Counterexample Guided Program Repair Using Zero-Shot Learning and MaxSAT-based Fault Localization

Pedro Orvalho [1] , Mikoláš Janota [2] and Vasco Manquinho [3]

[1]Department of Computer Science, University of Oxford, Oxford, UK
[2]CIIRC, Czech Technical University in Prague, Czechia
[3]INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal

OutSystems AI Reading Group

21 March 2025

Automated Program Repair (APR)

Given a buggy program P_o and a set of input-output examples T (test suite).

Automated Program Repair (APR)

Given a buggy program P_o and a set of input-output examples T (test suite).

The goal of *Automated Program Repair* is to find a program P_f by **semantically change a subset S_1 of P_o's statements** ($S_1 \subseteq P_o$) for another set of statements S_2, s.t.,

$$P_f = ((P_o \setminus S_1) \cup S_2)$$

and

$$\forall (t^i_{in}, t^i_{out}) \in T \ : \ P_f(t^i_{in}) = t^i_{out}$$

Motivation

1: Semantically incorrect program. Faults: {4,8}.

```
1  int main(){ //finds max of 3 nums
2     int f,s,t;
3     scanf("%d%d%d",&f,&s,&t);
4     if (f < s && f >= t)
5        printf("%d",f);
6     else if (s > f && s >= t)
7        printf("%d",s);
8     else if (t < f && t < s)
9        printf("%d",t);
10
11    return 0;
12 }
```

Motivation

2: Semantically incorrect program. Faults: {4,8}.

```
1  int main(){ //finds max of 3 nums
2     int f,s,t;
3     scanf("%d%d%d",&f,&s,&t);
4     if (f < s && f >= t)
5        printf("%d",f);
6     else if (s > f && s >= t)
7        printf("%d",s);
8     else if (t < f && t < s)
9        printf("%d",t);
10
11    return 0;
12 }
```

Symbolic-based APR Tools:

- APR tools **based on Automated Reasoning**, such as CLARA or VERIFIX, **cannot fix this program within 90s**;

Motivation

3: Semantically incorrect program. Faults: {4,8}.

```
1  int main(){ //finds max of 3 nums
2     int f,s,t;
3     scanf("%d%d%d",&f,&s,&t);
4     if (f < s && f >= t)
5        printf("%d",f);
6     else if (s > f && s >= t)
7        printf("%d",s);
8     else if (t < f && t < s)
9        printf("%d",t);
10
11    return 0;
12 }
```

Symbolic-based APR Tools:

- APR tools **based on Automated Reasoning**, such as CLARA or VERIFIX, **cannot fix this program within 90s**;
- CLARA **takes too long** to compute a 'minimal' repair;

Motivation

4: Semantically incorrect program. Faults: {4,8}.

```
1  int main(){ //finds max of 3 nums
2     int f,s,t;
3     scanf("%d%d%d",&f,&s,&t);
4     if (f < s && f >= t)
5        printf("%d",f);
6     else if (s > f && s >= t)
7        printf("%d",s);
8     else if (t < f && t < s)
9        printf("%d",t);
10
11    return 0;
12 }
```

Symbolic-based APR Tools:

- APR tools **based on Automated Reasoning**, such as CLARA or VERIFIX, **cannot fix this program within 90s**;
- CLARA **takes too long** to compute a 'minimal' repair;
- VERIFIX returns a **compilation error**.

Motivation

5: Semantically incorrect program. Faults: {4,8}.

```
1  int main(){ //finds max of 3 nums
2     int f,s,t;
3     scanf("%d%d%d",&f,&s,&t);
4     if (f < s && f >= t)
5        printf("%d",f);
6     else if (s > f && s >= t)
7        printf("%d",s);
8     else if (t < f && t < s)
9        printf("%d",t);
10
11    return 0;
12 }
```

LLMs for code (LLMCs)

- GRANITE and CODEGEMMA **cannot fix** the buggy program within 90 secs;

Motivation

6: Semantically incorrect program. Faults: {4,8}.

```
1  int main(){ //finds max of 3 nums
2     int f,s,t;
3     scanf("%d%d%d",&f,&s,&t);
4     if (f < s && f >= t)
5        printf("%d",f);
6     else if (s > f && s >= t)
7        printf("%d",s);
8     else if (t < f && t < s)
9        printf("%d",t);
10
11    return 0;
12 }
```

LLMs for code (LLMCs)

- GRANITE and CODEGEMMA **cannot fix** the buggy program within 90 secs;
- Even if we provide this assignment's **description and IO tests**.

