ConForM:
Concurrency and Formal Methods

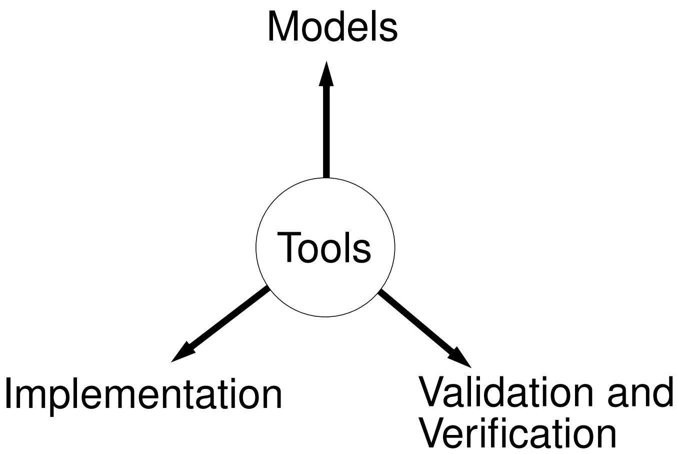
Didier Buchs

Current works and future directions

Goal of the main project of ConForM team

Applicable formal method for the development of large concurrent systems

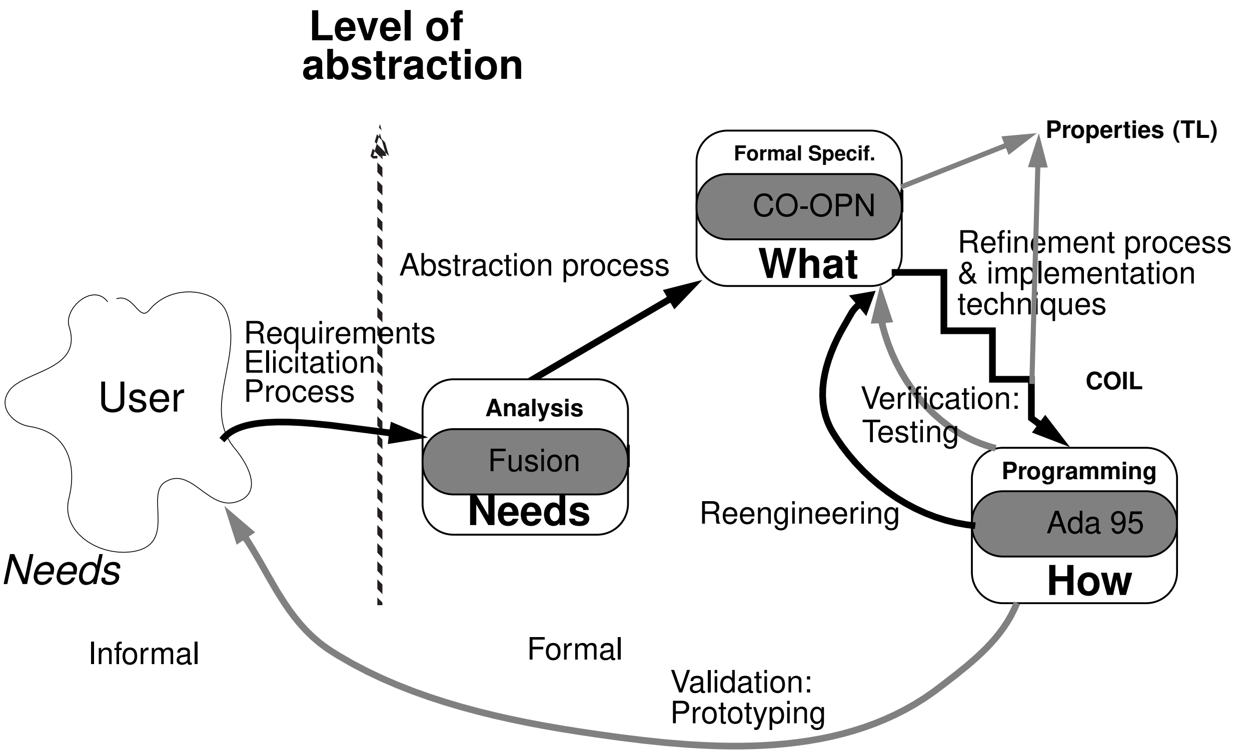
Research directions:



Main assumptions:

- Formal method based on a precise definition of the language and techniques that are proposed.
- Target systems: distributed or parallel systems.
- Object orientation used as structuring paradigm.
- Modelling of true concurrency.
- Applicable methods have to be supported by tools.
- Do not reinvent the wheel! Reuse already developed theory and tools.
- Check the validity of the method on concrete examples.

