Assignment 5
Code Generation in AToMPM

Randy Paredis
randy.paredis@uantwerpen.be

1 Practical Information

The goal of this assignment is to generate Petri-Nets and formal temporal logic for the production system modelling language in the visual modelling tool **AToMPM**, export these nets and logic to **LoLA** format, perform the analysis, and visualize the analysis results back in AToMPM.

This assignment builds on the previous assignment. Make sure you have fully finished that assignment before continuing, as it will heavily impact your workflow.

The different parts of this assignment:

1. Export your Petri-Net from AToMPM to metaDepth, and then transform to LoLA syntax.
2. Build a language for specifying formal temporal logic patterns within AToMPM.
3. Build your temporal logic formula within AToMPM, and (manually) transform it to LoLA syntax.
4. Verify the formula against the Petri-Net within LoLA, and obtain a trace.
5. Execute the trace within AToMPM on the production system, using the "fire transition" sequence you created in the previous assignment.
6. Write a report that includes a clear explanation of your complete solution and the modelling/analysis choices you made, as well as an explanation of your testing process. Also mention possible difficulties you encountered during the assignment, and how you solved them.

This assignment should be completed in groups of two if possible, otherwise individually is permissible.

Submit your assignment as a zip file (report in pdf + model files) on Blackboard before **21 December 2021, 23:59h**[1]. If you work in a group, only *one* person needs to submit the zip file, while all others *only* submit the report. Contact Randy Paredis if you experience any issues.

[1] Beware that BlackBoard's clock may differ slightly from yours.

2 Requirements

This section lists the requirements of the above steps and the report. Make sure to test each requirement with test models!

2.1 Setup

metaDepth Setup Download the file from: http://msdl.cs.mcgill.ca/people/hv/teaching/MSBDesign/exported_to_md.zip and place the contents in your `AToMPM/exported_to_md` folder. Also copy/move the `metaDepth.jar` in the `AToMPM/exported_to_md` folder.

LoLA Setup Then you'll need to install LoLA 2.0, a command-line Petri-Net analysis tool. LoLA also has a comprehensive manual available within the 'doc' folder describing the analysis possibilities and commands.

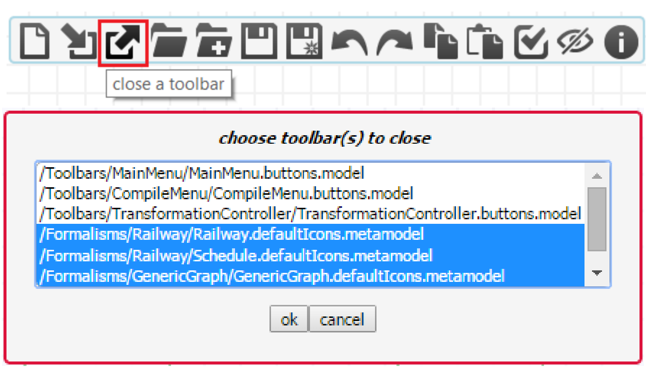
Download the files from https://msdl.uantwerpen.be/cloud/public/lola20 and see the `README` and/or `INSTALL` for compilation and install directions.

NOTE: LoLA is made specifically for Linux-distributions. On Windows, it can be executed by using Cygwin. It is your own responsibility to get it working.

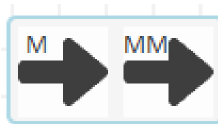
2.2 Exporting Petri-Nets to LoLA

1. Generate your Petri-Net model using your rule-based model transformation from the last assignment.

2. Remove all traceability links and production system elements by closing the respective toolbars.
   NOTE: Make sure you keep a saved version of your model, which allows for simulation (per assignment 4).



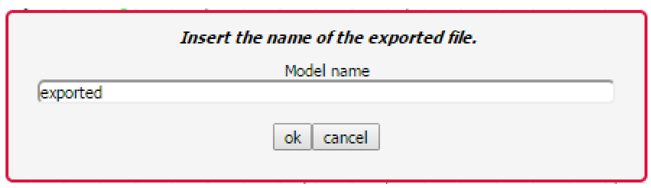
3. Load the metaDepth toolbar (inside of the `/Toolbars/MetaDepth/` folder, it is called `Export.buttons.model`). This toolbar has two buttons: one for exporting models, and one for exporting metamodels.



4. Export your custom Petri-Net metamodel by clicking the second button.

   - If you did not change the abstract syntax of the Petri-Net formalism, you can simply use the `PN.mdepth` file that is located in the zip.

5. Open your factory net and click on the button for exporting models. Ensure that “exported” is written in the text field. Click OK. This will generate a file with name “exported.mdepth” in the `AToMPM/exported_to_md` folder.



5. Write your EGL code in the file `AToMPM/exported_to_md/generate_LoLA.egl`. This file must generate a valid LoLA model using a template-based approach.

   - To see a template example, examine page 8 of `https://metadepth.org/papers/computer.pdf`
   - There is an example LoLA model for a Petri-Net of a chat room.
   - The repository `https://msdl.uantwerpen.be/git/bentley/lola_utils` has a Python script `LoLADraw.py` to visualize LoLA models.

