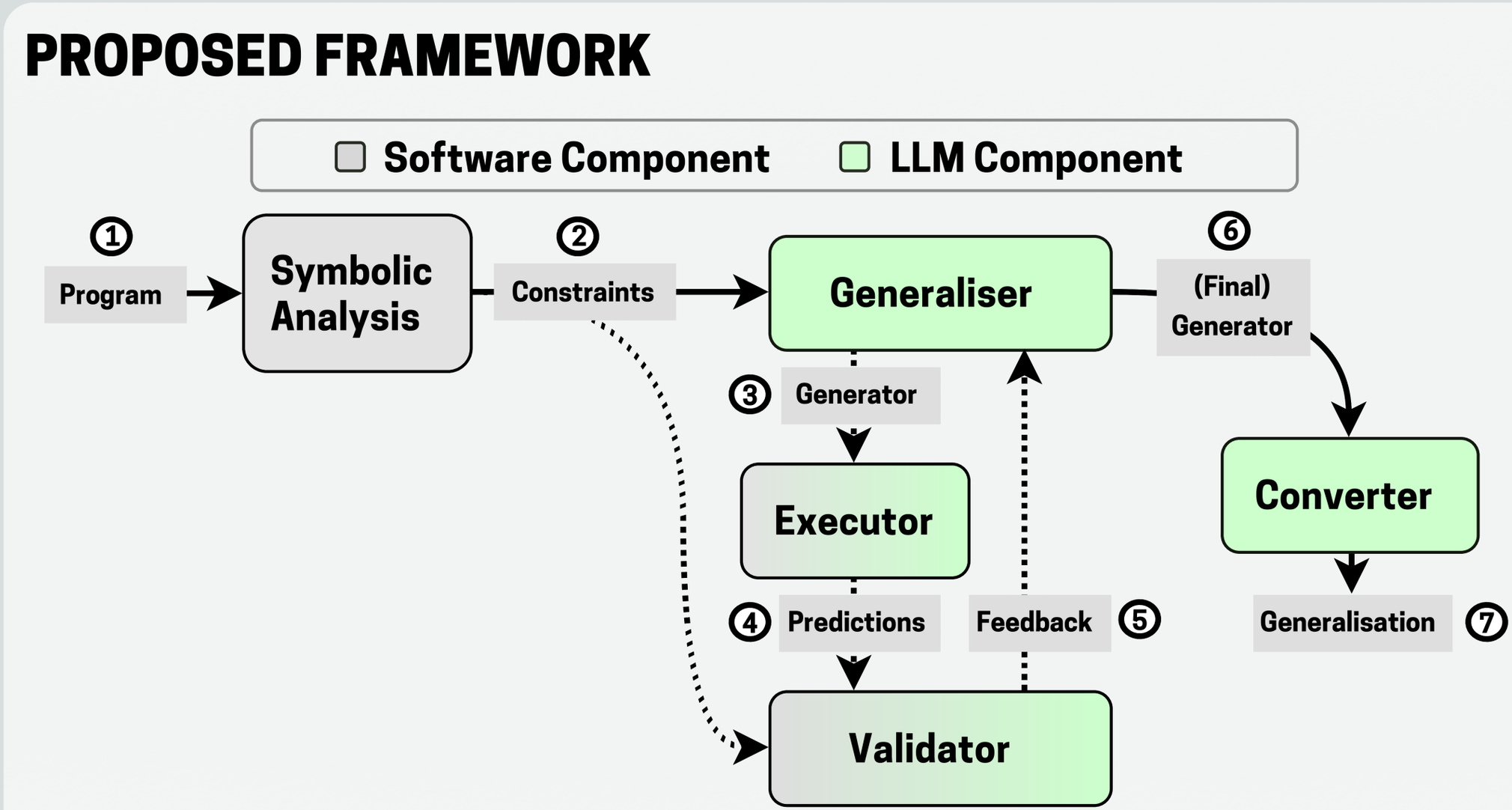
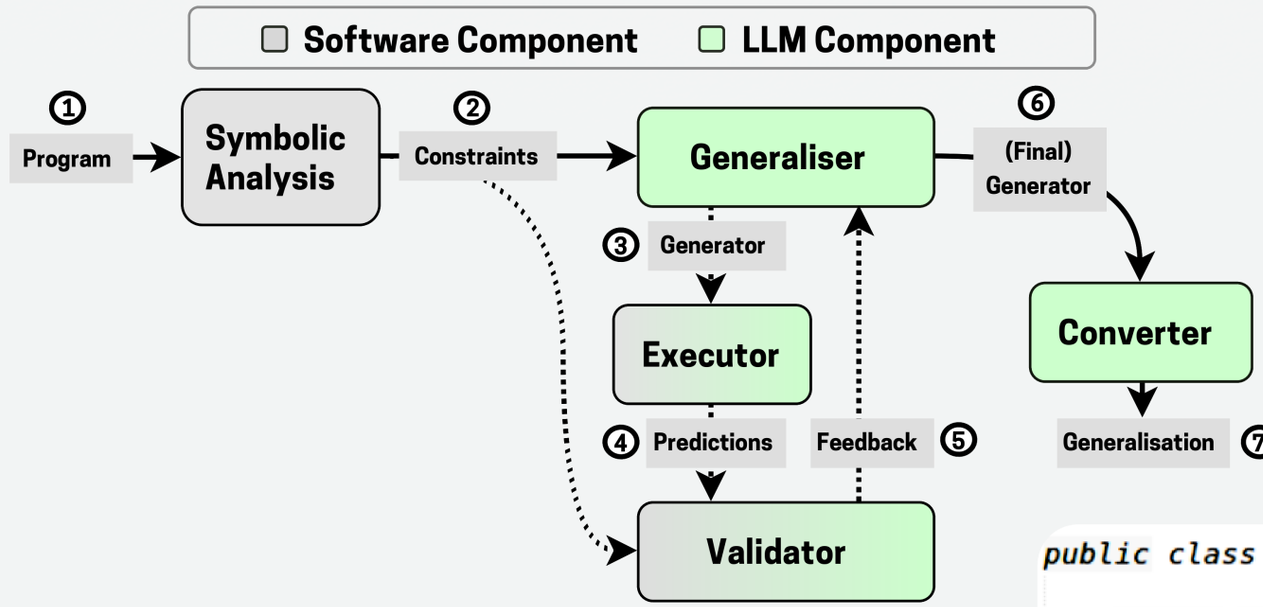
Complexity Estimation w/ Sym Exe and LLMs

