Ladder Metamodeling & PLC Program Validation through Time Petri Nets

Darlam Fabio Bender[1,2] Benoît Combemale[1] Xavier Crégut[1]
Jean-Marie Farines[2] Bernard Berthomieu[3] François Vernadat[3]

[1] *Institut de Recherche en Informatique de Toulouse* (CNRS UMR 5505)
Université de Toulouse, France.

[2] *Departamento de Automação e Sistemas*
Federal University of Santa Catarina. Florianopolis, Brazil.

[3] *Laboratoire d'Analyse et d'Architecture des Systemes* (CNRS)
Université de Toulouse, France.

This work is supported by the TOPCASED project, part of the French cluster *Aerospace Valley* (granted by the French DGE), cf. `http://www.topcased.org`

Motivation

- **Context:** PLC, Programmable Logical Controller
  - special purpose industrial computer used to automate industrial process
  - connected to inputs and outputs
  - controls the states of outputs according to inputs and internal state
  - programmed with Ladder Diagram (LD) and other languages [IEC 61131]
- **Problem:** Verification of Ladder Diagrams
  - actually, mainly achieved through exhaustive testing
  - purpose: experiment a model-checking approach
  - based on Model-Driven Engineering
- **Focus:** Race condition detection on Ladder diagrams
- **Approach**
  - Ladder translational semantics to Time Petri nets
  - use of Model-Driven Engineering (MDE)
  - use of Tina toolkit (model-checking)

