

Heterogeneous CASL

Till Mossakowski

30.03.2004



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).

Typcial Heterogeneous Situations

- **Viewpoint specifications**, e.g.: data types, process algebra and temporal logic
- First-order specifications with **some** higher order part (e.g. real numbers)
- **Wide-spectrum** specifications involving specifications and programs
- Specifications involving **different modalities** (e.g. deontic and temporal modalities)
- Different input languages for **tools**

A sample heterogeneous specification

```

spec BOOL = free type Bool ::= True | False
spec LIST[sort Elem] =
  free type List[Elem] ::= nil | cons(Elem; List[Elem])
logic HASCASL
spec FILTER = BOOL and LIST[sort Elem]
then
op   filter : (Elem → Bool) → List[Elem] → List[Elem]
var  p : Elem → Bool; x : Elem; l : List[Elem]
.     filter p [] = []
.     filter p (x :: l) = x :: filter p l if p x = True
.     filter p (x :: l) = filter p l if p x = False

```

A more complex example

logic CspCASL

spec BUFFER =

data LIST

channels *read, write : Elem*

process **let** *Buf*(*l : List[Elem]*) =

read?x → Buf(cons(x, nil))

\square *if l = nil then STOP*

else write!last(l) → Buf(rest(l))

in *Buf(nil)*

with **logic** → CASL-LTL

then %implies

$\forall x : ds . in_any_case(x, always\ eventually\ label_cond(y . fst(y) = write))$

Typical institution (co)morphisms

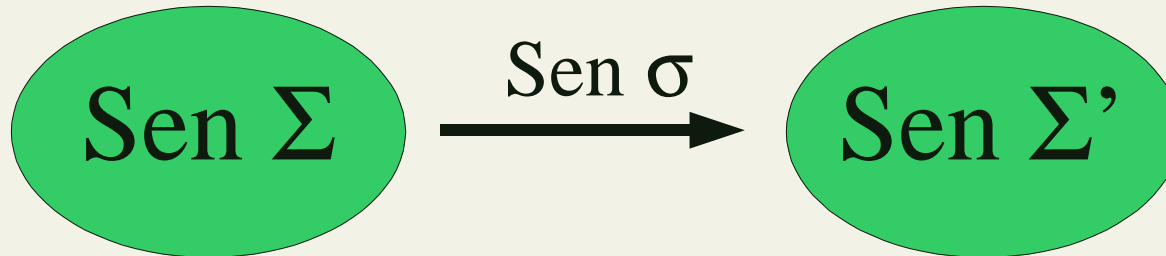
1. trivial **inclusions** (e.g. $FOL \hookrightarrow HOL$)
2. trivial **projections** (e.g. $HOL \rightarrow FOL$)
3. **encodings** (e.g. $CASL \rightarrow HOL$)
4. **feature interaction** (cf. example)
5. **implementation** going from specification to programming language (e.g. $HASCASL \rightarrow Haskell$)
6. . . . or the other way round (e.g. $Haskell\text{-subset} \rightarrow CASL$)

