A Lightweight Formal Encoding for a Constraint Language DSML Component

Arnaud Dieumegard[1,2] Marc Pantel[1] G. Babin[1] M. Carton[1]

[1]IRIT/ENSEEIHT - 2 rue Charles Camichel, 31000 Toulouse, France
first.last@enseeiht.fr

[2]IRT Saint Exupéry - 118 route de Narbonne, 31432 Toulouse, France
first.last@irt-saintexupery.com

September 28th 2015

Introduction

- Embedded systems development and verification is complex
- Need for safe and sound development and verification techniques
- State Of the Art : Domain Specific Modeling Languages (DSMLs)
  - Adapted to the needs
  - Adapted to the developer/designer knowledge
  - Ease of manipulation/transformation
  - Ease the verification
- Evolution of DSMLs : Language Component-based DSMLs
  - Capitalisation on the developer/designer knowledge
  - Integration of already existing and known concrete syntaxes/languages
  - Require means for semantics specification
- Problem : How to verify DSMLs instances

DSMLs

- Design
  - Well known techniques
    - Metamodel for the definition of the DSML structure (concepts and relations)
    - OCL constraints for the static semantics of the DSML
- Implementation
  - Model driven engineering based approach
- Verification
  - Transformations to formal methods/tools

OCL-like Component Constraint Language

- DSML are more and more complex
  - Need for the specification/development of more complex systems
  - Improvement of the specification for simpler/safer development
- Relying on an OCL-like language as a component
- Relates to the DSML data language component
  - Integrated in the DSML
  - Allows to dynamically constrain the DSML constructs
  - A rich/known/accessible language

Problem :

- Verification of the DSML instances including the constraints
  - Translation of the DSML constructs to the formal world
  - Translation of the OCL-like constraints to the formal world
- Automatic verification of the DSML properties
  - Allows the DSML user to interact with formal methods
  - No preliminary knowledge required

  $\Rightarrow$ Improvement of the DSML efficiency/confidence and thus its use/adoption

Verification Proposal

- A common formalisation of the DSML structure and constraints
  - Relying on the Why3 platform and the WhyML language
  - Verification of DSML properties with SMT solvers for automatic verification

$\Rightarrow$ An integrated verification framework

- Translation of the DSML for verification
  - Appropriate DSML elements from data component translated to WhyML types
  - Integrated constraints translated using a pre-defined formalisation of the constraint language
  - Automatic generation of verification properties

Constraint Language Scope

- Constraint language based on a subset of the OCL
- Relying on the OCL syntax to reduce the learning curve
- Limited support of the original OCL operations/constructs
  - No tuples (easy extension)
  - No messages (UML related)
  - No *Null* or *Invalid* values
  - Only *Bag* collections (easy extension)
  - No support for collection conversion operations (easy extension)
  - No support for failure prone operations (e.g. `div`, can be introduced with pre-condition : not(b=0) for div(a,b)).
  - No *Closure* operation
  - No support for Type manipulation operations (OclAsType, OclIsTypeOf, ...) (depends on the DSML data component)

Constraint Language Formalisation using WhyML

- OCL standard data types $\rightarrow$ WhyML standard data types
- Collections $\rightarrow$ lists

- Standard operations
- Collection operations
- Iteration operation

$\rightarrow$ { pre/post ; function implementation ; lemmas }

- Lemmas verification through SMT solvers or proof assistants

Extract of the Select Iteration Operation Formalisation

```
function select (l: list oclType) (p: HO.pred oclType): list oclType =
    match l with
    | Nil -> Nil
    | Cons hd tl -> if p hd then Cons hd (select tl p)
                            else select tl p
    end

lemma select_nil: forall p: HO.pred oclType.
  select Nil p = Nil
lemma select_cons_verified: forall e: oclType, l: list oclType,
                                   p: HO.pred oclType.
  p e -> select (Cons e l) p = Cons e (select l p)
lemma select_cons_not_verified: forall e: oclType, l: list oclType,
                                       p: HO.pred oclType.
  not (p e) -> select (Cons e l) p = select l p
lemma select_mem: forall l: list oclType, b: oclType,
                          p: HO.pred oclType.
  (mem b l /\ p b) -> mem b (select l p)
```