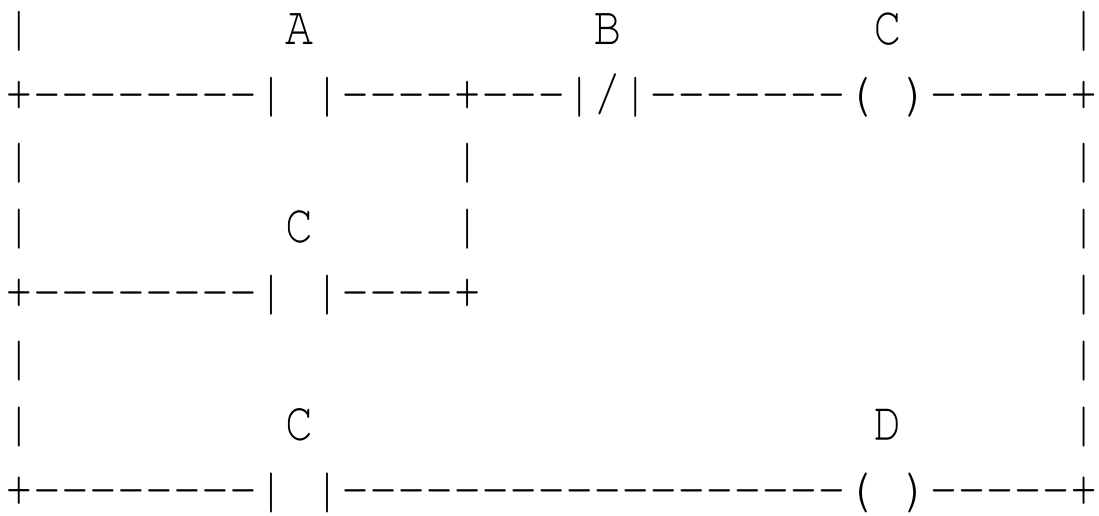
Outline

Plan

Ladder Diagrams

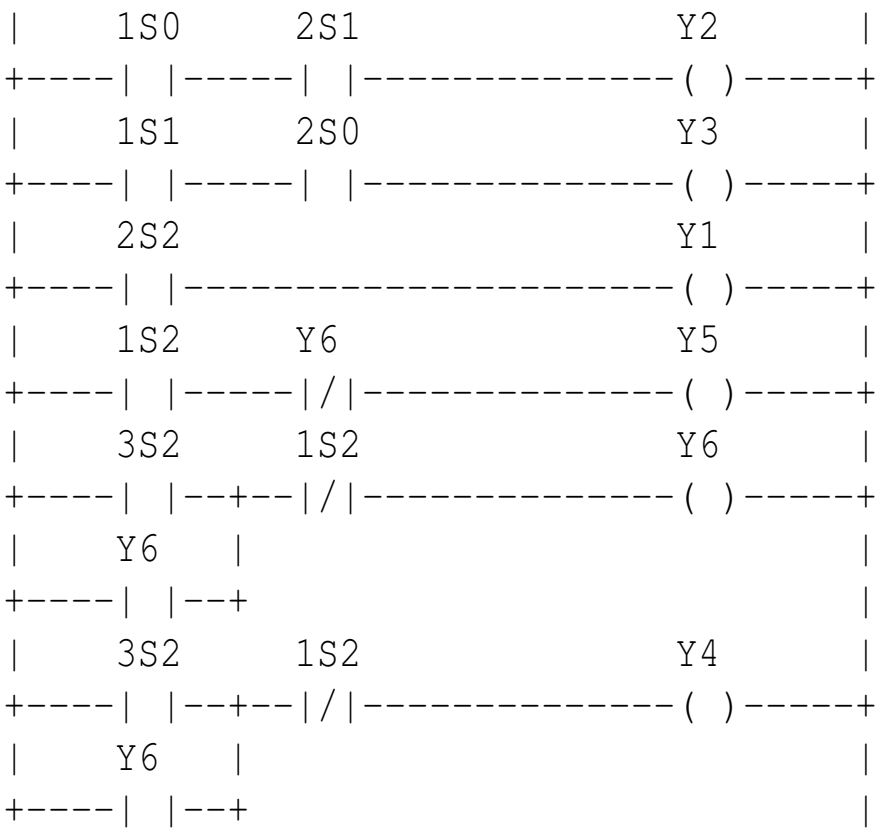
Main concepts

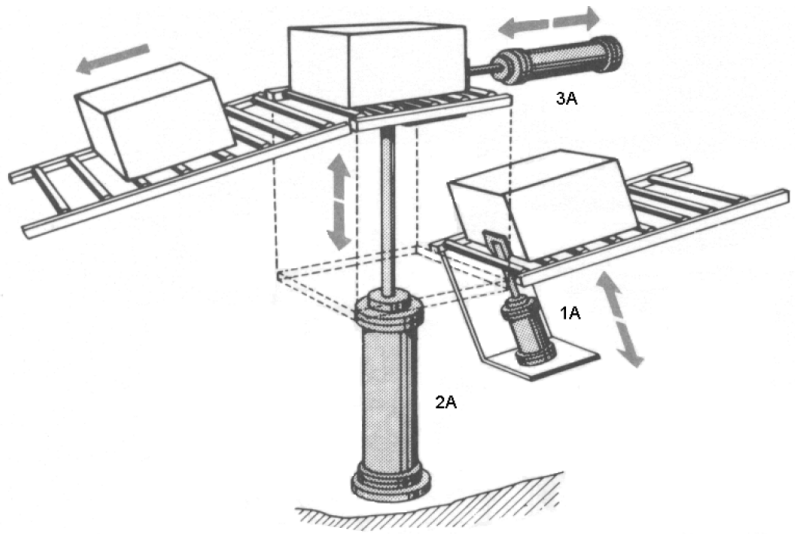
- **An example:**



- **Main concepts:** Rail, Rung, Contact (|| and |/|), Coil ( ), Variable
- **Interpretation:** from up to down.
  - $C = (A \vee C) \wedge \neg B = (A \wedge \neg B) \vee (C \wedge \neg B) = path1 \vee path2$,
  - $D = C$.
- **Other concepts:** (Not handled in this work)
  - Function Blocks with inputs, outputs, internal states.

A more realistic example

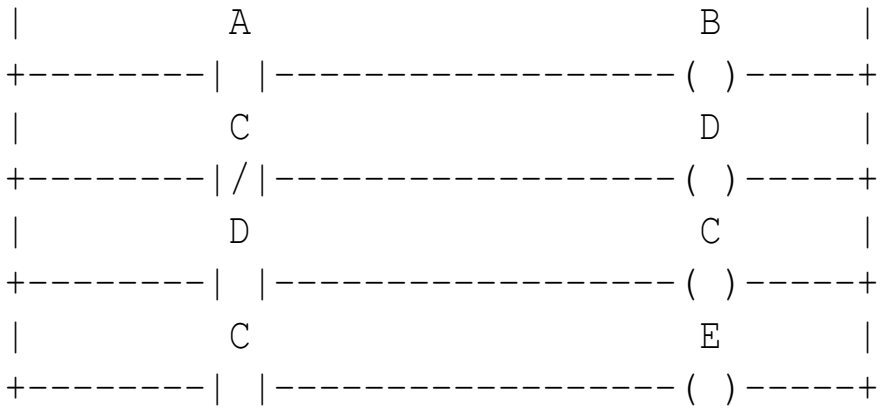




Ladder Diagrams

Race condition

- **Race condition:**
  Under fixed inputs (and function block states), one or more outputs keep changing their value.
- An **example**