- SPF WCA — used to search the space of fixed-size inputs to find the constraints for the worst-case execution paths
- Can LLMs predict the constraints for larger input sizes?
- Useful for generating worst-case inputs for actual problem size
- No known techniques for doing that

Approach


PROPOSED FRAMEWORK
Software Component
LLM Component
1
Program
Symbolic Analysis
2
Constraints
Generaliser
6
(Final) Generator
3
Generator
Executor
Converter
4
Predictions
Feedback
5
Generalisation
7
Validator

PROPOSED FRAMEWORK

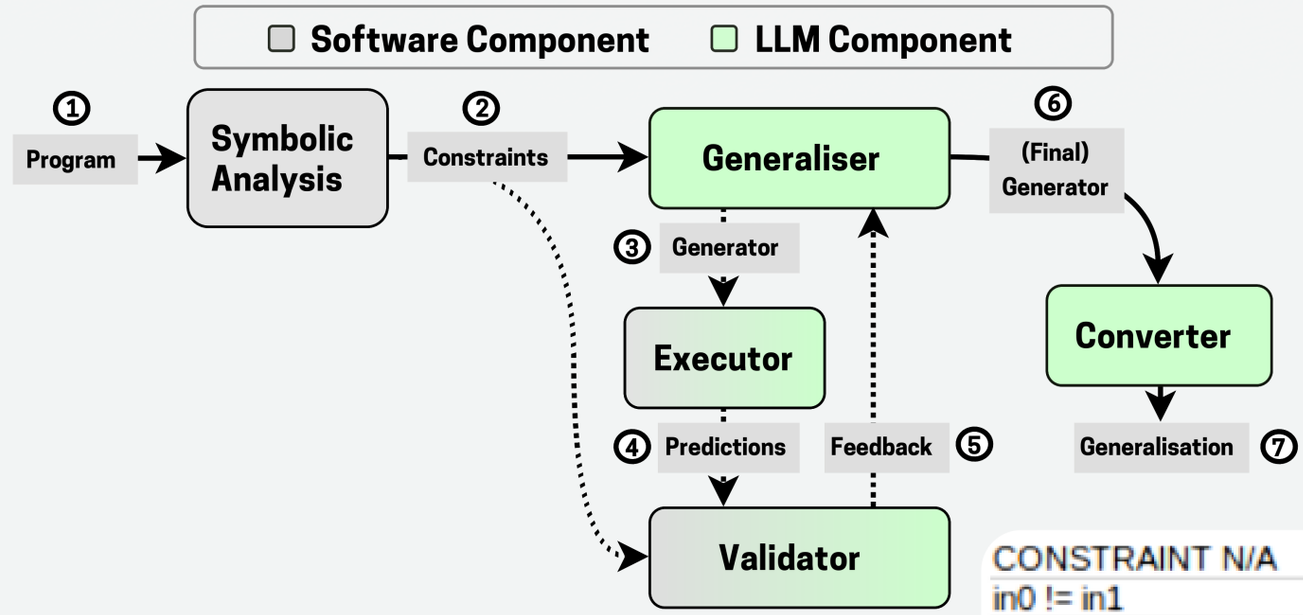



```java
public class SimpleUnique {

    public static void algo(char[] chars) {
        boolean fail = false;
        // Something is done with input variables
        for (int i = 0; i < N-1; i++) { ···
        }
        // Which determines whether expensive operation occur
        if (!fail) { ···
        }
    }
```

