**Hochschule Karlsruhe**
University of
Applied Sciences

Fakultät für
**Informatik und**
**Wirtschaftsinformatik**

# Statische Programmanalyse: Werkzeuge für C

Mika Amann

# Was ist statische Programmanalyse?

## Analyse von Quellcode – ohne ihn auszuführen

# Competition on Software Verification (SV-Comp)

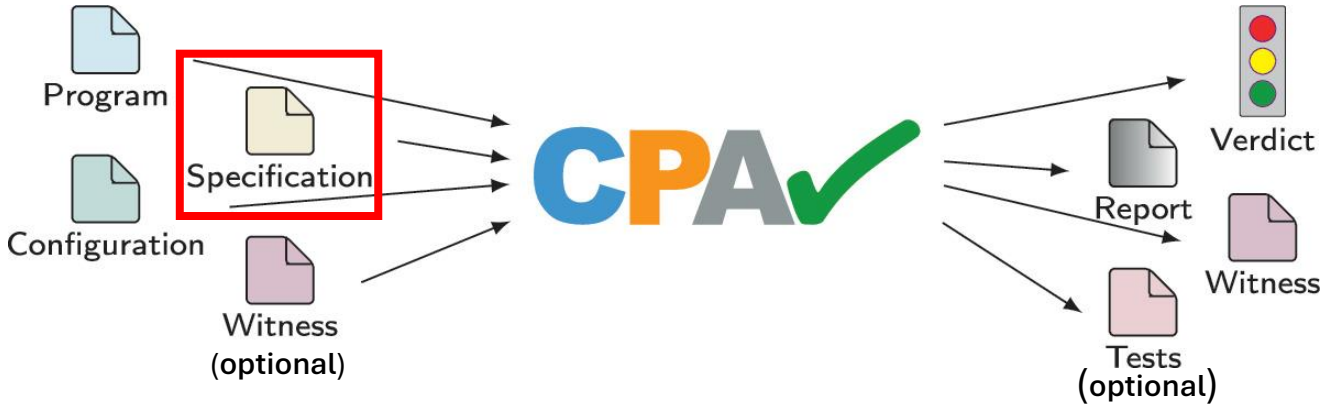
# Formale Verifikation vs. Sprachmodell


vs.

# Configurable Program Analysis (CPA)

# Configurable Program Analysis (CPA)

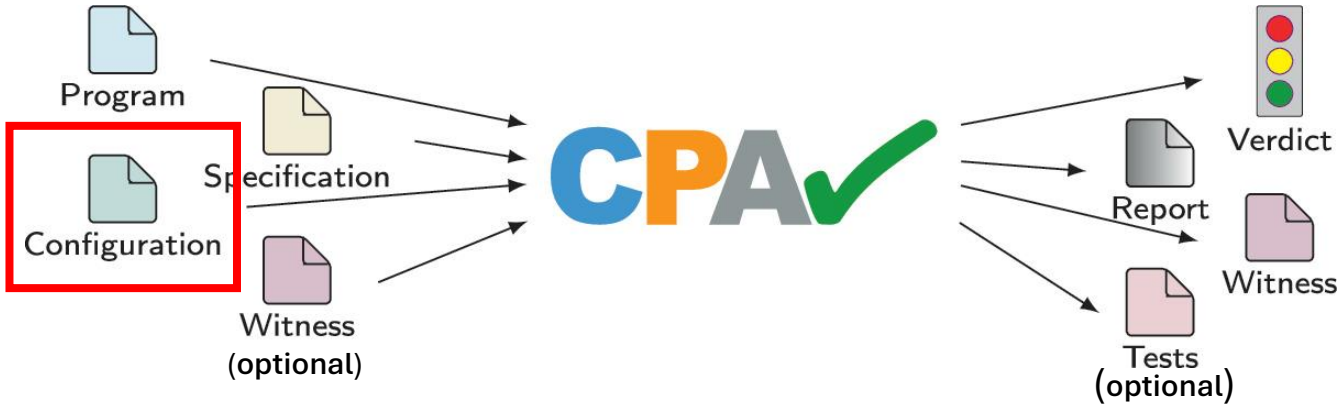
# Specification




## Welcher Fehlertyp wird gesucht?

- Overflows
- Memory Safety & Memory Cleanup
- Concurrency Safety
- Reachability
- …