Model Relations



Future works

Already started directions that will be completed:

- Models,
- V&V,
- Implementation by means of prototype generation, distributed implementation model for CO-OPN/2
- Tools: adaptation of the tools, distribution
- Links with analysis phase

New directions:

- Temporal CO-OPN, Stochastic Nets
- Proof of basic properties on SN
- Operational tool for test set extraction
- Model for specific domain: UI specification and generation.

Basic CO-OPN 1.5

Models	V&V	Implementation	Tools
ADT	TL proofs based on predicate structure	Centralized Operational model for prototyping	Checker for CO-OPN1.5 Syntax and static semantics
Object	O-O principles of test	O-O principles of mixed prototyping	CO-OPN1.0 Editor and Simulator
Concurrency and synchronization	Theory of test for object based systems	Principles of reverse engineering for Ada95	CO-OPN1.0 Transformer
Genericity	Object - Oriented test methodology	Transformation principles	Extended simulator for CO-OPN1.5
Heterogeneity	SN basic analysis	Distributed operational model	TL Verifier and basic properties prover
Full Semantics of CO-OPN1.5	SN structural analysis	Distributed model of implementation	Mixed prototyping generator
Circularity of synchronization	Operational test selection principles	Static or dynamic configuration control	Graphical tools and UI generator

Basic CO-OPN 2.0

Models	V&V	Implementation	Tools
ADT order sorted	TL proofs based on predicate structure	Centralized Operational model for prototyping	Checker for CO-OPN2.0 Syntax and static semantics
Class subtyping		O-O principles of mixed prototyping	CO-OPN2.0 Editor and Simulator
Concurrency and synchronization			CO-OPN2.0 Transformer
Genericity	Object - Oriented test methodology	Transformation principles	
Heterogeneity		Distributed operational model	TL Verifier and basic properties prover
Full Semantics of CO-OPN2.0		Distributed model of implementation	Mixed prototyping generator
Circularity of synchronization	Operational test selection principles	Static or dynamic configuration control	Graphical tools and UI generator

Conclusions

- New directions already started
- Several papers accepted for publication and/or presentation:

Stéphane Barbey, Didier Buchs and Cécile Péraire, A Theory of Specification-Based Testing for Object-Oriented Software, 2nd European Dependable Computing Conference, Taormina, Italy, Lecture Notes in Computer Science, 1996.

Stéphane Barbey, Didier Buchs and Cécile Péraire, ``Object Oriented Specification Based Testing,'' Workshop on “Object Orientation and Dependability”, 29th IFIP WG10.4 “Dependable Computing and Fault Tolerance” Meeting, La Martinique, January 1996, pp113-122.

Stéphane Barbey, Didier Buchs and Cécile Péraire, ``A Theory of Specification-Based Testing,'' Technical Report 96/163, EPFL-DI-LGL, January 1996.

Stéphane Barbey, Didier Buchs and Cécile Péraire, ``A Theory of Specification-Based Testing for Object-Oriented Software,'' ECOOP workshop W2 "Testing Object-Oriented Sooftware" in Linz, J. Overbeck, june 1996.

Olivier Biberstein, Didier Buchs and Nicolas Guelfi, ``COOPN/2 : A Specification Language for Distributed Systems Engineering,'' Technical Report 96/167, Software Engineering

Laboratory, Swiss Federal Institute of Technology, Lausanne, Switzerland, 1996.

Olivier Biberstein, Didier Buchs and Nicolas Guelfi, ``COOPN/2 : A Specification Language for Distributed Systems Engineering,'' Technical Report 96/167, Software Engineering Laboratory, Swiss Federal Institute of Technology, Lausanne, Switzerland, 1996.

Olivier Biberstein, Didier Buchs and Nicolas Guelfi, ``Modeling of Cooperative Editors Using COOPN/2 ,'' OOPMC Workshop, International Conference on Applications and Theory of Petri Nets; Gul Agha, Fiorella de Cindio, Akinori Yonezawa, Osaka Japan, 1996, pp 25-42.

