Evolution of the Formalism Language in CosyVerif

Benoît Barbot

LSV, ENS Cachan & CNRS & INRIA

Séminaire MeFoSyLoMa, Vendredi 20 Décembre 2013

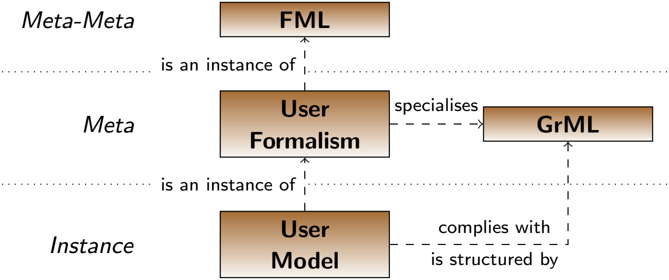
CosyVerif

Contributors

- Etienne André (LIPN)
- Maximilien Colange (LIP6)
- Clément Démoulins (LIP6-LSV)
- Serge Haddad (LSV)
- Lom Messan Hillah (LIP6)
- Fabrice Kordon (LIP6)
- Alban Linard (LSV)
- Laure Petrucci (LIPN)

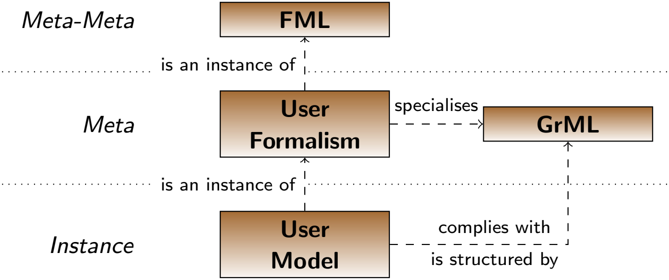
CosyVerif formalism/model architecture

- FML describes the structure of the model.
- GrML describes a model.
- Both are XML files.



CosyVerif formalism/model architecture

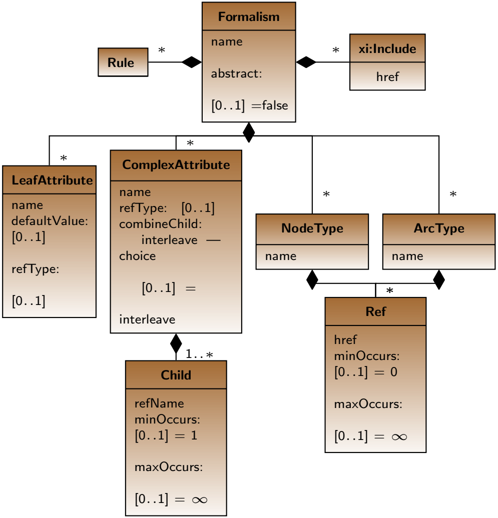
- FML describes the structure of the model.
- GrML describes a model.
- Both are XML files.



Conformance

The tool GrmlCheck allows to test the conformance of a model to its formalism

Recall on FMLv1



Formalism
name
abstract:
[0..1] =false

Rule
*

xi:Include
href
*

LeafAttribute
name
defaultValue:
[0..1]
refType:
[0..1]
*

ComplexAttribute
name
refType: [0..1]
combineChild:
interleave — choice
[0..1] = interleave
*

NodeType
name
*

ArcType
name
*

Ref
href
minOccurs:
[0..1] = 0
maxOccurs:
[0..1] = ∞
*

Child
refName
minOccurs:
[0..1] = 1
maxOccurs:
[0..1] = ∞
1..*

```xml
<?xml version="1.0" encoding="UTF-8"?>

<formalism name="Automaton"
       xmlns="http://cosyverif.org/ns/formalism">

    <leafAttribute name="name" defaultValue="" refType="
        Automaton"/>
    <leafAttribute name="initialState" />
    <leafAttribute name="finalState" />

    <complexAttribute name="type" refType="state">
        <child refName="initialState" minOccurs="0"
                maxOccurs="1"/>
        <child refName="finalState" minOccurs="0" maxOccurs
                ="1"/>
    </complexAttribute>

    <leafAttribute name="name" refType="state"/>
    <leafAttribute name="label" refType="transition"/>

    <nodeType name="state"/>
    <arcType name="transition"/>

    <!-- state names shoud all be unique -->
</formalism>
```

Weakness of FMLv1

Major limitations

- Inheritance by replacement
  ⇒ diamonds problem
- Data not properly typed. Requires additional rules which are hard to specify.
- Hierarchy of formalisms only used for modular definition.

