

Next Steps in LLM-Supported Java Verification

1st International Workshop on Neuro-Symbolic Software Engineering

Samuel Teuber, Bernhard Beckert

Institute of Information Security and Dependability (KASTEL)

May 3, 2025

Motivation: Can LLMs Generate Specifications?

- Large Language Models have seen tremendous success in recent years
- GitHub Copilot & Co show: LLMs can **generate code**
- But can they **specify code?**

Motivation: Can LLMs Generate Specifications?

```
/*@ normal_behavior
@ ensures (\forall int j; j >= 0 && j < a.length; \result >= a[j]);
@ ensures a.length > 0
@ ==> (\exists int j; j >= 0 && j < a.length; \result == a[j]);
*/
public static /*@ pure */ int max(int[] a) {
    if (a.length == 0) return 0;
    int max = a[0], i = 1;
    /*
    @
    @
    @
    @
    */
    while (i < a.length) {
        if (a[i] > max) max = a[i];
        ++i;
    }
    return max;
}
```

Verification requires Loop Invariant

- Holds before first loop iteration
- Preserved by loop iteration
- Implies post condition

Additionally:

- Loop Variant
- Assignable Heap Variables

Motivation: Can LLMs Generate Specifications?

```
/*@ normal_behavior
@ ensures (\forall int j; j >= 0 && j < a.length; \result >= a[j]);
@ ensures a.length > 0
@     ==> (\exists int j; j >= 0 && j < a.length; \result == a[j]);
@*/
public static /*@ pure */ int max(int[] a) {
    if (a.length == 0) return 0;
    int max = a[0], i = 1;
    /*@ loop_invariant 0 <= i && i <= a.length;
     @ loop_invariant (\forall int k; 0 <= k && k < i; max >= a[k]);
     @ loop_invariant (\exists int k; 0 <= k && k < i; max == a[k]);
     @ decreases a.length - i;
     @ assignable max, i;
     @*/
    while (i < a.length) {
        if (a[i] > max) max = a[i];
        ++i;
    }
    return max;
}
```

Motivation: Can LLMs Generate Specifications?

- Large Language Models have seen tremendous success in recent years
- GitHub Copilot & Co show: LLMs can **generate code**
- But can they **specify code?**



Writing auxiliary spec yourself [ChatGPT]

Motivation: Can LLMs Generate Specifications?

- Large Language Models have seen tremendous success in recent years
- GitHub Copilot & Co show: LLMs can **generate code**
- But can they **specify code?**



Having ChatGPT write auxiliary spec [ChatGPT]

Motivation: Can LLMs Generate Specifications?

- Large Language Models have seen tremendous success in recent years
- GitHub Copilot & Co show: LLMs can **generate code**
- But can they **specify code**?

Let's ask ChatGPT:

Do you know JML, the Java Modeling Language?

Yes, I am familiar with JML (Java Modeling Language). JML is a formal specification language for Java programs. [...] JML is typically used in conjunction with formal verification tools, such as ESC/Java or KeY, to check that the code meets its specifications.

The Program Verifier KeY

Deductive verification

100% Java Card

Java Modeling
Language (JML)

Numerous Case Studies:

- TimSort (OpenJDK)
- LinkedList (OpenJDK)
- Super Scalar Sample Sort



Modular
Reasoning

collaboration of KIT, TU Darmstadt, Chalmers University

Ahrendt u. a. 2016

Java Modelling Language

- Specification Language for Java
- Design by Contract Paradigm
- Rich set of possible first-order annotations:
 - Hoare-Style pre- and post-conditions
 - Invariants
 - Asserts
 - Class-Invariants
- Supported by numerous tools for Java verification

```
/*@ normal_behavior
@ ensures (\forall int j; j >= 0 && j <
@ ensures a.length > 0
@      ==> (\exists int j; j >= 0 && j <
@*/
public static /*@ pure */ int max(int[] a)
{
    if (a.length == 0) return 0;
    int max = a[0], i = 1;
    /*@ loop_invariant 0 <= i && i <= a.length
     @ loop_invariant (\forall int k; 0 <= k < i \implies max >= a[k])
     @ loop_invariant (\exists int k; 0 <= k < i \implies max = a[k])
     @ decreases a.length - i;
     @ assignable max, i;
    */
    while (i < a.length) {
        if (a[i] > max) max = a[i];
        ++i;
    }
    return max;
}
```