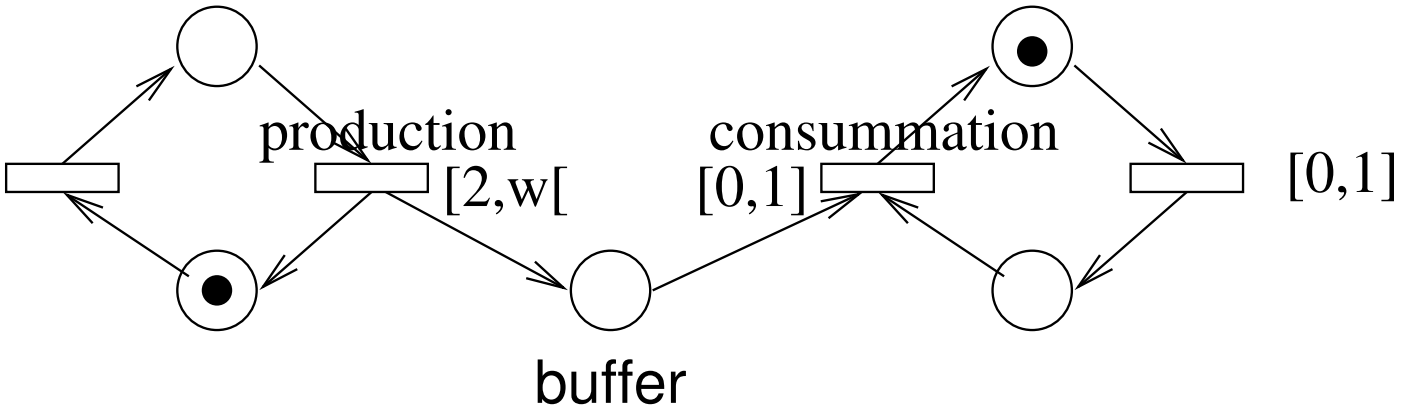
  - $B = A$
  - $D = \neg C$
  - $C = D$
  - $E = C$