Motivation

7: Semantically incorrect program. Faults: {4,8}.

```
1  int main(){ //finds max of 3 nums
2     int f,s,t;
3     scanf("%d%d%d",&f,&s,&t);
4     if (f < s && f >= t)
5        printf("%d",f);
6     else if (s > f && s >= t)
7        printf("%d",s);
8     else if (t < f && t < s)
9        printf("%d",t);
10
11    return 0;
12 }
```

LLMs for code (LLMCs)

- GRANITE and CODEGEMMA **cannot fix** the buggy program within 90 secs;
- Even if we provide this assignment's **description and IO tests**.
- Suggesting a correct implementation, **both LLMs copy the correct program**, ignoring instructions not to do so.

Motivation

8: Semantically incorrect program. Faults: {4,8}.

```
1  int main(){ //finds max of 3 nums
2     int f,s,t;
3     scanf("%d%d%d",&f,&s,&t);
4     if (f < s && f >= t)
5        printf("%d",f);
6     else if (s > f && s >= t)
7        printf("%d",s);
8     else if (t < f && t < s)
9        printf("%d",t);
10
11    return 0;
12 }
```

- Symbolic approaches demand an **excessive amount of time to produce an answer**;

Motivation

9: Semantically incorrect program. Faults: {4,8}.

```
1  int main(){ //finds max of 3 nums
2     int f,s,t;
3     scanf("%d%d%d",&f,&s,&t);
4     if (f < s && f >= t)
5        printf("%d",f);
6     else if (s > f && s >= t)
7        printf("%d",s);
8     else if (t < f && t < s)
9        printf("%d",t);
10
11    return 0;
12 }
```

- Symbolic approaches demand an **excessive amount of time to produce an answer**;
- LLMs, while fast, **often produce incorrect fixes**.

Program Sketches

10: Semantically incorrect program. Faults :{4,8}.

```
1  int main(){ //finds max of 3 nums
2     int f,s,t;
3     scanf("%d%d%d",&f,&s,&t);
4     if (f < s && f >= t)
5        printf("%d",f);
6     else if (s > f && s >= t)
7        printf("%d",s);
8     else if (t < f && t < s)
9        printf("%d",t);
10
11    return 0;
12 }
```

11: Program sketch with holes.

```
1  int main(){
2     int f,s,t;
3     scanf("%d%d%d",&f,&s,&t);
4     @ HOLE 1 @
5        printf("%d",f);
6     else if (s > f && s >= t)
7        printf("%d",s);
8     @ HOLE 2 @
9        printf("%d",t);
10
11    return 0;
12 }
```

Our Work

- Combines the strengths of **Formal Methods (FM) and LLM-based approaches**;

Our Work

- Combines the strengths of **Formal Methods (FM) and LLM-based approaches**;
- Uses **MaxSAT-based fault localization to rigorously identify buggy lines**, which can then be highlighted in the LLM prompt;

Our Work

- Combines the strengths of **Formal Methods (FM) and LLM-based approaches**;
- Uses **MaxSAT-based fault localization to rigorously identify buggy lines**, which can then be highlighted in the LLM prompt;
- For instance, **instructing both LLMs to complete the previous sketch allows them to fix the buggy program** in a single interaction;

MaxSAT-Based Fault Localization

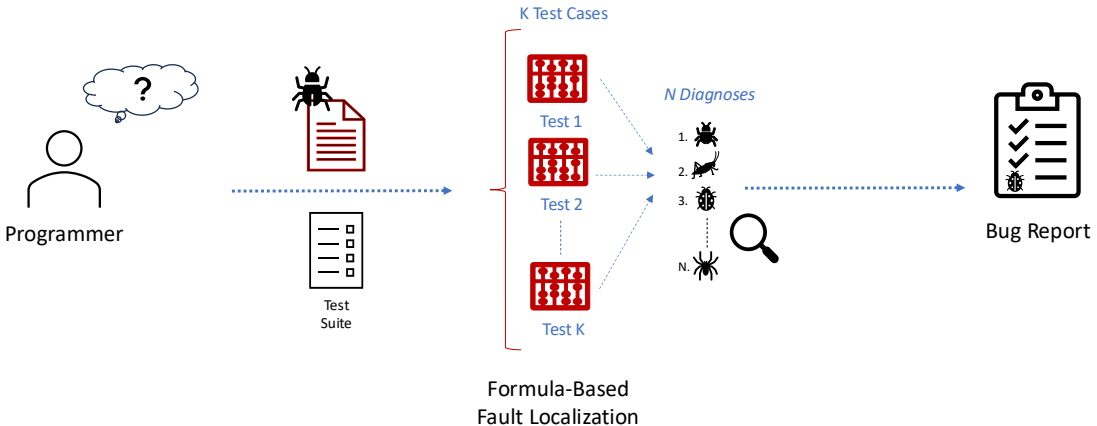
- **FM24** - CFaults: Model-Based Diagnosis for Fault Localization in C with Multiple Test Cases.