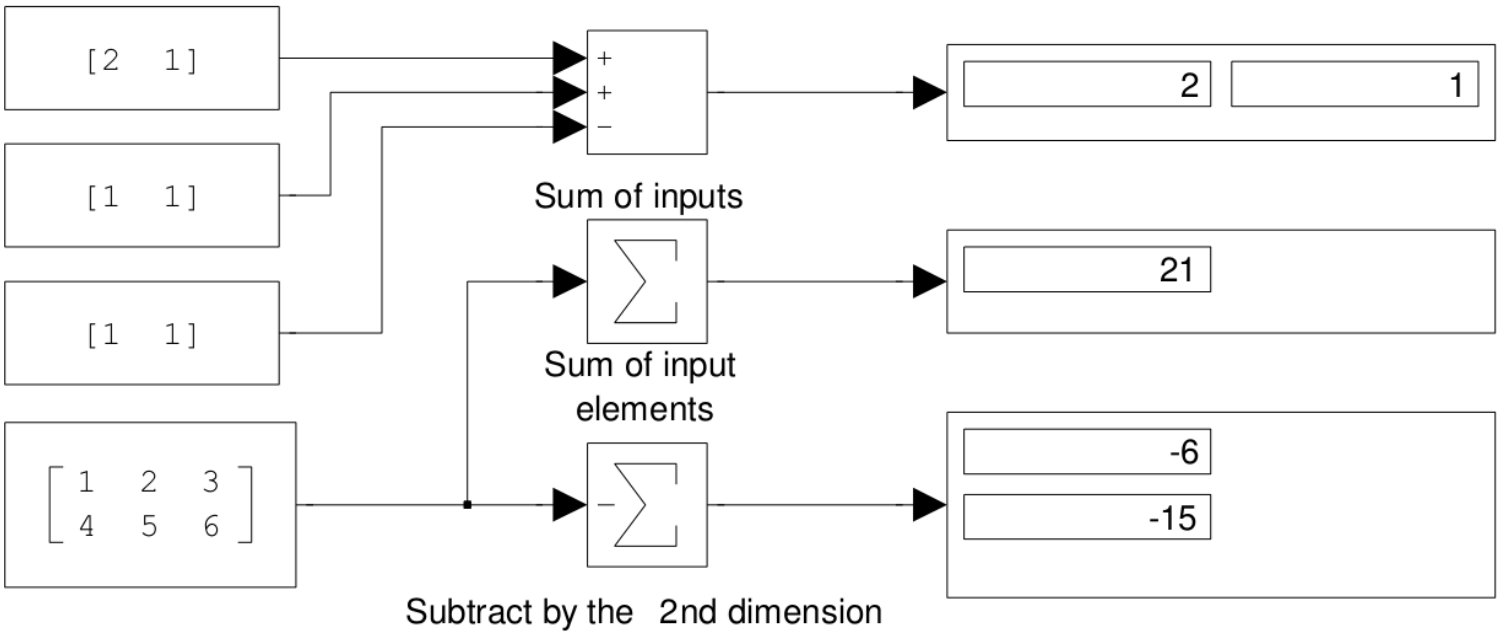
Proof obligations	**Alt-Ergo-Pro (1.0.0)**	**Coq (8.4pl3)**
lemma `select_nil`	0.03	
lemma `select_cons_verified`	0.03	
lemma `select_cons_not_verified`	0.04	
lemma `select_mem`		2.31