  The values of D, C and E keep changing.
- See later

Time Petri Net

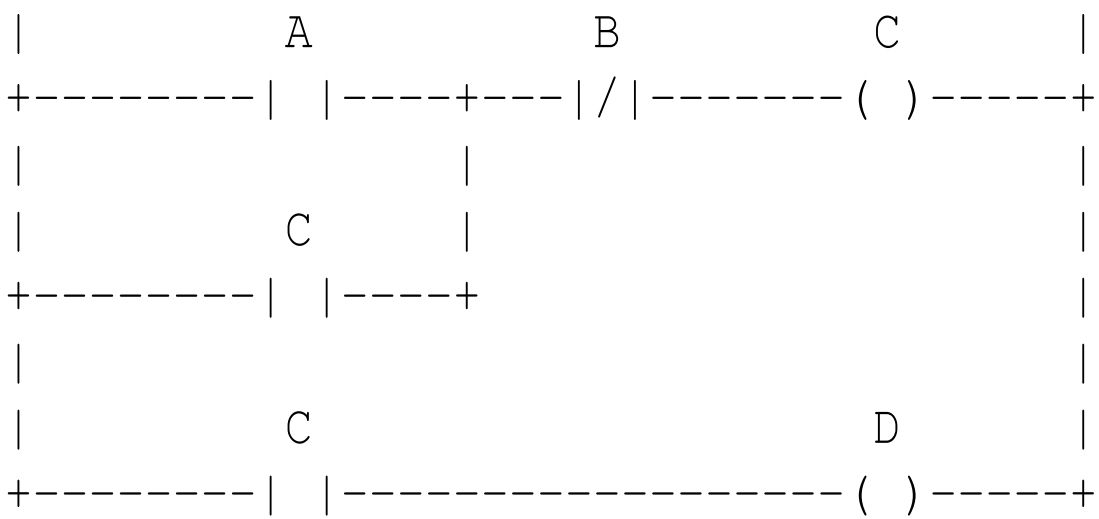
Main concepts

- **An example:**

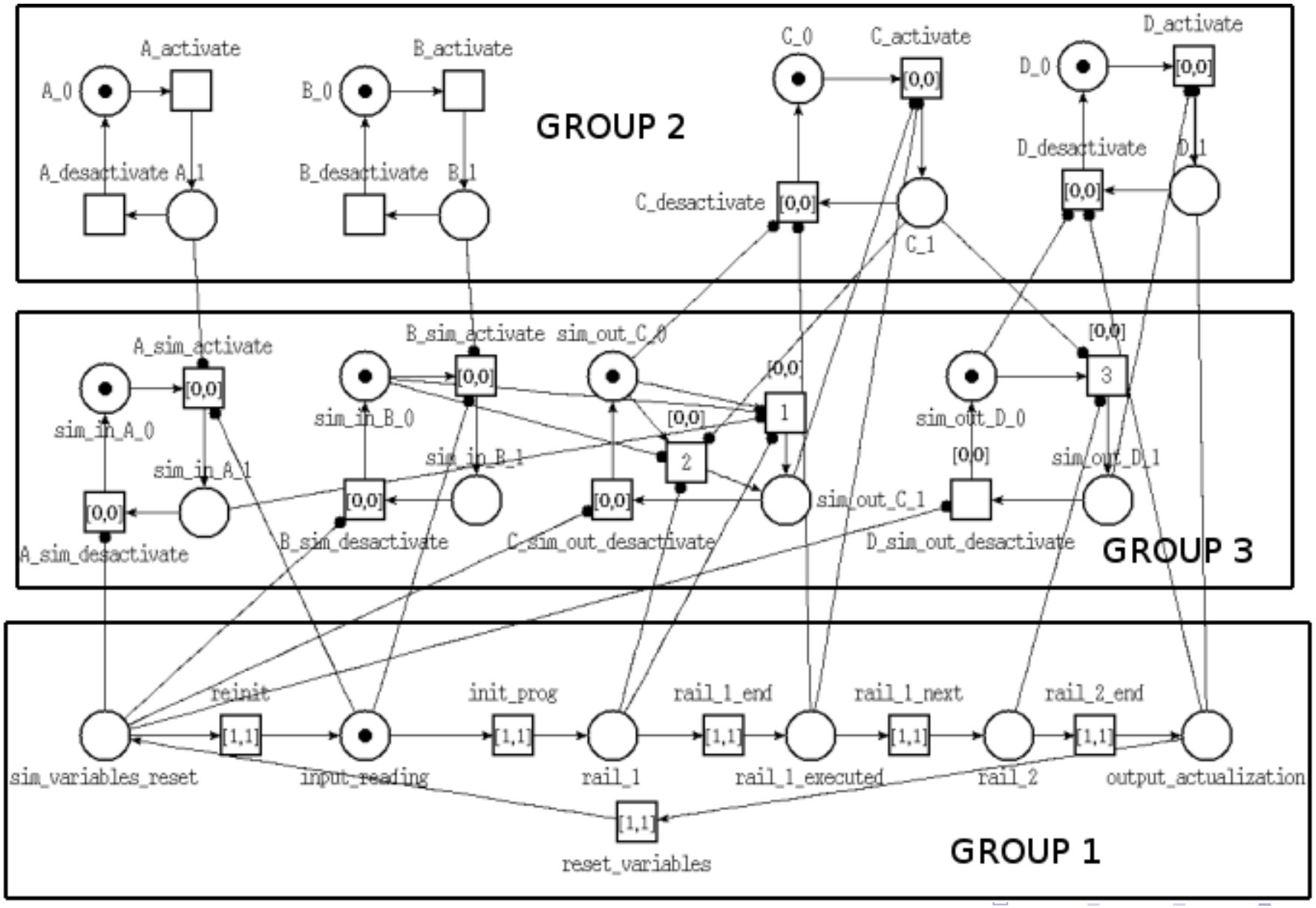


- **Main concepts:** Place, Transition, Arc, Token, temporal constraint.

Translation of Ladder Diagram into Time Petri Net



Translation of Ladder Diagram into Time Petri Net



Translation of Ladder Diagram into Time Petri Net

Some explanations

1. **Three main parts:** Group 1, Group 2 and Group 3
2. **Group 2**: Variables
   - two places for one variable: set (*_1) and unset (*_0) places.
   - *memory* and *output* variables may be updated from simulation variables.
3. **Group 1**: Control one computation of all rungs, one at a time
   - initialize simulation variables from Group 2 Variables
   - compute the first rung
   - update Group 2 variables according to simulation variables
   - the same two steps for all other rungs
   - sequencing is done through [1,1] transition
4. **Group 3**: "simulation" variables used during a rung computation
   - the simulation variables
   - the way to update them from the Group 2 variables
   - the computation of their new values according to rungs :
     - rungs are decomposed into paths
     - $C = (A \vee C) \wedge \neg B = (A \wedge \neg B) \vee (C \wedge \neg B) = \text{path1} \vee \text{path2}$
     - each path becomes one transition

Race Condition Formalisation

Definition (race condition)

An LD program is free of race condition if

$$\Box\,(\mathit{stable\_inputs} \Rightarrow \Diamond\ \mathit{stable\_outputs})$$

stable_inputs = logical *AND* between the stability condition for every input variable

stable_outputs = the same for every output and memory variable.