LLMs for Deductive Java Verification

Large Language Models

- May produce output that **is not correct**
- “Reasoning” is **not rigorous**
- **Inconsistent Answers**

LLMs for Deductive Java Verification

Large Language Models

- May produce output that **is not correct**
- “Reasoning” is **not rigorous**
- **Inconsistent Answers**

Deductive Verifiers

- Lack “common sense”
- Symbolic techniques:
Not good at “**guessing**” annotations
from context

LLMs for Deductive Java Verification

Large Language Models

- May produce output that **is not correct**
- “Reasoning” is **not rigorous**
- **Inconsistent Answers**

Deductive Verifiers

- Lack “common sense”
- Symbolic techniques:
Not good at “**guessing**” annotations
from context

Objective: An Intersymbolic AI approach to Program Verification

Combine LLMs and Deductive Verification so that **weaknesses cancel out**

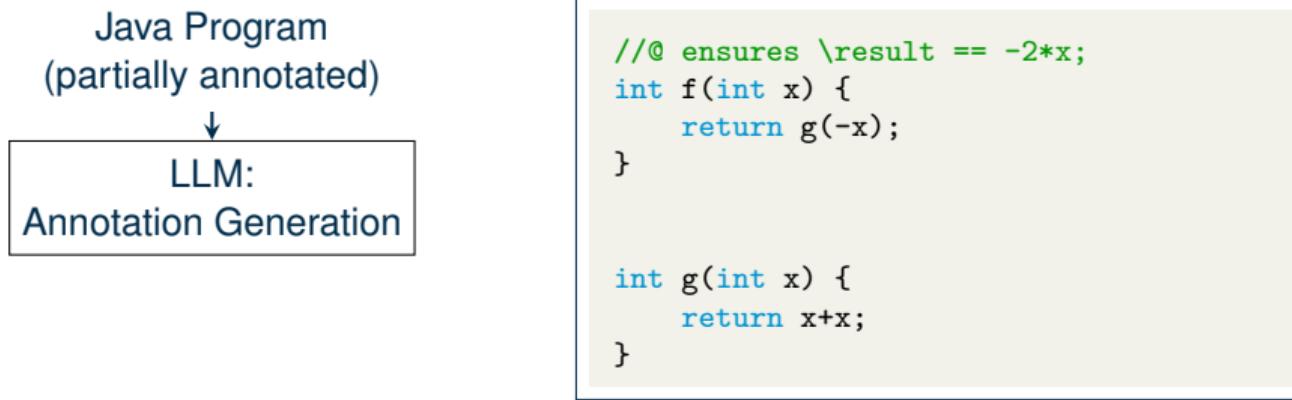
LLM-based generation of JML: Approach

Java Program
(partially annotated)

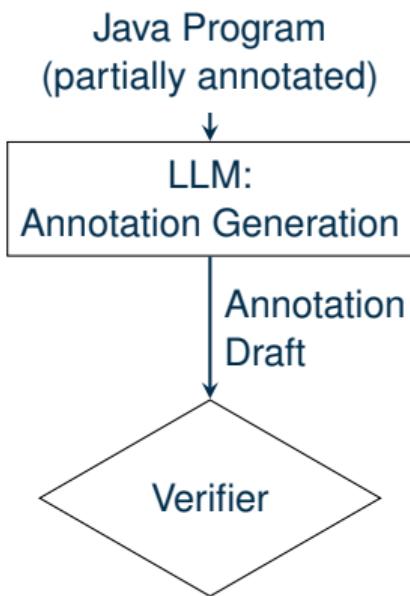
```
//@ ensures \result == -2*x;
int f(int x) {
    return g(-x);
}

int g(int x) {
    return x+x;
}
```

