06Lab
System modelling, a Programming Language approach: non-determinism, unbounded size, Petri Nets

Mirko Viroli
mirko.viroli@unibo.it

C.D.L. Magistrale in Ingegneria e Scienze Informatiche
Alma Mater Studiorum—Università di Bologna, Cesena

a.a. 2023/2024

One slide sum-up on Modelling

Concepts

- inspired by MDE, models are key to design (hence implementation), and verification (hence validation)
- behavioural models are systems per se, hence can be run/animated/simulated/analysed
- good engineering of behavioural models can be done at ProgLang level, and can lead to nice reusable meta-models

A stack of "models": example

- Model: Petri-Net modelling of "readers and writers" (specific safety properties, design template)
- Meta-model: Petri Nets (captures parallelism, unbounded size, synchronisation, ..., could be extended)
- Meta-meta-model: transition systems (captures non-deterministic state transition)

API organisation

- syntax of the model: via a DSL, or expressive API
- transition system: compiled from the DSL (states and transition rules)
- possibly adding analysis methods (carefully addressing loops/non-termination/divergence)

Starting point and goals

References

- 06-repo: from virtuale

General goals

- get acquainted with our modelling stack, possibly improve it
- play with Petri Nets
- design new models

Tasks

VERIFIER

- Code and do some analysis on the Readers & Writers Petri Net. Add a test to check that in no path long at most 100 states mutual exclusion fails (no more than 1 writer, and no readers and writers together). Can you extract a small API for representing safety properties? What other properties can be extracted? How the boundness assumption can help?

DESIGNER

- Code and do some analysis on a variation of the Readers & Writers Petri Net: it should be the minimal variation you can think of, such that if a process says it wants to read, it eventually (surely) does so. How would you show evidence that your design is right? What about a variation where at most two process can write?

ARTIST

- Create a variation/extension of `PetriNet` meta-model, with `priorities`: each transition is given a numerical priority, and no transition can fire if one with higher priority can fire. Show an example that your pretty new "abstraction" works as expected. Another interesting extension is "coloring": tokens have a value attached, and this is read/updated by transitions.

TOOLER

- The current API might be re-organised: can we generate/navigate all paths thanks to caching and lazy evaluation? can we use monads/effects to capture non-determinism? can we generate paths and capture safety properties by ScalaCheck?

PETRINET-LLM

We know that LLMs/ChatGPT can arguably help in write/improve/complete/implement/reverse-engineer standard ProgLangs. But is it of help in designing Petri Nets? Does it truly "understand" the model? Does it understand our DSL by examples?