COMPARE 2012

30th June 2012

Towards Unified Mechanisms for Defining and Sharing Formal Notations for Concurrency

Étienne André[1], Benoît Barbot[2], Clément Démoulins[2], Lom Messan Hillah[3,4], Francis Hulin-Hubard[2], Fabrice Kordon[3], and Laure Petrucci[1]

[1] Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, France
[2] LSV, CNRS & ENS de Cachan, France
[3] LIP6, CNRS UMR 7606, UPMC, France
[4] Université Paris Ouest Nanterre La Défense, France

Motivation

Main goal

Scalable reference platform for automated reasoning

- Wide range of tools
- Heterogeneous formalisms
- Chaining of processes of verification in order to allow certification of models
- Tool comparison and evaluation with homogeneous criteria

Motivation

Main goal

Scalable reference platform for automated reasoning

- Wide range of tools
- Heterogeneous formalisms
- Chaining of processes of verification in order to allow certification of models
- Tool comparison and evaluation with homogeneous criteria

Problems

- Difficulty to conciliate different formalisms and tools into one common platform
- Even harder to consider end-to-end verification in a toolchain combining different formalisms and tools

Outline

1. Related Work
2. Approach
3. Integration into CosyVerif
4. Perspectives

Outline

Related Work: Purely Syntactic Approaches

- OMDoc
  - Markup format and data model for Open Mathematical Documents
  - Ontology language for mathematical knowledge
  - No associated platform, but interfaces for existing tools
- MoWGLI
  - Management and publishing of mathematical documents (MathML, OpenMath, OMDoc)
  - XML-based technologies (XSLT, RDF, etc.)
  - Not maintained anymore?

Related Work: Syntax and Toolkits (1/2)

- Prosper: Proof and Specification Assisted Design Environments
  - Extensible, proof tool architecture for formal design and verification
  - Tools with graphical (textual) interface
  - Promising but outdated
- CASL: Common Algebraic Specification Language
  - Functional requirements and modular design language for software systems
  - HetCASL platform: Heterogeneous Tool Set
  - Logic- and theorem prover-oriented (Isabelle, Maude, etc.)
- Diabelli [Urbas and Jamnik, 2012]
  - Heterogeneous reasoning (theorem proving with both diagrammatic and sentential formulae, and proof steps)
  - Standalone tool combining Isabelle and Speedith
  - Graphical interface, but textual models
  - Not that flexible (requires translations), not in the cloud

Related Work: Syntax and Toolkits (2/2)

- LTSmin: Meta toolkit [Blom et al., 2010]
  - Different input language modules (mCRL2, Promela, etc.)
  - LTS-based semantic exchange of state space between different tools (Partitioned Next-State function)
  - Allows the end user to apply different verification algorithms than their native tool
- Rich-model Toolkit
  - Standardization of formal languages: common formats for systems, formulae, proofs and counterexamples
  - SAT and SMT oriented, built-in algorithms (?)
  - Recent initiative

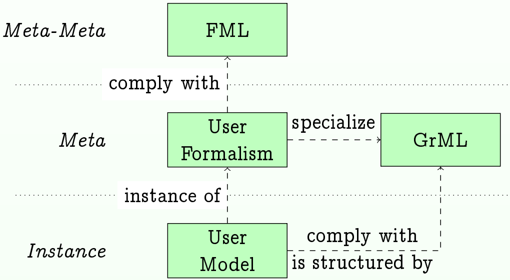
Outline

Challenges

- Flexible and extensible mechanism for describing formalisms
  - Should allow well-formatted files
  - Should be based on technologies supported by tools and libraries for file manipulation
- Composition and hierarchy of formalisms
  - Formalisms are not independent from each other: need for factoring, and maintaining precise relations between formalisms
  - Formalisms should be composed and reused
  - Formalisms should be easily extensible

Two-layered Formalism Approach

- Separating concerns
  - Formalisms: FML
  - Models descriptions: GrML



FML: Formalism Markup Language

- Defines the concepts of a graph-based formalism
  - Nodes and arcs
  - Complex attributes can be attached
- Based on XML
  - Favor reusability
  - Numerous existing tools and libraries
- Allows formalism inclusion
  - A formalism can include one or several other formalism definition(s)
  - Favor reusability
  - Favor inheritance
  - Favor easy definition of new formalisms using composition of existing ones

Example: FML Description for Directed Graphs

```
<?xml version="1.0" encoding="UTF-8"?>
<formalism name="Graph" xmlns="http://cosyverif.org/ns/formalism">
    <nodeType name="vertex"/>
    <arcType name="transition"/>
    <leafAttribute name="name" refType="vertex"/>
</formalism>
```

