Foundations of Heterogeneous Specification, Illustrated with an Example from Process Algebra

Till Mossakowski, BISS, University of Bremen, Germany
examples are joint work with Markus Roggenbach, Swansea
IFIP meeting, Menorca, June 2003
Motivation

- Multi-logic specifications are needed, since complex problems have different aspects that are best specified in different logics.
- A combination of all the used logics would become too complex in many cases.
- Different approaches can be compared.
- Formal interoperability among languages and tools.
- We aim at heterogeneous theorem proving ⇒ able to exploit the power of specialized proof tools.
Comparison with logic combination

Compared with logic combination, heterogeneous specification

- has only weaker forms of feature interaction, but
- is easier, more flexible and wider applicable (one metaformalism for all logics), and
- supports better re-use of existing proof tools (no need to implement new calculi).
An example from the literature

Feature Interactions: A Mixed Semantic Model Approach
(Paul Gibson, Bruno Mermet, Dominique Méry)

“The three types of formal models are used:

1. Firstly, we require an executable model (written in LOTOS using an object-based style) which is useful for constructing an executable model for validation.

2. Secondly, we have a logical model (based on the B method) which is used to verify the state invariant properties of our system (statically).

3. Finally, we use TLA to provide semantics for a static analysis of liveness and fairness properties. No one model can treat each of these aspects, yet each of these aspects of the conceptualisation are necessary in the formal development of features.”
An example using \texttt{CSP-CASL} and \texttt{CASL-LTL}

\begin{verbatim}
logic CSP-CASL = spec Buffer =
data List
channels read, write : Elem
process let Buf(l : List[Elem]) =
  read?x \rightarrow Buf(cons(x, nil))
  \Box \text{if } l = \text{nil} \text{ then STOP}
  \text{else write!last(l)} \rightarrow Buf(rest(l))
in Buf(nil)
with logic \rightarrow \text{CASL-LTL}
then \%implies
forall x : ds. in\_any\_case(x,
  always eventually label\_cond(y . fst(y) = write))
\end{verbatim}
Institutions

- category $\text{Sign}$ of signatures,
- sentence functor $\text{Sen}: \text{Sign} \rightarrow \text{Set}$
- a functor $\text{Mod}: \text{Sign}^{op} \rightarrow \mathcal{CAT}$

satisfaction relation $\models_{\Sigma} \subseteq |\text{Mod}(\Sigma)| \times \text{Sen}(\Sigma)$, such that

$$M' \models_{\Sigma'} \text{Sen}(\sigma)(\varphi) \Leftrightarrow \text{Mod}(\sigma)(M')' \models_{\Sigma} \varphi$$

or shortly

$$M' \models_{\Sigma'} \sigma(\varphi) \Leftrightarrow M'|_{\sigma} \models_{\Sigma} \varphi$$

(satisfaction condition).
Objection #1 (by a referee)

Most of those aspects are naturally described by languages (and logics) that do not form an institution in the obvious way, though they can be (painfully) coded to become an institution.

Then, an institution based mechanism to compose heterogeneous specifications seems to me a fake, because the component specifications are not expressed in the “natural” language for the particular issue they are addressing, have to be already coded in the institution version of the “natural” language.

I have the strong belief that the heterogeneity that can be naturally achieved within this kind of composition is limited to different subsets of logical languages.
“Logical” and “programming” institutions

A $\Sigma$-sentence $\varphi$ in an institution is called a program, if

- it is non-trivial, i.e. does not hold in every model, and
- there is some signature morphism $\sigma: \Sigma' \rightarrow \Sigma$ such that any $\Sigma'$-model $M$ has a unique $\sigma$-expansion satisfying $\varphi$.

(In this case, it follows that $\varphi \notin \sigma(\text{Sen}(\Sigma'))$.)

Otherwise, $\varphi$ is called logical.