# Configuration




## Wie soll der Code Analysiert werden?

- Value Analysis

- Predicate Analysis

# Verdict


## Ist der Code entsprechend der Spezifikation fehlerfrei?

- TRUE
- FALSE
- UNKNOWN
- ERROR

# Report


## Was ist das Ergebnis der Analyse?

- Zusammenfassung der Analyse
- Informationen und Statistiken für Entwickler

# Witness

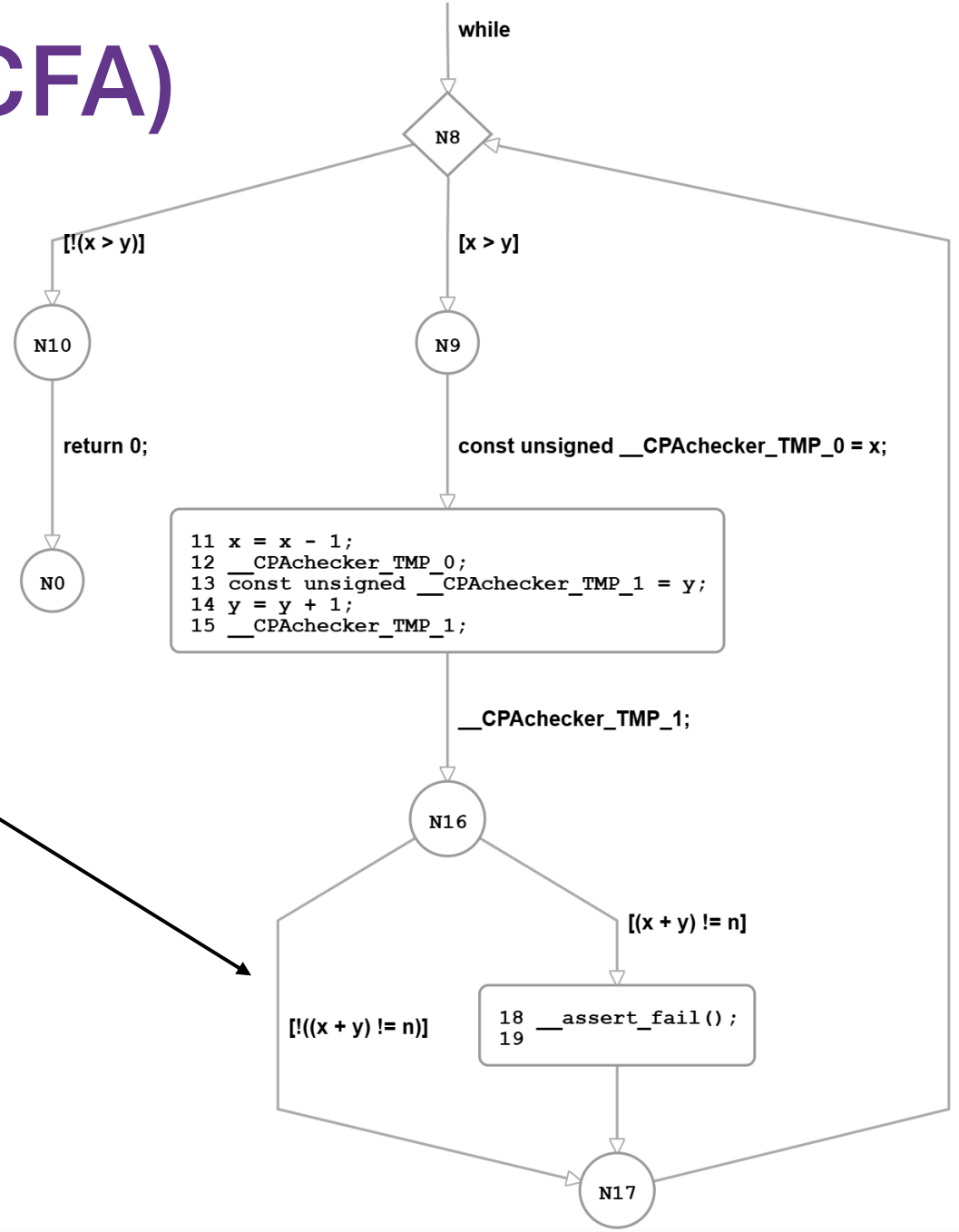

## Wie wird der Fehler belegt?

- Maschinell überprüfbarer Beweis
- Enthält konkreten Fehlerpfad

# Kontrollfluss-Automat (CFA)

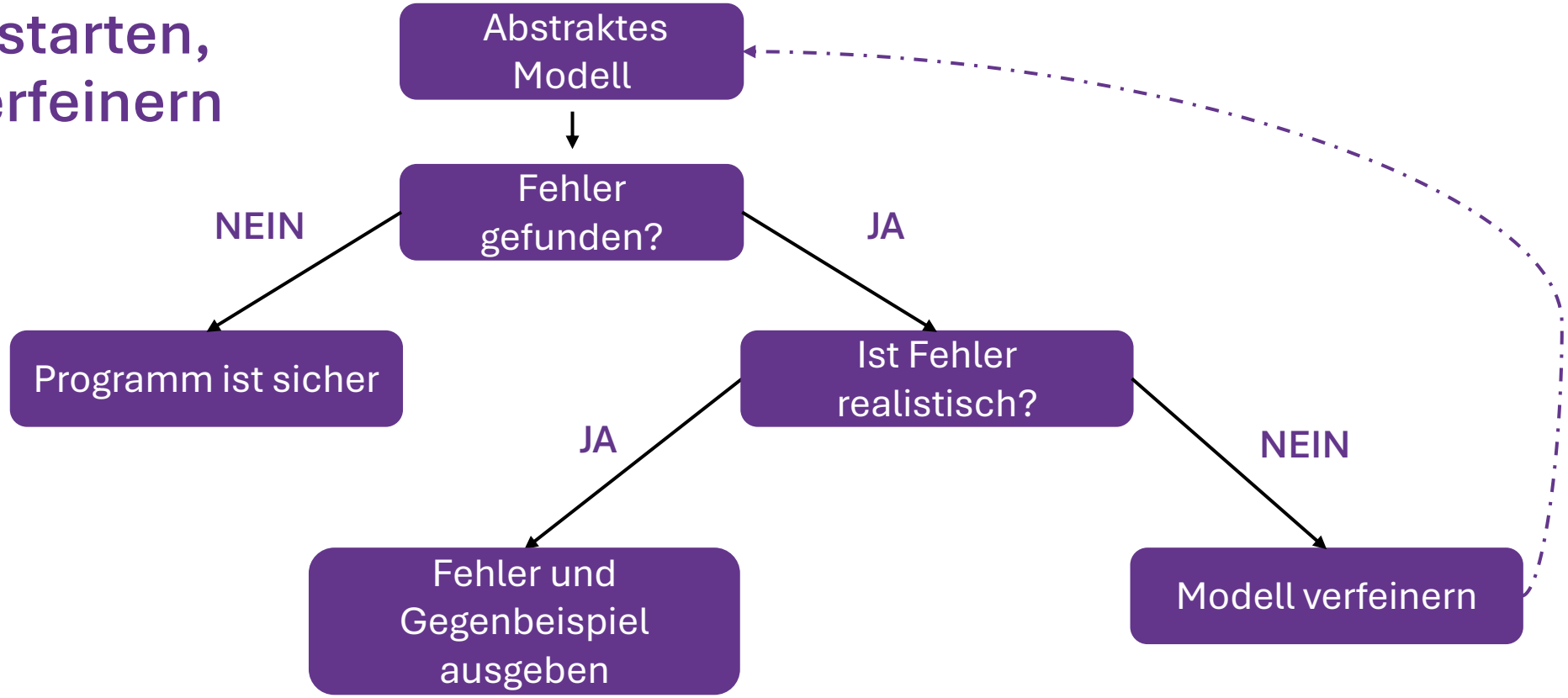
## Knoten: Programmpositionen

## Kanten: Programmoperationen



# Counterexample-Guided Abstraction Refinement (CEGAR)

**Abstrakt starten, gezielt verfeinern**



## Settings

**CPAchecker revision**

cpachecker-3.0

**Configuration**

predicateAnalysis

**Properties/Specification** CPAchecker | sv-repo | plaintext | file

sv-benchmarks.no-overflow

**Program File** plaintext | file

```
  unsigned y = 0;
  while(x > n) {
    x--; n++;
    y = n - x;
  }
  return 0;
}
```

**Witness Validation** on | off

☐ *I hereby grant to the operators of this service the unrestricted and permanent right to store, use, copy, publish and distribute all data and files that I submit to this service. I assure that I have the necessary rights on all the material to do so and agree to the BenchCloud's terms of service.*

Submit run

## Options

☐ Disable output files

**Log level**

INFO

**Machine model**

Linux32

**Additional options**

**Time limitation**

05min 00s

**Memory limitation**

2.0 GB

**CPU core limitation**

2

Link zum Tool

## Run id: 43687f9f-0de5-4df3-8a54-6322a281de7b

| | |
|---|---|
| Result: | FALSE |
| Message: | FALSE. Property violation (no-overflow: integer overflow in line 8) found by chosen configuration. |

Download run result

View report output/Counterexample.1.html

## Run description

| | |
|---|---|
| cores | 4 |
| cpuModel | AMD EPYC 7443 24-Core |
| frequency | 2 GHz |
| memory | 10428407808 byte |
| name | benchcloud-worker01 |
| os | Linux 6.8.0-62-generic |
| turboboost-enabled | false |
| turboboost-supported | false |

## Host information

| | |
|---|---|
| cores | 4 |
| cpuModel | AMD EPYC 7443 24-Core |
| frequency | 2 GHz |

## Run information

| | |
|---|---|
| pressure-memory-some | 0s |
| pressure-cpu-some | 0.588307s |
| additionalEnvironment | |
| debug | false |
| memory | 155574272B |
| coreLimit | 2 |
| blkio-write | 0B |
| exitcode | 0 |
| starttime | 2025-06-26T10:58:15.872284+02:00 |
| generatedFilesCount | 33 |
| walltime | 1.935086346929893s |
| newEnvironment | |
| timeLimit | 05min 00s |
| pressure-io-some | 0s |
| matchedResultFilesCount | 33 |
| memoryLimit | 1000000000 |
| cputime | 3.408413s |
| runId | 43687f9f-0de5-4df3-8a54-6322a281de7b |
| blkio-read | 0B |

## Individual output files

- output/ARG.dot
- output/ARG.svg
- output/ARGRefinements.dot
- output/ARGSimplified.dot
- output/CPALog.txt
- output/CPALog.txt.lck
- output/Counterexample.1.assignment.txt
- output/Counterexample.1.c
- output/Counterexample.1.core.txt
- output/Counterexample.1.dot
- output/Counterexample.1.graphml.gz
- output/Counterexample.1.harness.c
- output/Counterexample.1.html
- output/Counterexample.1.smt2
- output/Counterexample.1.spc
- output/Counterexample.1.txt
- output/Counterexample.1.witness-2.0.yml
- output/Counterexample.1.witness.dot.gz
- output/Statistics.txt
- output/UsedConfiguration.properties
- output/VariableClassification.log
- output/VariableTypeMapping.txt
- output/abstractions.txt
- output/cfa.dot
- output/cfa/cfa__main.dot

Prev Start Next

Search for...