Fault Localization

- Given a buggy program, *fault localization (FL)* involves identifying locations in the program that could cause a faulty behaviour (bug).

Formula-Based Fault Localization (FBFL)

- FBFL methods encode the localization problem into **several optimization problems** to identify a minimal set of bugs (diagnoses).



Fault Localization

- We formulate the FL problem as a **single optimization problem**;

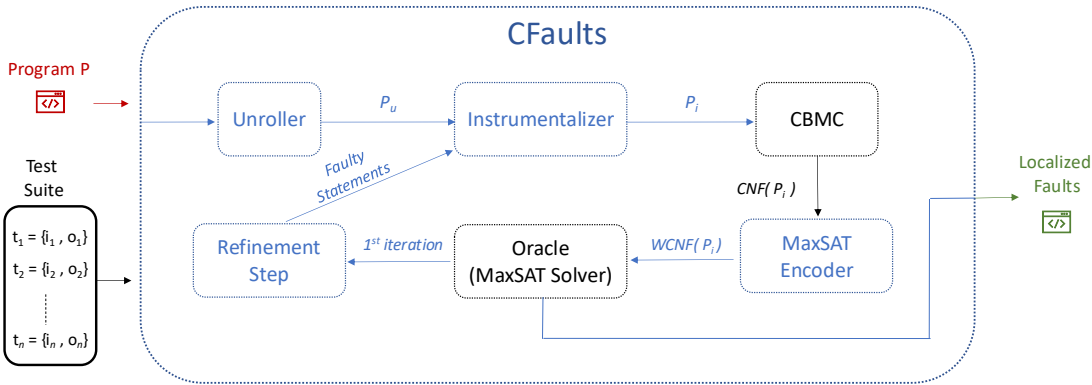
Fault Localization

- We formulate the FL problem as a **single optimization problem**;
- We leverage MaxSAT and the theory of *Model-Based Diagnosis (MBD)* [Reiter et al., 1987, Ignatiev et al., 2019], **integrating all failing test cases simultaneously**;

Fault Localization

- We formulate the FL problem as a **single optimization problem**;
- We leverage MaxSAT and the theory of *Model-Based Diagnosis (MBD)* [Reiter et al., 1987, Ignatiev et al., 2019], **integrating all failing test cases simultaneously**;
- We implement this MBD approach in a publicly available tool called CFaults [Orvalho et al., 2024].

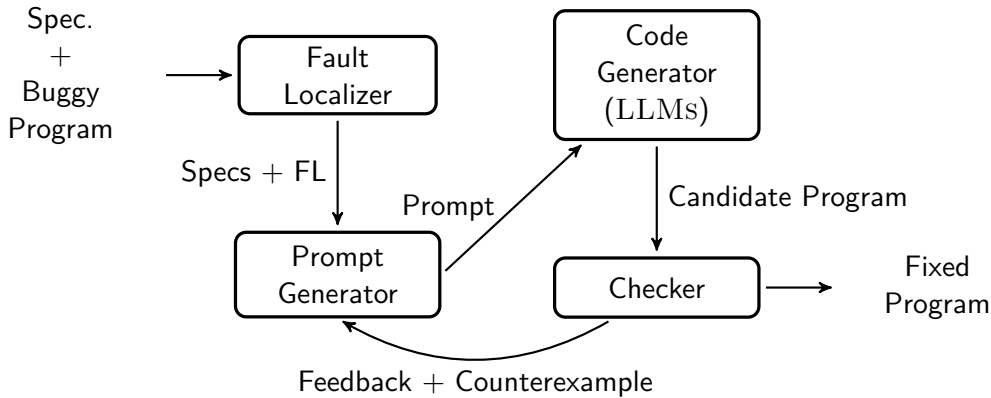
CFaults


CFaults
Program P
Test Suite
$t_1 = \{i_1 , o_1\}$
$t_2 = \{i_2 , o_2\}$
$t_n = \{i_n , o_n\}$
Unroller
P_u
Instrumentalizer
P_i
CBMC
Faulty Statements
CNF(P_i)
Refinement Step
1st iteration
Oracle (MaxSAT Solver)
WCNF(P_i)
MaxSAT Encoder
Localized Faults

Counterexample Guided Program Repair

- **AAAI 2025** - Counterexample Guided APR Using MaxSAT-based Fault Localization.