LLM-based generation of JML: Approach



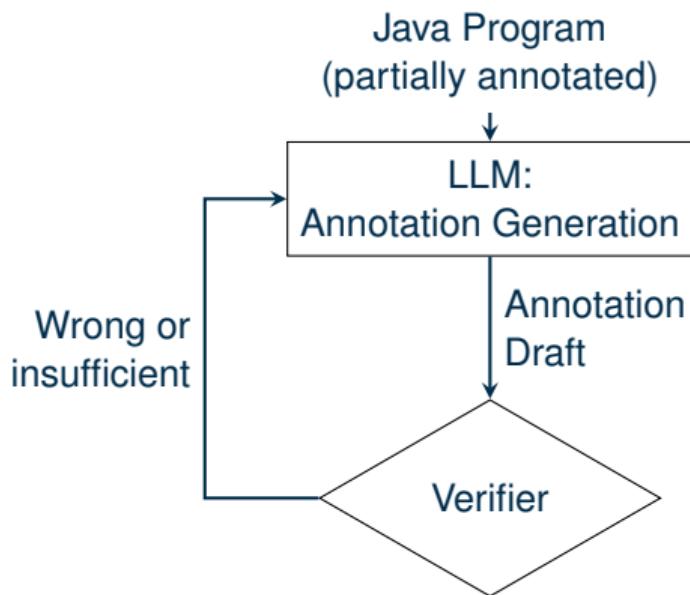
LLM-based generation of JML: Approach



```
//@ ensures \result == -2*x;
int f(int x) {
    return g(-x);
}

//@ ensures x == 2 ==> \result == 4;
int g(int x) {
    return x+x;
}
```

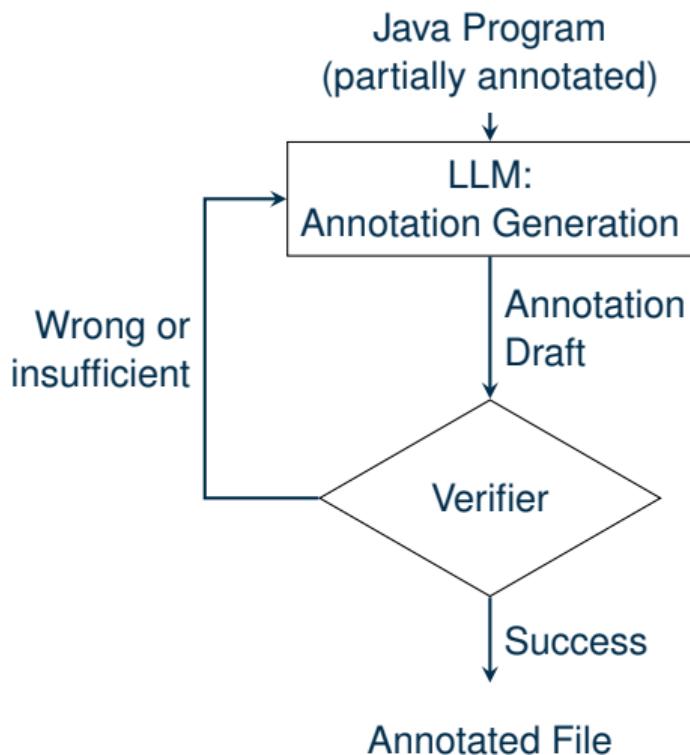
LLM-based generation of JML: Approach



```
//@ ensures \result == -2*x;
int f(int x) {
    return g(-x);
}

//@ ensures x == 2 ==> \result == 4;
int g(int x) {
    return x+x;
}
```

LLM-based generation of JML: Approach



```
//@ ensures \result == -2*x;  
int f(int x) {  
    return g(-x);  
}  
  
//@ ensures \result == 2*x;  
int g(int x) {  
    return x+x;  
}
```

[ISoLA 2024]

Evaluation

Curation of a first, small benchmark set:
KeY repository and old exercise sheets

Evaluation

Curation of a first, small benchmark set:
KeY repository and old exercise sheets

Benchmark Categories:

- Generate top-level contract for isolated method
(Java \Rightarrow JML)

Evaluation

Curation of a first, small benchmark set:
KeY repository and old exercise sheets

Benchmark Categories:

- Generate top-level contract for isolated method
(Java \Rightarrow JML)
- Generate auxilliary annotations for given top-level spec
(Java+JML \Rightarrow JML)
 - Loop invariant (given method contract)
 - Contract of callee-method (given caller contract)