DSML Translation to WhyML

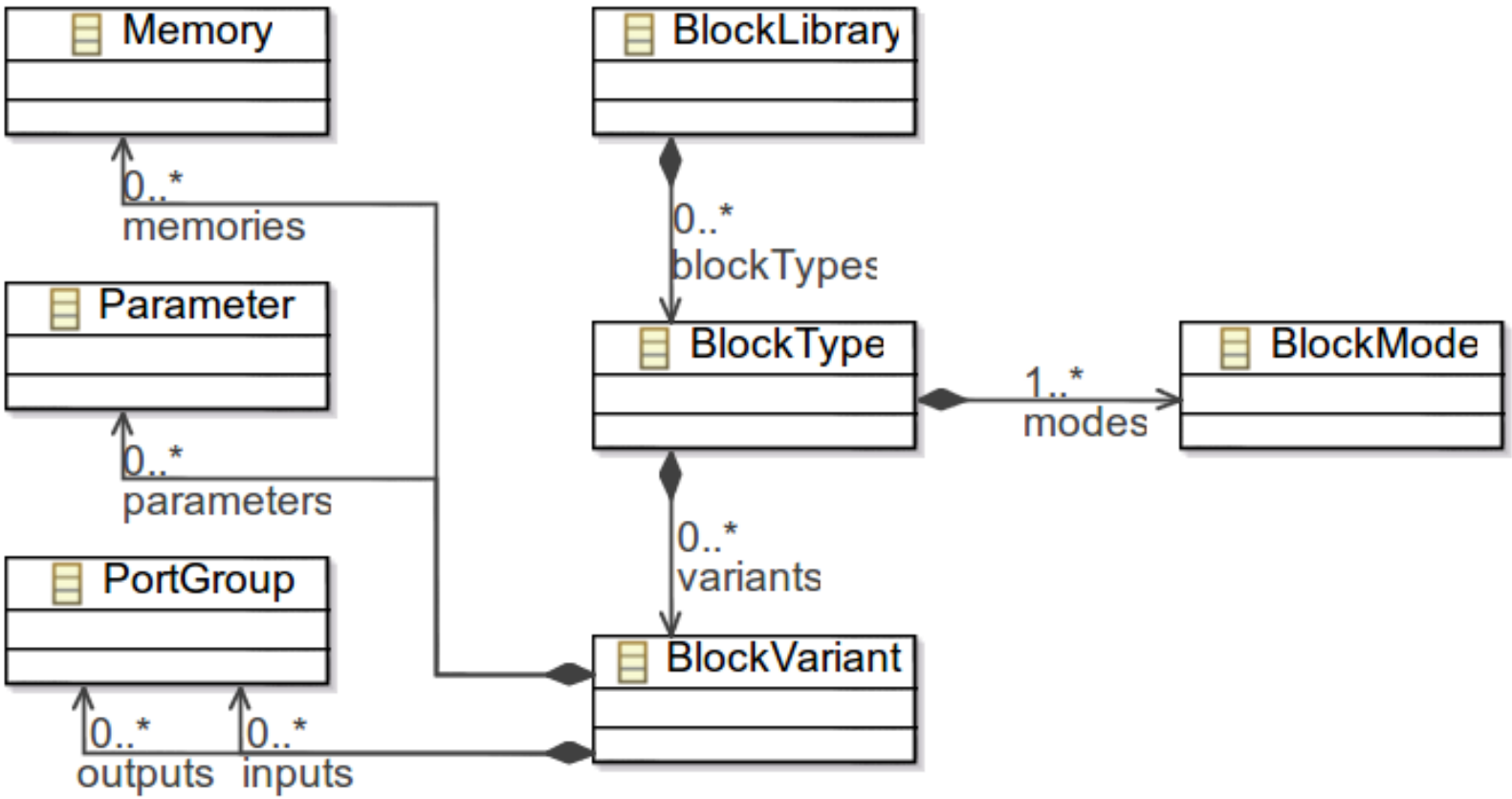
- Direct translation of the constraint relying on
  - DSML data component elements formalisation
  - Constraint component language formalisation
- Translation of the verification properties
- "Out of formalisation" constraints are necessarily false
  - Pre/post conditions and lemmas constrains the use of the operations
  - Constraints out of the formalisation scope lead to a verification failure
- Advantages
  - Relying on an already verified formalisation
  - Simpler translation
  - Simpler verification of the properties

BlockLibrary Specification DSL (OCL2012,SPLC2014)

