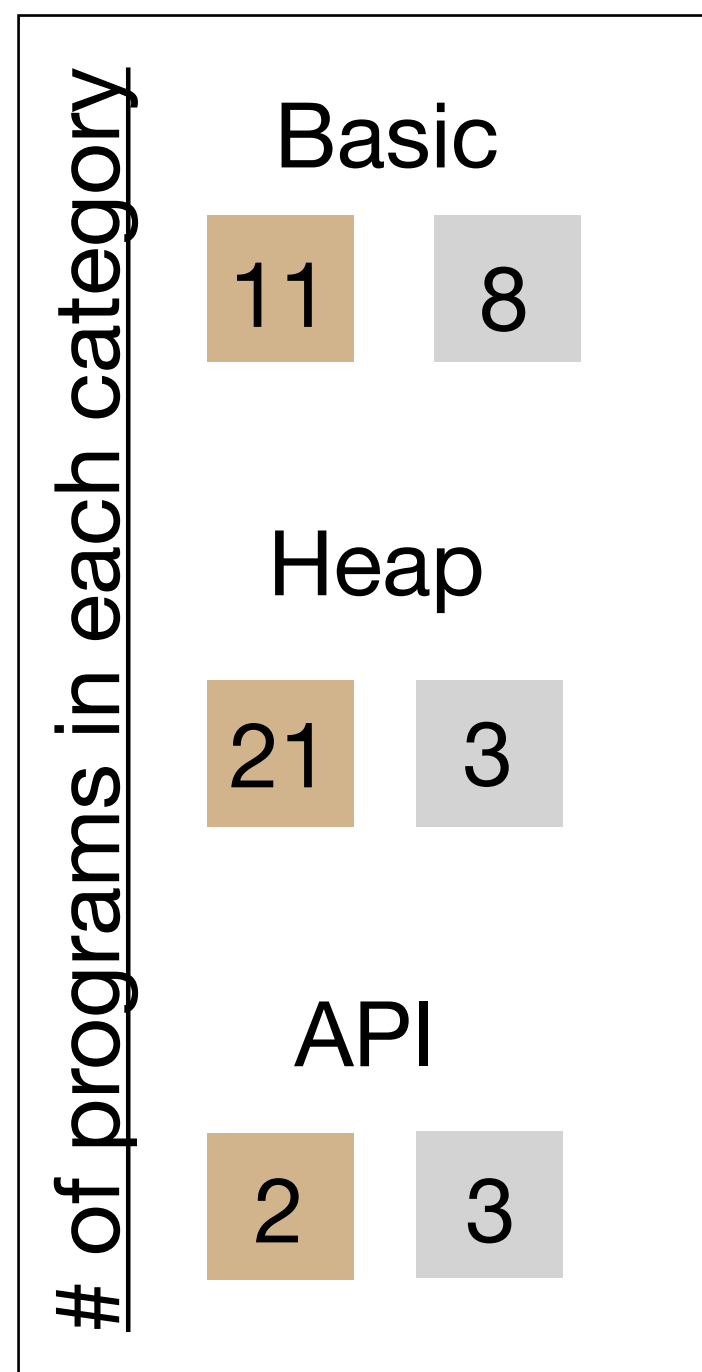
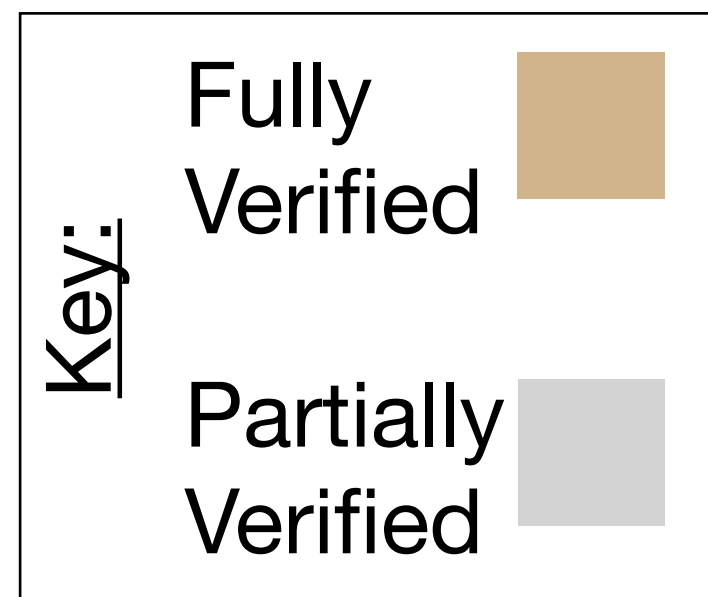
Contact