1

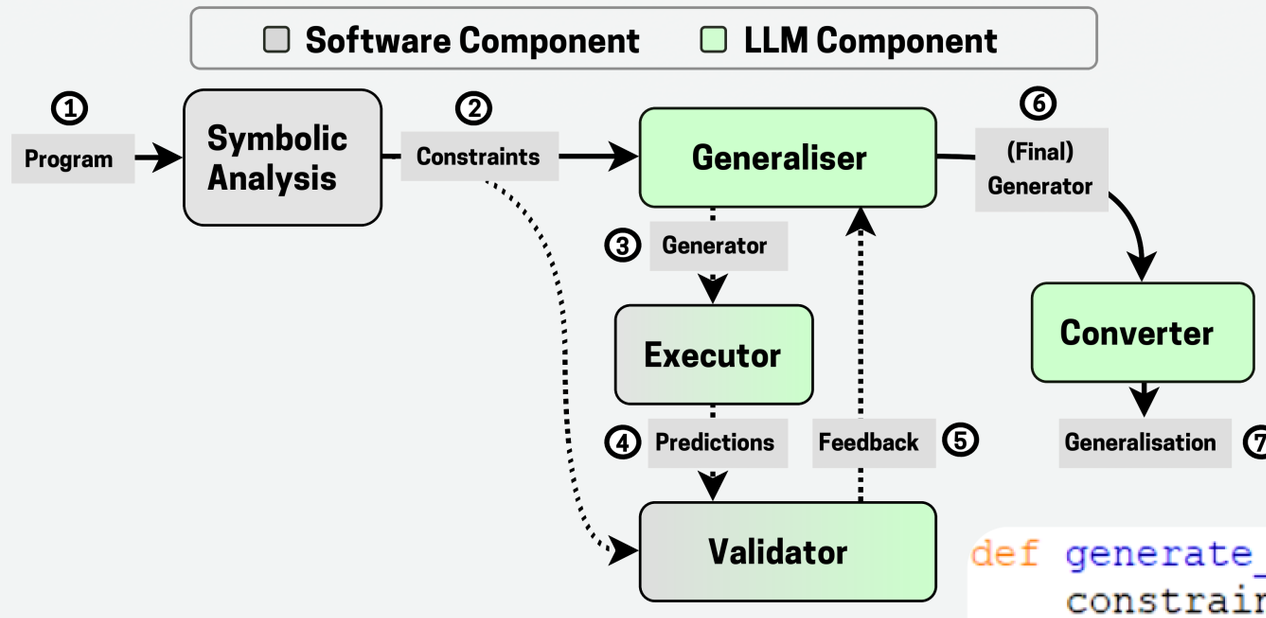
PROPOSED FRAMEWORK



② CONSTRAINT N/A

```
in0 != in1
in1 != in2 && in0 != in2 && in0 != in1
in2 != in3 && in1 != in3 && in1 != in2 && in0 != in3 && in0 != in2 && in0 != in1
in3 != in4 && in2 != in4 && in2 != in3 && in1 != in4 && in1 != in3 && in1 != in2 && i
in4 != in5 && in3 != in5 && in3 != in4 && in2 != in5 && in2 != in4 && in2 != in3 && i
in5 != in6 && in4 != in6 && in4 != in5 && in3 != in6 && in3 != in5 && in3 != in4 && i
in6 != in7 && in5 != in7 && in5 != in6 && in4 != in7 && in4 != in6 && in4 != in5 && i
in7 != in8 && in6 != in8 && in6 != in7 && in5 != in8 && in5 != in7 && in5 != in6 && i
in8 != in9 && in7 != in9 && in7 != in8 && in6 != in9 && in6 != in8 && in6 != in7 && i
```

