

Can ChatGPT Support CPAchecker with Useful Loop Invariants?

CPAchecker Workshop 2023

Cedric Richter, 11.09.2023

GPT-2 (2019)

System Prompt (human-written)

In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Model Completion (machine-written, 10 tries)

The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

GPT-2 (2019)

System Prompt (human-written)

In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Model Completion (AI-written)



<https://drawception.com/game/Bx1SezmLtw/a-quadricorn/>

of the population, after their distinctive horn, these four-horned, silver-white unicorns were brought to science.

four-horned, silver-white unicorns

ChatGPT



Can you list 5 interesting facts about CPAchecker?



CPAchecker is a widely used tool for the verification of software programs, particularly those written in the C programming language. Here are five interesting facts about CPAchecker:

- 1. Model Checking Engine:** CPAchecker is primarily a model checking engine [...]
- 2. Configurable and Extensible:** CPAchecker is highly configurable and extensible, [...]
- 3. Open-Source:** CPAchecker is an open-source tool, released under the GNU General Public License (GPL). [...]
- 4. Competition Success:** CPAchecker has participated in various verification competitions, [...]
- 5. Integration with Other Tools:** CPAchecker can be integrated with other verification tools and frameworks, such as SMT solvers and software model checkers. [...]

ChatGPT can write programs from descriptions!

Write a Python function to compute the Fibonacci numbers.



```
python Copy code  
  
def fibonacci_recursive(n):  
    if n <= 0:  
        return 0  
    elif n == 1:  
        return 1  
    else:  
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)  
  
# Example usage:  
n = 10 # Change this to compute Fibonacci number for a different value of n  
result = fibonacci_recursive(n)  
print(f"The {n}-th Fibonacci number is: {result}")
```

With up to 86% accuracy* on coding benchmarks (HumanEval).

*„OctoPack: Instruction Tuning Code Large Language Models“ (2023)

ChatGPT can debug its code!

But Fibonacci numbers are not defined for negative numbers!



```
python Copy code  
  
def fibonacci_recursive(n):  
    if n < 0:  
        raise ValueError("Fibonacci numbers are not defined for negative numbers")  
    elif n == 0:  
        return 0  
    elif n == 1:  
        return 1  
    else:  
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)  
  
# Example usage:  
n = 10 # Change this to compute Fibonacci number for a different value of n  
result = fibonacci_recursive(n)  
print(f"The {n}-th Fibonacci number is: {result}")
```

Achieves up to 91% accuracy* on coding benchmarks (HumanEval) with self-debugging.

*, „Reflexion: Language Agents with Verbal Reinforcement Learning“ (2023)

ChatGPT show some form of logical reasoning!

Henry made two stops during his 60-mile bike trip. He first stopped after 20 miles. His second stop was 15 miles before the end of the trip. How many miles did he travel between his first and second stops?



Henry traveled 60 miles on his bike trip. He first stopped after 20 miles, and his second stop was 15 miles before the end of the trip.

To find out how many miles he traveled between his first and second stops, you can subtract the distances of these two stops from the total distance:

60 miles (total distance) - 20 miles (first stop) - 15 miles (second stop) = 60 - 20 - 15 = 25 miles

Henry traveled 25 miles between his first and second stops.



ChatGPT show some form of logical reasoning!

Henry made two stops during his 60-mile bike trip. He first stopped after 20 miles. His second stop was 15 miles before the end of the trip. How many miles did he travel between his first and second stops?



Henry traveled 60 miles on his bike trip. He first stopped after 20 miles, and his second stop was 15 miles before the end of the trip.

To find out how many miles he traveled between his first and second stops, you can subtract the distances of these two stops from the total distance:

60 miles (total distance) - 20 miles (first stop) - 15 miles (second stop) = 60 - 20 - 15 = 25 miles

Henry traveled 25 miles between his first and second stops.



Achieves more than 90% accuracy* on logical reasoning benchmarks (GSM8K, ARC)

Can ChatGPT be useful for **formal** verification?



Work in
Progress



Task: Loop Invariant Generation

```
int main()
{
    unsigned int n = __VERIFIER_nondet_uint();
    unsigned int x=n, y=0;
    while(x>0)
    {
        x--;
        y++;
    }
    __VERIFIER_assert(y==n);
}
```

Task: count_up_down-1

Task: Loop Invariant Generation

```
int main()
{
    unsigned int n = __VERIFIER_nondet_uint();
    unsigned int x=n, y=0;
    while(x>0)
    {
        x--;
        y++;
    }
    __VERIFIER_assert(y==n);
}
```

Loop invariant:
 $x + y == n$

Task: count_up_down-1

Loop invariants can be “easily” validated

```
int main()
{
    unsigned int n = __VERIFIER_nondet_uint();
    unsigned int x=n, y=0;
    while(x>0)
    {
        x--;
        y++;
    }
    __VERIFIER_assert(y==n);
}
```

Loop invariant:
 $x + y == n$

Many tools such as Frama-C, CPAchecker and others support the validation of loop invariants.