Find only exact matches

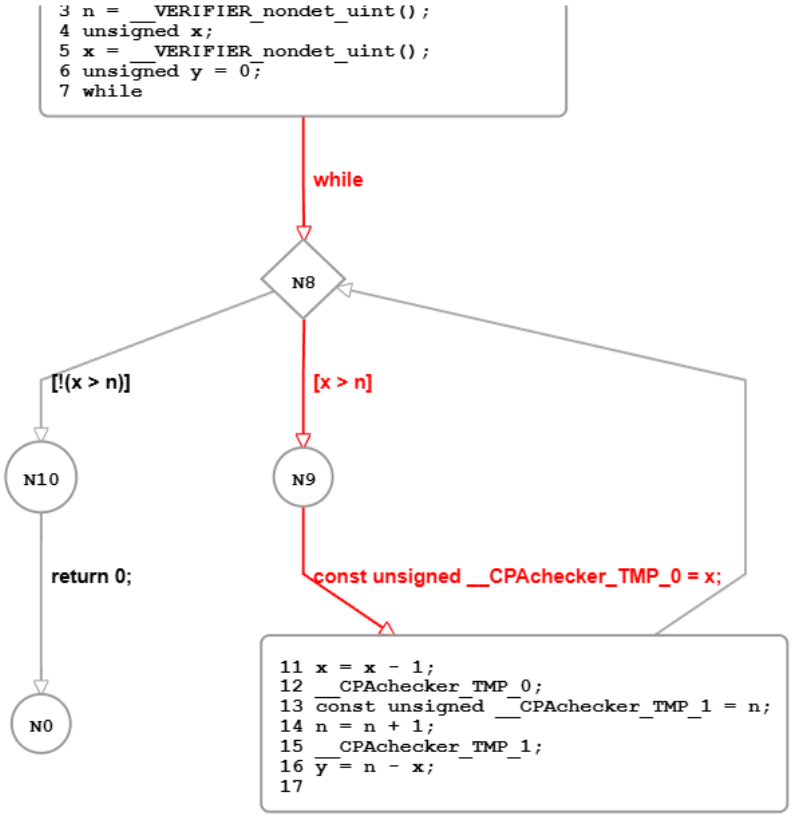
```
INIT GLOBAL VARS
unsigned int __VERIFIER_nondet_uint(
int main();
   unsigned n;
-V-   n = __VERIFIER_nondet_uint();
-V-   unsigned x;
-V-   x = __VERIFIER_nondet_uint();
-V-   unsigned y = 0;
-V-   while
-V-   [x > n]
-V-   const unsigned __CPAchecker_TMP_0
-V-   x = x - 1;
-V-   __CPAchecker_TMP_0;
-V-   const unsigned __CPAchecker_TMP_1
-V-   n = n + 1;
-V-   __CPAchecker_TMP_1;
-V-   y = n - x;
```

n: 1U
x: 2U
y: 4294967295U

Show Error-Path Section | CFA | ARG | Source | Log | Statistics | Configurations | About