Definition (stable variable)

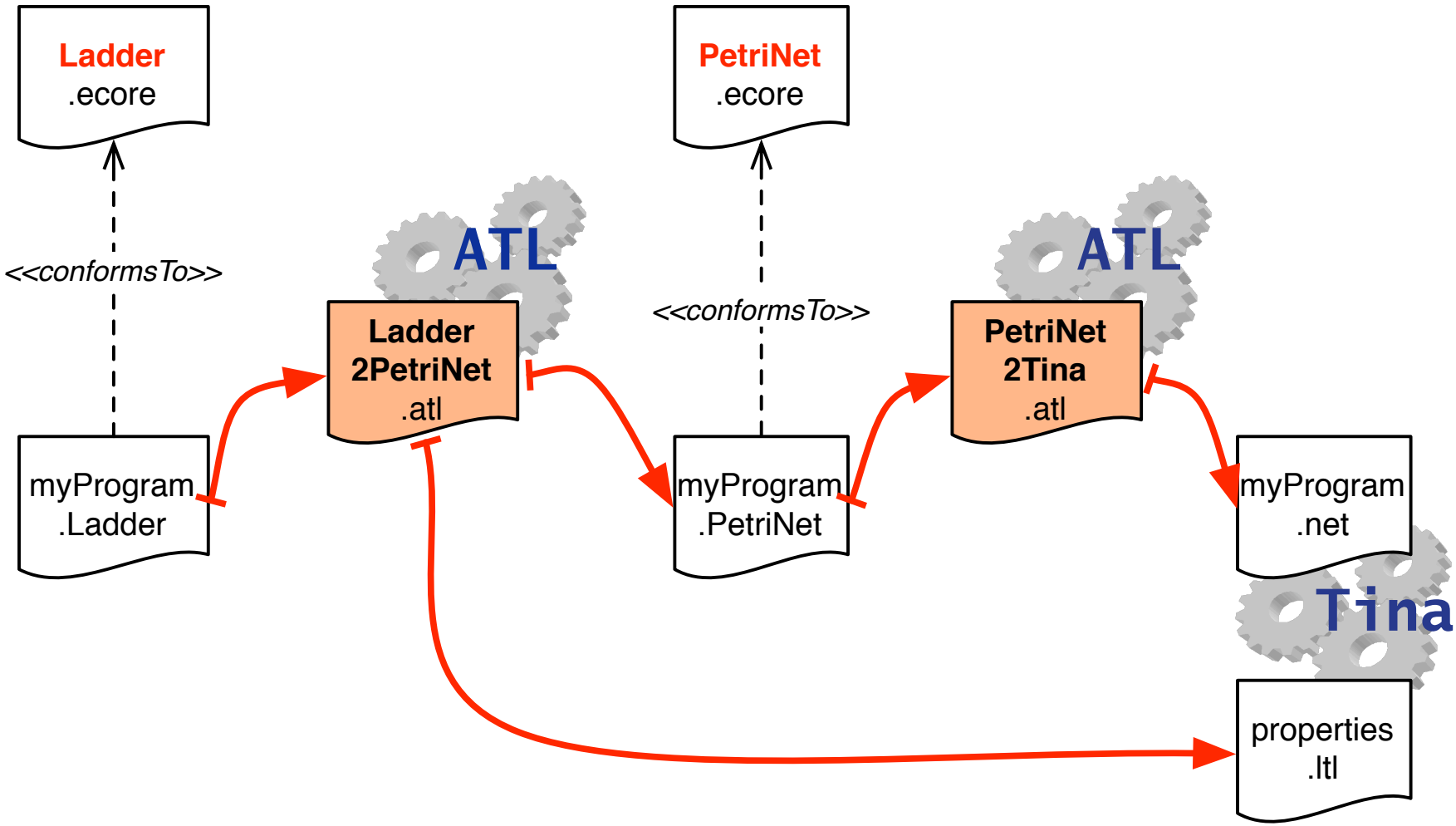
An LD variable called x is *stable* if $((\Box\ x\_0) \vee (\Box\ x\_1))$.