Task: count_up_down-1

Can ChatGPT generate loop invariants?

Compute a loop invariant for the following program:

```
int main()
{
    unsigned int n = __VERIFIER_nondet_uint();
    unsigned int x=n, y=0;
    while(x>0)
    {
        x--;
        y++;
    }
    __VERIFIER_assert(y==n);
}
```

Loop Invariant: At the beginning of each iteration of the loop, the value of y is equal to the initial value of n minus the current value of x .

Mathematically, you can represent the loop invariant as:

```
y + x == n
```

Copy code



Can ChatGPT generate **valid** loop invariants?

Compute a loop invariant for the following program:

```
int main(void) {
    unsigned int x = 0;
    unsigned int y = __VERIFIER_nondet_uint();

    while (x < 99) {
        if (y % 2 == 0) {
            x += 2;
        } else {
            x++;
        }
    }

    __VERIFIER_assert((x % 2) == (y % 2));
}
```

Loop Invariant: At the beginning of each iteration of the loop, the following condition holds true:

```
x % 2 == y % 2
```

Copy code



Can ChatGPT generate **valid** loop invariants?

```
int main(void) {  
    unsigned int x = 0;  
    unsigned int y = __VERIFIER_nondet_uint();  
  
    while (x < 99) {  
        if (y % 2 == 0) {  
            x += 2;  
        } else {  
            x++;  
        }  
    }  
  
    __VERIFIER_assert((x % 2) == (y % 2));  
}
```

Generate a loop invariant for the following program:

```
int main(void) {  
    unsigned int x = 0;  
    unsigned int y = __VERIFIER_nondet_uint();  
  
    while (x < 99) {  
        if (y % 2 == 0) {  
            x += 2;  
        } else {  
            x++;  
        }  
    }  
  
    __VERIFIER_assert((x % 2) == (y % 2));  
}
```

At the beginning of each iteration of the loop, the following condition holds true:

```
x % 2 == y % 2
```



Loop invariant generation via ACSL

```
int main(void) {
    unsigned int x = 0;
    unsigned int y = __VERIFIER_nondet_uint();

    //@ loop invariant x % 2 == y % 2 || (y % 2 == 1 && x <= 99);
    while (x < 99) {
        if (y % 2 == 0) {
            x += 2;
        } else {
            x++;
        }
    }

    //@ assert x % 2 == y % 2;
    __VERIFIER_assert((x % 2) == (y % 2));
}
```

1. Automatically prove the validity via Frama-C.
2. ChatGPT can likely handle it

ACSL infilling (120 Tasks from SV-COMP Loops)

Compute a loop invariant for the following program*:

```
int main(void) {
  unsigned int x = 0;
  unsigned int y = __VERIFIER_nondet_uint();

  //@ loop invariant [redacted];
  while (x < 99) {
    if (y % 2 == 0) {
      x += 2;
    } else {
      x++;
    }
  }

  //@ assert x % 2 == y % 2;
  __VERIFIER_assert((x % 2) == (y % 2));
}
```



GPT-3.5

[...] From the code, we can observe that the value of `x` is incremented by 2 if `y` is even, and by 1 if `y` is odd. Therefore, the loop invariant can be stated as:[...]

```
`//@ loop invariant x % 2 == y % 2 ;`
```

*“Compute a loop invariant for the annotated loop including [invariant] in the following C code. Please use the format of ACSL annotations and always end your response with `//@ loop invariant X ;` where X is the computed invariant.”

ACSL infilling (120 Tasks from SV-COMP Loops)

Compute a loop invariant for the following program*:

```
int main(void) {
  unsigned int x = 0;
  unsigned int y = __VERIFIER_nondet_uint();

  //@ loop invariant           ;
  while (x < 99) {
    if (y % 2 == 0) {
      x += 2;
    } else {
      x++;
    }
  }

  //@ assert x % 2 == y % 2;
  __VERIFIER_assert((x % 2) == (y % 2));
}
```



About **45%** of all loop invariants can be validated by **Frama-C**.
Around **19%** match the human annotated loop invariant.