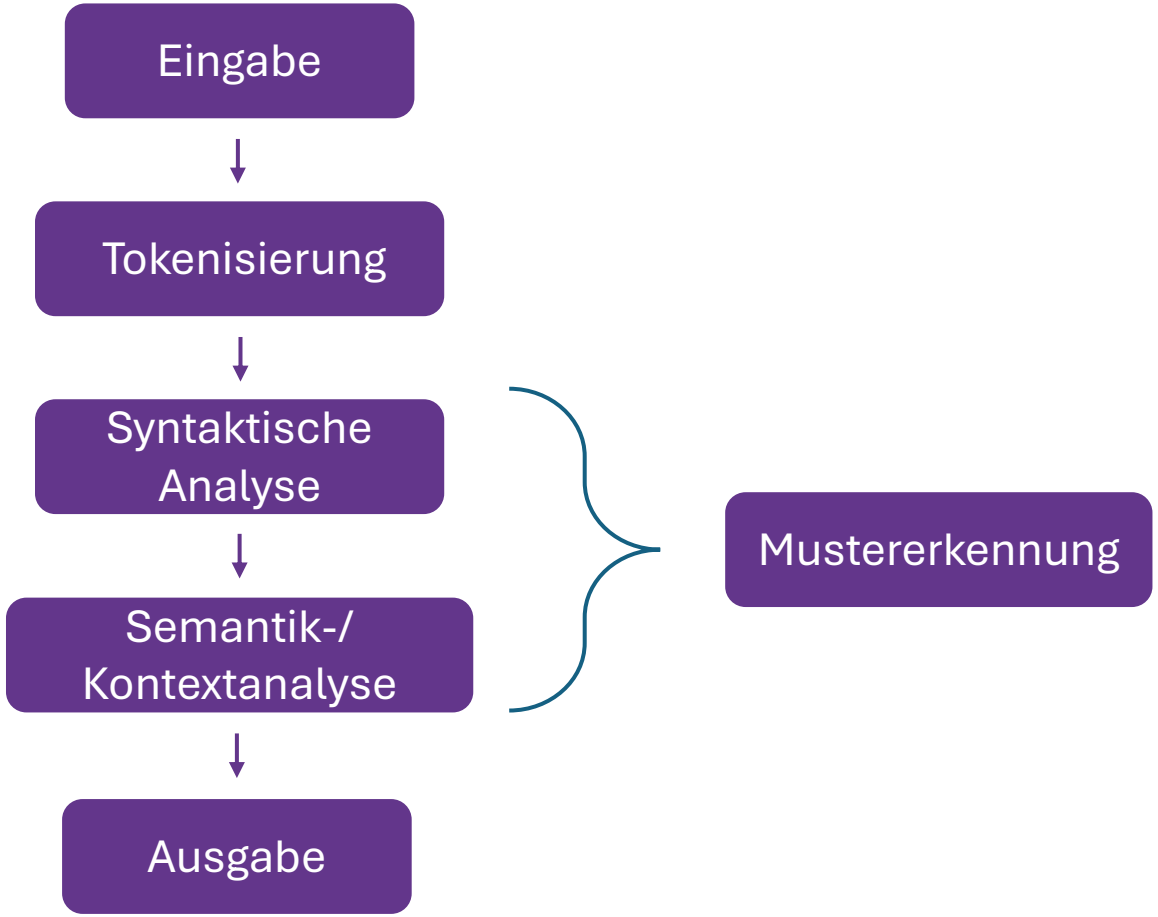
Full Screen Mode

Displayed CFA: main | Mouse Wheel Zoom | Split Threshold

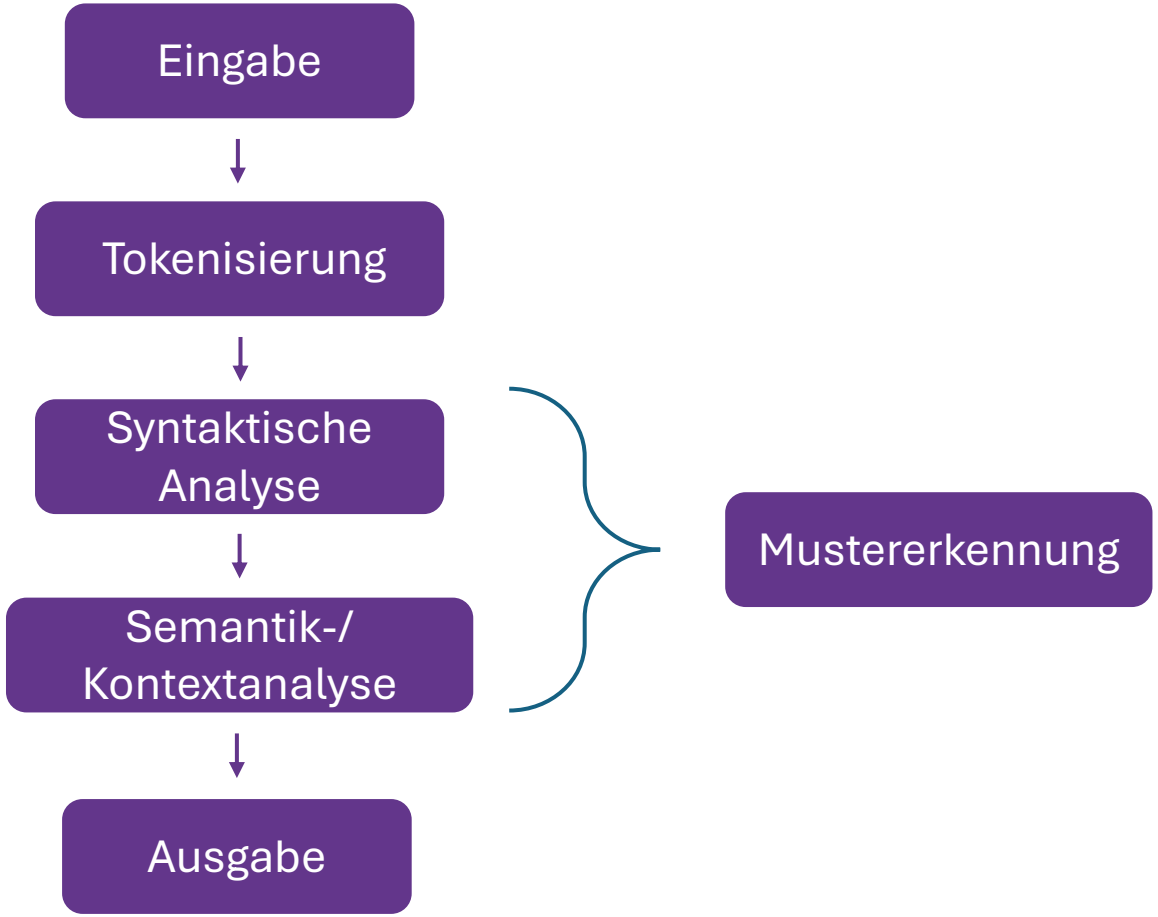


# Funktionsweise eines LLM bei der Codeanalyse

# Funktionsweise eines LLM bei der Codeanalyse



# Funktionsweise eines LLM bei der Codeanalyse



# LLM Codeanalyse angewandt

```c
int main() {
    int age = 25;
    printf("Your age is: %s\n", age);  // Fehler: %s statt %d
    return 0;
}
```

# Tokenisierung und Syntaxanalyse

```
[
"int", "main", "(", ")", "{",
 "int", "age", "=", "25", ";",
 "printf", "(", "\"Your age is: %s\\n\"", ",", "age", ")", ";",
 "return", "0", ";",
 "}"
]
```

- int main() {...} → Hauptfunktion
- int age = 25; → gültige Deklaration einer Ganzzahl
- printf(...) → bekannter Funktionsaufruf aus C-Standardbibliothek

=> Keine Syntaxfehler

# Semantik- und Kontextanalyse

- Kontextanalyse: „Was soll der Code tun?“
  - Deklaration einer Variable „age“ mit Wert „25“ und anschließender Ausgabe

- Semantikanalyse: „Tut der Code das was er soll?“
  - Versuch einen Integer mit printf(...) und %s auszugeben
  - %s erwartet einen String
  - Für Integer %d erwartet ...

  => Semantisch widersprüchlich!

# Aufbau der Teststudie

## Repo der SV-COMP

## ~ 650 Codebeispiele

**=> 33.353 Testdurchläufe verteilt auf 6 Kategorien**

# Kategorien

Reach Safety

Memory Safety

Concurrency Safety

No Overflows

Termination

Software Systems

# LLM Prompt

Analysiere den folgenden Code auf **formale**, **semantische** oder **sicherheitsrelevante** Fehler. Gib ein Verdict zurück:

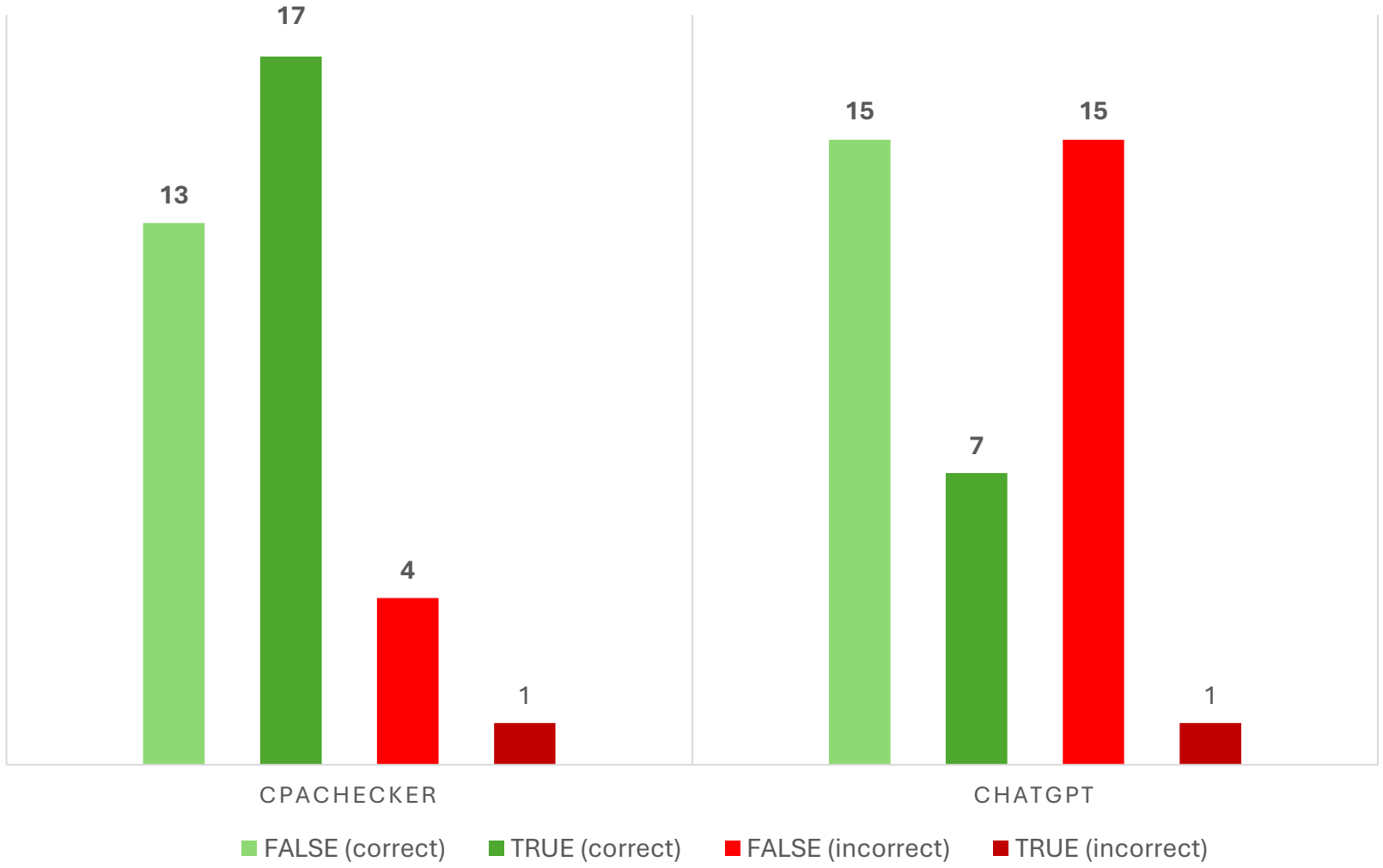
**TRUE**, wenn der Code fehlerfrei ist

**FALSE**, wenn du einen oder mehrere Fehler erkennst

**UNKNOWN**, wenn du dir unsicher bist oder nur spekulierst

Halte deine Antwort exakt in diesem Format und bleibe bei der Begründung deines Verdicts kurz.

# Testresultate



# Punktevergabe nach SV-Comp-Regelwerk

| Ergebnis | Punktzahl |
|---|---|
| FALSE (correct) | + 1 |
| TRUE (correct) | + 2 |
| FALSE (incorrect) | - 16 |
| TRUE (incorrect) | - 32 |
| UNKNOWN | 0 |

# Testresultate

## CPAchecker

**-49**

**(~ 76,9%)**

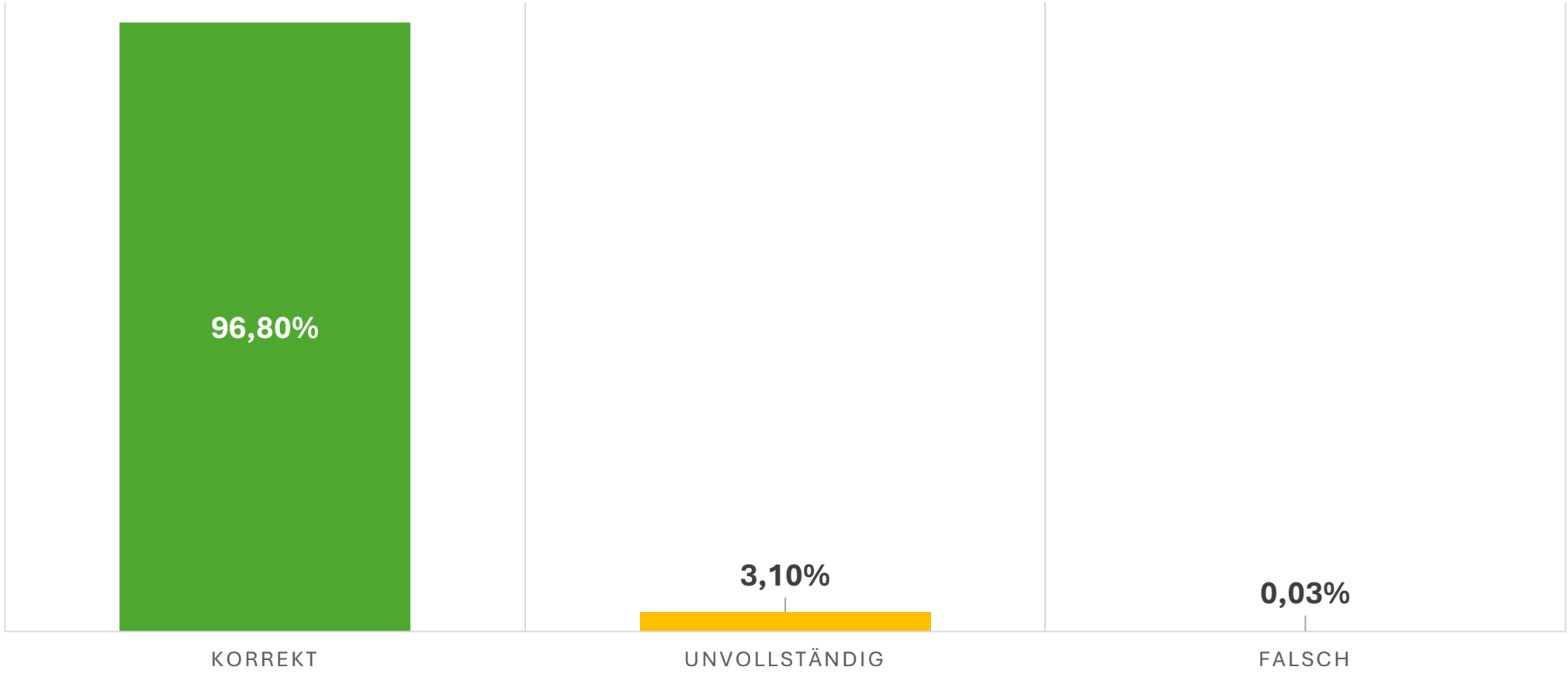
## ChatGPT

**-243**

**(~ 56,4%)**

**Ergebnisse dienen nur dem internen Vergleich und sind nicht repräsentativ für reale Tool-Leistung**

# CPAchecker bei SV-Comp 2025



# Bewertung CPAchecker

- Verlässlichkeit und Präzision
- Formaler Beweis (Witness)
- Verarbeitung großer Codebasen

- Funktioniert primär nur für C - Code
- Bibliothekscode oft nicht prüfbar
- Einarbeitungsaufwand

# Bewertung ChatGPT

- Erkennt viele Sprachen, Bibliotheken und Frameworks
- Intuitive Verwendung
- Flexibel Einsetzbar

- Keine Garantien
- False Positives und Halluzinationen
- Begrenztes Kontextfenster

# Vielen Dank für Ihre Aufmerksamkeit!

## Gibt es noch Fragen?

# Beispiel

```
1   extern unsigned __VERIFIER_nondet_uint();
2   int main() {
3     unsigned n =  __VERIFIER_nondet_uint();
4     unsigned x =  __VERIFIER_nondet_uint();
5     unsigned y = 0;
6     while(x > n) {
7       x--; n++;
8       y = n - x;
9     }
10    return 0;
11  }
```

# Transformer Architektur

- Ziel:
  - Eingaben verstehen & passende Ausgaben erzeugen
- Self Attention
  - Jeder Token wird unter Berücksichtigung aller anderen Tokens verarbeitet
  - Erkennung von Zusammenhängen in Sequenzen
- Resultat
  - LLM erkennt Bedeutungen und Zusammenhänge