- Each vertex is a node
- Each transition is an arc
- Each vertex has a name

Example: FML Description for Directed Graphs

```
<?xml version="1.0" encoding="UTF-8"?>
<formalism name="Graph" xmlns="http://cosyverif.org/ns/formalism">
    <nodeType name="vertex"/>
    <arcType name="transition"/>
    <leafAttribute name="name" refType="vertex"/>
</formalism>
```

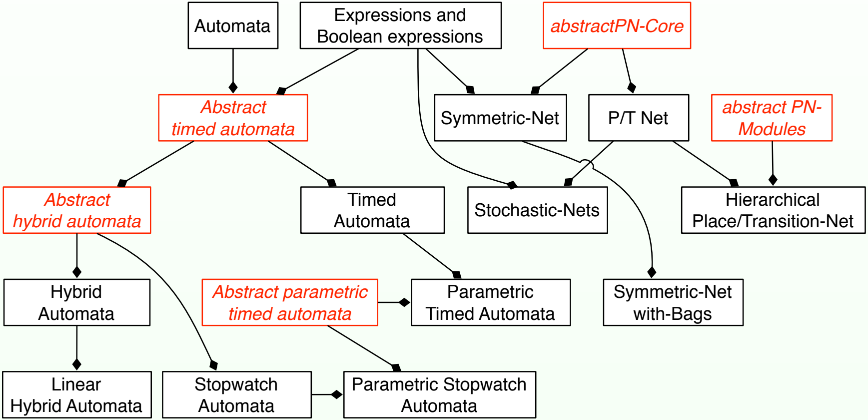
- Each vertex is a node
- Each transition is an arc
- Each vertex has a name

One could add:

- Initial and final vertexes
- Transition labels
- And so on

An Example of Hierarchy of Formalisms

- Formalisms for classes of automata and Petri nets
- Available on the Web



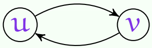
GrML: GRaph Markup Language

- A GrML file describes a model
- References a FML formalism
  - Instance of a FML formalism
  - Automated conformance check for any FML formalism and any GrML model
- Analogies
  - With UML: FML defines the *superstructure*, and GrML the *infrastructure*
  - With DSL: FML is a meta meta model, and GrML a meta model

Example of GrML Model

```xml
<?xml version="1.0" encoding="UTF-8"?>
<model formalismUrl="http://formalisms.cosyverif.org/graph.fml"
xmlns="http://cosyverif.org/ns/model">
<node id="1" nodeType="vertex">
<attribute name="name">u</attribute>
</node>
<node id="2" nodeType="vertex">
<attribute name="name">v</attribute>
</node>
<arc id="101" arcType="transition" source="1" target="2"/>
<arc id="102" arcType="transition" source="2" target="1"/>
</model>
```

- Syntactically conforms to the FML model previously given
- Corresponds to the following graph



Outline

CosyVerif: Architecture

- A flexible server: Alligator
  - Contains the integrated tools
- A flexible client: Coloane
  - Contains a graphical interface for the models
  - Available as an Eclipse plugin or an RCP application
  - Can be easily extended (plugin architecture)
- Distributed architecture (in the cloud)
  - A client automatically (or manually) connects to an available server through a Web service
  - Advantage: no charge on the user computer

CosyVerif: Features

- Generic and open platform
  - Depends neither on the formalisms nor on the tools and their algorithms
- Very flexible
  - Easy to add a new formalism
  - Easy to integrate a new tool: one parser and one printer (one day of work with no specific knowledge)
  - Other clients can be implemented

CosyVerif: Community

- Widely used
  - Frequent meetings (steering committee, one-day workshops, integration parties, PN model checking competition, etc.)
  - Based on CPN-AMI (since 1987): more than 260 sites licenses in 50 countries
  - Benchmarks library in GrML
- 100% open source
  - Server, client and tools are in GNU GPL

CosyVerif: Community

- Widely used
  - Frequent meetings (steering committee, one-day workshops, integration parties, PN model checking competition, etc.)
  - Based on CPN-AMI (since 1987): more than 260 sites licenses in 50 countries
  - Benchmarks library in GrML
- 100% open source
  - Server, client and tools are in GNU GPL
- Try it!

www.cosyverif.org

CosyVerif: Currently Integrated Tools

- Cosmos [Ballarini et al., 2011], a statistical model checker for Petri net with general distribution
- Crocodile [Colange et al., 2011], a model checker for Symmetric Nets with bags
- Imitator [André et al., 2012], a tool for synthesizing timing parameters for Timed Automata with stopwatches
- PNXDD [Kordon et al., 2012], a model checker for Place/Transition Petri nets based on Hierarchically Structured Decision Diagrams

CosyVerif: Currently Integrated Tools

- Cosmos [Ballarini et al., 2011], a statistical model checker for Petri net with general distribution
- Crocodile [Colange et al., 2011], a model checker for Symmetric Nets with bags
- Imitator [André et al., 2012], a tool for synthesizing timing parameters for Timed Automata with stopwatches
- PNXDD [Kordon et al., 2012], a model checker for Place/Transition Petri nets based on Hierarchically Structured Decision Diagrams

…And more to come!