Counterexample Guided Automated Repair



Prompt Example without Fault Localization

````
Fix all semantic bugs in the buggy program
below. Modify the code as little as possible.
Do not provide any explanation.

### Problem Description ###
Write a program that determines and
prints the largest of three integers
given by the user.

### Test Suite
#input:
6 2 1
#output:
6
// The other input-output tests

# Reference Implementation
(Do not copy this program) <c> #
```c
int main(){
  // Reference Implementation
}
```

### Buggy Program <c> ###
```c
int main(){
  // Buggy program from Listing 1
}
```

### Fixed Program <c> ###
```c
````
````

Prompt with Fault Localization (FIXME)

Fix all buggy lines with '/* FIXME */' comments in the buggy program below.
Modify the code as little as possible.
Do not provide any explanation.

Problem Description
Write a program that determines and prints the largest of three integers given by the user.

Test Suite
#input:
6 2 1
#output:
6
// The other input-output tests

Reference Implementation
(Do not copy this program) <c> #
```c
int main(){
  // Reference Implementation
}
```

Buggy Program <c>
```c
int main(){
  // Buggy program from Listing 1
}
```

Fixed Program <c>
```c
```

Prompt with Fault Localization (Sketches)

Complete all the '@ HOLES N @' in the incomplete program below.
Modify the code as little as possible.
Do not provide any explanation.

Problem Description
Write a program that determines and prints the largest of three integers given by the user.

Test Suite
#input:
6 2 1
#output:
6
// The other input-output tests

Reference Implementation
(Do not copy this program) <c> #
```c
int main(){
  // Reference Implementation
}
```

Incomplete Program <c>

```c
int main(){
  // Buggy program from Listing 1
}
```

Complete Program <c>
```c
```

Experimental Results

Experimental Setup

- **Evaluation Benchmark:** C-Pack-IPAs, a set of twenty-five **introductory programming assignments**, comprising **1431 faulty programs**;

Experimental Setup

- **Evaluation Benchmark:** C-Pack-IPAs, a set of twenty-five **introductory programming assignments**, comprising **1431 faulty programs**;
- **Large Language Models (LLMs):** We evaluated **six different LLMs**.

Experimental Setup

- **Evaluation Benchmark:** C-Pack-IPAs, a set of twenty-five **introductory programming assignments**, comprising **1431 faulty programs**;
- **Large Language Models (LLMs):** We evaluated **six different LLMs**.
  - **Three of these models are LLMCs**, i.e., LLMs fine-tuned for coding tasks:

Experimental Setup

- **Evaluation Benchmark:** C-Pack-IPAs, a set of twenty-five **introductory programming assignments**, comprising **1431 faulty programs**;
- **Large Language Models (LLMs):** We evaluated **six different LLMs**.
  - **Three of these models are LLMCs**, i.e., LLMs fine-tuned for coding tasks:
    - IBM's Granite;

Experimental Setup

- **Evaluation Benchmark:** C-Pack-IPAs, a set of twenty-five **introductory programming assignments**, comprising **1431 faulty programs**;
- **Large Language Models (LLMs):** We evaluated **six different LLMs**.
  - **Three of these models are LLMCs**, i.e., LLMs fine-tuned for coding tasks:
    - IBM's Granite;
    - Google's CodeGemma;

Experimental Setup

- **Evaluation Benchmark:** C-Pack-IPAs, a set of twenty-five **introductory programming assignments**, comprising **1431 faulty programs**;
- **Large Language Models (LLMs):** We evaluated **six different LLMs**.
  - **Three of these models are LLMCs**, i.e., LLMs fine-tuned for coding tasks:
    - IBM's Granite;
    - Google's CodeGemma;
    - Meta's CodeLlama.