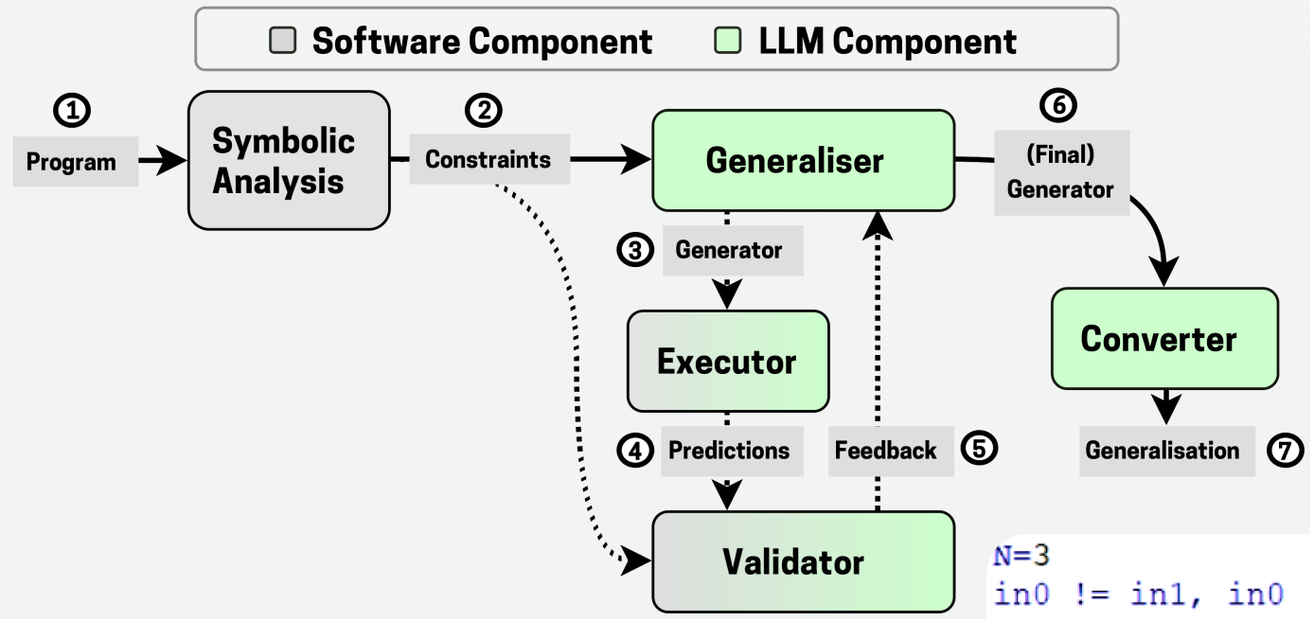
PROPOSED FRAMEWORK



③

```python
def generate_constraints(N):
    constraints = []
    for i in range(N):
        for j in range(i+1, N):
            constraints.append(f'in{i} != in{j}')
    return constraints
```