Weakness of FMLv1

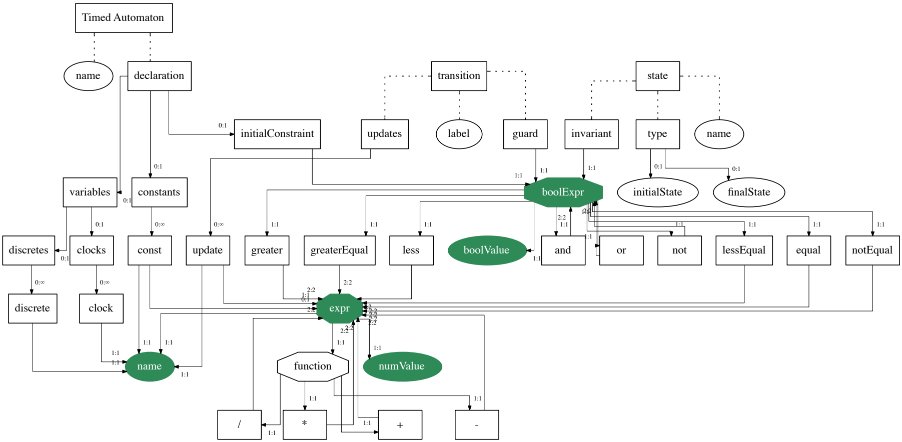
Major limitations

- Inheritance by replacement
  $\Rightarrow$ diamonds problem
- Data not properly typed. Requires additional rules which are hard to specify.
- Hierarchy of formalisms only used for modular definition.

Additional limitations

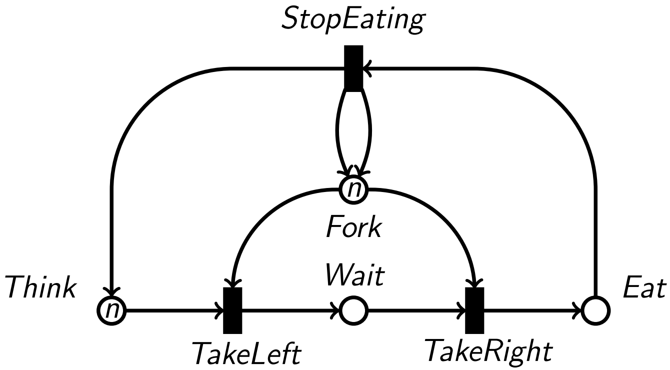
- XML not convenient for specifying formalism
- No syntactic information (leads to additionnal formalisms in Coloane)
- Hierarchical formalisms too difficult to use

Bad handling of data type



- No specification of terminal type
- Mix of graphical and textual data

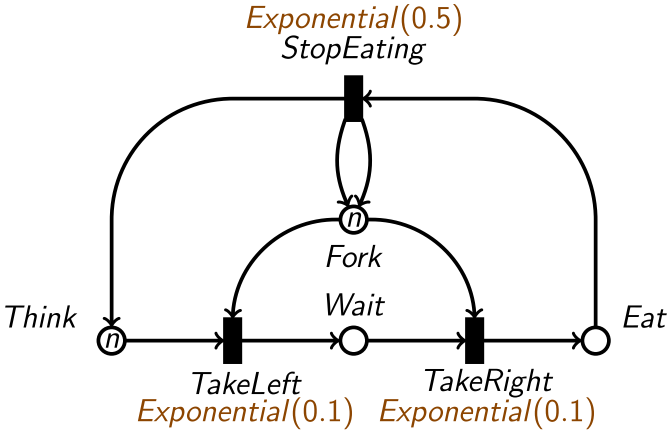
Ordinary Petri Net (PT)



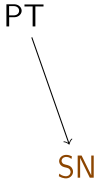
PT

Diamond problem

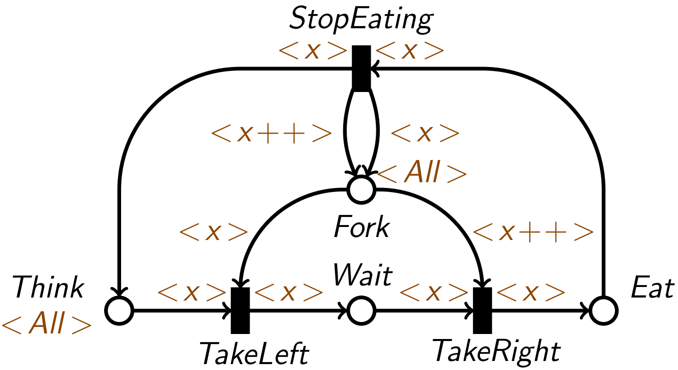
Stochastic Net (SN)