An institution is a programming institution, if all of its sentences are programs. Otherwise, it is said to be logical.
A logical institution: CASL-LTL (Reggio, Astesiano, Choppy)

- Signatures: subsorted partial first-order signatures with distinguished ternary labeled transition system (LTS) predicates $\text{__} \xrightarrow{__} \text{__}: s \times lab \times s$
- Models: subsorted partial first-order structures
- Sentences: first-order sentences and CTL*-formulas
- Satisfaction: first-order and CTL*-satisfaction, where the LTS predicates are used to generate traces
Two kinds of programming institutions

1. Programs are definitions of just one global object. Recursion is solely dealt with by let-expressions.

2. Programs are definitions of some (named) global object. Recursion among global objects is allowed. This requires a pre-order on object names.

We subsequently will use the second kind.
The **CSP-CASL** institution with LTS semantics

CSP is a programming institution, and so is **CSP-CASL**.

Basic idea: use **CASL** models to provide the CSP alphabet.

Labelled transition systems (LTS) based models are the most general ones: all the different kinds of denotational semantics (like trace, failure and failure/divergence semantics) can be derived from it.
\textbf{Csp-Casl signatures and signature morphisms}

Signatures are triples \((\Sigma, N, \leq)\), where

- \(\Sigma\) is a \textit{Casl} signature, and
- \((N, \leq)\) is a pre-ordered set of process names (the pre-order also induces an equivalence relation).

Signature morphisms are pairs \((\sigma, f)\), where

- \(\sigma\) is a sort-injective, subsorting-preorder reflecting \textit{Casl} signature morphism,
- and
- \(f\) is a monotone function.
**CSP-CASL: LTS-based models**

A \((\Sigma, N, \leq)\)-model \((M, L)\) consists of

- a CSP \(\Sigma\)-model \(M\),
- for each process name \(n\), a labelled transition system \(L(p)\), with labels in

\[
Lab(M) := \left( \biguplus_{s \in \text{Sorts}(\Sigma)} M_s \right) / \sim \uplus \{ \bot, \sqrt{\_}, \tau, \top \}
\]

\(\sim\) is the equivalence relation generated by

\[
x \sim \text{inj}_{s,s'}(x) \text{ for } s \leq s'
\]
• $\bot$ is an additional alphabet letter standing for undefinedness;

• $\sqrt{}$ stands for successful termination, generated by the usual CSP semantics;

• $\tau$ stands for hiding via the CSP-operator;

• $\top$ stands for hiding via model reducts.
CSP-CASL: model reducts

\[(M, L)|_{(\sigma, f)} = (M|_\sigma, LTS(\hat{\sigma}) \circ L \circ f)\]

Note that due to the restrictions on signature morphisms, there is an injection \(l_\sigma: Lab(M|_\sigma) \rightarrow Lab(M)\).

\[\hat{\sigma}(x) := \begin{cases} y, & \text{if } l_\sigma(y) = x \\ \top, & \text{if no such } y \text{ exists} \end{cases}\]

\(LTS(\hat{\sigma})\) maps an LTS by changing labels along \(\hat{\sigma}\).
**CSP-CASL: sentences**

A \((\Sigma, N, \leq)\)-sentence is either

- a \(\text{CASL} \ \Sigma\)-sentence, or
- a system of equations \(\langle p_1 = t_1, \ldots, p_n = t_n \rangle\), where \(\{p_1, \ldots, p_n\}\) is an equivalence class of process names, and the \(t_i\) are CSP process terms with \(\text{CASL}\) terms in place of alphabet letters, possibly involving process names less then \(p_1, \ldots, p_n\).