# Training

- Training auf riesigen Text- und Code-Datenmengen
  - Texte, Dokumentationen, Quellcodes

- Feinabstimmung auf Dialoge & Anweisungen (RLHF)
  - Menschliches Feedback verfeinert das Modellverhalten
    => Menschlich wirkende Antwort

- Ergebnis: Sprachmodell mit statistischem Codeverständnis
  - Statistisch erlerntes Wissen über Sprache & Code aus Trainingsdaten
  - Keine formale Verifikation sondern Mustererkennung auf Basis von Trainingsbeispielen
  - Kein „echtes“ Verständnis; schätzt basierend auf Wahrscheinlichkeiten

# Kategorien

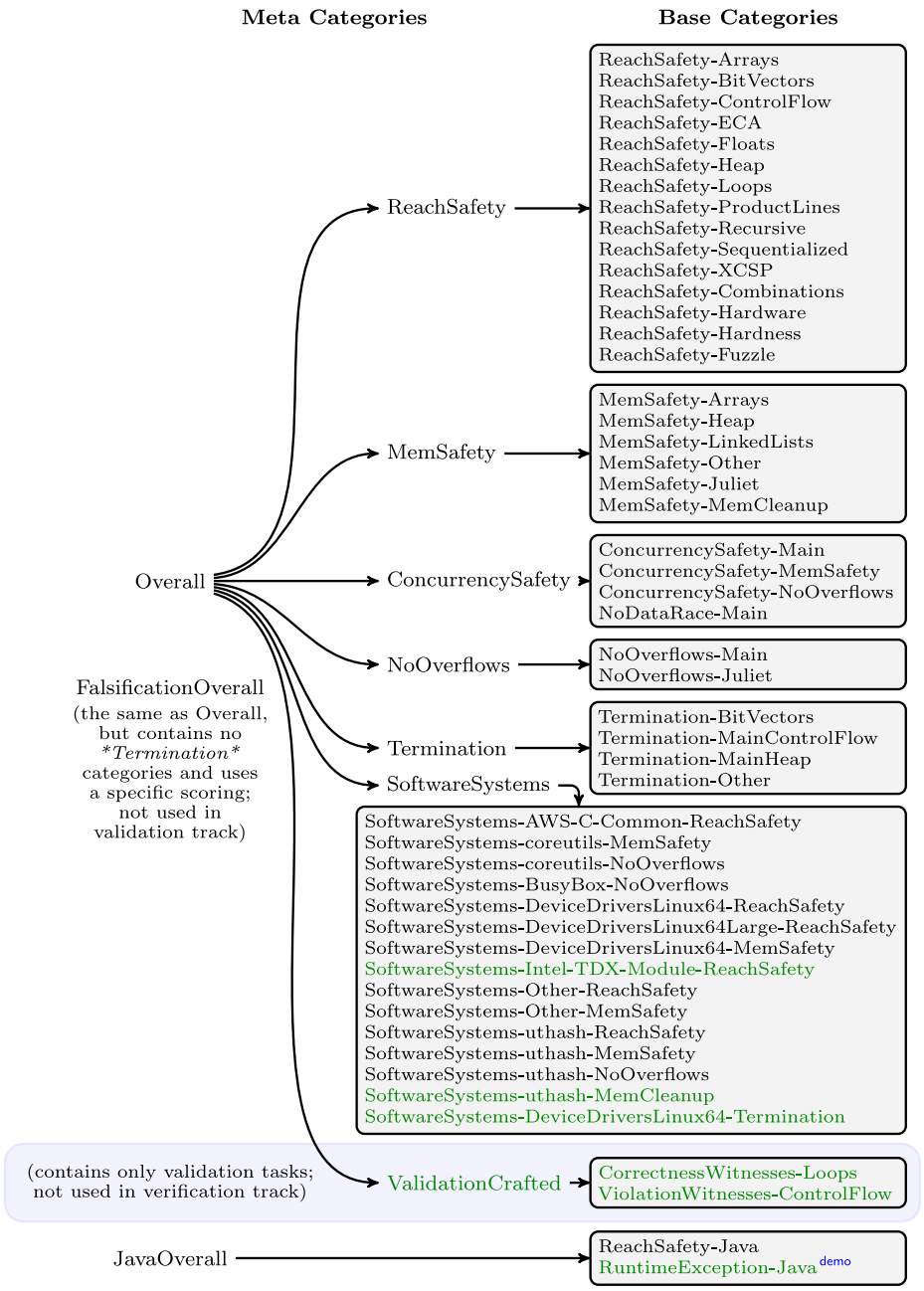
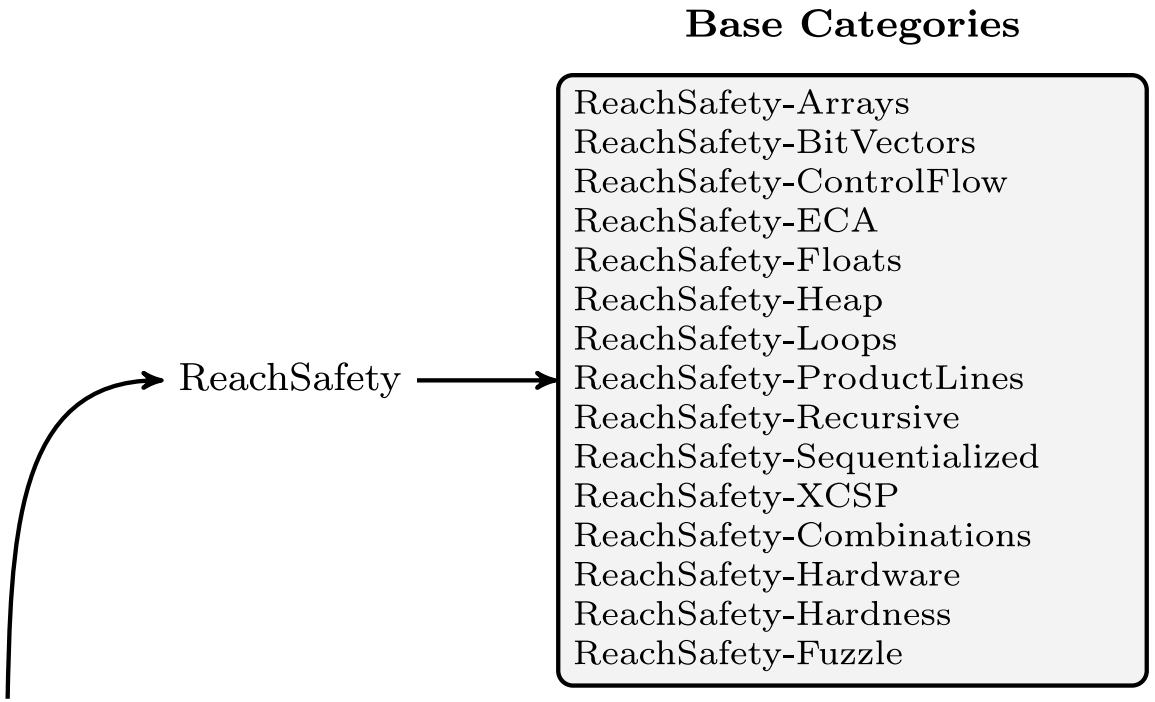