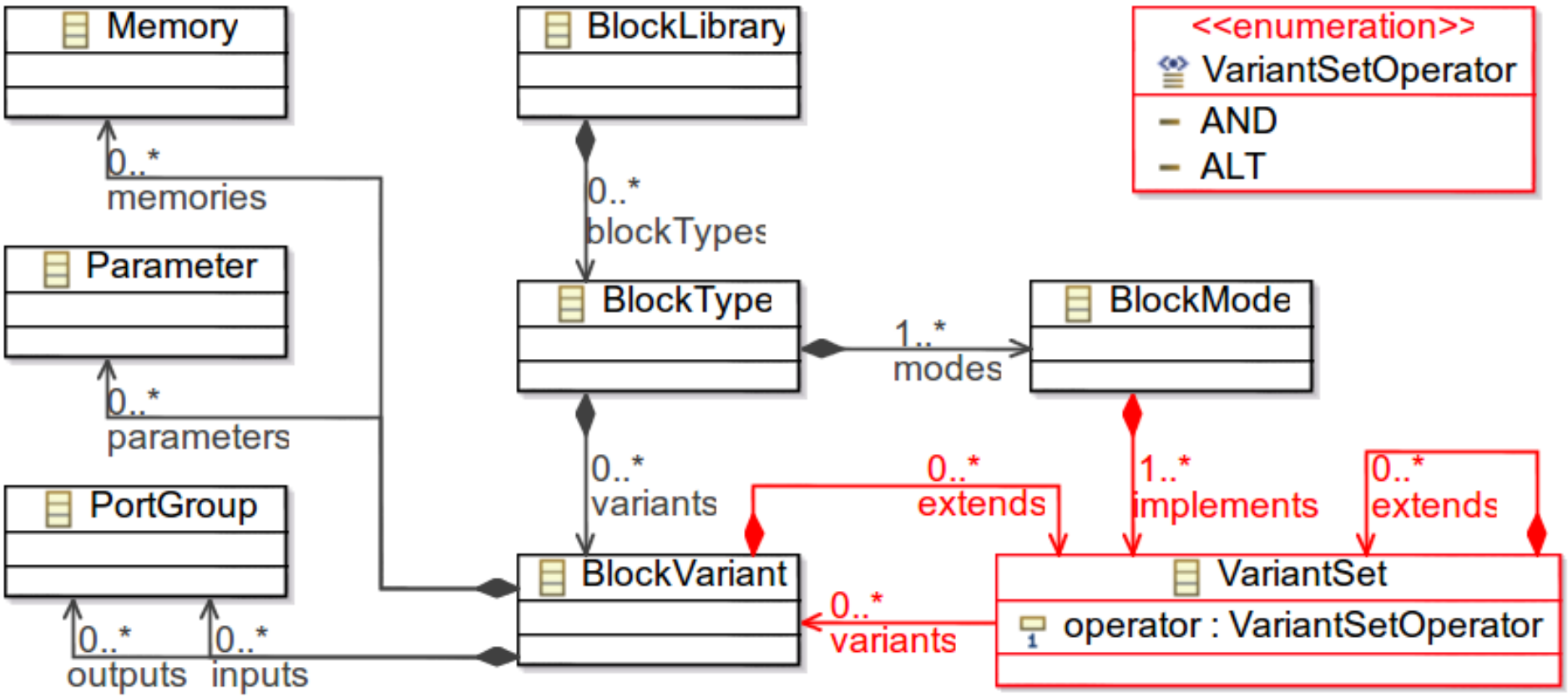
- Specificition of dataflow languages (e.g. Simulink, SCADE, . . .) block libraries
- Complex blocks (e.g. Sum operation documentation in Simulink around 20 pages)
- Requires verification of Consistency and Completeness



BlockLibrary Specification DSL – Metamodel



BlockLibrary Specification DSL – Metamodel



VariantSet : composition of block interfaces (BlockVariants) and attachement of block semantics (BlockMode)

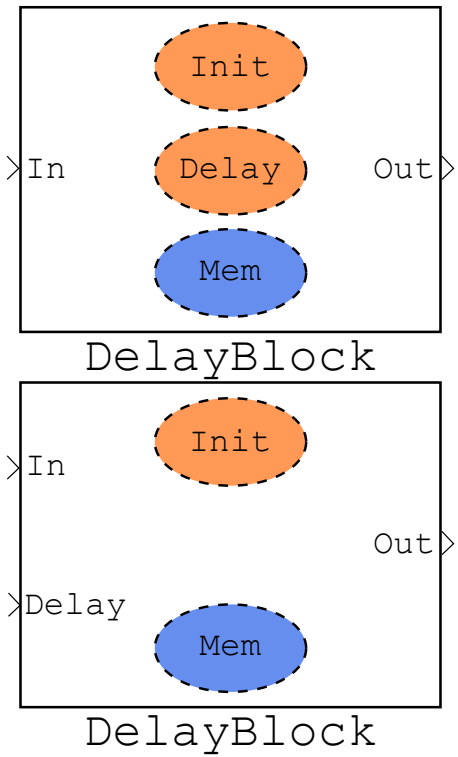
BlockLibrary Specification DSL – Block Interface