Sentence translation is straightforward.
**CSP-CASL: satisfaction**

A model \((M, L)\) satisfies

- a CASL sentence \(\varphi\), if \(M\) satisfies \(\varphi\) in CASL,
- a system of equations \(\langle p_1 = t_1, \ldots, p_n = t_n \rangle\), if

\(L(p_1), \ldots, L(p_n)\) is the least solution of

\(\langle p_1 = t_1, \ldots, p_n = t_n \rangle\) using CSP semantics and \(L(p)\) for \(p\)

less than \(p_1, \ldots, p_n\).
The institutions involved so far

$$\mathbf{CSP-CASL}$$

$$\mathbf{CASL} =$$

$$\mathbf{CASL-LTL}$$
Institution (co)morphisms


\[
\begin{array}{ccc}
\text{morphism} & \quad & \text{comorphism} \\
\text{Sign} & \rightarrow & \text{Sign}' \\
\text{Sen} & \leftarrow & \text{Sen}' \circ \Phi \\
\text{Mod} & \rightarrow & \text{Mod}' \circ \Phi \\
\end{array}
\]
A small logic graph

\[
\begin{align*}
\text{morphism: } pr \quad & \quad \text{CSP-CASL} \\
\text{morphism: embed} \quad & \quad (\text{theoroidal partial semi-}) \text{ comorphism: toLTL} \\
\text{CASL}^= & \quad \text{CASL-LTL}
\end{align*}
\]
Problem

How to provide a formal basis for heterogeneous specification?
Solution

- **Indexed institution** = diagram of institutions and morphisms
- **Indexed coinstitution** = diagram of coinstitutions and morphisms = diagram of institutions and comorphisms
- For both, we have a Grothendieck construction (Diaconescu, Applied categorical structures 10, 2002)

Slogan:

Heterogeneous specification is structured specification over the Grothendieck institution
Comorphism based Grothendieck institution

Given an indexed coinstitution $\mathcal{I}: \text{Ind}^{\text{op}} \rightarrow \text{CoIns}$, define the Grothendieck institution $\mathcal{I}^\#$ as follows:

- signatures in $\mathcal{I}^\#$ are pairs $(\Sigma, i)$, where $i \in |\text{Ind}|$ and $\Sigma$ a signature in the institution $\mathcal{I}(i)$,
- signature morphisms $(\sigma, e): (\Sigma_1, i) \rightarrow (\Sigma_2, j)$ consist of a morphism $e: j \rightarrow i \in \text{Ind}$ and a signature morphism $\sigma: \Phi_{\mathcal{I}(e)}(\Sigma_1) \rightarrow \Sigma_2$,
- the $(\Sigma, i)$-sentences are the $\Sigma$-sentences in $\mathcal{I}(i)$,
- the $(\Sigma, i)$-models are the $\Sigma$-models in $\mathcal{I}(i)$,
- satisfaction w.r.t. $(\Sigma, i)$ is satisfaction w.r.t. $\Sigma$ in $\mathcal{I}(i)$.
Problem

How to combine the **morphism-based** and the **comorphism-based** Grothendieck construction?
The Bi-Grothendieck institution

Let \((\mathcal{I}_m, \mathcal{I}_c, \mathcal{I}_0)\), with \(\mathcal{I}_m\) an indexed institution, \(\mathcal{I}_c\) an indexed coinstitution and \(\mathcal{I}_0\) a discrete indexed institution be given, such that \(|\text{Ind}_m| = |\text{Ind}_c| = |\text{Ind}_0|\), and \(\mathcal{I}_m, \mathcal{I}_c\) and \(\mathcal{I}_0\) agree on these. Then we form the Grothendieck institutions \(\mathcal{I}_0^\#, \mathcal{I}_m^\#\) and \(\mathcal{I}_c^\#\). Since \(\mathcal{I}_0^\#\) obviously is included in \(\mathcal{I}_m^\#\) and \(\mathcal{I}_c^\#\) via a (co)morphism, we can take the pushout

\[
\begin{array}{ccc}
\mathcal{I}_0^\# & \rightarrow & \mathcal{I}_m^# \\
\downarrow & & \downarrow \\
\mathcal{I}_c^# & \rightarrow & \mathcal{J}
\end{array}
\]

in the category of institutions and institution morphisms. \(\mathcal{J} =: (\mathcal{I}_m, \mathcal{I}_c)^\#\) is called the Bi-Grothendieck institution.
Inducibility