* "Compute annotations and always end your response with //@ loop invariant X ; where X is the computed invariant."

The other 55% not validated by Frama-C

```
int main(void) {  
    unsigned int x = 0;  
    unsigned int y = 1;  
  
    //@ loop invariant x >= 0 && x <= 6 && y == pow(2, x);  
    while (x < 6) {  
        x++;  
        y *= 2;  
    }  
  
    //@ assert x == 6;  
    __VERIFIER_assert(x == 6);  
}
```



Frama-C

The other 55% not validated by Frama-C

```
int main(void) {  
    unsigned int w = __VERIFIER_nondet_uint();  
    unsigned int x = w;  
    unsigned int y = w + 1;  
    unsigned int z = x + 1;  
    //@ loop invariant y == z - 1;  
    while (__VERIFIER_nondet_uint()) {  
        y++;  
        z++;  
    }  
    //@ assert y == z;  
    __VERIFIER_assert(y == z);  
    return 0;  
}
```



Not correct

Generated loop invariants are often non-trivial!

```
int main() {  
    int i, j;  
    i = __VERIFIER_nondet_int();  
    j = __VERIFIER_nondet_int();  
    if (!(i >= 0 && i <= 1000000)) return 0;  
    if (!(j >= 0)) return 0;  
    int x = i;  
    int y = j;  
    int z = 0;  
    //@ loop invariant x >= 0 && y == j - 2 * z && z >= 0;  
    while (x != 0) {  
        x--;  
        y -= 2;  
        z++;  
    }  
    if (i == j) {  
        //@ assert y == -z;  
        __VERIFIER_assert(y == -z);  
    }  
    return 0;  
}
```

But not all invariants are **useful**...

```
int main(){  
    unsigned int x = __VERIFIER_nondet_int();  
    unsigned int y = 1;  
    ..  
    //@ loop invariant x % 3 == 0 || x % 3 == 1 || x % 3 == 2;  
    while(__VERIFIER_nondet_int()){  
        if(x % 3 == 1){  
            x += 2; y = 0;  
        }  
        else{  
            if(x % 3 == 2){  
→ x += 1; y = 0;  
            }  
            else{
```

Can ChatGPT (GPT-3.5) generate **useful** loop invariants?

Subfolder	Frama-C				CPAchecker (Interactive Verifier)	
	T	V	Baseline	ChatGPT-inv / Human	Baseline	ChatGPT-inv / Human
loop-ac						
loop						
loop-						
pa						
loop-i						
loop						
loop-lit	13	4	0	1 / 4	1	3 / 7
loo						
loop						
loc						
ld						
loops-						
total	120	55	0	12 / 22	19	28 / 44

Frama-C verifies 12 of 55 programs (ChatGPT).
Frama-C verifies 22 of 55 programs (Human).
CPAchecker verifies 28 (ChatGPT) of 44 (Human) programs.

The invariants generated by ChatGPT can be useful for both Frama-C and CPAchecker.

Failures of ChatGPT

```
int main() {  
  int i = __VERIFIER_nondet_int();  
  int n = __VERIFIER_nondet_int();  
  int sum = __VERIFIER_nondet_int();  
  if (!(i==0 && n>=0 && n<=100 && sum==0)) return 0;  
  //@ loop invariant i >= 0 && i <= n && sum == \sum_{k=0}^{i-1} k;  
  while (i < n) {  
    sum = sum + i;  
    i++;  
  }  
  //@ assert sum >= 0;  
  __VERIFIER_assert(sum >= 0);  
  return 0;  
}
```

Target invariant:
 $2 * \text{sum} == (i - 1) * i$

References to unknown variables

```
int main() {  
  int i = __VERIFIER_nondet_int();  
  int j = __VERIFIER_nondet_int();  
  int r = __VERIFIER_nondet_int();  
  if (!(r > i + j)) return 0;  
  //@ loop invariant i + j == i_0 + j_0;  
  while (i > 0) {  
    i = i - 1;  
    j = j + 1;  
  }  
  //@ assert r > i + j;  
  __VERIFIER_assert(r > i + j);  
  return 0;  
}
```

Target invariant:

$r > i + j$

Invariants via copy heuristic

```
int main(){  
    unsigned int x = 0;  
  
    //@ loop invariant x % 2 == 0;  
    while (x < 100000000) {  
        if (x < 10000000) {  
            x++;  
        } else {  
            x += 2;  
        }  
    }  
  
    //@ assert x % 2 == 0;  
    __VERIFIER_assert((x%2)==0);  
    return 0;  
}
```