mukher39@purdue.edu

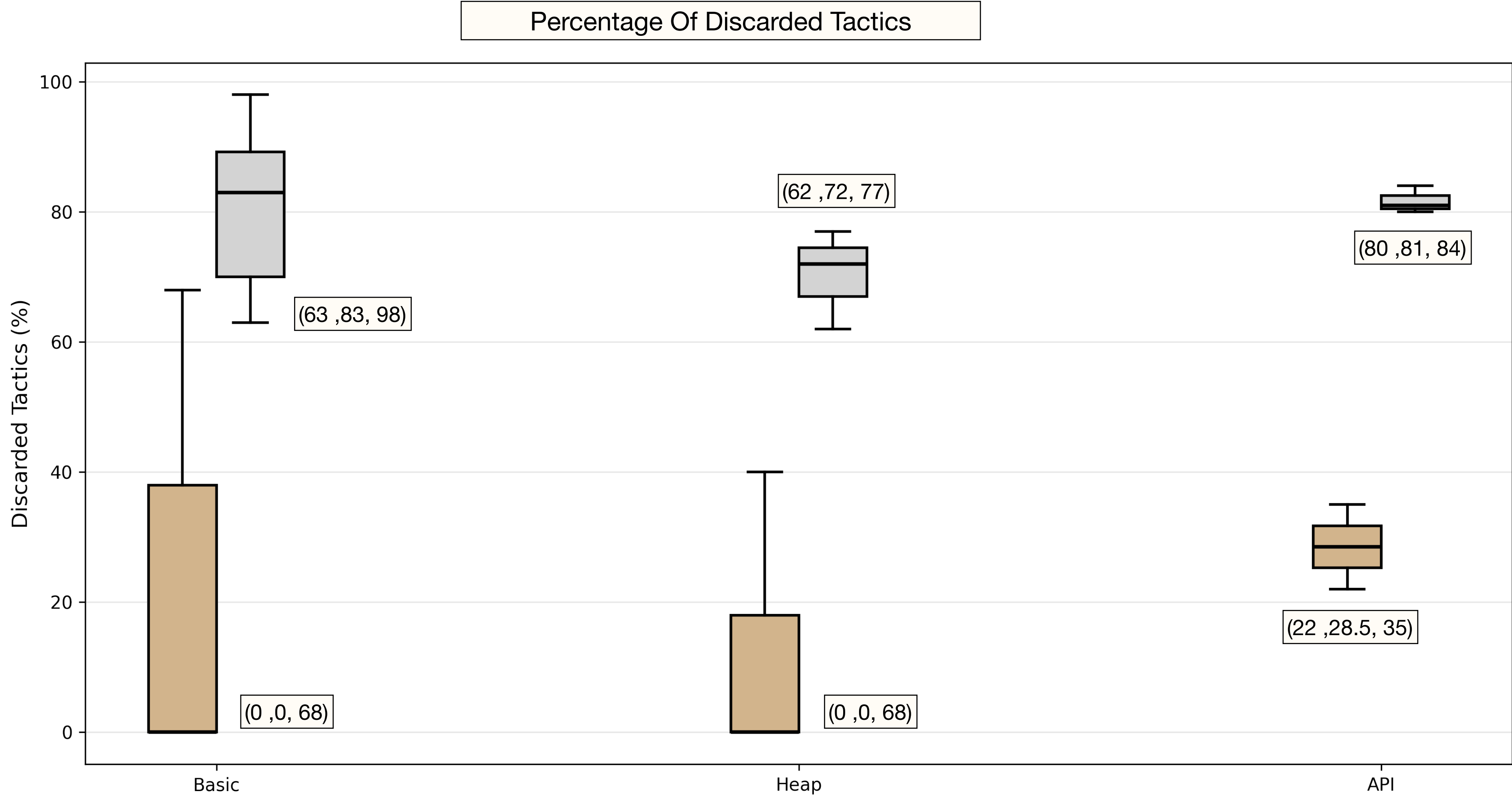
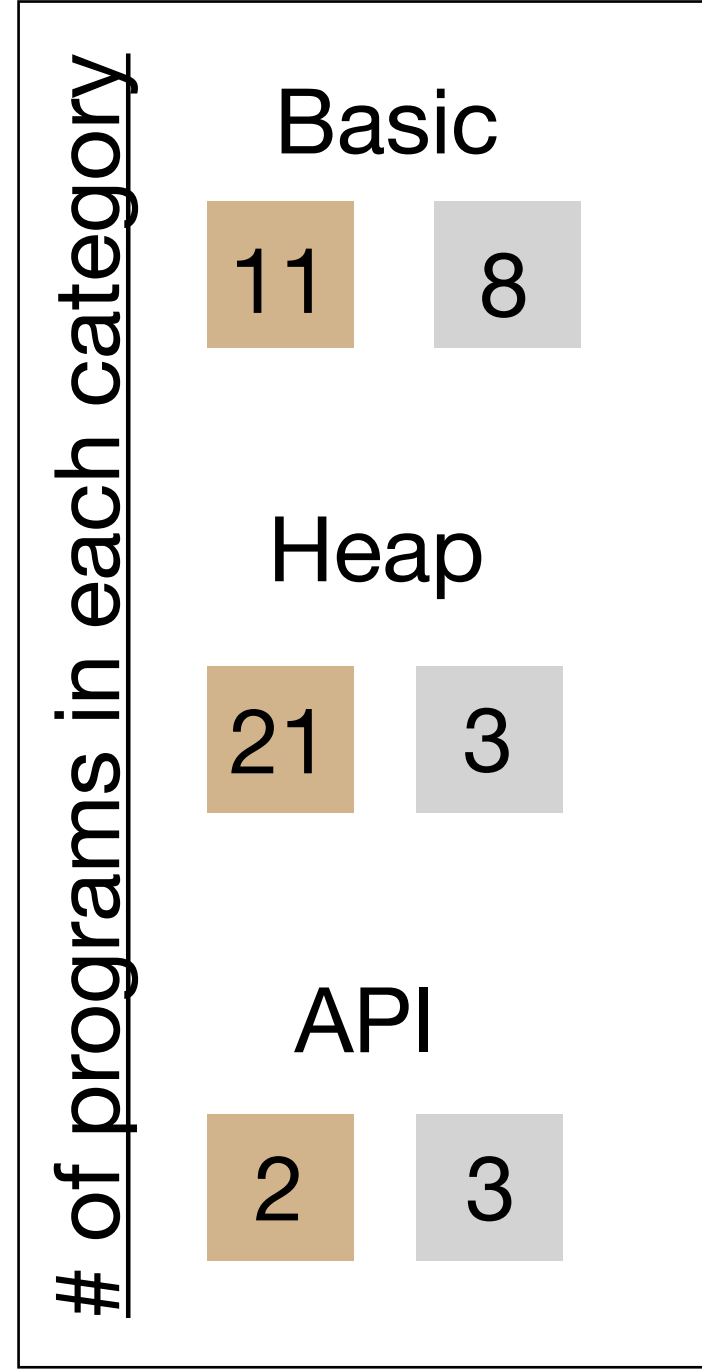
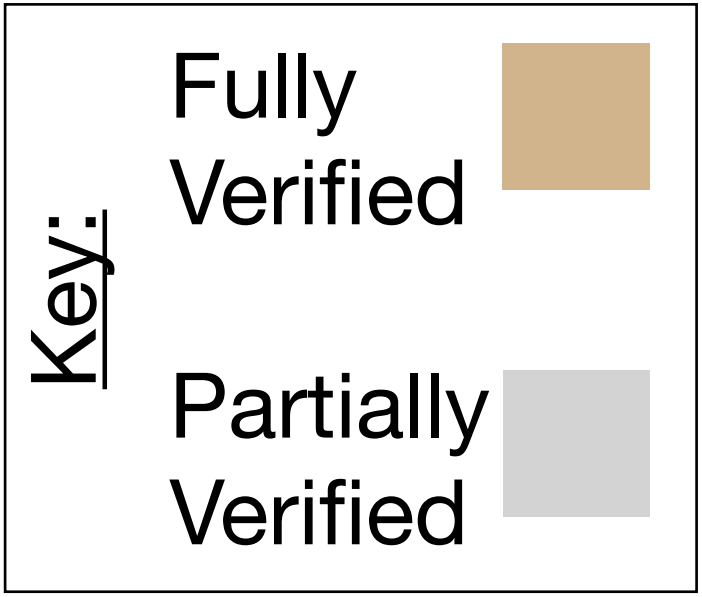


Code

Backup: How effective is SYNVER?



Backup: How effective is SYNVER?



Boxes are labelled with the minimum, median, and maximum percentage