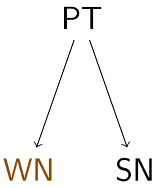


Transitions have additional attributes



Well Formed Net (WN)

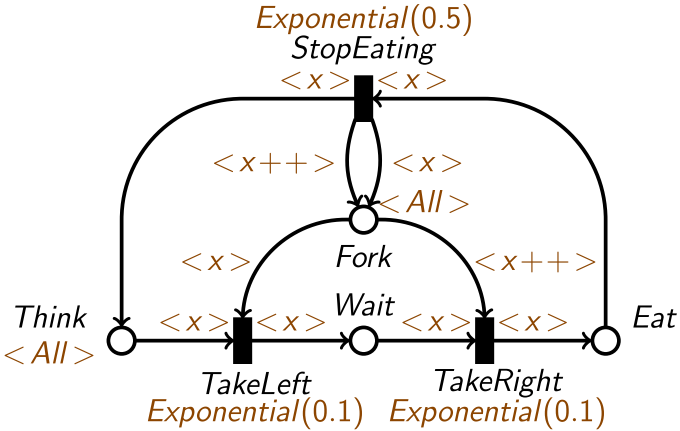


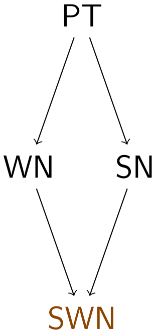


Valuation and Marking have a different type

Diamond problem

Stochastic Well Formed Net (SWN)





Requires to redefine Place and Transition

FMLv2 - Main Ideas

What we keep from FMLv1

- Graph-based formalisms
- Only syntactic

FMLv2 - Main Ideas

What we keep from FMLv1

- Graph-based formalisms
- Only syntactic

Separation of structure and datatype

- Structure: a graph with attributes
- Datatype: an algebraic expression

FMLv2 - Main Ideas

What we keep from FMLv1

- Graph-based formalisms
- Only syntactic

Separation of structure and datatype

- Structure: a graph with attributes
- Datatype: an algebraic expression
- Inheritance of the structure
- Modular definition of datatype
- Late binding of datatype and structure

FMLv2 - Main Ideas

What we keep from FMLv1

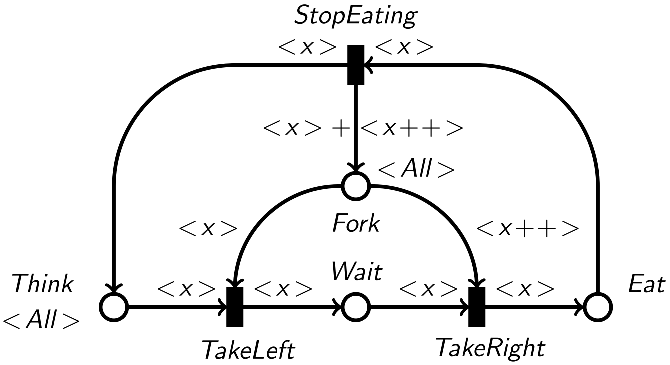
- Graph-based formalisms
- Only syntactic

Separation of structure and datatype

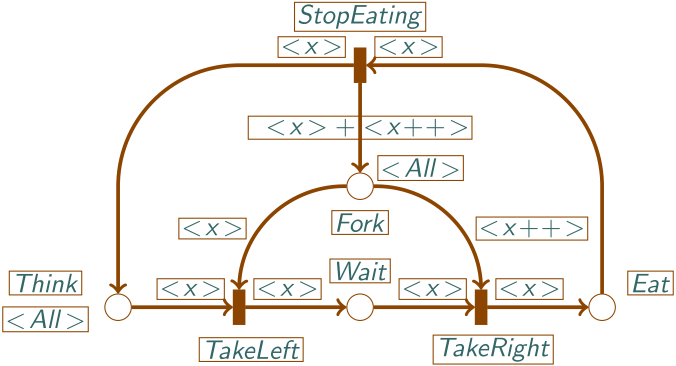
- Structure: a graph with attributes
- Datatype: an algebraic expression
- Inheritance of the structure
- Modular definition of datatype
- Late binding of datatype and structure

Use of a script language

- Avoids XML for formalism definition
- Uses an API in LUA for defining formalism
- Allows to have upcasted views of a model


StopEating
<x>
<x>
<x> + <x++>
<All>
Fork
<x>
<x++>
Think
<All>
<x>
<x>
Wait
<x>
<x>
Eat
TakeLeft
TakeRight