Example

$$\Box\,((((\Box\ A\_0) \vee (\Box\ A\_1)) \wedge ((\Box\ B\_0) \vee (\Box\ B\_1))) \Rightarrow \Diamond\,(((\Box\ C\_0) \vee (\Box\ C\_1)) \wedge ((\Box\ D\_0) \vee (\Box\ D\_1))))$$

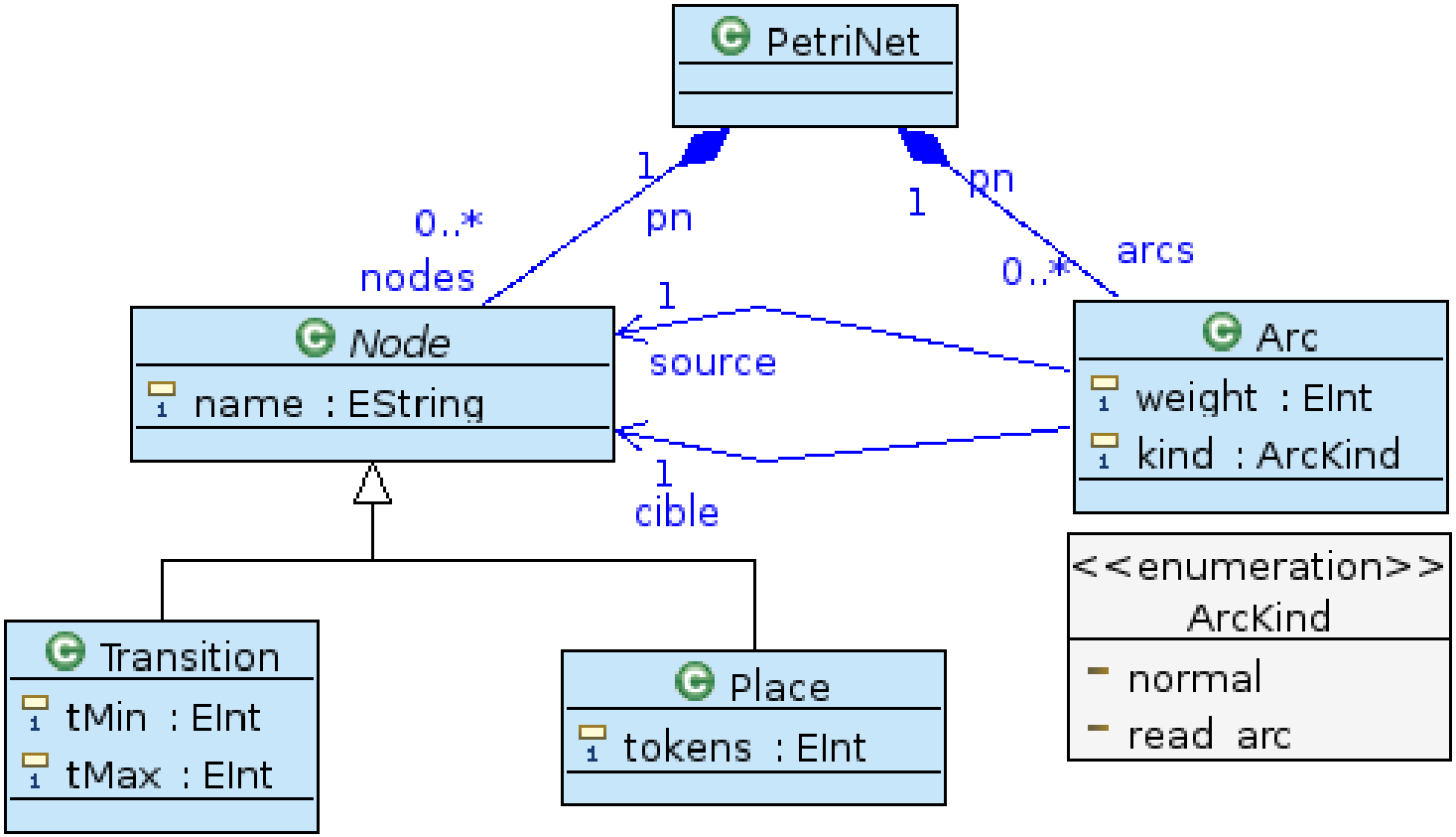
Plan

Approach : Metamodels and Transformations



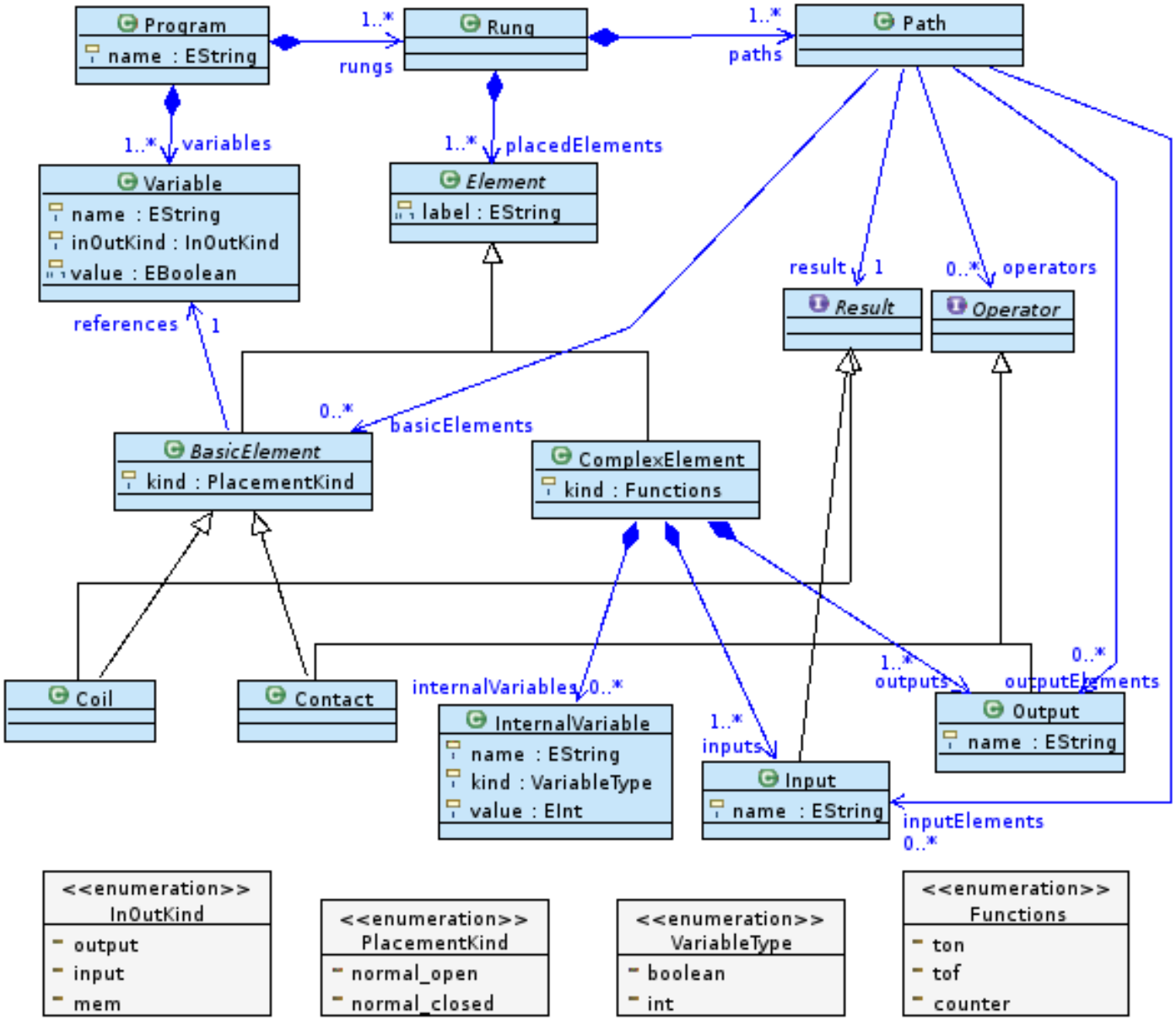
Time Petri Net Metamodel

- A **metamodel**



- and **OCL constraints**.

