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

Pedro Orvalho, Mikoláš Janota, Vasco M. Manquinho
pedro.orvalho@cs.ox.ac.uk
Department of Computer Science, University of Oxford, Oxford, UK

Listing 1 Semantically incorrect program. Faulty lines: {4,8}.

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

Listing 2 Reference implementation.

```c
int main() {
   int m1,m2,m3,m;
   scanf("%d%d%d",&m1,&m2,&m3);
   m = m1 > m2 ? m1 : m2;
   m = m3 > m ? m3 : m;
   printf("%d\n", m);
   return 0;
}
```

Listing 3 Program sketch with holes.

```c
int main(){
   int f,s,t;
   scanf("%d%d%d",&f,&s,&t);
   @ HOLE 1 @
      printf("%d",f);
   else if (s > f && s >= t)
      printf("%d",s);
   @ HOLE 2 @
      printf("%d",t);
   return 0;
}
```

Listing 4 GRANITE's fix using the program sketch.

```c
int main(){
   int f,s,t;
   scanf("%d%d%d",&f,&s,&t);
   if (f >= s && f >= t)
      printf("%d",f);
   else if (s > f && s >= t)
      printf("%d",s);
   else
      printf("%d",t);
   return 0;
}
```

Motivation

- Listing 1 aims to **determine the maximum among three given numbers**;
- Traditional Automated Program Repair **(APR) tools** for introductory programming assignments (IPAs) **based on Formal Methods**, such as CLARA or VERIFIX, **cannot fix this program within 90s**.
- **CLARA takes too long to compute a 'minimal' repair** by considering several correct implementations for the same IPA, while **VERIFIX returns a compilation error**.
- Using **LLMs trained for coding tasks (LLMCs)**, GRANITE or CODEGEMMA, would involve providing the **description of the IPA and some IO tests**.
- Nonetheless, **neither LLM could fix the buggy program in Listing 1 within 90s**.
- Suggesting the program in Listing 2 as a correct implementation, **both LLMs simply copy the correct program**, ignoring instructions not to do so.
- Thus, **symbolic approaches demand an excessive amount of time to produce an answer, and LLMs, while fast, often produce incorrect fixes**.

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** to focus only on these lines;
- **Listing 3 shows an example of a program sketch, which is a partially incomplete program** where each buggy statement is replaced with a `@ HOLE @`;
- **Instructing the LLMs to complete this sketch allows both LLMs to fix the buggy program** in a single interaction, returning the program in Listing 4.

Contributions

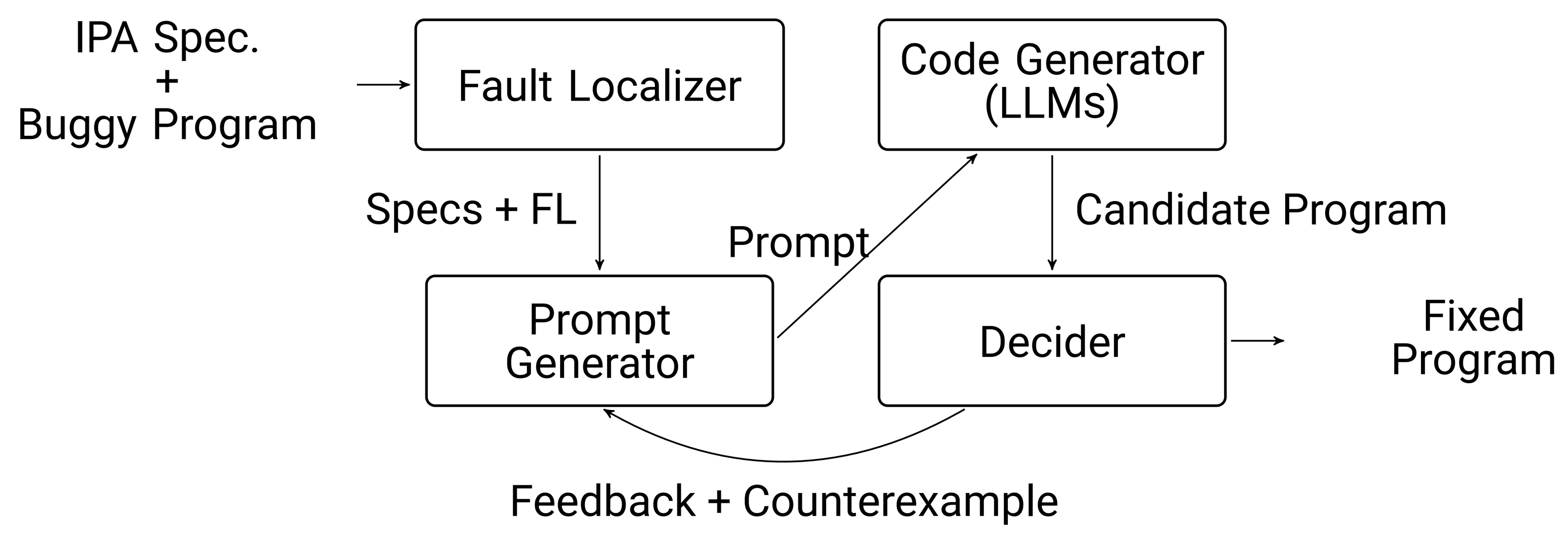
- We tackle the Automated Program Repair (APR) problem using an **LLM-Driven Counterexample Guided Inductive Synthesis (CEGIS) approach**;
- We employ **MaxSAT-based Fault Localization to guide and minimize LLMs' patches** to incorrect programs by feeding them bug-free program sketches;
- Experiments show that 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.

