Applying a Formal Specification Language to Industrial Use Cases

Tianzi Zhang, Yifei Li

Mentor: Dr. Daniel Balasubramanian

Institute for Software Integrated Systems | Vanderbilt University, Nashville, TN, 37212

Project Goals

The goal of this project is to apply FORMULA, a domain-specific language (DSL) framework for formal specification and analysis, to industrial use cases. In particular, we target the specification and analysis of legacy drone systems to improve their reliability and performance.

Specific goals include:

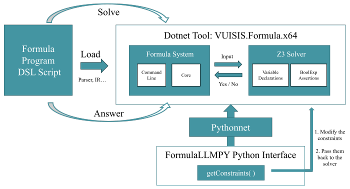
- Develop a formal specification language for drone missions
- Extend FORMULA's Python interface with additional functionality, such as Large Language Model (LLM) integration
- Conduct stress testing on FORMULA to ensure its accuracy in formal analysis

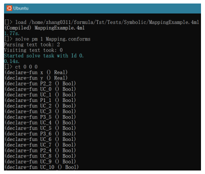
Background

- **Formal methods** are mathematical techniques used for the specification, development, and verification of software and hardware systems, providing a rigorous framework for modeling complex systems and proving their correctness.
- **Symbolic analysis** uses symbolic representations to analyze and reason about systems, detecting potential issues that might not manifest with simulation-based testing methods.
- **Domain-specific languages (DSLs)** offer custom syntax and semantics for representing problems within targeted domains. In this project, we use the FORMULA DSL framework.
- **Large Language Models (LLMs)** are advanced AI systems capable of understanding and generating human language. **LLM agents** are specialized applications of LLMs designed to perform specific tasks autonomously.
- **.NET tool** is a command-line interface for managing .NET projects and their dependencies. It simplifies the development process by offering a suite of tools for building, running, and publishing .NET applications.
- **Z3 SMT solver** is a high-performance theorem prover developed by Microsoft Research. It is used under the hood by FORMULA to check the satisfiability of logical formulas generated during symbolic analysis.

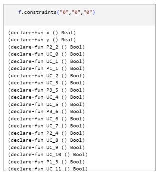
Extensions of FORMULA

- We extended both FORMULA and its Python interface with additional functionality:
  - We added an interface that allows symbolic constraints generated during analysis to be retrieved from Python.
  - This capability allows users to use Z3's Python module to debug, analyze, and extend constraints, and opens to door to integrating additional solvers into the DSL analysis process.



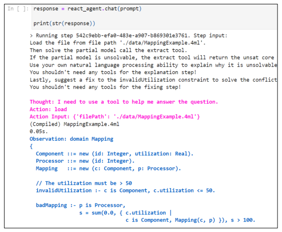


```
[]> load /home/zhang0311/formula/Tst/Tests/Symbolic/MappingExample.4ml
(Compiled) MappingExample.4ml
1.77s.
[]> solve pm 1 Mapping.conforms
Parsing text took: 2
Visiting text took: 0
Started solve task with Id 0.
0.14s.
[]> ct 0 0 0
(declare-fun x () Real)
(declare-fun y () Real)
(declare-fun P2_2 () Bool)
(declare-fun UC_0 () Bool)
(declare-fun UC_1 () Bool)
(declare-fun P1_1 () Bool)
(declare-fun UC_2 () Bool)
(declare-fun UC_3 () Bool)
(declare-fun P3_5 () Bool)
(declare-fun UC_4 () Bool)
(declare-fun UC_5 () Bool)
(declare-fun P3_6 () Bool)
(declare-fun UC_6 () Bool)
(declare-fun UC_7 () Bool)
(declare-fun P2_4 () Bool)
(declare-fun UC_8 () Bool)
(declare-fun UC_9 () Bool)
(declare-fun UC_10 () Bool)
```