Structure/Datatype - Separation



Datatype

Based on hierarchy of algebraic expressions

- Constructors
- Operators

Datatype

Based on hierarchy of algebraic expressions

- Constructors
- Operators

```
local constant = new (constructor) {
  value = new (parameter) { minimum = 1, maximum = 1 },
  text_syntax = "${value}",
}
local param = ...

local plus = new (constructor) {
  operands = new (parameter) {minimum=1, maximum= oo},
  text_syntax =
    " ${operands} + ... + ${operands} "
}
local times = ...

local arithmetic = new (datatype) {
  terminal_t = abstract_type,
  param_t = abstract_type,
  constructors = {
    constant { type_of(value) = terminal_t },
    param { type_of(value) = param_t },
    plus { type_of(operands) = self },
    times { type_of(operands) = self }
  }
}

local natural = new (arithmetic) {
  terminal_t = "xsd:NonNegativeInteger"
}
```

Structure - Definition

Hypergraph

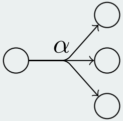
- General structure of hypergraphs
- Specialization of edges and vertices

Structure - Definition

Hypergraph

- General structure of hypergraphs
- Specialization of edges and vertices

Example:MDP

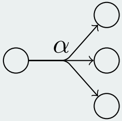


Structure - Definition

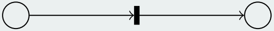
Hypergraph

- General structure of hypergraphs
- Specialization of edges and vertices

Example:MDP



Example:Petri Net



Structure - Definition

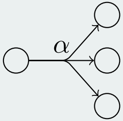
Hypergraph

- General structure of hypergraphs
- Specialization of edges and vertices

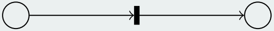
Inheritance

- Simple inheritance
- Multiple inheritance
- Renaming
- Add attribute

Example:MDP



Example:Petri Net



Structure - Example

```
local pn = require "Hierarchy".pn
```

Structure - Example

```
local pn = require "Hierarchy".pn
```

```
local marked_net = new (pn) {
  marking_type = abstract_type,
  place_type.marking =
    instance_of (marking_type) {
      container = labels
    }
}
```

Structure - Example

```
local pn = require "Hierarchy".pn
```

```
local marked_net = new (pn) {
  marking_type = abstract_type,
  place_type.marking =
    instance_of (marking_type) {
      container = labels
    }
}
```

```
local weighted_net = new (pn) {
  valuation_type = abstract_type,
  pre_arc_type = with {
    valuation = instance_of (
         valuation_type) {
      container = labels
    }
  },
  post_arc_type = with {
    valuation = instance_of (
         valuation_type) {
      container = labels
    }
  },
}
```

Structure - Example

```
local pn = require "Hierarchy".pn
```

```
local weighted_net = new (pn) {
  valuation_type = abstract_type,
  pre_arc_type = with {
    valuation = instance_of (
          valuation_type) {
      container = labels
    }
  },
  post_arc_type = with {
    valuation = instance_of (
          valuation_type) {
      container = labels
    }
  },
}
```

```
local marked_net = new (pn) {
  marking_type = abstract_type,
  place_type.marking =
    instance_of (marking_type) {
      container = labels
    }
}
```

```
local pt = new (weighted_net, marked_net) {
  token_type = abstract_type,
  marking_type = token_type,
  valuation_type = token_type,
}
```

Binding

- Bindings of all abstract types
- Define default value for each bindings

Binding

- Bindings of all abstract types
- Define default value for each bindings

```
local PT = new (pt,parameters) {
  token_type = natural,
  natural.param_t = parameters_t,
  place_type.marking.default = " 0 " ,
  arc_type.valuation.default = " 1 "
}
```

Evaluation

Structure

- Easy to upcast
- Easy to downcast with default value
- Easy to compose

⇒ handled by FMLv2

Evaluation

Structure

- Easy to upcast
- Easy to downcast with default value
- Easy to compose

⇒ handled by FMLv2

Datatype

- Upcast requires semantic
- Dowcast requires semantic
- Composition can make no sense

⇒ Requires external tools

Examples of utilization

- Allows to add parameters to any formalism
- Tools can add results to models
- Allows to specialize formalism
- Allows to script construction of formalism (ex k-partite graph)
- Allows to upcast anything to graph, can be used for placement

Conclusion

Conclusion

- Unified syntax for user
- Reliability of library syntax for tool developers
- Adapted to a lightweight graphical interface.
- Helps tools to communicate.

What need to be done

- Finalize the FMLv2 syntax
- Design the model syntax
- Specify a complete hierarchy of Petri nets and automata
- Implement API for handling models in those formalisms