This work was supported by ERC AdG FUN2MODEL (Grant agreement No. 834115).

Counterexample Guided Automated Repair

Our approach follows a **Counterexample Guided Inductive Synthesis (CEGIS) [1] loop** to iteratively refine the program.
The **input is a buggy program and the specifications for an IPA**, including its description, a test suite, and a correct solution. Then, we:

1. Employ MaxSAT-based fault localization to **rigorously identify the minimal set of buggy parts of a program**;
2. Generate **a prompt based on the specifications of the IPA and a bug-free program sketch**, then feed this information to the LLM;
3. **The LLM generates a program based on the provided prompt**;
4. **The Decider evaluates the synthesized program** against a test suite;
5. **If the program is incorrect, a counterexample is sent to the prompt generator**, which then feeds this counterexample to the LLM to prompt a revised synthesis.



Experimental Evaluation

- **Evaluation Benchmark:** We used C-PACK-IPAS [2], which consists of **1431 semantically incorrect student C programs**.
- **Large Language Models (LLMs):** We evaluated **six different LLMs through iterative querying**. **Three of these models are LLMCs**, i.e., LLMs fine-tuned for coding tasks: IBM's GRANITE, Google's CODEGEMMA and Meta's CODELLAMA. **The other three models are general-purpose LLMs**: Google's GEMMA, Meta's LLAMA3 and Microsoft's PHI3.
- **Fault Localization (FL):** We used **CFAULTS [3], a MaxSAT-based FL tool that pinpoints bug locations within the programs**.

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%)

Discussion:

- **CLARA repairs 495 programs (34.6%)**, times out on 154 (10.8%), and fails to repair 738 programs (54.7%);
- **VERIFIX repairs 91 programs (6.3%)**, reaches the time limit on 0.6%, and fails to repair 1338 programs (93.5%);
- **All six LLMs using different prompt configurations repair more programs than traditional APR tools**;
- Prompt configurations with **FL-based Sketches, IPA description and test suite fix more programs**.
- Incorporating **FL-based Sketches (or FIXME annotations) allows the LLMs to repair more programs** than only providing the buggy program.
- 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**.

References

[1] Armando Solar-Lezama et al. "Combinatorial sketching for finite programs". In: *ASPLOS 2006*.
[2] Pedro Orvalho, Mikoláš Janota, and Vasco Manquinho. "C-Pack of IPAs: A C90 Program Benchmark of Introductory Programming Assignments". In: *Automated Program Repair (APR) 2024*.
[3] Pedro Orvalho, Mikoláš Janota, and Vasco Manquinho. "CFaults: Model-Based Diagnosis for Fault Localization in C Programs with Multiple Test Cases". In: *Formal Methods (FM) 2024*.

LaTeX TikZposter