Experimental Setup

- **Evaluation Benchmark:** C-Pack-IPAs, a set of twenty-five **introductory programming assignments**, comprising **1431 faulty programs**;
- **Large Language Models (LLMs):** We evaluated **six different LLMs**.
  - **Three of these models are LLMCs**, i.e., LLMs fine-tuned for coding tasks:
    - IBM's Granite;
    - Google's CodeGemma;
    - Meta's CodeLlama.
  - **The other three models are general-purpose LLMs**:

Experimental Setup

- **Evaluation Benchmark:** C-Pack-IPAs, a set of twenty-five **introductory programming assignments**, comprising **1431 faulty programs**;
- **Large Language Models (LLMs):** We evaluated **six different LLMs**.
  - **Three of these models are LLMCs**, i.e., LLMs fine-tuned for coding tasks:
    - IBM's Granite;
    - Google's CodeGemma;
    - Meta's CodeLlama.
  - **The other three models are general-purpose LLMs**:
    - Google's Gemma;

Experimental Setup

- **Evaluation Benchmark:** C-Pack-IPAs, a set of twenty-five **introductory programming assignments**, comprising **1431 faulty programs**;
- **Large Language Models (LLMs):** We evaluated **six different LLMs**.
  - **Three of these models are LLMCs**, i.e., LLMs fine-tuned for coding tasks:
    - IBM's Granite;
    - Google's CodeGemma;
    - Meta's CodeLlama.
  - **The other three models are general-purpose LLMs**:
    - Google's Gemma;
    - Meta's Llama3;

Experimental Setup

- **Evaluation Benchmark:** C-Pack-IPAs, a set of twenty-five **introductory programming assignments**, comprising **1431 faulty programs**;
- **Large Language Models (LLMs):** We evaluated **six different LLMs**.
  - **Three of these models are LLMCs**, i.e., LLMs fine-tuned for coding tasks:
    - IBM's Granite;
    - Google's CodeGemma;
    - Meta's CodeLlama.
  - **The other three models are general-purpose LLMs**:
    - Google's Gemma;
    - Meta's Llama3;
    - Microsoft's Phi3.

Experimental Setup

- **Evaluation Benchmark:** C-Pack-IPAs, a set of twenty-five **introductory programming assignments**, comprising **1431 faulty programs**;
- **Large Language Models (LLMs):** We evaluated **six different LLMs**.
  - **Three of these models are LLMCs**, i.e., LLMs fine-tuned for coding tasks:
    - IBM's Granite;
    - Google's CodeGemma;
    - Meta's CodeLlama.
  - **The other three models are general-purpose LLMs**:
    - Google's Gemma;
    - Meta's Llama3;
    - Microsoft's Phi3.
- Experiments were conducted using a memory limit of **10GB**, and a timeout of **90s**.

LLM-Driven APR with CFaults

LLMs	De-TS	De-TS-CE	FIXME_De-TS	FIXME_De-TS-CE	Sk_De-TS	Sk_De-TS-CE	Portfolio (All Configurations)
CodeGemma	597 (41.7%)	606 (42.3%)	592 (41.4%)	601 (42.0%)	682 (47.7%)	**688 (48.1%)**	823 (57.5%)
CodeLlama	492 (34.4%)	500 (34.9%)	481 (33.6%)	463 (32.4%)	**573 (40.0%)**	561 (39.2%)	712 (49.8%)
Gemma	496 (34.7%)	492 (34.4%)	446 (31.2%)	444 (31.0%)	532 (37.2%)	**534 (37.3%)**	670 (46.8%)
Granite	626 (43.7%)	624 (43.6%)	566 (39.6%)	583 (40.7%)	**691 (48.3%)**	681 (47.6%)	846 (59.1%)
Llama3	564 (39.4%)	590 (41.2%)	535 (37.4%)	557 (38.9%)	578 (40.4%)	**591 (41.3%)**	851 (59.5%)
Phi3	494 (34.5%)	489 (34.2%)	460 (32.1%)	474 (33.1%)	**547 (38.2%)**	535 (37.4%)	621 (43.4%)
Portfolio (All LLMs)	842 (58.8%)	846 (59.1%)	796 (55.6%)	820 (57.3%)	900 (62.9%)	**907 (63.4%)**	1013 (70.8%)
Verifix	90 (6.3%)						
Clara	495 (34.6%)						

Table 1: The number of programs fixed by each LLM under various configurations. Mapping abbreviations to configuration names: **De** - IPA *Description*, **TS** - *Test Suite*, **CE** - *Counterexample*, **FIXME** - *FIXME Comments*, **SK** - *Sketches*.