Evaluation Benchmark: Features

	Isolated Methods	Sub-Methods	Invariants	Total	
No. of benchmarks	36	27	14	77	
JML features					
Quantifiers	32	24	14	70	91%
Non-empty assignable	24	17	11	52	68%
Reference to pre-state (<code>\old</code>)	18	11	8	37	48%
(Pure) Method calls in spec	10	7	7	24	31%
Java features					
Arrays	30	23	13	66	86%
Field access	11	7	3	21	27%

Evaluation

Curation of a first, small benchmark set:
KeY repository and old exercise sheets

Benchmark Categories:

- Generate top-level contract for isolated method
(Java \Rightarrow JML)
- Generate auxilliary annotations for given top-level spec
(Java+JML \Rightarrow JML)
 - Loop invariant (given method contract)
 - Contract of callee-method (given caller contract)

Category	Instances	$\mu \pm \sigma$ of success rate (%)	
		GPT 3.5 (n = 10)	GPT 4o (n = 3)
Isolated Method	36	52.2 \pm 4.3	62.0 \pm 1.6
Submethods	14	19.3 \pm 12.1	40.5 \pm 4.1
Invariants	27	37.0 \pm 7.4	67.9 \pm 5.7

Evaluation

Curation of a first, small benchmark set:
KeY repository and old exercise sheets

Benchmark Categories:

- Generate top-level contract for isolated method
(Java \Rightarrow JML)
- Generate auxilliary annotations for given top-level spec
(Java+JML \Rightarrow JML)
 - Loop invariant (given method contract)
 - Contract of callee-method (given caller contract)

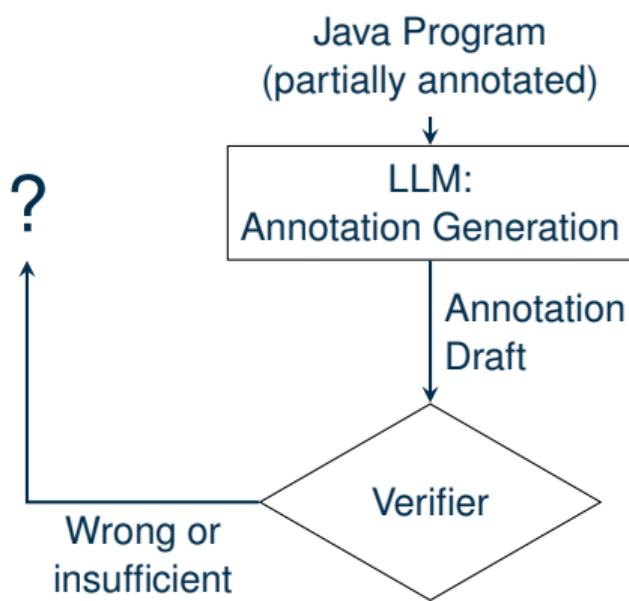
Category	Instances	$\mu \pm \sigma$ of success rate (%)	
		GPT 3.5 (n = 10)	GPT 4o (n = 3)
Isolated Method	36	52.2 \pm 4.3	62.0 \pm 1.6
Submethods	14	19.3 \pm 12.1	40.5 \pm 4.1
Invariants	27	37.0 \pm 7.4	67.9 \pm 5.7

No elaborate prompt engineering yet!

Objective: Evaluate baseline performance

Not enough benchmarks

LLM-based generation of JML: Approach



```
//@ ensures \result == -2*x;  
int f(int x) {  
    return g(-x);  
}  
  
//@ ensures x == 2 ==> \result == 4;  
int g(int x) {  
    return x+x;  
}
```

What steps to take?

Grasping Feedback

We can provide feedback on failed proof attempts,
but **does this benefit** the specification generation?

Grasping Feedback

We can provide feedback on failed proof attempts,
but **does this benefit** the specification generation?

Concrete Counterexamples

Via bounded model checking

Grasping Feedback

We can provide feedback on failed proof attempts,
but **does this benefit** the specification generation?

Concrete Counterexamples Via bounded model checking

The provided invariant **does not hold for the following instantiations of variables:**

k = 0
a = [0, 0, 0]
i = 1

You're right; we still need to refine the loop invariant to handle the specific case where **the array length is odd**. In such cases, the middle element should remain unchanged.

...

Here's the corrected code:

<wrong invariant>

Grasping Feedback

We can provide feedback on failed proof attempts,
but **does this benefit** the specification generation?

