Chorer: a tool for Program Understanding [1]

G. Genovese[1,2], I. Lanese[3], C. Di Giusto[2], E. Tuosto[4], G. Vidal[5]

1 University of Bologna, Bologna, Italy 2 Université Cote d'Azur, CNRS, I3S, France
3 Olas Team, University of Bologna, Bologna, Italy 4 Gran Sasso Science Institute, Italy
5 VRAIN, Universitat Polit`ecnica de Val`encia, Spain

This tool could help you debug your program!

The idea

We wanted to build a tool that could assist developers in analyzing existing codebases by exploiting the **Choreography Automata** [2] theory to uncover classic bugs in distributed systems, particularly in the context of *actor-based programming*. To achieve this, our extraction algorithm follows a **bottom-up approach** with **over-approximation**, ensuring that every possible behavior (good or bad) is highlighted.

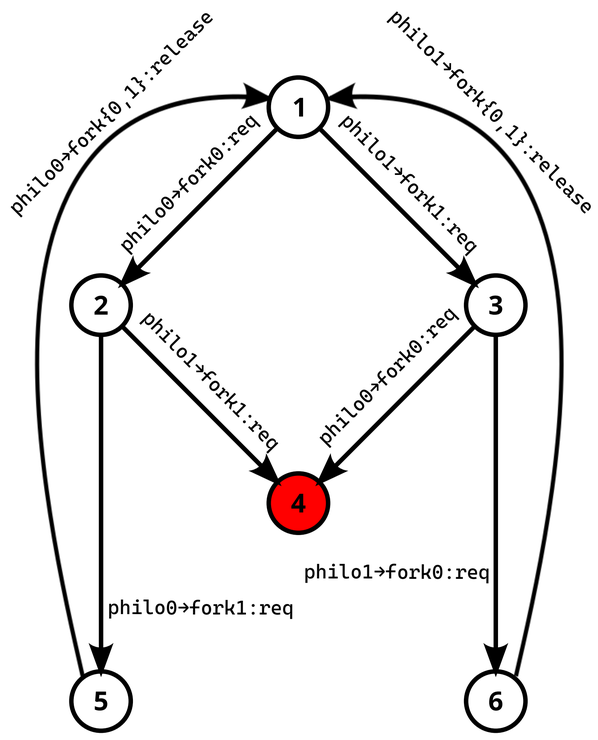
Extraction steps

1. Take the file input and perform some simple analysis
2. Extract a local-view for each actor found
3. Create a global-view using the local-views
4. Minimize and analyse the output graph

Pseudocode

```
philosopher(Fork1, Fork2) →
  send req to Fork1,
  receive ack from Fork1,
  send req to Fork2,
  receive ack from Fork2,
  eat(),
  send release to Fork1,
  send release to Fork2,
  philosopher(Fork1, Fork2).

fork() →
  receive req from Phil,
  send ack to Phil,
  receive release from Phil,
  fork().
```



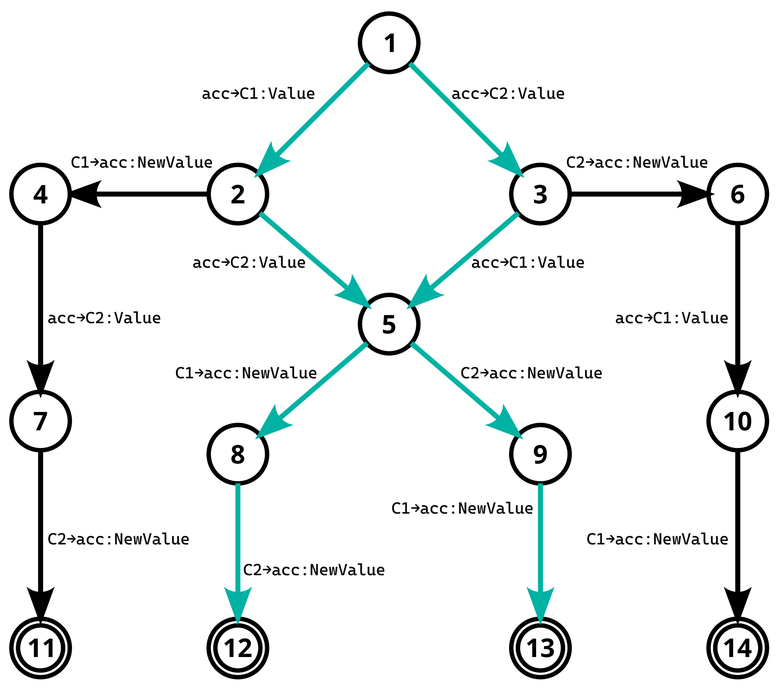
A motivating example

On the left, you can see an implementation of the **dining philosophers** problem written in *Erlang-like* pseudocode with its corresponding Choreography Automaton extracted by the tool. The system consists of four actors: two philosophers and two forks. The *red* state highlights a **deadlock** situation, which occurs when both philosophers attempt to take the forks simultaneously.

Pseudocode

```
account(Value) →
  receive
    read from Client →
      send Value to Client,
      account(Value),
    NewValue from Client →
      account(NewValue).

client() →
     receive read from Acc,
     receive Value from Acc,
     % operation on Value
     send NewValue to Acc.
```



Another example

We now demonstrate how the tool can highlight a **race condition**. Suppose a *banking application* attempts to read a client's credit information. If two types of client actors exist (let's call them *C1* and *C2*) the system may encounter a race condition. This potential behavior is illustrated by the *light blue* path in the automaton.

Conclusion

Building an effective tool that truly achieves its goals involves several challenges. The problem is inherently **undecidable**, as it would require solving termination (an already known undecidable problem). Moreover, the extraction process can produce extremely **large automata**. Significant effort has been dedicated to ensuring a correct and practical synthesis of the automaton.

Referance
[1] Genovese, et al. "Choreographies for Program Understanding", International Conference on Formal Techniques for Distributed Objects, Components, and Systems. 2025.
[2] Barbanera, et al. "Choreography automata" International Conference on Coordination Languages and Models. Cham: Springer International Publishing, 2020.

Repository

github.com/gabrielegenovese/chorer

gabriele.genovese2@studio.unibo.it