Block interfaces building inspired from product line technologies

- n-ary AND, n-ary ALT (XOR) composition operators

Interface and features constraining

```
variant InternalDelay {
  parameter Delay : TInt32
}
variant ExternalDelay {
  in data Delay : TInt32
}
variant DelayVar extends ALT (InternalDelay,
                               ExternalDelay) {
  invariant ocl { Delay.value > 0 }
}
variant ListDelay_ScalarInput extends AND (
  ResetParam, DelayVar, InputScalar
) {
  invariant ocl { Delay.value > 1 }
  invariant ocl { Init.value.size() = Delay.value }
  memory Mem {
    datatype auto ocl {In.value}
    length auto ocl {0}
  }
}
```



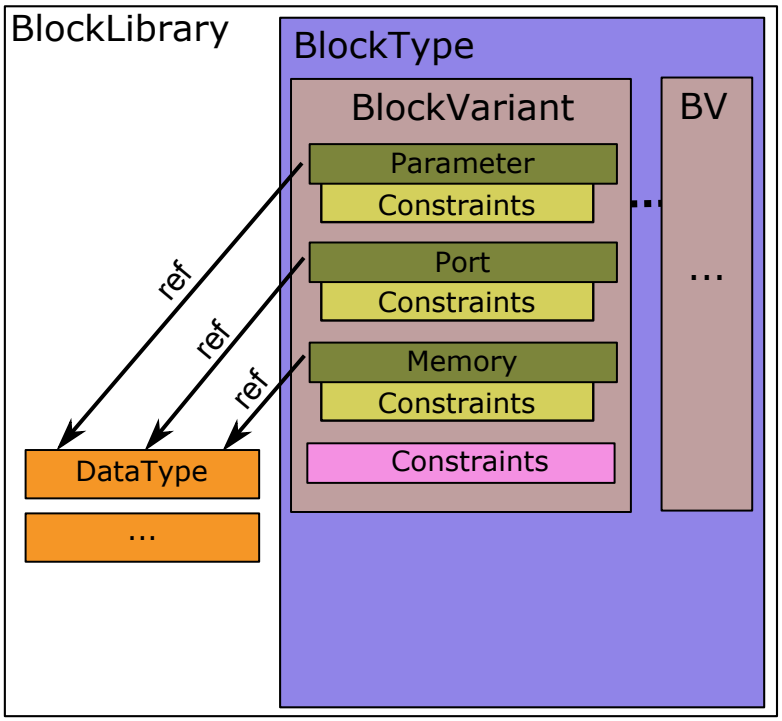
BlockLibrary Specification DSL – Block Interface

Block interfaces building inspired from product line technologies

- n-ary AND, n-ary ALT (XOR) composition operators

Interface and features constraining

```
variant InternalDelay {
  parameter Delay : TInt32
}
variant ExternalDelay {
  in data Delay : TInt32
}
variant DelayVar extends ALT (InternalDelay,
                               ExternalDelay) {
  invariant ocl { Delay.value > 0 }
}
variant ListDelay_ScalarInput extends AND (
  ResetParam, DelayVar, InputScalar
) {
  invariant ocl { Delay.value > 1 }
  invariant ocl { Init.value.size() = Delay.value }
  memory Mem {
    datatype auto ocl {In.value}
    length auto ocl {0}
  }
}
```