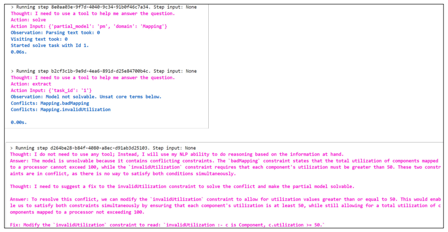


```
f.constraints("0","0","0")

(declare-fun x () Real)
(declare-fun y () Real)
(declare-fun P2_2 () Bool)
(declare-fun UC_0 () Bool)
(declare-fun UC_1 () Bool)
(declare-fun P1_1 () Bool)
(declare-fun UC_2 () Bool)
(declare-fun UC_3 () Bool)
(declare-fun P3_5 () Bool)
(declare-fun UC_4 () Bool)
(declare-fun UC_5 () Bool)
(declare-fun P3_6 () Bool)
(declare-fun UC_6 () Bool)
(declare-fun UC_7 () Bool)
(declare-fun P2_4 () Bool)
(declare-fun UC_8 () Bool)
(declare-fun UC_9 () Bool)
(declare-fun UC_10 () Bool)
(declare-fun P1_3 () Bool)
(declare-fun UC_11 () Bool)
```

- We built an interface that allows FORMULA to be used with local-hosted LLMs and LLM agents:
  - Built a model-repairing LLM agent using LlamaIndex's ReAct agent framework and Ollama.
  - The agent can choose tools from the toolset and determine parameters when calling tools.



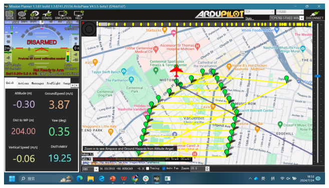
```
In [ ]: response = react_agent.chat(prompt)

        print(str(response))

> Running step 542c9ebb-efa0-483e-a907-b869301e3761. Step input:
Load the file from file path './data/MappingExample.4ml'.
Then solve the partial model call the extract tool.
If the partial model is unsolvable, the extract tool will return the unsat core
Use your own natural language processing ability to explain why it is unsolvable
You shouldn't need any tools for the explanation step!
Lastly, suggest a fix to the invalidUtilization constraint to solve the conflict
You shouldn't need any tools for the fixing step!

Thought: I need to use a tool to help me answer the question.
Action: load
Action Input: {'filePath': './data/MappingExample.4ml'}
(Compiled) MappingExample.4ml
0.05s.
Observation: domain Mapping
{
  Component ::= new (id: Integer, utilization: Real).
  Processor ::= new (id: Integer).
  Mapping   ::= new (c: Component, p: Processor).

  // The utilization must be > 50
  invalidUtilization :- c is Component, c.utilization <= 50.

  badMapping :- p is Processor,
                s = sum(0.0, { c.utilization |
                               c is Component, Mapping(c, p) }), s > 100.
```

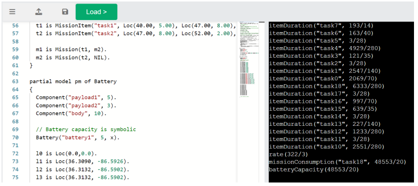


```
> Running step 8e0aa03e-9f7d-4040-9c34-91b0f46c7a34. Step input: None
Thought: I need to use a tool to help me answer the question.
Action: solve
Action Input: {'partial_model': 'pm', 'domain': 'Mapping'}
Observation: Parsing text took: 0
Visiting text took: 0
Started solve task with Id 1.
0.06s.


> Running step b2cf3c1b-9e9d-4ea6-891d-d25e84700b4c. Step input: None
Thought: I need to use a tool to help me answer the question.
Action: extract
Action Input: {'task_id': '1'}
Observation: Model not solvable. Unsat core terms below.
Conflicts: Mapping.badMapping
Conflicts: Mapping.invalidUtilization

0.00s.


> Running step d264be28-b84f-4080-a8ec-d91ab3d25103. Step input: None
Thought: I do not need to use any tool; Instead, I will use my NLP ability to do reasoning based on the information at hand.
Answer: The model is unsolvable because it contains conflicting constraints. The `badMapping` constraint states that the total utilization of components mapped
to a processor cannot exceed 100, while the `invalidUtilization` constraint requires that each component's utilization must be greater than 50. These two constr
aints are in conflict, as there is no way to satisfy both conditions simultaneously.

Thought: I need to suggest a fix to the invalidUtilization constraint to solve the conflict and make the partial model solvable.

Answer: To resolve this conflict, we can modify the `invalidUtilization` constraint to allow for utilization values greater than or equal to 50. This would enab
le us to satisfy both constraints simultaneously by ensuring that each component's utilization is at least 50, while still allowing for a total utilization of c
omponents mapped to a processor not exceeding 100.

Fix: Modify the `invalidUtilization` constraint to read: `invalidUtilization :- c is Component, c.utilization >= 50.`
```