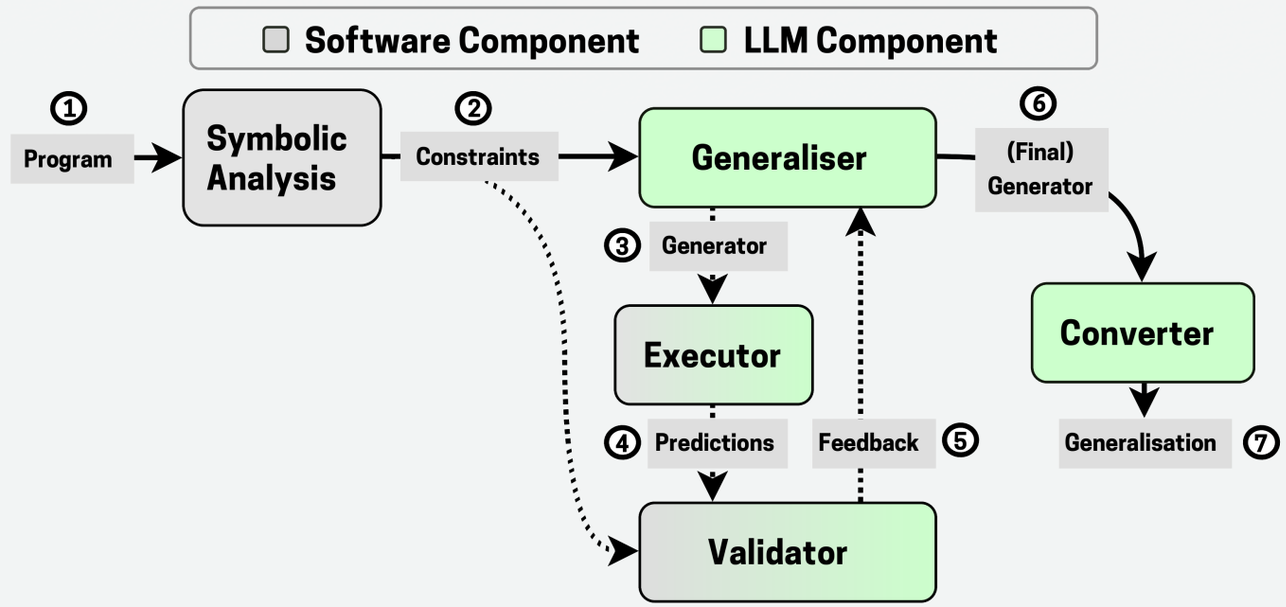
PROPOSED FRAMEWORK



④

```
N=3
in0 != in1, in0 != in2, in1 != in2
N=4
in0 != in1, in0 != in2, in0 != in3, in1 != in2, in1 != in3,
!= in3
N=5
in0 != in1, in0 != in2, in0 != in3, in0 != in4, in1 != in2,
!= in3, in1 != in4, in2 != in3, in2 != in4, in3 != in4
N=6
in0 != in1, in0 != in2, in0 != in3, in0 != in4, in0 != in5,
!= in2, in1 != in3, in1 != in4, in1 != in5, in2 != in3, in2
in4, in2 != in5, in3 != in4, in3 != in5, in4 != in5
```

PROPOSED FRAMEWORK

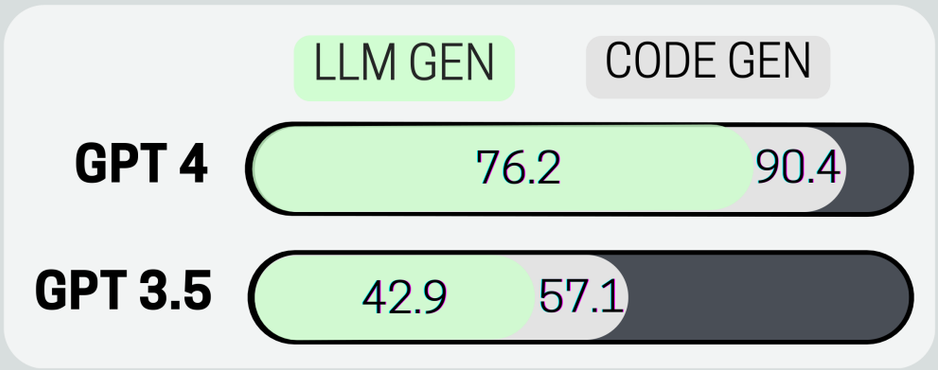


7

```
# {ss[i] != ss[j] | 0 ≤ i < j < N} for all i, j in Z+
# for all i, for all j, ((0 ≤ i < j < N) -> ss[i] != ss[j])
cond1 = ForAll([i, j], Implies(And(0 <= i, i < j, j < N), ss(i) != ss(j)))
```

METHOD AND PRELIMINARY FINDINGS

- A small test dataset of 21 Java programs created for evaluation.
- Allowed a maximum of 10 rounds of feedback before being marked unsuccessful.
- The larger GPT 4 performs much better than 3.5, possibly indicating that the real-world application of this framework only became viable recently.
- Generator evaluation has a surprising amount of influence on performance, expressing generators as executable Python code being faster to evaluate and more accurate.
- LLMs can accurately predict the constraints of program inputs over which it could not generalise,indicating the possibility for further improvements.



Generalisation success rate % over dataset, under various framework configurations

FUTURE WORK

- We Made use of Z3 Solver to prove equivalence of predictions against ground truths to improve evaluation.

- We plan to use theorem provers to show that our outputs are indeed valid. Alternatively, we may provide extensive statistical verification.
- Continuing work can benchmark against our results, but we are also interested in comparing against existing methods for finding worst-case inputs.
- Our approach is limited by an LLMs context-window size, which could be partially remedied by normalising constraints to shorter formats.
- We also aim to minimise the number of LLM API calls and improve reliability.

Thank you

Contact information: corina.s.pasareanu@nasa.gov