Institutions

Signatures

$$\Sigma \xrightarrow{\sigma} \Sigma'$$

Sentences

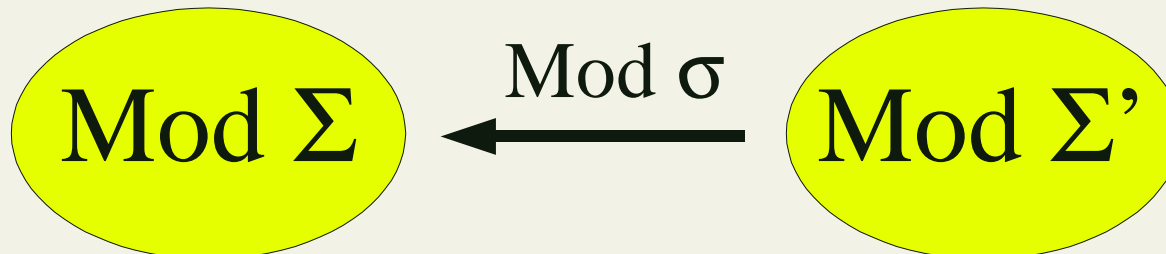


Satisfaction

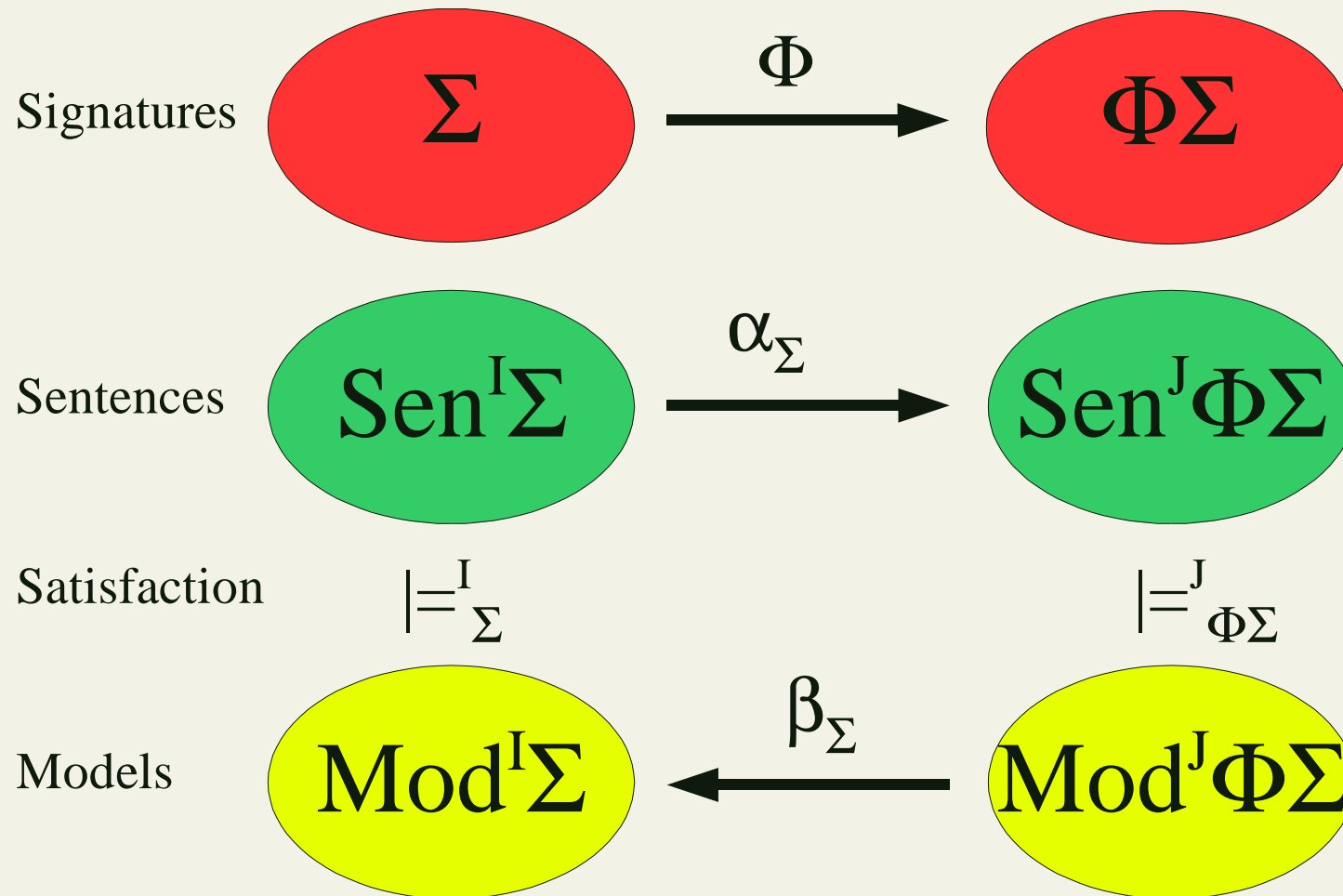
$$\models_{\Sigma}$$

$$\models_{\Sigma'}$$

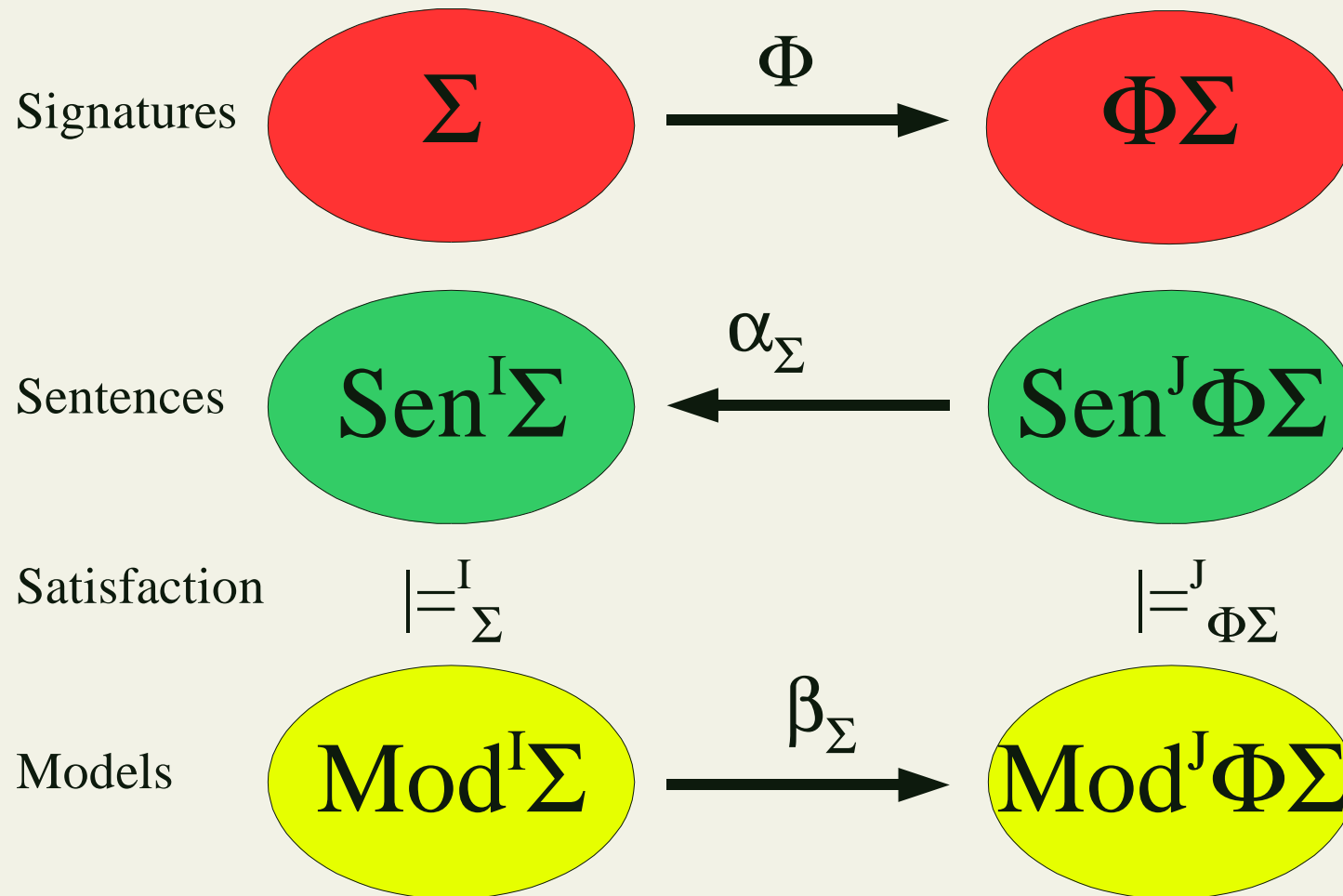
Models



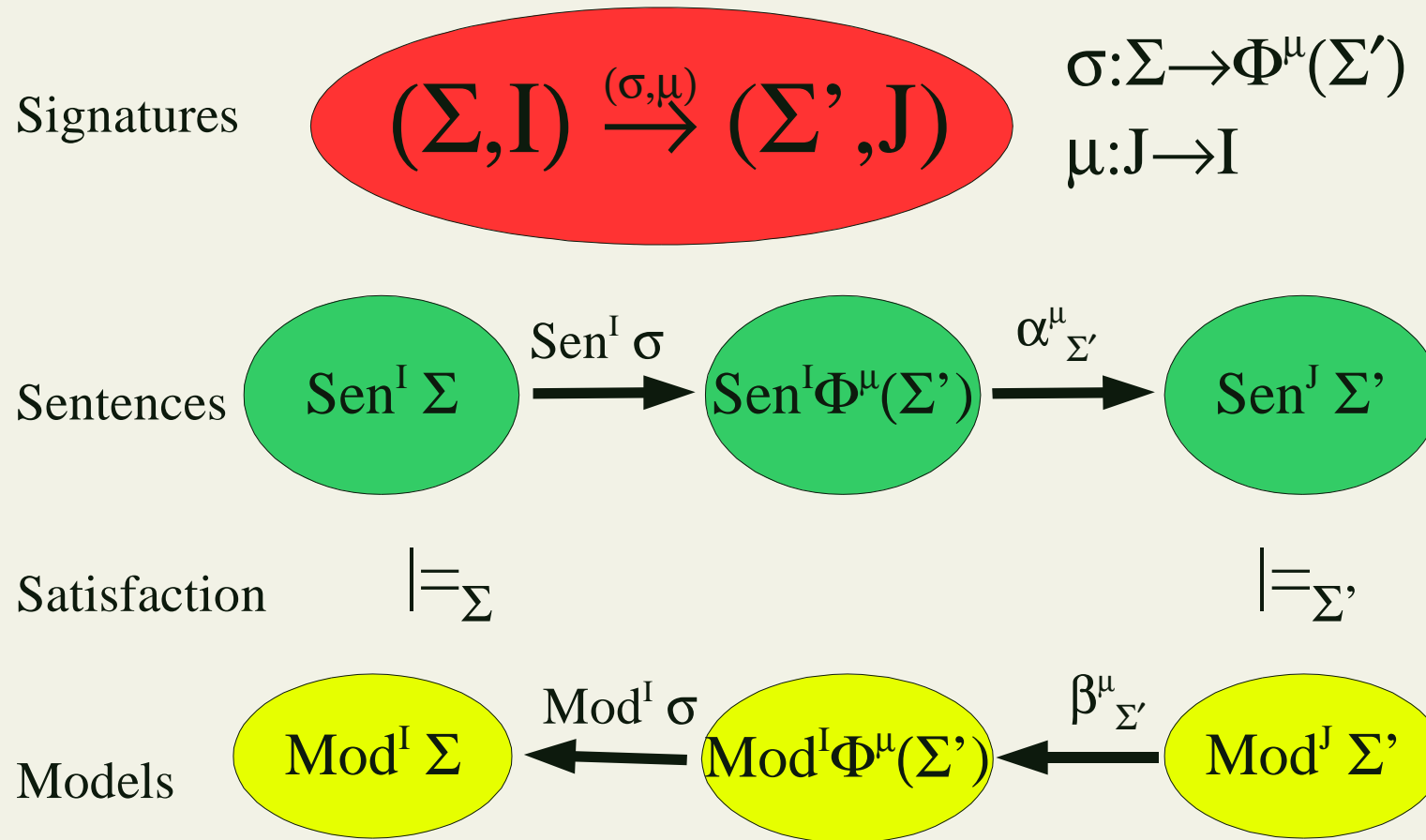
Institution representations



Institution morphisms



The Grothendieck Institution



Semantic basis of HETCASL

- **Graph** of institutions/logics and (co)morphisms with marked **default inclusions** and **projections**