Didier Buchs, Pascal Racloz, Mathieu Buffo, Jacques Flumet and Erik Urland, ``Deriving Parallel programs using SANDS tools,'' Transputer Communication Journal, vol. 3, no. 1, 1996, pp. 23-32.

Didier Buchs, ``Les spécifications algébriques, un modèle formel pour le développement de logiciels,'' Génie logiciel: principes, méthodes et techniques, Alfred Strohmeier and Didier Buchs (Ed.), Presses Polytechniques et Universitaires Romandes, Lausanne, 1996, pp. 137-158.

Jarle Hulaas, ``An Evolutive Distributed Algebraic Petri Nets Simulator,'' Modelling and Simulation 1996, A. Javor and A. Lehmann and I. Molnar (Ed.), Society for Computer Simulation International, Budapest, Hungary, June 2-6 1996, pp. 348-352.

Pascal Racloz, ``Introduction aux réseaux de Petri,'' Génie logiciel: principes, méthodes et techniques, Alfred Strohmeier and Didier Buchs (Ed.), Presses Polytechniques et Universitaires Romandes, Lausanne, 1996, pp.207-240.

- Tool distribution will be provided asap.

Some Ideas about Parametric Finite Structures in CO-OPN/2

Didier Buchs

EPFL-DI-LGL

ConForM team

Plan

Introduction

Bounded Algebraic Specification

Parametric Bounded Algebraic Specification

Algebraic Specification in CO-OPN/2

Bounded Classes and Collections

Finite Structures in Algebraic Specifications

Algebraic specification allow to describe types:

- Infinite types using recursive definitions e.g. Stack, Naturals,...
- Finite types using recursive definitions and quotient with finite number of equivalence classes.
- Finite types using explicit enumeration of values.

Problems:

The quotient equations must be fixed and then it is difficult to parametrize the size of the finite structure.

Examples:

Description of modulo 3 integers using finite enumeration.

```
ADT Naturals;
INTERFACE
  SORTS naturals;
  GENERATORS
    f0 : -> naturals;
    f1 : -> naturals;
    f2 : -> naturals;
  OPERATIONS
    succ  : naturals -> naturals;
    prev  : naturals -> naturals;
BODY
  AXIOMS
    succ f2  = f0;    succ f0  = f1;    succ f1  = f2;

    prev f1  = f0;    prev f2  = f1;    prev f0  = f2;
END Naturals;
```


Description of modulo 3 integers using recursive definition.

```
ADT Naturals;
INTERFACE
  SORTS naturals;
  GENERATORS
    f0 : -> naturals;
    succ  : naturals -> naturals;
  OPERATIONS
    f1 : -> naturals;
    f2 : -> naturals;
    prev  : naturals -> naturals;
BODY
  AXIOMS
   succ succ succ f  = f;
   prev succ f  = f;
   f1 = succ f0;
   f2 = succ f1;
  WHERE
   f  : naturals;
END Naturals;
```

Problems:

- Difficult to generalize to modulo n for both ways

How can we generalize to other finite structures

- Add the possibility to express parametrized size of terms in equation i.e. $\texttt{succ}^n\ \texttt{x}$ for n time application of `succ` on `x`.

- Introduce additional constraints to the usual axioms

Bounded data type specification

Bounded data types:

- **Ideal part** which is an algebraic specification with a set of generators.

- **Constraint part** which is a set of constraint formulae on constructor terms.

  Presburger arithmetic is chosen by Arnould, Le Gall and Marre to express the constraints:

  - First order formulas over the operation 0,1,+ and <=
  - Sort measures inductively defined over the constructors

Presburger Arithmetic

Syntax

0,1,+ and <=

Properties:

Syntax of bounded specification

SP = (Σ, Ax, M, C)

Σ = signature = (S, G, F)

Ax = positive conditional axioms

M = S - indexed set of sort measure (on generators)
m_s: $(T_G)_s$ -> N

C = $(C_s)_{s \in S}$ is a set of bounding constraints (formulae expressed in the language 0,1,<=, and, or,...).