6. To actually generate the `*.lola` file, execute the command `java -jar metaDepth.jar < commandlist_lola` inside of the `AToMPM/exported_to_md/` folder. This will generate a file called `exported.lola`.

7. This new file can be loaded in LoLA to perform your analysis as described below.

2.3 Building the Temporal Logic Pattern Language

In this part of the assignment, you will create the abstract syntax, concrete syntax, and example models for temporal logic property formulas. These formulas will be at the level of abstraction of the production system, and will be based on property specification patterns.

Property specification patterns specify high-level properties about the events or states of a system. For example, here is a property we are interested to verify on our system: "**Globally**, if *a cube and a sphere are at the assembler*, then in **Response** *an assembled item is produced at the assembler* eventually."

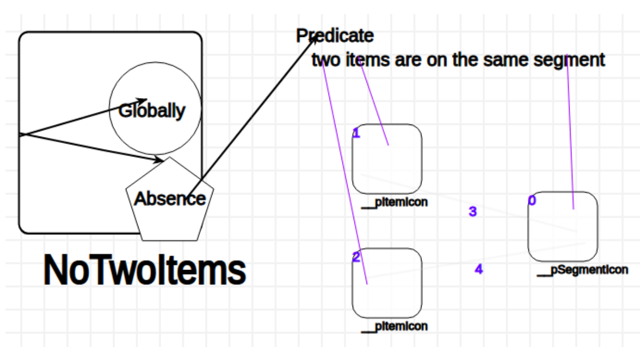
From this example, we see the two components of property specification patterns: *Scopes* and *Patterns*

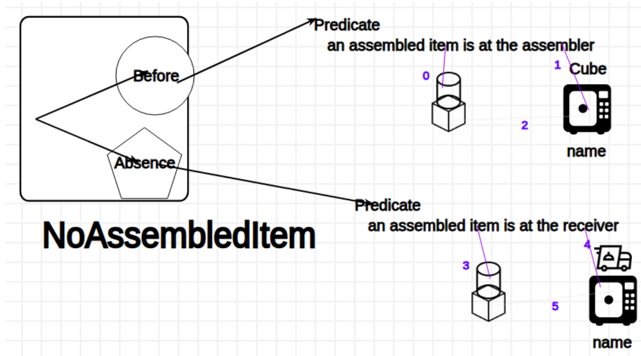
- A **Scope** defines in what conditions the pattern should hold
  - In this assignment, we will consider three scopes: *Globally*, *Before* a state/event, and *After* a state/event.
- **Pattern**: which defines which states/events should hold.
  - In this assignment, we will consider four patterns: *Universality* 'always a state occurs', *Absence* 'never a state occurs', *Existence* 'eventually a state/event occurs', *Response* 'after state/event P occurs, then state/event S occurs'.

A few examples of interesting properties:

- Example 1: "**Globally**, it is never the case (**Absence**) that *two items are on the same segment*".
- Example 2: "**Globally**, if *an assembled item is at the loading bay*, then in **Response** *the loading bay becomes empty*".
- Example 3: "**Before** *an assembled item is at the assembler*, it is never the case (**Absence**) that *an assembled item is at the loading bay*".
- Example 4: "**Globally**, it is never the case (**Absence**) that *the factory terminates*".

The abstract and concrete syntaxes of your language will therefore have to represent these simple temporal logic properties. To represent the state of the production system, the graphical syntax of the production system must be connected in some way. For example, the top part of this figure shows the statement for Example 1, and the bottom part of this figure is for Example 3.





- Note: The predicates above refer to a particular *state* or *event* of the production system. That is, we do not refer to the firing of rules on the production system.

- You will need a special predicate type to handle the state of *the factory stops*. This construct will correspond to the Petri Net being deadlocked.

Add constraints and cardinalities as appropriate to the abstract syntax of the pattern language. Also, try to introduce the informal English statements (*it is never the case*, etc.) into the concrete syntax.

2.4 Defining Temporal Logic Properties and Exporting

2.4.1 Creating the Properties

Now that you have a temporal logic language, you will create temporal logic properties for the production system as models in that language.

Produce **six** temporal logic properties which are interesting to prove on your production systems. One must be about the factory running forever/stopping. In your report, show the property models, write the entire informal English statement for each property, and describe why it is interesting and what you expect to happen.

Feel free to reuse the examples above and/or modify them as needed. Note also that it's *not an issue if one of the properties above cannot be expressed for your production system, or if the properties you create fail to hold on your production system.* In this assignment, we are concerned with how to check interesting properties of our model, not on whether your production system has the correct semantics.

2.4.2 Exporting the Properties

These properties must be exported to the LoLA formula syntax. Note that this could be achieved by utilizing the Petri-Net creation transformation created in the last assignment to get the equivalent place markings from the production system elements, and then using a model-to-text exporter to generate the formulas. To simplify matters, you can just create the equivalent LoLA formula (or series of formulas) manually for each temporal logic property. Make sure to add them to your report!