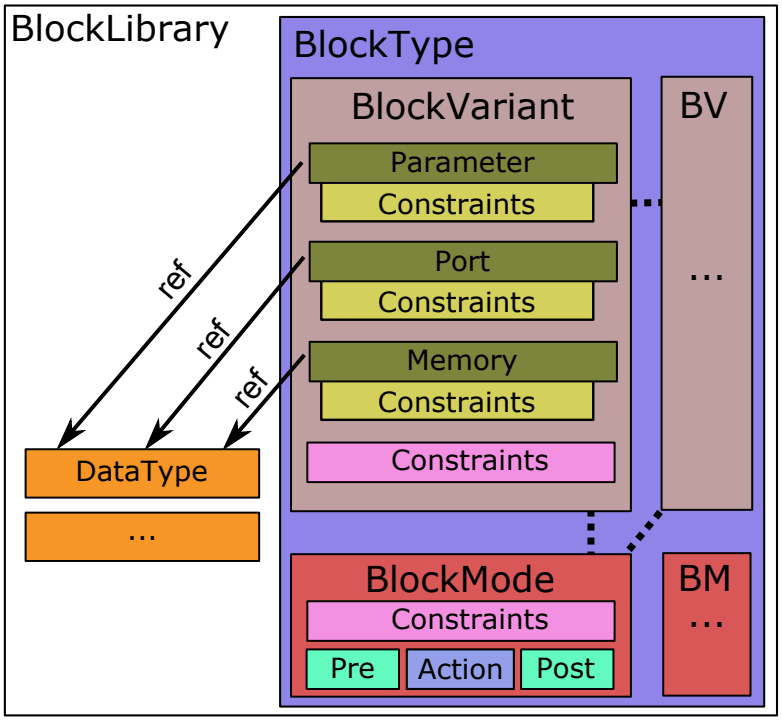


BlockLibrary Specification DSL – Block Configuration

Block semantics definition

- Gives semantics to block interface
- Attached to block interface using same composition operators

```
mode SimpleDelay implements ALT (
  SimpleDelay_ScalarInput,
  SimpleDelay_VectorInput,
  SimpleDelay_MatrixInput)
{
  definition bal = init {
    precondition ocl { ... }
    postcondition ocl { Mem == Init }
    Mem = Init;
  }
  definition bal = compute {
    precondition ocl { ... }
    postcondition ocl { Output == Mem }
    Output = Mem;
  }
  definition bal = update {
    precondition ocl { ... }
    postcondition ocl { Mem == In }
    Mem = In;
  }
}
```



BlockLibrary Verification

Structural correctness (syntax, structure)

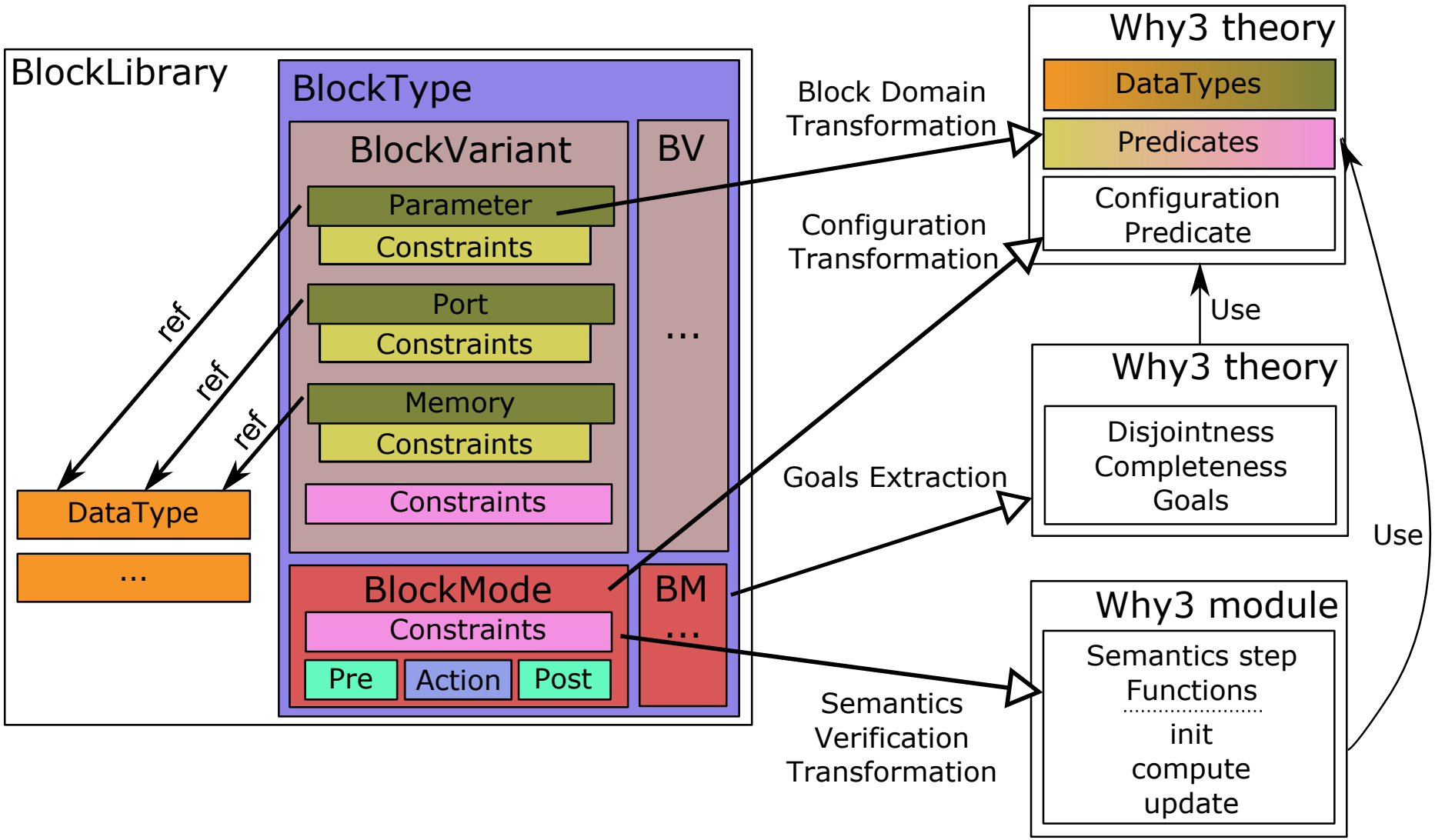
$\Rightarrow$ Metamodel conformance and OCL constraints on models