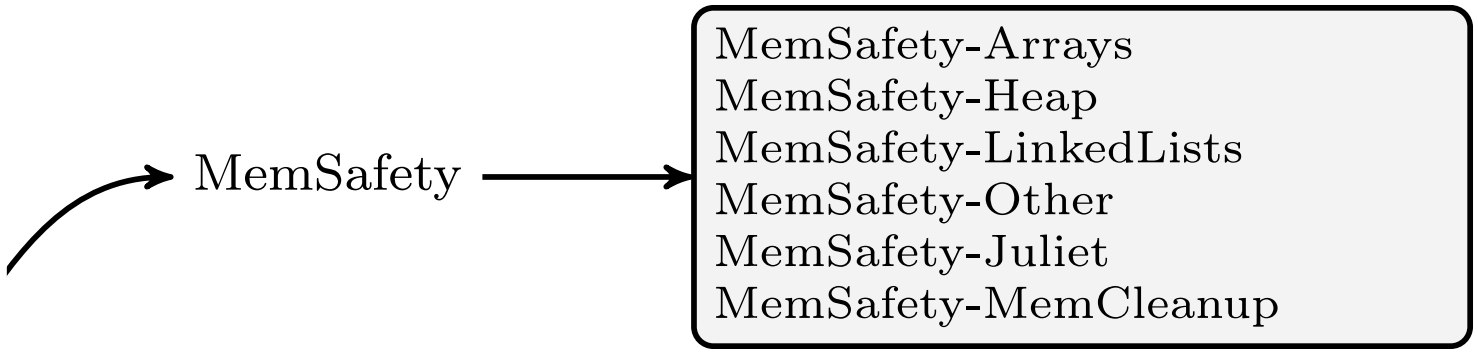

Fig. 1: Category structure for SV-COMP 2025; demo marks the demo category, new categories are green

Figure taken from report (Proc. TACAS, Springer, 2025)