- Distinguished **main** institutions with subinstitutions
- Some institutions come with **proof system**
- Heterogeneous specification = CASL structured specification in the **Grothendieck institution** [Dia01]
- We here need **Bi-Grothendieck institution** = pushout of comorphism-based and morphism-based Grothendieck institutions [Mos02]

Heterogeneous constructs: the current logic

- The **current** logic is part of the local environment
- **However**: there is no default “empty” logic!
- The current logic may be selected using “**logic L**”
- **logic L {SP}** interprets SP in L
- implicit coercions via unions with the local environment

Heterogeneous Translations and Reductions

- Translations along comorphisms:

SP with logic $c:L_1 \rightarrow L_2$,

SP with logic $\rightarrow L_2$

SP with logic $L_1 \rightarrow L_2$,

- Reductions along morphisms:

SP hide logic $c:L_1 \rightarrow L_2$,

SP hide logic $\rightarrow L_2$

SP hide logic $L_1 \rightarrow L_2$,

- logic $c:L_1 \rightarrow L_2$ may also be used in symbol lists and maps

Process Specifications

- $\text{data } SP_1 \text{ } SP_2$ is shorthand for
 $\text{logic } L_1 \text{ } SP_1 \text{ with logic } L_2 \text{ then } SP_2$

Naming sublogics

- Sublogics are named $L . SL$
- SL can be a sublogic of more than one main logic L
- Main logic is used to decide which tool to use

Comorphism transformations

Example: we can go from *FOL* to *HOL* either **directly** or **via CASL**.

Problem: the two ways **differ**, but this should not matter in the Grothendieck institutiton.