Semantics behavior correctness criteria (verification)

- Disjointness : Configurations are unique
- Completeness : All possible configurations are specified
- Hoare-based semantics consistency

$\Rightarrow$ Translation to a formal verification platform

Verification Strategy : Translation to Why3



$\Rightarrow$ SMT solver/Proof assistant to verify goals and functions

Disjointness/Completeness Properties Verification

1 – Block specification (extract)

```
variant InternalDelay {
  parameter Delay : TInt32
}
variant ExternalDelay {
  in data Delay : TInt32
}
variant DelayVar extends ALT(
  InternalDelay, ExternalDelay) {
  invariant ocl { Delay.value > 0 }
}
variant ListDelay_ScalarInput extends AND(
  ResetParam, DelayVar, InputScalar
) {
  invariant ocl { Delay.value > 1 }
  invariant ocl {
    Init.value.size() = Delay.value }
  memory Mem {
    datatype auto ocl {In.value}
    length auto ocl {0}
  }
}
```

2 – Generated Why predicates (extract)

```
type tDelay_InternalDelay_TInt32 = tParameter (tRealInt32)
type tDelay_ExternalDelay_TInt32 = tInPortGroup (tRealInt32)
type tMem_ListDelay_ScalarInput_TArray1Dim_TDouble =
    tMemoryVariable (list (tRealDouble))
(* Theory 1 *)
  predicate delayvar_inv_1 (...) =
    delay.value_pt > 0
  predicate listdelay_scalarinput_modeInv_1 (...) =
    delay.value_pt > 1
  predicate listdelay_scalarinput_modeInv_2 (...) =
    let init_0 = init.value_pt in length init_0 = delay.value_pt

(* Theory 2*)
  predicate listdelay_scalarinput_modeInv_2 (...) =
    let init_0 = init.value_pt in length init_0 = delay.value_inpg
```

3 – Generated verification goal (extract)

```
goal Delay_completeness :
  forall reset_Algo : tReset_Algo_ResetParam_TResetAlgo ,
         delay : tDelay_InternalDelay_TInt32 ,
         init : tInit_InternalICMatrix_TMatrixDouble, ...
delay_Simple_sig0 reset_Algo delay iC mem output input block \/
delay_Simple_sig1 reset_Algo delay mem iC output input block ...
```

4 – Verification output

```
DelayComplet_Delay.why Delay_Verif Delay_completeness : Valid (2.01s)
DelayComplet_Delay.why Delay_Verif Delay_disjointness : Valid (7.85s)
```

Conclusion

Current status

- A lightweight formalisation of a constraint language component inspired by OCL
- Automatic verification of DSML properties achievable with SMT solvers

Future works

- Automatic translation of the DSML constructs to WhyML
- Widening of the constraint language scope
  - Extensions based on DSML needs
  - Do not target full OCL (HOL-OCL fills the need)
- Verification feedback to the DSML user