For example, the top statement in Figure 2.3 is represented by the LoLA boundness checks formulas "`AG PLACEX <= 1`" where `PLACEX` is the name of every place in the Petri-Net which represents a Segment in the production system.

For the bottom statement in Figure 2.3, the formula is more interesting:

1. First, map the predicates to the equivalent Petri-Net markings:
   - `Pred_AtAssembler = (P_Assembler_HasAssembledItem == 1)`
   - `Pred_AtReceiver = (P_Receiver_HasAssembledItem == 1)`
   - This depends on how you encode information in your Petri-Net.
2. Then look at the statement as the scope and pattern
   - *Before* `Pred_AtAssembler`, *Absence* `Pred_AtReceiver`
3. Map the scopes and patterns to the underlying temporal logic using Table 1.
   - "*Before R, Absence P*" becomes "*Eventually R* $\rightarrow$ *(not P until R)*"
4. Plug in the Petri Net markings and convert to LoLA syntax
   - *EVENTUALLY (P_Assembler_HasAssembledItem = 1) $\rightarrow$ (NOT (P_Receiver_HasAssembledItem = 1) UNTIL (P_Assembler_HasAssembledItem = 1))*

2.5 Verification and Executing the Trace

For each one of the temporal patterns and LoLA formulas from the last section, write in your report the command used to run LoLA for that formula and the formula result. See the LoLA documentation in the `docs/` folder for more information.

Note that verifying formulas for each place in the Petri-Net is best done with a (trivial) helper script[2]: https://msdl.uantwerpen.be/git/bentley/lola_utils/src/master/LoLARunner.py

Some of the analyses may run for a while on your solution. Try running the command with and without the '–search='cover'' flag to help with this. Consider a formula to run forever if it has searched hundreds of thousands or millions of states. If this is the case, argue why this formula could take forever as well as if/why this proves the formula. Hint: Section 5.2 of the LoLA documentation discusses flags for early termination of formulas.

If possible, show the marking or firing path for formulas. Section 8.1 and 8.2 of the LoLA manual specify flags for producing markings and paths for formulas.

[2]You may edit this script however you like.

	Universality P
Globally	always(P)
Before R	eventually → (P until R)
After Q	always(Q → always(P))
	Absence P
Globally	always(not P)
Before R	eventually R → (not P until R)
After Q	always(Q → always(not P))
	Existence P
Globally	eventually(P)
Before R	always(not R) or (not R until (P and not R))
After Q	always(not Q) or eventually(Q and eventually P))
	Response P then S
Globally	always(P → eventually S)
Before R	eventually R→(P→(not R until(S and not R))) until R
After Q	always (Q → always(P → eventually S))

Table 1: Scope and pattern mapping to temporal logic.

Executing the Trace For at least one formula, obtain an interesting firing path and execute it within AToMPM on your production system. This should be done using the mechanism for selecting transitions as described in the last assignment. Document a few steps of this execution in your report, and record this firing as a video.

Again, it is interesting and you will not lose marks if the trace shows that something is incorrect in your system, such as your Petri-Net deadlocking. The point of the assignment is to explore how to use round-trip transformations to verify your models.

If none of the analyses you perform produces a path, intentionally break your Petri-Net to produce a deadlocking net. Report how you broke your solution, and how the trace helps debug the error.

3 Report

There are a number of requirements for the report. Above all, the marker must be able to read the report and have a clear understanding of all aspects of the assignment, without having to investigate the model files. I.e., your model files will only be used as a support for your report, not the other way around!

Specifically, the report must contain:

- A brief outline of how the code template, example models, and formulas meet the requirements of the assignment as described in each section above.

- In particular,
  - Show your template file in a listing, with syntax colouring and comments.
  - Show the property specification language in AToMPM (AS and CS) and the formula models.
  - Show the LoLA formulas and the commands used to verify them.
- A discussion of any interesting decisions made and possible improvements to any model or language.
- Two example production systems.
- For each production system, show:
  - The results and discussion of formula checking for each formula.
  - The traces (if any) reported by LoLA.
- Choose one production system and produce a short screen recording of the Petri-Net execution transformation running and showing interesting behaviour on a trace produced by LoLA.
  - This video should not be submitted with your assignment (due to a large file size), but a link to where your video can be watched should be placed in your report. Note: this is not a download link!
  - For instance, you can upload it to YouTube, Vimeo, Google Drive, Dropbox,... Note that it should be unlisted, so it cannot be found, except when using the link.
  - A short description could be provided below the video, but no captions/voiceover/editing is required.
  - You can use OBS (`https://obsproject.com/`) or any other screen recording software.

4 Useful Links and Tips

- LoLA - Petri Net analyser: `https://msdl.uantwerpen.be/cloud/public/lola20`
- AToMPM main page: `https://atompm.github.io/`
- Download and code: `https://github.com/AToMPM/atompm`
- Documentation: `https://atompm.readthedocs.io/en/latest/`

Acknowledgements

Based on an earlier assignment by Bentley Oakes.