Examples:

```
ADT Naturals;
INTERFACE
  SORTS naturals;
  GENERATORS
    0 : -> naturals;
    succ  : naturals -> naturals;
  OPERATIONS
    _ + _  : naturals, naturals -> naturals;
  MEASURES
    size  : naturals -> Meta-nat;
BODY
  AXIOMS
   0 + g = g;    (succ f  ) + g = succ (f + g);
   size(0) = 0;
   size(succ f) = size(f) + 1;
  CONSTRAINTS
    Constraint(f) = size(f) <= 2;
  WHERE
   f,g  : naturals;
END Naturals;
```


Interpretation of bounds on terms

- $C(T_G)$ is the set of ground generator terms satisfying the constraints

Example:

`succ(succ(0))` $\in C(T_G)$

`succ(succ(succ(succ(0))))` $\notin C(T_G)$

- A term $t \in T_G$ is below bounds <=> all sub-terms t' of t are equivalent to terms u composed with generators and they are below bounds. $Ax \vdash t' = u$ and $u \in C(T_G)$
  (opposite is beyond bounds)

Example:

`succ(succ(0)) + succ(0)` is below bound

`succ(succ(0)) + succ(succ(0))` is beyond bound

- A ground formulae is below bounds <=> all its terms are below bounds, the set of formulae below bound is $B(Ax(\Sigma))$

Semantics

Given SP = (Σ, Ax, M, C) a specification with bound

A $\in$ Mod(Σ) is a model of SP <=> $\exists$ B $\in$ $Mod_{Gen}(SP_{ideal})$ s.t.

$$\forall \phi \in B(Ax(\Sigma)) \; (A \models \phi \iff B \models \phi)$$

Results: Models are finitely generated, and there exist a correct and complete calculus.

It means that A $\models$ Ax <=> A $\models$ B(Th(SP)) where B(Th(SP)) is the subset of B(Ax(Σ)) containing all consequences of Ax.

$$B(Th(SP)) = B(Ax(\Sigma)) \cap Th(SP)$$

Example of Models:

Z_k (k>2) are appropriate models for the previous specification.

What is the meaning of being beyond bounds?

Parametric Bounded Algebraic Specification

Conventional genericity management can be used to give more flexibility to the management of system constants.

```
PARAMETER Bound;
INTERFACE
  MEASURES
    n :  -> Meta-nat;
BODY
END Bound;
```

```
Adt 2 ;
INTERFACE
  MEASURES
    2 :  -> Meta-nat;
BODY
  AXIOMS
   2 = 1+1;
END Nat-3;
```


Instantiation

This can be used to instantiate, as needed, bounded naturals to specific size.

```
Adt Nat-3 as Naturals(2);
INTERFACE
BODY
END Nat-3;
```

```
GENERIC ADT Naturals(Bound);
INTERFACE
  SORTS naturals;
  GENERATORS
    0 : -> naturals;
    succ  : naturals -> naturals;
  OPERATIONS
    _ + _  : naturals, naturals -> naturals;
  MEASURES
    size  : naturals -> Meta-nat;
BODY
  AXIOMS
   0 + g = g;    (succ f ) + g = succ (f + g);
   size(0) = 0;
   size(succ f)) = size(f) + 1;
  CONSTRAINTS
    Constraint(f) = size(f) <= n;
  WHERE
   f,g  : naturals;
END Naturals;
```


Algebraic Specifications in CO-OPN/2

Used to describe:

- Values in the various computations.
- References to objects.
- Contents of places (multisets or bags).

Remark: References and contents are predefined types that are instantiated implicitly.

Bounded Collections

This can be used to instantiate the declaration:

```
...
PLACES
  P : T (n);
...
```

to specific type and size.

```
Adt TBagCapacity2 as Bags(T,2);
INTERFACE
BODY
END TBagCapacity2;
```


Bounded Classes

This can be used to instantiate, as needed for the declaration:

```
...
TYPE
   T (2);
...
```

to specific type and size.

```
Adt TRef as Ref(2);
INTERFACE
BODY
END TRef;
```

Parametric Bounded Classes and Collections

:

```
GENERIC CLASSES T (Constraint);
...
TYPE
   T (Constraint);
...
```

:

```
GENERIC CLASSES U (Constraint);
...
USE T(Constraint)
TYPE
   U (Constraint);
...
```

:

```
Class System as U(5);
......
```


Conclusion

Elegant way for introducing parametrizations which are difficult to express in the current version of CO-OPN/2

- can improve the test set generation
- easier exhaustive proofs
- more realistic modelling

but needs:

- Introduction of Meta-types (already defined for Meta-Bool)
- Generalization of the instantiation mechanism