Outline

Towards Models for Composition

- Horizontal composition
  - Several models can be synchronized together (usually on-the-fly)
  - Example: Timed automata
- Vertical composition: heterogeneous hierarchy
  - Subparts of a model can refer to another model
  - Example: what if a Petri net place is refined by a timed automaton?
- Need for models for composition

Towards Semantic Models

- Semantic bridges between formalisms
  - Allow automated model translation
  - Allow tool comparison even on different formalisms
  - Allow tool orchestration
    - Sequence of calls using different formalisms
    - Parallel with LTSmin, but more complicated than LTSs
- Handling inconsistencies
  - Not every model in a formalism can be translated to any other formalism
  - Automated detection of possible incompatibilities
  - Or loss controlled semantic mapping

References I

André, É., Fribourg, L., Kühne, U., and Soulat, R. (2012).
IMITATOR 2.5: A tool for analyzing robustness in scheduling problems.
In *FM'12*, LNCS. Springer.
To appear.

Ballarini, P., Djafri, H., Duflot, M., Haddad, S., and Pekergin, N. (2011).
HASL: An expressive language for statistical verification of stochastic models.
In *VALUETOOLS'11*.
To appear.

Blom, S., van de Pol, J., and Weber, M. (2010).
LTSmin: Distributed and symbolic reachability.
In *CAV'10*, volume 6174 of *LNCS*, pages 354–359. Springer.

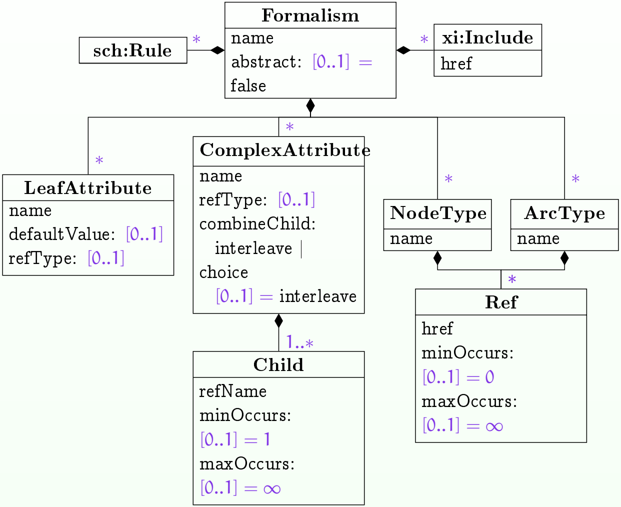
Colange, M., Baarir, S., Kordon, F., and Thierry-Mieg, Y. (2011).
Crocodile: a symbolic/symbolic tool for the analysis of symmetric nets with bag.
In *ICATPN'11*, volume 6709 of *LNCS*, pages 338–347. Springer.

Kordon, F., Linard, A., Buchs, D., Colange, M., Evangelista, S., Lampka, K., Lohmann, N., Paviot-Adet, E., Thierry-Mieg, Y., and Wimmel, H. (2012).
Report on the model checking contest at Petri Nets 2011.
ToPNoC, V:121–140.

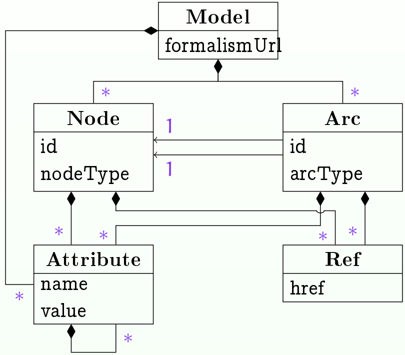
References II

Urbas, M. and Jamnik, M. (2012).
Diabelli: A heterogeneous reasoning framework.
In *IJCAR'12*.
To appear.

FML Concepts



GrML Concepts



Abstract vs. Concrete Formalisms

- Abstract formalism
  - Root (or intermediate) formalism for the hierarchy
  - Should not have GrML instance
- Concrete formalism
  - Inherits one or several abstract formalism(s)
  - May add constraints to the abstract formalism
- Good design practice
- Parallel with object-oriented software design
  - Abstract classes factor common features
  - Concrete classes refine them, and can be instantiated

Technologies

- Inclusion of formalisms is performed using XInclude
- Constraints are specified using Schematron
- Model validation and conformity is performed using XSLT