Concrete Counterexamples Via bounded model checking

The provided invariant **does not hold** for the following instantiations of variables:

k = 0
a = [0, 0, 0]
i = 1

You're right; we still need to refine the loop invariant to handle the specific case where **the array length is odd**. In such cases, the middle element should remain unchanged.

...

Here's the corrected code:
<wrong invariant>

Feedback from Proof State

- Pass on parser errors
- Pass on information on open proof branches

Grasping Feedback

We can provide feedback on failed proof attempts,
but **does this benefit** the specification generation?

Concrete Counterexamples Via bounded model checking

The provided invariant **does not hold** for the following instantiations of variables:

k = 0
a = [0, 0, 0]
i = 1

You're right; we still need to refine the loop invariant to handle the specific case where **the array length is odd**. In such cases, the middle element should remain unchanged.

...

Here's the corrected code:
<wrong invariant>

Feedback from Proof State

- Pass on parser errors
- Pass on information on open proof branches

Focus: Auxilliary Specifications

Sampling (pass@N) vs. Feedback

10 rounds with GPT 4o

Grasping Feedback

We can provide feedback on failed proof attempts,
but **does this benefit** the specification generation?

Concrete Counterexamples Via bounded model checking

The provided invariant **does not hold** for the following instantiations of variables:

k = 0
a = [0, 0, 0]
i = 1

You're right; we still need to refine the loop invariant to handle the specific case where **the array length is odd**. In such cases, the middle element should remain unchanged.

...

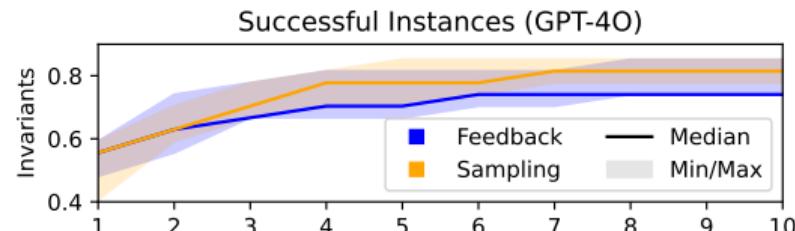
Here's the corrected code:
<wrong invariant>

Feedback from Proof State

- Pass on parser errors
- Pass on information on open proof branches

Focus: Auxilliary Specifications

Sampling (pass@N) vs. Feedback
10 rounds with GPT 4o



Grasping Feedback

We can provide feedback on failed proof attempts,
but **does this benefit** the specification generation?

Concrete Counterexamples
Via **bounded model checking**

The provided invariant **does not hold** for the following instantiations of variables:

$k = 0$
 $a = [0, 0]$
 $i = 1$

You're invariant to **length**.
...
should remain unchanged.

Here's the corrected code:
<wrong invariant>

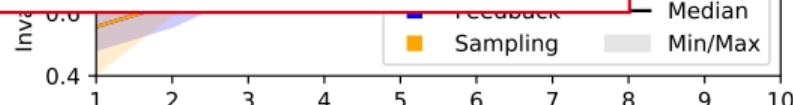
Feedback from Proof State

- Pass on parser errors
- Pass on information on open proof branches

Focus: Auxilliary Specifications

We need more benchmarks for conclusive results

Prompt Engineering for proof state descriptions requires more examples



Feedback vs. Sampling

What is the right metric?

Classic Verification

Two verification techniques:

- (A) 5 iterations, 2 seconds CPU time/iteration
- (B) 3 iterations, 4 seconds CPU time/iteration

⇒ **(A) is better**

Feedback vs. Sampling

What is the right metric?

Classic Verification

Two verification techniques:

- (A) 5 iterations, 2 seconds CPU time/iteration
- (B) 3 iterations, 4 seconds CPU time/iteration

⇒ **(A) is better**

API usage **hides** computational cost!

Feedback vs. Sampling

What is the right metric?

Classic Verification

Two verification techniques:

- (A) 5 iterations, 2 seconds CPU time/iteration
 - (B) 3 iterations, 4 seconds CPU time/iteration
- ⇒ (A) is better

API usage **hides** computational cost!