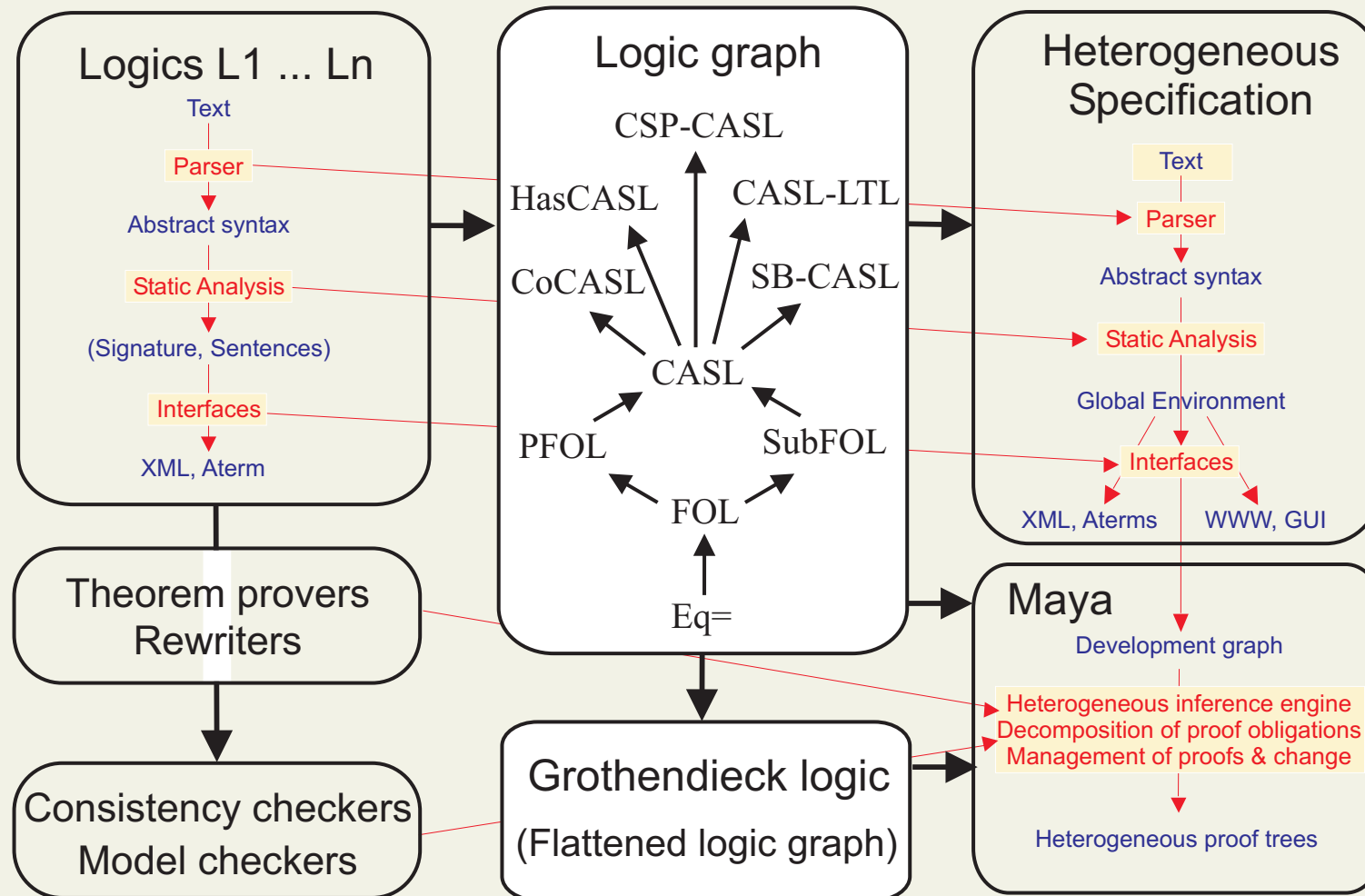
Solution: work with **comorphism transformations** (2-cells between comorphisms), which lead to a **quotient** on the Grothendieck institutiton.

Observation: Tarlecki's **representation maps** for morphism-comorphism triangles translate to **comorphism transformations** for comorphism squares.

Further results

- Bi-Grothendieck institution can be **simplified** to span-based Grothendieck institution [Mossakowski WADT 02]
- **Development graph proof calculus** has been adapted to the heterogeneous case [Mossakowski MFCS 02]
- **Heterogeneous borrowing**: see [Mossakowski Fossacs 02]
- **Tool support**: the heterogeneous tool set [WADT 2004]

Architecture of the heterogeneous tool set Hets



Conclusion and future work

- HETCASL allows us to deal with all the CASL extensions!
- More examples; aiming at heterogeneous developments
- Make heterogeneous development graph calculus better implementable

<http://www.tzi.de/cofi/hetcasl>

Results about spans

Theorem: Structured specifications
over the **Bi-Grothendieck institution**
can be translated into structured specifications
over the **span comorphism-based Grothendieck institution**,
such that model categories are preserved.

Key idea: Translate

SP hide logic (Ψ, α, β)

into

SP hide logic (id, α, β) **with logic** (Ψ, id, id)

The Bi-Grothendieck institution

- Take both a **morphism-based** indexed institution \mathcal{I}_m and a **comorphism-based** indexed coinstitution \mathcal{I}_c (both being built over the same discrete indexed institution \mathcal{I}_0),
- form their **Grothendieck institutions** $\mathcal{I}_m^\#$ and $\mathcal{I}_c^\#$ and
- take the **pushout**

$$\begin{array}{ccc} \mathcal{I}_0^\# & & \mathcal{I}_m^\# \\ & \searrow & \downarrow \\ \mathcal{I}_c^\# & & (\mathcal{I}_m, \mathcal{I}_c)^\# \end{array}$$

in the category of institutions and institution morphisms.