Institutions:
an abstract framework for foundations of software specification and logic

Andrzej Tarlecki

Institute of Informatics, University of Warsaw

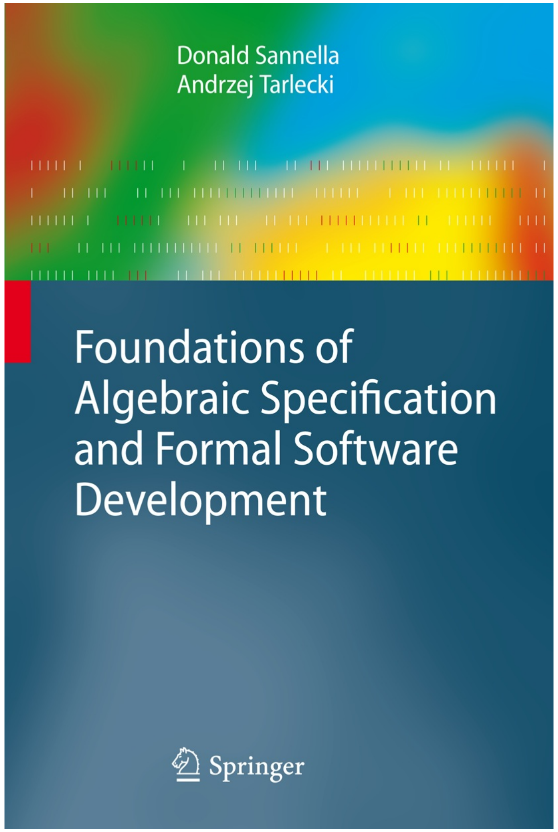
tarlecki@mimuw.edu.pl

http://www.mimuw.edu.pl/~tarlecki

Institutions:

theoretical foundations to frame practical issues

Foundations of Software Specification and Development



Ultimate goal

A *formal basis*

for *systematic development*

of *correct software systems*

from *requirements specifications*

by *verified refinement steps*.

so that we can sleep at night

Formal basis:

- Mathematical structures to model software systems
- Logical systems to capture their properties
- Formal semantics to assign meanings to syntax
- Proofs to facilitate certainty and understanding

Software models

Programs should be:

- *clear; efficient; robust; reliable; user friendly; well documented; . . .*
- *but first of all, CORRECT*
- *don't forget though: also, executable. . .*

First approximation:

Software system (module, program, database, . . .):

modelled as an algebra

$=$ *sets of data values with operations on them*

- Disregarding: code (and efficiency, robustness, reliability, . . .)
- Focusing on: semantics (and input/output behaviour)

Correctness

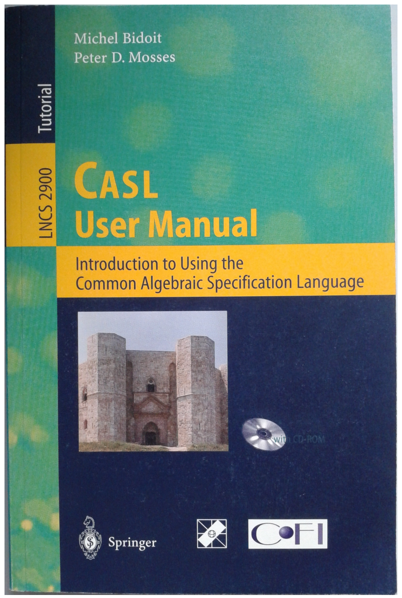
Software correctness makes sense only
w.r.t. a precise *specification* of the requirements.