Ladder Metamodel



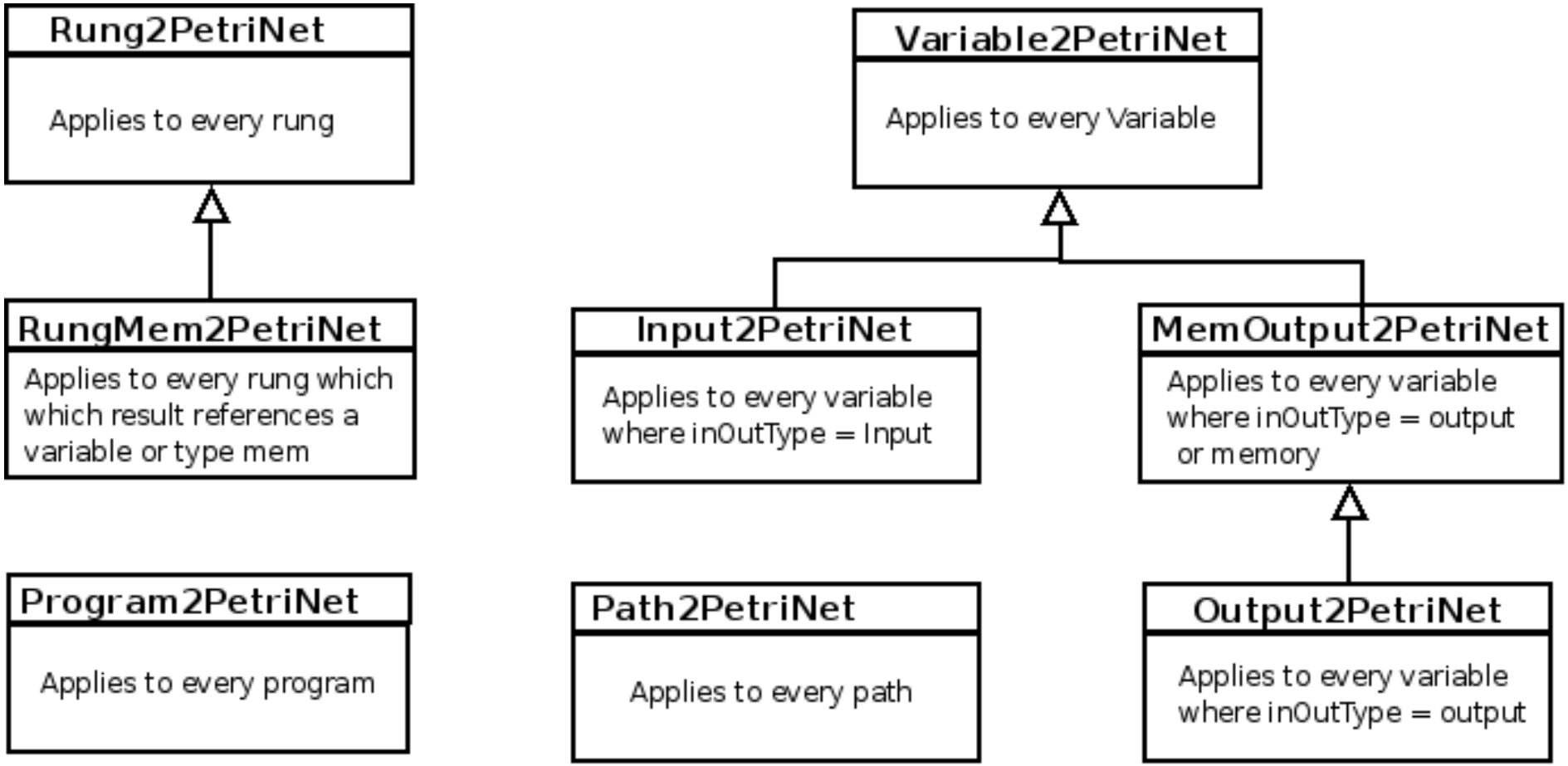
Ladder2PetriNet

Translational Semantics

- Transformation **Model to Model**
- Use of a hybrid transformation language : **ATL**
- Mainly relying on the **declarative style** (rules)
- **Rule inheritance** to structure and factorize code
  Used to simulate a role of one source metamodel element (e.g. *Variable*)
- Use of the *resolveTemp* ATL operator (because of inheritance and to identify the right element generated by another rule)

Ladder2PetriNet

Rules architecture



Ladder2LTL

Properties generation

- Achieved through an ATL query (Model to Text)
- Requires to know the names used in the translational semantics

Tina Projector

Transformation from PetriNet model to Tina concrete syntax

1. Using an ATL query (Model2Text)
   - using helpers on metamodel elements
2. Using TCS (Textual Concrete Syntax) [Jouault *et al*. 2006]
   - Main characteristics:
     - defines textual concrete syntax for a DSL/metamodel
     - based on ANTLR
     - provides both an injector and a projector
   - Results:
     - wide gap between Tina syntax and PetriNet metamodel
     - able to write a TCS model with one warning
     - the injector works :-)
     - the projector does not work :(

Plan

Conclusion

Contribution:

- Ladder metamodel,
- Ladder translational semantics through Petri nets,
- Race condition formalisation.

Conclusion:

- On the verification side:
  - Race conditions are indeed detected
  - Initial coding has been changed to avoid useless behaviours
  - $\Longrightarrow$ great improvement
- On the MDE side:
  - Should be able to handle large models
  - Easy to implement the evolutions of the Ladder2PetriNet mapping

Future Works

- Use FIACRE as an intermediate language to verification tools (Topcased),
- Handle other Ladder concepts like function blocks,
- Ladder injector to automate the building of the Ladder model,
- Find the changing variable in the counter-example provided by the model-checker.

Thank you
for your attention...

Questions?

Metamodels and transformations are available at

`http://combemale.svn.enseeiht.fr/proto/fr.irit.acadie.ladder2tina/`