Discussion

- CLARA repairs 495 programs (34.6%);

Discussion

- CLARA repairs 495 programs (34.6%);
- VERIFIX repairs only 91 programs (6.3%);

Discussion

- CLARA repairs 495 programs (34.6%);
- VERIFIX repairs only 91 programs (6.3%);
- **All six LLMs using different prompt configurations repair more programs than traditional APR tools**;

Discussion

- CLARA repairs 495 programs (34.6%);
- VERIFIX repairs only 91 programs (6.3%);
- **All six LLMs using different prompt configurations repair more programs than traditional APR tools**;
- Incorporating **FL-based Sketches (or FIXME annotations) allows LLMs to repair more programs**;

Discussion

- CLARA repairs 495 programs (34.6%);
- VERIFIX repairs only 91 programs (6.3%);
- **All six LLMs using different prompt configurations repair more programs than traditional APR tools**;
- Incorporating **FL-based Sketches (or FIXME annotations) allows LLMs to repair more programs**;
- Including **a reference implementation allows for more repaired programs but with less efficient fixes** (see our paper);

Discussion

- CLARA repairs 495 programs (34.6%);
- VERIFIX repairs only 91 programs (6.3%);
- **All six LLMs using different prompt configurations repair more programs than traditional APR tools**;
- Incorporating **FL-based Sketches (or FIXME annotations) allows LLMs to repair more programs**;
- Including **a reference implementation allows for more repaired programs but with less efficient fixes** (see our paper);
- Our CEGIS approach **significantly improves the accuracy of LLM-driven APR across various configurations**;

Takeaway Message

- We tackle the APR problem using an **LLM-Driven Counterexample Guided Inductive Synthesis (CEGIS) approach** [Solar-Lezama et al., 2006];

Takeaway Message

- We tackle the APR problem using an **LLM-Driven Counterexample Guided Inductive Synthesis (CEGIS) approach** [Solar-Lezama et al., 2006];
- We employ **MaxSAT-based Fault Localization to guide and minimize LLMs' patches** to incorrect programs by feeding them bug-free program sketches;

Takeaway Message

- We tackle the APR problem using an **LLM-Driven Counterexample Guided Inductive Synthesis (CEGIS) approach** [Solar-Lezama et al., 2006];
- We employ **MaxSAT-based Fault Localization to guide and minimize LLMs' patches** to incorrect programs by feeding them bug-free program sketches;
- With our approach **all six evaluated LLMs fix more programs and produce smaller patches** than other configurations and symbolic tools;

Takeaway Message

- We tackle the APR problem using an **LLM-Driven Counterexample Guided Inductive Synthesis (CEGIS) approach** [Solar-Lezama et al., 2006];
- We employ **MaxSAT-based Fault Localization to guide and minimize LLMs' patches** to incorrect programs by feeding them bug-free program sketches;
- With our approach **all six evaluated LLMs fix more programs and produce smaller patches** than other configurations and symbolic tools;
- Our code is available on GitHub and on Zenodo.

Thank you!

https://cs.ox.ac.uk/people/pedro.orvalho

References

Ahmed, Umair Z and Fan, Zhiyu and Yi, Jooyong and Al-Bataineh, Omar I and Roychoudhury, Abhik (2022)

Verifix: Verified repair of programming assignments.

TOSEM 22 12(3), 45 – 678.

Gulwani, Sumit and Radiček, Ivan and Zuleger, Florian (2018)

Automated clustering and program repair for introductory programming assignments.

PLDI 18 52(4), 465 – 480.

Armando Solar-Lezama and Liviu Tancau and Rastislav Bodík and Sanjit A. Seshia and Vijay A. Saraswat (2018)

Combinatorial sketching for finite programs.

ASPLOS 2006.

Reiter, Raymond (1987)

A Theory of Diagnosis from First Principles.

Artif. Intell. 1987.

References


Ignatiev, Alexey and Morgado, António and Weissenbacher, Georg and Marques-Silva, João (2019)
Model-Based Diagnosis with Multiple Observations.
IJCAI 2019.


P. Orvalho and M. Janota and V. Manquinho (2024)
CFaults: Model-Based Diagnosis for Fault Localization in C with Multiple Test Cases.
Formal Methods (FM) 2024.


P. Orvalho and M. Janota and V. Manquinho (2024)
C-Pack of IPAs: A C90 Program Benchmark of Introductory Programming Assignments.
Automated Program Repair (APR) 2024.


P. Orvalho and M. Janota and V. Manquinho (2025)
Counterexample Guided Program Repair Using Zero-Shot Learning and MaxSAT-based Fault Localization.
AAAI 2025.