What we know:

Computational cost increases with token count

- Initial Query: I tokens
- LLM Output: O tokens
- Feedback: F tokens

Sampling: $n(I + O) \in \mathcal{O}(n)$ tokens

Feedback: $n(I + O) + \frac{n(n-1)}{2} (O + F) \in \mathcal{O}(n^2)$ tokens

Feedback vs. Sampling

What is the right metric?

Classic Verification

Two verification techniques:

- (A) 5 iterations, 2 seconds CPU time/iteration
 - (B) 3 iterations, 4 seconds CPU time/iteration
- ⇒ (A) is better

API usage **hides** computational cost!

What we know:

Computational cost increases with token count

- Initial Query: I tokens
- LLM Output: O tokens
- Feedback: F tokens

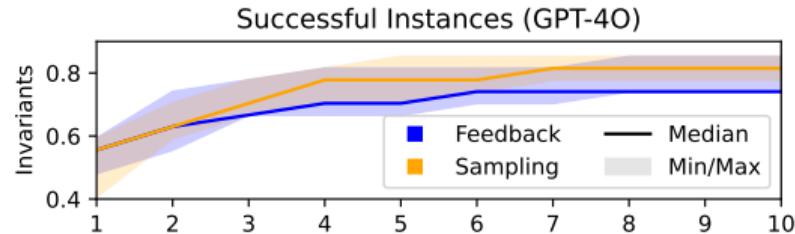
Sampling: $n(I + O) \in \mathcal{O}(n)$ tokens

Feedback: $n(I + O) + \frac{n(n-1)}{2} (O + F) \in \mathcal{O}(n^2)$ tokens

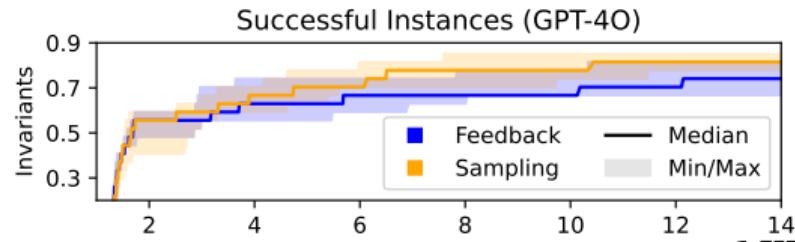
Another evaluation:

Iterations → Normalized Token Count ($\frac{\# \text{Tokens}}{I}$)

Iteration based:



Token based:



Related Work

This is a **rapidly** growing research field

Related Work

This is a **rapidly** growing research field

- A lot of work on **Dafny** (e.g. due to DafnyBench)

Loughridge u. a. 2025

Related Work

This is a **rapidly** growing research field

- A lot of work on **Dafny** (e.g. due to DafnyBench) Loughridge u. a. 2025
- AUTOSPEC for C and ACSL with Frama C Wen u. a. 2024
 - Beyond *filling the gap*:
Strategy for generating all method/invariant annotations
 - No proof state feedback for individual annotations
 - Also use ChatGPT API

Related Work

This is a **rapidly** growing research field

- A lot of work on **Dafny** (e.g. due to DafnyBench) Loughridge u. a. 2025
- AUTOSPEC for C and ACSL with Frama C
 - Beyond *filling the gap*:
Strategy for generating all method/invariant annotations
 - No proof state feedback for individual annotations
 - Also use ChatGPT APIWen u. a. 2024
- Joint generation of code and specification Sun u. a. 2024
 - No soundness guarantees
 - Empirically: Consistency between code and spec helps

Related Work

This is a **rapidly** growing research field

- A lot of work on **Dafny** (e.g. due to DafnyBench) Loughridge u. a. 2025
- AUTOSPEC for C and ACSL with Frama C
 - Beyond *filling the gap*:
Strategy for generating all method/invariant annotations
 - No proof state feedback for individual annotations
 - Also use ChatGPT APIWen u. a. 2024
- Joint generation of code and specification Sun u. a. 2024
 - No soundness guarantees
 - Empirically: Consistency between code and spec helps
- Dataset for JML via GitHub scraping Greiner u. a. 2024

Demo
Backup

Conclusion

- OpenAI's models are **surprisingly good** at JML
- **Foundation Models:**
 - Make effective usage of ML for niche languages like JML possible
 - Cannot solve all data problems (we still need well-curated benchmark sets)
- Iteration helps, but feedback not (yet)

Open Questions

- Effective Feedback from the verifier's proof state
- Prompt Engineering in the presence of “*perfect checkers*”
Sampling may not be that bad afterall?
- What is a fair comparison between an LLM approach and a “classic” approach?