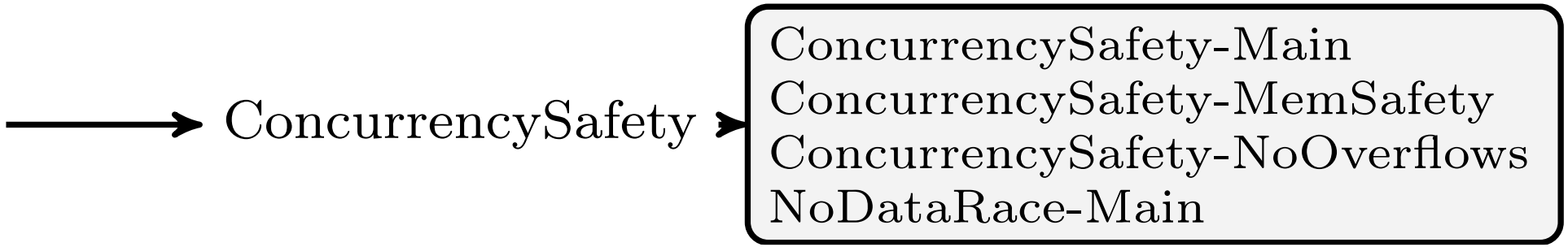
# Reach Safety



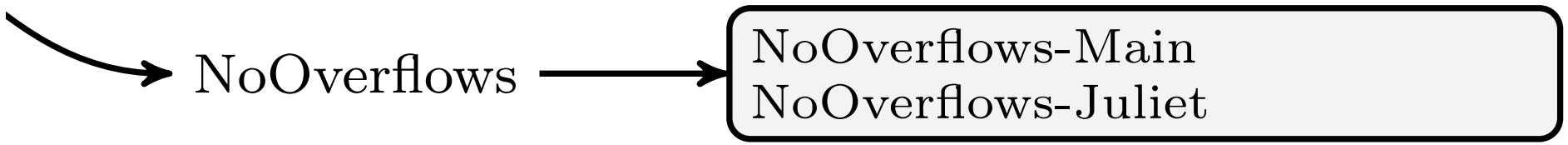
# Memory Safety



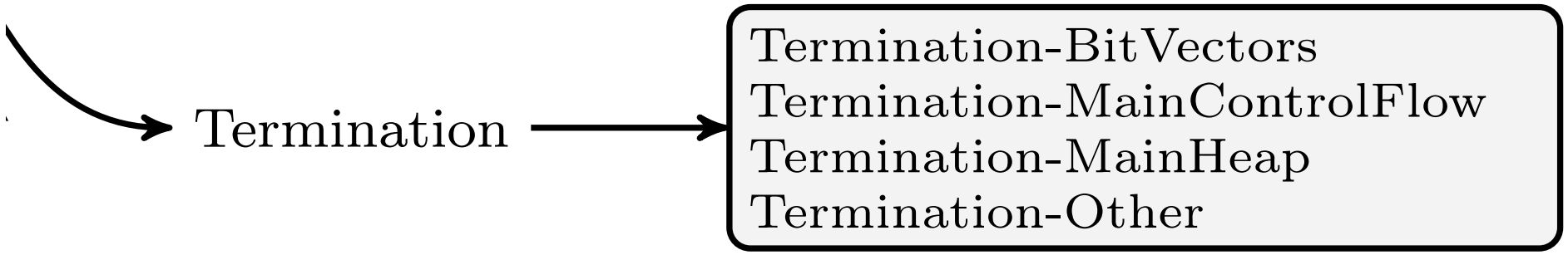
# Concurrency Safety



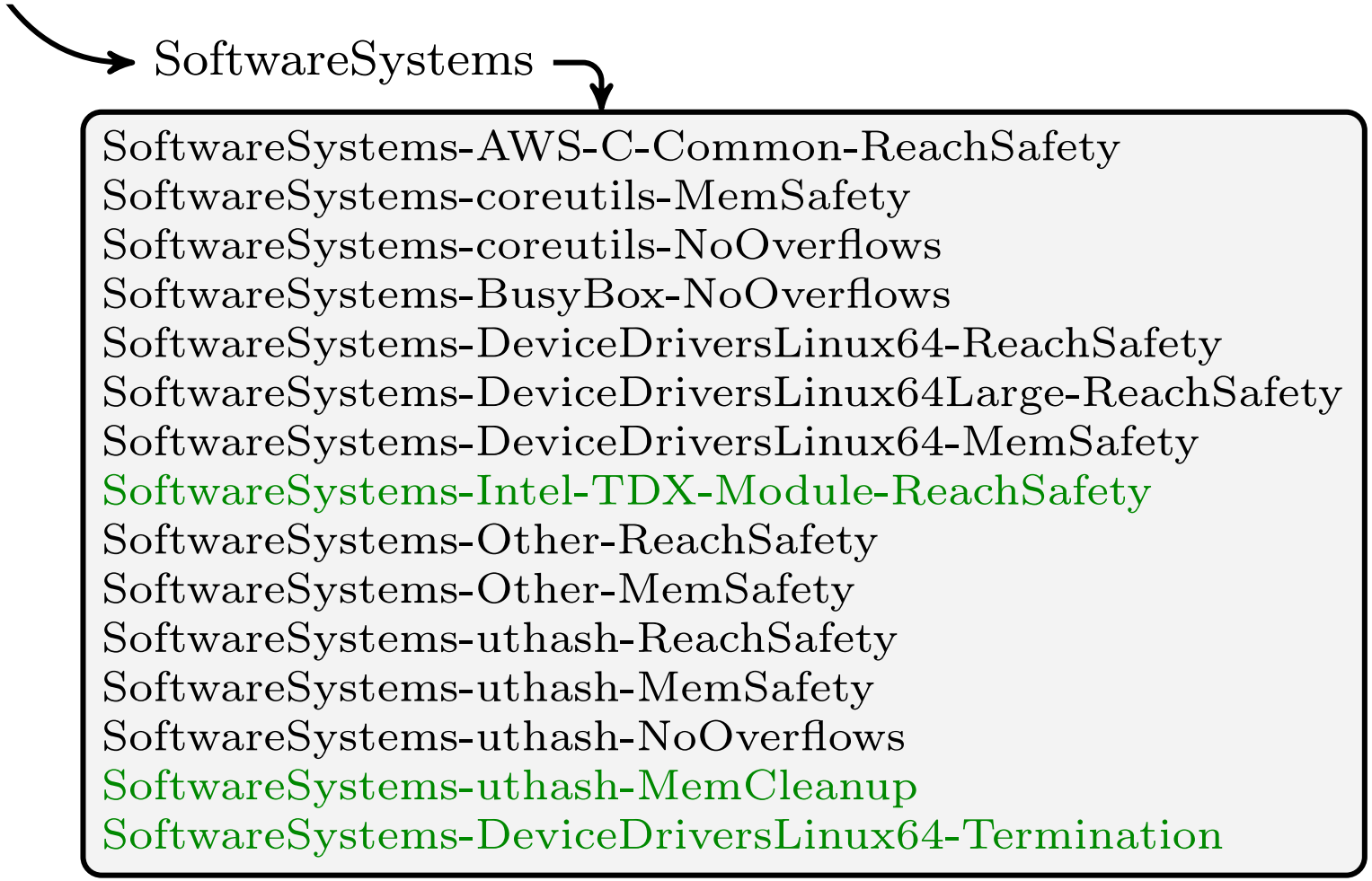
# No Overflows



# Termination



# Software Systems



# Bewertungsschema

| Ergebnis | Beschreibung |
|---|---|
| FALSE (correct) | Fehler korrekt erkannt |
| TRUE (correct) | Korrektheit erkannt |
| FALSE (incorrect) | Fehler gemeldet, obwohl keiner da |
| TRUE (incorrect) | Fehler übersehen |
| UNKNOWN | Keine Entscheidung getroffen (Fehler, Absturz) |

# Programmbeispiel: Fibonacci

```c
int fibonacci(int n) {
    if (n < 1) {
        return 0;
    } else if (n == 1) {
        return 1;
    } else {
        return fibonacci(n-1) + fibonacci(n-2);
    }
}

int main() {
    int x = 9;
    int result = fibonacci(x);
    if (result == 34) {
        return 0;
    } else {
        ERROR: {reach_error();abort();}
    }
}
```

# Programmbeispiel: Fibonacci

Konfiguration: **No-Overflow**

Erwartetes Ergebnis: ✅

CPAchecker Ergebnis: ✅

ChatGPT Ergebnis: ✅

```c
int fibonacci(int n) {
    if (n < 1) {
        return 0;
    } else if (n == 1) {
        return 1;
    } else {
        return fibonacci(n-1) + fibonacci(n-2);
    }
}

int main() {
    int x = 9;
    int result = fibonacci(x);
    if (result == 34) {
        return 0;
    } else {
        ERROR: {reach_error();abort();}
    }
}
```

# Testresultate

| Ergebnis | CPAchecker - Anzahl | LLM - Anzahl |
|---|---|---|
| FALSE (correct) | 13 | 15 |
| TRUE (correct) | 17 | 7 |
| FALSE (incorrect) | 4 | 15 |
| TRUE (incorrect) | 1 | 1 |
| UNKNOWN | 4 | 1 |

# CPAchecker bei SV-Comp 2025

**Auswertbare Tasks**
21.188

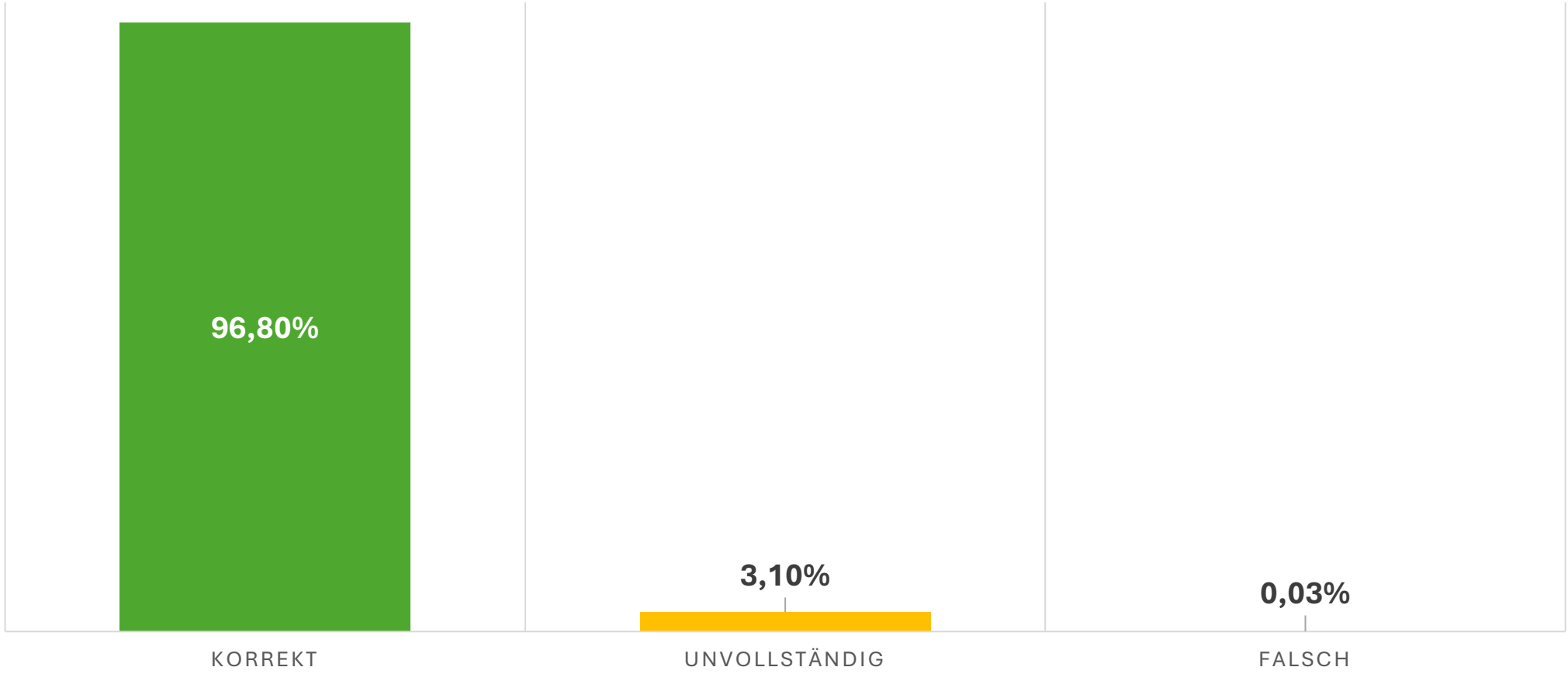
**Korrekt gelöste Tasks**
20.506

## Gesamtpunktzahl
## 26.786

**Unvollständige Lösungen**
675

**Falsch gelöste Tasks**
7

# CPAchecker bei SV-Comp 2025



# CPAchecker bei SV-Comp 2025