Specification: defines which software systems are acceptable
$=$ *description of a set (class) of algebras*

- Mainly: listing PROPERTIES that an acceptable system must satisfy
  - often: equational, first-order, etc, properties that characterise the results of the operations of the system
- Separates WHAT system should do from HOW it works

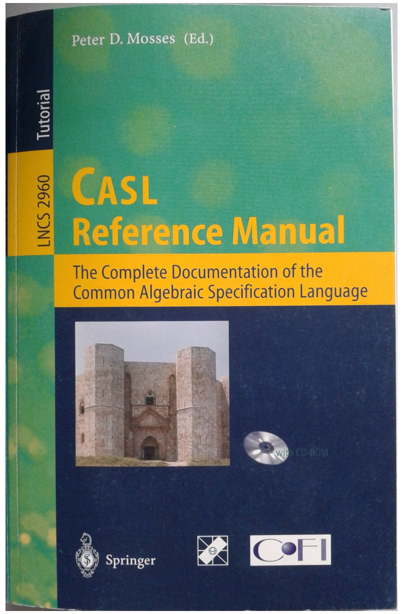
Rough analogy

module interface	$\leadsto$	signature
module	$\leadsto$	algebra
module specification	$\leadsto$	class of algebras



CASL

Common
Algebraic
Specification
Language



Generality and abstraction

There are many choices:

- Software systems: Non-termination allowed? Exceptions? Non-determinism? Higher-order functions? Concurrency? etc.
- Specifications: Logical language to capture basic required properties? Equational? First-order? Higher-order? Temporal formulae? LTL, CTL, CTL*?
- Proofs: Logical calculi for building proofs (of properties, of refinement steps, etc.)

Most of the theory is independent of most of these choices!

We try to make this explicit:

rely only on basic common features

Crash course I

Universal algebra

Trivial data type

Its *signature* Σ (syntax):

$$
\begin{array}{ll}
\textbf{sorts} & \mathit{Int}, \mathit{Bool}; \\
\textbf{opns} & 0, 1 \colon \mathit{Int}; \\
 & \mathit{plus}, \mathit{times}, \mathit{minus} \colon \mathit{Int} \times \mathit{Int} \to \mathit{Int}; \\
 & \mathit{false}, \mathit{true} \colon \mathit{Bool}; \\
 & \mathit{lteq} \colon \mathit{Int} \times \mathit{Int} \to \mathit{Bool}; \\
 & \mathit{not} \colon \mathit{Bool} \to \mathit{Bool}; \\
 & \mathit{and} \colon \mathit{Bool} \times \mathit{Bool} \to \mathit{Bool};
\end{array}
$$

and Σ-*algebra* A (semantics):

$$
\begin{array}{ll}
\textbf{carriers} & A_{\mathit{Int}} = \mathbf{Int}, A_{\mathit{Bool}} = \mathbf{Bool} \\
\textbf{operations} & 0_A = 0, 1_A = 1 \\
 & \mathit{plus}_A(n,m) = n+m, \mathit{times}_A(n,m) = n*m \\
 & \mathit{minus}_A(n,m) = n-m \\
 & \mathit{false}_A = \mathbf{ff}, \mathit{true}_A = \mathbf{tt} \\
 & \mathit{lteq}_A(n,m) = \mathbf{tt} \textbf{ if } n \le m \textbf{ else } \mathbf{ff} \\
 & \mathit{not}_A(b) = \mathbf{tt} \textbf{ if } b = \mathbf{ff} \textbf{ else } \mathbf{ff} \\
 & \mathit{and}_A(b,b') = \mathbf{tt} \textbf{ if } b = b' = \mathbf{tt} \textbf{ else } \mathbf{ff}
\end{array}
$$

Signatures & algebras

- *Algebraic signature*:

$$\boxed{\Sigma = (S, \Omega)}$$

  – *sort names*: S

  – *operation names, classified by their argument and result sorts*: $\Omega = \langle \Omega_{w,s} \rangle_{w \in S^*, s \in S}$

- Σ-*algebra*:

$$\boxed{A = (|A|, \langle f_A \rangle_{f \in \Omega})}$$

  – *carrier sets:* $|A| = \langle |A|_s \rangle_{s \in S}$

  – *operations*: $f_A \colon |A|_{s_1} \times \ldots \times |A|_{s_n} \to |A|_s$, for $f \in \Omega_{s_1 \ldots s_n, s}$

- $f \colon s_1 \times \ldots \times s_n \to s$ stands for $s_1, \ldots, s_n, s \in S$ and $f \in \Omega_{s_1 \ldots s_n, s}$

Fix a signature $\Sigma = (S, \Omega)$ for a while.