References I

- [1] Wolfgang Ahrendt u. a., Hrsg. *Deductive Software Verification - The KeY Book - From Theory to Practice*. Bd. 10001. LNCS. Cham: Springer, 2016. ISBN: 978-3-319-49811-9. DOI: [10.1007/978-3-319-49812-6](https://doi.org/10.1007/978-3-319-49812-6).
- [2] Bernhard Beckert. "A dynamic logic for the formal verification of Java Card programs". In: *International Java Card Workshop*. Springer, 2000, S. 6–24.
- [3] Bernhard Beckert und André Platzer. "Dynamic Logic with Non-rigid Functions: A Basis for Object-oriented Program Verification.". In: *IJCAR*. Hrsg. von Ulrich Furbach und Natarajan Shankar. Bd. 4130. LNCS. ISSN: 0302-9743. Springer, 2006, S. 266–280. ISBN: 3-540-37187-7. DOI: [10.1007/11814771_23](https://doi.org/10.1007/11814771_23).
- [4] Sandra Greiner u. a. "Automated Generation of Code Contracts: Generative AI to the Rescue?" In: *Proceedings of the 23rd ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*. GPCE '24. Pasadena, CA, USA: Association for Computing Machinery, 2024, S. 1–14. ISBN: 9798400712111. DOI: [10.1145/3689484.3690738](https://doi.org/10.1145/3689484.3690738). URL: <https://doi.org/10.1145/3689484.3690738>.
- [5] K Rustan M Leino und Michał Moskal. "Usable auto-active verification". In: *Usable Verification Workshop*. Hrsg. von Tom Ball, Lenore Zuck und Natarajan Shankar. 2010. URL: <https://fm.csl.sri.com/UV10>.
- [6] Chloe R Loughridge u. a. "DafnyBench: A Benchmark for Formal Software Verification". In: *Transactions on Machine Learning Research* (2025). ISSN: 2835-8856. URL: <https://openreview.net/forum?id=yBgTVWcclx>.
- [7] Chuyue Sun u. a. "Clover: Closed-Loop Verifiable Code Generation". In: *AI Verification - First International Symposium, SAIV 2024, Montreal, QC, Canada, July 22-23, 2024, Proceedings*. Hrsg. von Guy Avni u. a. Bd. 14846. Lecture Notes in Computer Science. Springer, 2024, S. 134–155. DOI: [10.1007/978-3-031-65112-0_7](https://doi.org/10.1007/978-3-031-65112-0_7). URL: https://doi.org/10.1007/978-3-031-65112-0%5C_7.

References II

- [8] Cheng Wen u. a. "Enchanting program specification synthesis by large language models using static analysis and program verification". In: *International Conference on Computer Aided Verification*. Springer. 2024, S. 302–328.

Demo (Backup)

[Back to Slides](#)

Example: Find the Maximum

```
/*@ normal_behavior
 @ ensures (\forall int j; j >= 0 && j < a.length; \result >= a[j]);
 @ ensures a.length > 0 ==>
 @ (\exists int j; j >= 0 && j < a.length; \result == a[j]);
 @ */
public static /*@ pure */ int max(int[] a) {
    if (a.length == 0) return 0;
    int max = a[0], i = 1;
    while (i < a.length) {
        if (a[i] > max) max = a[i];
        ++i;
    }
    return max;
}
```

Example: Find the Maximum (1st try)

Given the following Java class:

[...]

Please provide a loop invariant for the loop construct with the comment '`//Add invariant here`' of the method 'max'. Beware this annotation has to be a loop invariant. A loop invariant typically has the following structure:
[Template: **loop_invariant, decreases, assignable**]

Based on the method 'max', the goal is to find the maximum element in the given array 'a'. [...]

```
/*@ loop_invariant 0 <= i && i <= a.length;
 @ loop_invariant (\forall int k; 0 <= k && k < i; max
     >= a[k]);
 @ decreases a.length - i;
 @ assignable i, max;
 @ */
```

Problem: `max` not required to be in the array

Example: Find the Maximum (2nd try)

The provided JML does not solve the task. [...] **some proof goals were not closed:**

[...]