Target invariant:
 $x < 100000000 \parallel x \% 2 == 0$

AI in isolation



AI

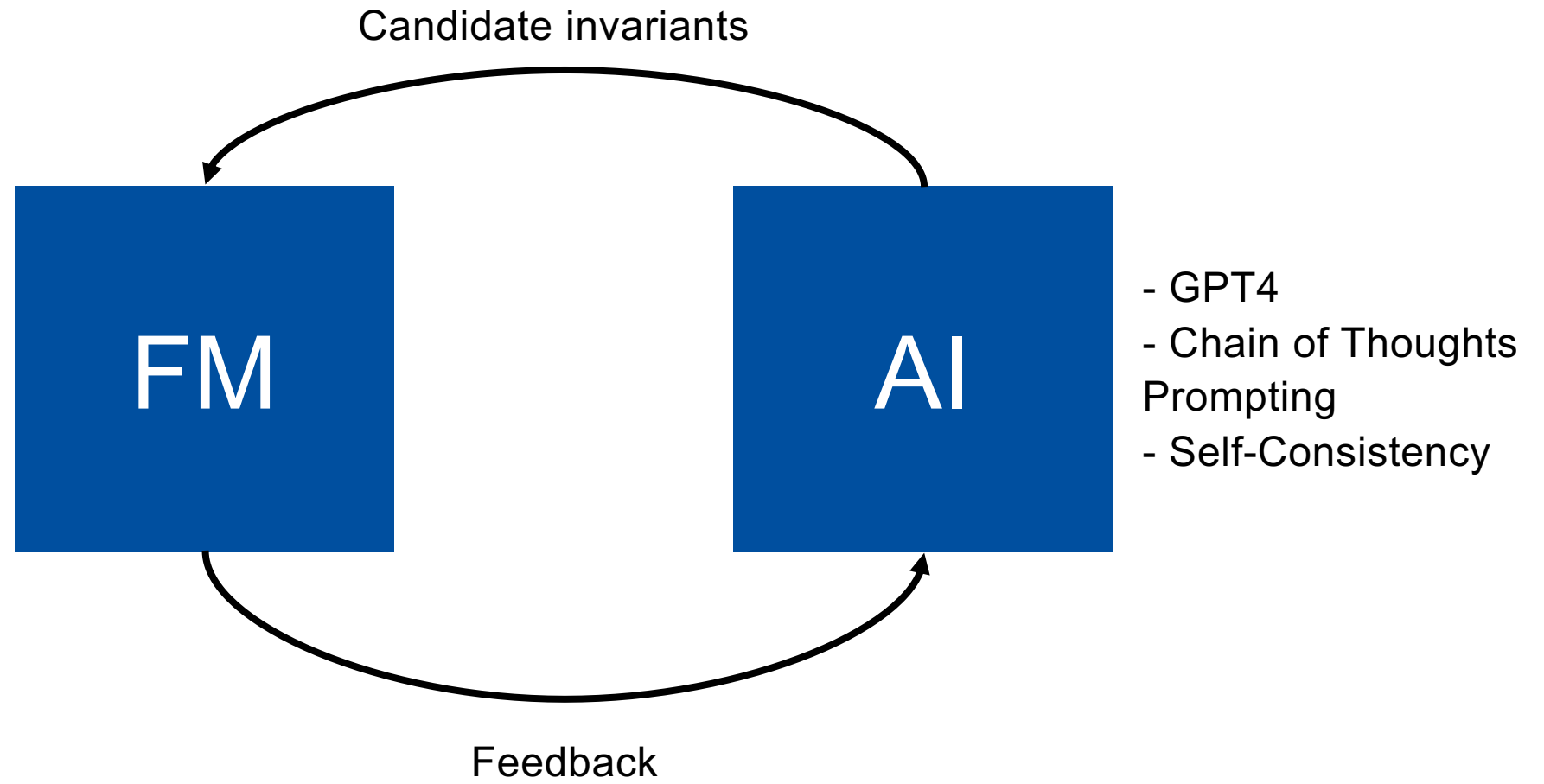
Future Work: Improvements for AI



AI

- GPT4
- Chain of Thoughts Prompting
- Self-Consistency

Future Work: Self-debug, Student-Teacher Loop, CEGIR



Can ChatGPT be useful for **formal** verification?

ChatGPT can generate valid and useful invariants!

The output is still noisy!
We need formal methods to find the real invariants!

There is potential for ChatGPT
to be useful in formal verification!