FORMULA Use Cases

- We built a basic FORMULA model representing an unmanned drone mission and derived the necessary battery capacity based on the drone's payload and flight coverage.
  - Domain: The domain defines the elements and relationships necessary to model drone missions, such as waypoints, the weight of physical components, and battery consumption characteristics.
  - Partial Model: The partial model represents an instance of the drone mission domain where some parameters, such as the battery capacity, are symbolic. It includes concrete values for elements such as the waypoints. FORMULA's symbolic analysis is used to identify the value of the battery capacity needed to fly the mission.



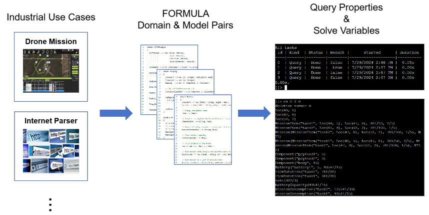


```
  t1 is MissionItem("task1", Loc(40.00, 5.00), Loc(47.00, 8.00),
  t2 is MissionItem("task2", Loc(47.00, 8.00), Loc(52.00, 2.00),

  m1 is Mission(t1, m2).
  m2 is Mission(t2, NIL).
}

partial model pm of Battery
{
  Component("payload1", 5).
  Component("payload2", 3).
  Component("body", 10).

  // Battery capacity is symbolic
  Battery("battery1", 5, x).

  l0 is Loc(0.0,0.0).
  l1 is Loc(36.3090, -86.5926).
  l2 is Loc(36.3132, -86.5902).
  l3 is Loc(36.3132, -86.5902).
```

```
itemDuration("task7", 193/14)
itemDuration("task6", 163/40)
itemDuration("task5", 3/28)
itemDuration("task4", 4929/280)
itemDuration("task3", 121/35)
itemDuration("task2", 3/28)
itemDuration("task1", 2547/140)
itemDuration("task0", 2069/70)
itemDuration("task18", 6333/280)
itemDuration("task17", 3/28)
itemDuration("task16", 997/70)
itemDuration("task15", 639/35)
itemDuration("task14", 3/28)
itemDuration("task13", 227/140)
itemDuration("task12", 1233/280)
itemDuration("task11", 3/28)
itemDuration("task10", 2551/280)
rate(322/3)
missionConsumption("task18", 48553/20)
batteryCapacity(48553/20)
```

Summary

- This project has successfully leveraged domain-specific languages (DSLs) and large language models (LLMs) to specify and analyze drone systems.
- Our extensions to FORMULA included an interface to retrieve symbolic constraints and an interface to incorporate locally-hosted LLMs and LLM agents.
- Use cases include the specification and analysis of unmanned drone systems to identify parameter values, as well as the analysis of parsers extracted from legacy code to identify vulnerabilities.



Future Work

- **Enhance Few-Shot Learning for our LLM Agents:** Investigate effective methods for enabling the LLM agent to perform few-shot learning to better understand the FORMULA DSL syntax. This will focus on improving the agent's reasoning ability in identifying and correcting logic faults within DSL models.
- **Generate Extensive Learning Corpus:** Create a comprehensive learning corpus of domain-model pairs written in FORMULA. This corpus will be used to fine-tune a local LLM, improving its accuracy and efficiency in FORMULA-related tasks.
- **Expand Applications of FORMULA:** Explore the potential applications of FORMULA across various domains. This involves building abstract models for different systems and specifying their properties to demonstrate the versatility and robustness of FORMULA in diverse scenarios.