During verification, the following proof branches could not be closed:

Normal Execution (`_a != null`)

Please fix the JML loop invariant.

Let's refine the loop invariant [...]

```
/*@ loop_invariant a != null;
 @ loop_invariant 0 <= i && i <= a.length;
 @ loop_invariant (\forall int k; 0 <= k && k < i; max
      >= a[k]);
 @ decreases a.length - i;
 @ assignable i, max;
 @*/

```

Problem: same (only added `a != null`)

Example: Find the Maximum (3rd try)

[Same message]

To refine the loop invariant further [...]

```
/*@ loop_invariant a != null;
 @ loop_invariant 0 <= i && i <= a.length;
 @ loop_invariant (\forall int k; 0 <= k && k < i; max
                  >= a[k]);
 @ loop_invariant (i > 0) ==>
 @ (\exists int k; 0 <= k && k < i; max == a[k]);
 @ decreases a.length - i;
 @ assignable i, max;
 @*/

```

Invariant that is strong enough to prove the post-condition.

Problem: Invariant is not inductive (because of $i > 0$)

Example: Find the Maximum (4th try)

[Same message]

Let's ensure that the JML loop invariant comprehensively captures [...]

```
/*@ loop_invariant a != null;
 @ loop_invariant 0 <= i && i <= a.length;
 @ loop_invariant (\forall int k; 0 <= k && k < i; max
                  >= a[k]);
 @ loop_invariant
 @ \exists int m; m >= 0 && m < i && max == a[m];
 @ decreases a.length - i;
 @ assignable i, max;
 @ */
```

Correct invariant

Syntax error: ACSL `\exists ...` vs. JML `(\exists ...)`

Example: Find the Maximum (5th try)

The provided code is not valid JML.

[...]

Error during JML parsing: Failed to parse JML fragment: **Encountered unexpected token: “\exists” “\exists” at line 4, column 20.**

Was expecting one of: [...]

[unfortunately still fails]

Example: Find the Maximum (restart)

Second Attempt...

[Same initial question]

In the given 'max' method, the loop is responsible for [...]

```
/*@ loop_invariant 0 <= i && i <= a.length;
 @ loop_invariant (\forall int k; 0 <= k && k < i; max
                  >= a[k]);
 @ decreases a.length - i;
 @ assignable max, i;
 */
```

Problem: `max` not required to be in array.

[Same feedback]

[Correct answer]

More Difficult Example: Rotate an Array

```
public static void rotate(int[] a, int len) {  
    int[] b = new int[a.length];  
    int i = 0;  
    /*@ [...] @*/  
    for (i = 0; i < len; i++) {  
        b[i] = a[a.length - len + i];  
    }  
    /*@ [...] @*/  
    for (i = len; i < a.length; i++) {  
        b[i] = a[i - len];  
    }  
    /*@ [...] @*/  
    for (i = 0; i < a.length; i++) {  
        a[i] = b[i];  
    }  
}
```

More Difficult Example: Rotate an Array

Correct specification generated by GPT 4o

```
/* @ normal_behavior
 *
 * @ requires a != null && 0 <= len && len <= a.length;
 *
 * @ assignable a[*];
 *
 * @ ensures (\forall int i; 0 <= i && i < len;
 *           a[i] == \old(a[a.length - len + i]));
 * @ ensures (\forall int i; len <= i && i < a.length;
 *           a[i] == \old(a[i - len]));
 */

```

Demo (Backup)

[Back to Slides](#)

Evaluation: Isolated Methods

Category	# Benchmarks	$\mu \pm \sigma$ of success rate (%)	
		GPT 3.5 (n = 10)	GPT 4o (n = 3)
Isolated Method	36	52.2 \pm 4.3	62.0 \pm 1.6

Success criterion: KeY proves code satisfies generated spec

Manual Inspection: Spec adequately specifies code behavior

Sometimes incomplete

Repetition/Feedback helps

- 75% of benchmarks successful
(over 10 runs, GPT 3.5)
- Feedback from the verifier can help