Given

- an institution comorphism $\rho = (\Phi, \alpha, \beta): I \rightarrow J$,
- a functor $\Psi: \text{Sign}^J \rightarrow \text{Sign}^I$,
- and a natural transformation $\varepsilon: \Phi \circ \Psi \rightarrow \text{Id}$,

we say that $\rho$ $\varepsilon$-induces the institution morphism $\mu = (\Psi, \bar{\alpha}, \bar{\beta}): J \rightarrow I$ given by

\[
\bar{\alpha} = (\text{Sen}^J \cdot \varepsilon) \circ (\alpha \cdot \Psi) \\
\bar{\beta} = (\beta \cdot (\Psi^\text{op}) \circ (\text{Mod}^J \cdot \varepsilon^\text{op})
\]

A morphism that is $\varepsilon$-induced by some $\rho$ is called inducible.
Simplification of the Bi-Grothendieck institution

Theorem.
Let \((\mathcal{I}_m, \mathcal{I}_c, \mathcal{I}_0)\) as above be given.

If each morphism in \(\mathcal{I}_m\) is induced by some comorphism in \(\mathcal{I}_c\), then there is a retraction of \((\mathcal{I}_m, \mathcal{I}_c)^\#\) onto \(\mathcal{I}_c^\#\).

Dually, if each comorphism in \(\mathcal{I}_c\) is induced by some morphism in \(\mathcal{I}_m\), then there is a retraction of \((\mathcal{I}_m, \mathcal{I}_c)^\#\) onto \(\mathcal{I}_m^\#\).
But . . .

Proposition

Neither

- the morphism $pr: \text{CSP-CASL} \rightarrow \text{CASL}$, nor
- the theoroidal semi-comorphism $\text{toLTL}: \text{CSP-CASL} \rightarrow \text{CASL-LTL}$ is inducible.
Better solution: Work with spans

Each morphism \( I \xrightarrow{\alpha} J \)

\[
\begin{array}{c}
\Psi \\
\downarrow \beta \\
\end{array}
\]

can be translated into a span of comorphisms

\[
\begin{align*}
\text{Sign} & \xleftarrow{id} \text{Sign}^I \\
\text{Sen}^I & \xleftarrow{\alpha} \text{Sen}^J \circ \Psi \\
\text{Mod}^I & \xrightarrow{\beta} \text{Mod}^J \circ \Psi \\
\text{Sign}^J & \xrightarrow{\Psi} \\
\text{Sen}^J \circ \Psi & \xrightarrow{id} \\
\text{Mod}^J \circ \Psi & \xrightarrow{id}
\end{align*}
\]
The Bi-Grothendieck institution can be recovered

Theorem.
Given an indexed institution \( \mathcal{I}_m \) and an indexed coinstitution \( \mathcal{I}_c \) (both over the same set of institutions in the above sense),
then each development graph (or structured specification) over the Bi-Grothendieck institution \( (\mathcal{I}_m, \mathcal{I}_c)^\# \) can be translated into a development graph over the Grothendieck institution over the span-based indexed coinstitution \( \text{Span}(\mathcal{I}_m, \mathcal{I}_c)^\# \), such that model categories are preserved.
Also can be modeled as spans of comorphisms:

- partial comorphisms
- semi-morphisms
- semi-comorphisms
- certain forward morphisms and comorphisms
Objection #2 (by another referee)

[The author] makes . . . the claim that one can restrict oneself to just comorphisms, because the others can be transformed to comorphisms. Due to the inherent duality between morphisms and comorphism, exactly the same thing, but dual, can be done for morphisms.
Comorphisms are simpler than morphisms

Proposition
The Grothendieck institution $\mathcal{I}^\#$ of an indexed coinstitution $\mathcal{I} \colon \text{Ind}^{\text{op}} \rightarrow \text{CoIns}$ consisting of comorphisms with cocontinuous signature translation is (weakly) semi-exact if and only if
- $\mathcal{I}$ is (weakly) locally semi-exact,
- $\mathcal{I}$ is (weakly) semi-exact, and
- all institution comorphisms in $\mathcal{I}$ are (weakly) exact.