Few further notions

- the class of all Σ-algebras: $\mathbf{Alg}(\Sigma)$
- *subalgebra* $A_{sub} \subseteq A$: given by subset $|A_{sub}| \subseteq |A|$ closed under the operations
- *homomorphism* $h\colon A \to B$: map $h\colon |A| \to |B|$ that preserves the operations
- *isomorphism* $i\colon A \to B$: bijective homomorphism
- *congruence* $\equiv$ on A: equivalence $\equiv \subseteq |A| \times |A|$ closed under the operations
- *quotient algebra* $A/{\equiv}$: built in the natural way on the equivalence classes of $\equiv$
- *product algebra* $\prod_{i \in \mathcal{I}} A_i$: built on the Cartesian product of algebra carriers, with operations defined componentwise

Subalgebras

- for $A \in \mathbf{Alg}(\Sigma)$, a Σ-*subalgebra* $A_{sub} \subseteq A$ is given by subset $|A_{sub}| \subseteq |A|$ closed under the operations:
  - for $f\colon s_1 \times \ldots \times s_n \to s$ and $a_1 \in |A_{sub}|_{s_1}, \ldots, a_n \in |A_{sub}|_{s_n}$,
  $$f_{A_{sub}}(a_1, \ldots, a_n) = f_A(a_1, \ldots, a_n)$$
- for $A \in \mathbf{Alg}(\Sigma)$ and $X \subseteq |A|$, the *subalgebra of A genereted by X*, $\langle A \rangle_X$, is the least subalgebra of A that contains X.
- $A \in \mathbf{Alg}(\Sigma)$ is *reachable* if $\langle A \rangle_\emptyset$ coincides with A.

Fact: *For any $A \in \mathbf{Alg}(\Sigma)$ and $X \subseteq |A|$, $\langle A \rangle_X$ exists.*

Boringly known . . .

Proof (idea):

- generate the generated subalgebra from X by closing it under operations in A; or
- the intersection of any family of subalgebras of A is a subalgebra of A.

Homomorphisms

Boringly known ...

- for $A, B \in \mathbf{Alg}(\Sigma)$, a Σ-*homomorphism* $h\colon A \to B$ is a function $h\colon |A| \to |B|$ that preserves the operations:
  - for $f\colon s_1 \times \ldots \times s_n \to s$ and $a_1 \in |A|_{s_1}, \ldots, a_n \in |A|_{s_n}$,
  $$h_s(f_A(a_1, \ldots, a_n)) = f_B(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$$

Fact: *Given a homomorphism $h : A \to B$ and subalgebras A_{sub} of A and B_{sub} of B, the image of A_{sub} under h, $h(A_{sub})$, is a subalgebra of B, and the coimage of B_{sub} under h, $h^{-1}(B_{sub})$, is a subalgebra of A.*

Fact: *Given a homomorphism $h : A \to B$ and $X \subseteq |A|$, $h(\langle A\rangle_X) = \langle B\rangle_{h(X)}$.*

Fact: *Identity function on the carrier of $A \in \mathbf{Alg}(\Sigma)$ is a homomorphism $id_A : A \to A$. Composition of homomorphisms $h : A \to B$ and $g : B \to C$ is a homomorphism $h;g : A \to C$.*

Boringly known ...

Isomorphisms

- for $A, B \in \mathbf{Alg}(\Sigma)$, a Σ-*isomorphism* is any Σ-homomorphism $i\colon A \to B$ that has an *inverse*, i.e., a Σ-homomorphism $i^{-1}\colon B \to A$ such that $i;i^{-1} = id_A$ and $i^{-1};i = id_B$.
- Σ-algebras are *isomorphic* if there exists an isomorphism between them.

Fact: *A Σ-homomorphism is a Σ-isomorphism iff it is bijective ("1-1" and "onto").*

Fact: *Identities are isomorphisms, and any composition of isomorphisms is an isomorphism.*

Boringly known ...