Diaconescu’s corresponding result for morphisms needs to assume that these are induced by comorphisms.
The heterogeneous specification language

- uses institution-specific syntax for basic specifications
- uses CASL structuring constructs
- the current institution is chosen using `logic <logic-name>`
- heterogeneous constructs:
  - `SP with logic <comorphism-name>`
  - `SP hide logic <morphism-name>`
- implicit coercions along standard inclusions
- reactive institutions have an underlying data institution
An example using $\textbf{CSP-CASL}$ and $\textbf{CASL-LTL}$

**logic** $\textbf{CSP-CASL} =$  

**spec** $\textbf{BUFFER} =$  

**data** $\textbf{LIST}$  

**channels** $\text{read}, \text{write} : \text{Elem}$  

**process** 

let $\text{Buf}(l : \text{List}[\text{Elem}]) =$  

\[
\text{read}\,?\,x \rightarrow \text{Buf}(\text{cons}(x, \text{nil})) \\
\square \text{if } l = \text{nil} \text{ then STOP}
\]

else $\text{write}!\text{last}(l) \rightarrow \text{Buf}(\text{rest}(l))$

\[
\text{in} \quad \text{Buf}(\text{nil})
\]

**with** $\text{logic} \rightarrow \textbf{CASL-LTL}$  

**then**  

$\%$implies  

forall $x : ds$ . in\_any\_case$(x,$  

always eventually label\_cond$(y \cdot \text{fst}(y) = \text{write})$
Proof systems

- **Heterogeneous borrowing:** uses encoding in some “universal” institution with good proof support
- **Heterogeneous proof calculus:** via heterogeneous development graphs, also allows heterogeneous proofs involving different calculi. This also is a generalization of heterogeneous bridges (G. Bernot, S. Coudert, and P. Le Gall: Towards heterogeneous formal specifications. *AMAST 96*.)
Tools

- the **heterogeneous tool set** (hets), via multiparameter type classes and existential types in Haskell, see technical report
- is based on parser, static analyser and prover for each institution
- **heterogeneous parser** and **static checker** are already available!
- **Proof support:** **MAYA**, with an XML-RPC interface for provers (see FroCoS 2002) — needs to be made heterogeneous
Architecture of the heterogeneous tool set Hets

Logics L1 ... Ln
- Text
- Parser
- Abstract syntax
- Static Analysis
- (Signature, Sentences)
- Interfaces
- XML, Aterm
- Prover1
- Provern
- Consistency checker

Logic comorphism L1 → L2

Logic comorphism transformation L1 ↓ L2

Logic graph
- L1 → L3 → L5
- L2 → L4 → L6

Grothendieck logic
- (Σ1,L1)
- Φ(Σ1) → L1
- σ → L2
- (Φ,α,β) → Σ2 → L2
- (Σ2,L2)

Heterogen. Spec
- Text
- Parser
- Abstract syntax
- Static Analysis
- Global Environment
- Interfaces
- XML, Aterms
- WWW, GUI

Maya
- Development graph
- Heterogeneous inference engine
- Management of proofs & change
- Heterogeneous proof trees

daVinci

Till Mossakowski: Heterogeneous specification, IFIP meeting, Menorca, June 2003
Hets-supported institutions so far

- **CASL**
- **HasCASL**
- Haskell (only parser so far)
- **CSP-CASL** (only parser so far)
Relevant references

Conclusion and future work

- Heterogeneous specifications can deal with practically relevant examples
- Spans of comorphisms suffice for the Grothendieck construction
- Heterogeneous specification: Concepts and theory are there —
  now: more examples & tool development

http://www.tzi.de/cofi/