Congruences

- for $A \in \mathbf{Alg}(\Sigma)$, a Σ-*congruence on* A is an equivalence $\equiv \subseteq |A| \times |A|$ that is closed under the operations:
  - for $f\colon s_1 \times \ldots \times s_n \to s$ and $a_1, a'_1 \in |A|_{s_1}, \ldots, a_n, a'_n \in |A|_{s_n}$,
    if $a_1 \equiv_{s_1} a'_1, \ldots, a_n \equiv_{s_n} a'_n$ then $f_A(a_1, \ldots, a_n) \equiv_s f_A(a'_1, \ldots, a'_n)$.

Fact: *For any relation $R \subseteq |A| \times |A|$ on the carrier of a Σ-algebra A, there exists the least congruence on A that conatins R.*

Fact: *For any Σ-homomorphism $h : A \to B$, the kernel of h, $K(h) \subseteq |A| \times |A|$, where $a \; K(h) \; a'$ iff $h(a) = h(a')$, is a Σ-congruence on A.*

Quotients

Boringly known . . .

- for $A \in \mathbf{Alg}(\Sigma)$ and Σ-congruence $\equiv \subseteq |A| \times |A|$ on A, the *quotient algebra* $A/\!\equiv$ is built in the natural way on the equivalence classes of $\equiv$:
  - for $s \in S$, $|A/\!\equiv|_s = \{[a]_\equiv \mid a \in |A|_s\}$, with $[a]_\equiv = \{a' \in |A|_s \mid a \equiv a'\}$
  - for $f\colon s_1 \times \ldots \times s_n \to s$ and $a_1 \in |A|_{s_1}, \ldots, a_n \in |A|_{s_n}$,
  $$f_{A/\equiv}([a_1]_\equiv, \ldots, [a_n]_\equiv) = [f_A(a_1, \ldots, a_n)]_\equiv$$

Fact: *The above is well-defined; moreover, the natural map that assigns to every element its equivalence class is a Σ-homomorphisms $[\_]_\equiv : A \to A/\!\equiv$.*

Fact: *Given two Σ-congruences $\equiv$ and $\equiv'$ on A, $\equiv \subseteq \equiv'$ iff there exists a Σ-homomorphism $h : A/\!\equiv \to A/\!\equiv'$ such that $[\_]_\equiv;h = [\_]_{\equiv'}$.*

Fact: *For any Σ-homomorphism $h : A \to B$, $A/K(h)$ is isomorphic with $h(A)$.*

Products

Boringly known ...

- for $A_i \in \mathbf{Alg}(\Sigma)$, $i \in \mathcal{I}$, the *product of* $\langle A_i \rangle_{i \in \mathcal{I}}$, $\prod_{i \in \mathcal{I}} A_i$ is built in the natural way on the Cartesian product of the carriers of A_i, $i \in \mathcal{I}$:
  - for $s \in S$, $|\prod_{i \in \mathcal{I}} A_i|_s = \prod_{i \in \mathcal{I}} |A_i|_s$
  - for $f\colon s_1 \times \ldots \times s_n \to s$ and $a_1 \in |\prod_{i \in \mathcal{I}} A_i|_{s_1}, \ldots, a_n \in |\prod_{i \in \mathcal{I}} A_i|_{s_n}$, for $i \in \mathcal{I}$, $f_{\prod_{i \in \mathcal{I}} A_i}(a_1, \ldots, a_n)(i) = f_{A_i}(a_1(i), \ldots, a_n(i))$

Fact: *For any family $\langle A_i \rangle_{i \in \mathcal{I}}$ of Σ-algebras, projections $\pi_i(a) = a(i)$, where $i \in \mathcal{I}$ and $a \in \prod_{i \in \mathcal{I}} |A_i|$, are Σ-homomorphisms $\pi_i : \prod_{i \in \mathcal{I}} A_i \to A_i$.*

Define the product of the empty family of Σ-algebras.

When the projection π_i is an isomorphism?

Terms

Consider an S-sorted set X of variables, Σ-algebra A and valuation $v\colon X \to |A|$.

- *term* $t \in |T_\Sigma(X)|$: built using variables X, constants and operations from Ω in the usual way
- *term algebra* $T_\Sigma(X)$: with the set of terms as the carrier, and operations defined "syntactically"
- *term evaluation* $v^\#\colon T_\Sigma(X) \to A$: the unique homomorphism from $T_\Sigma(X)$ to A that extends v
- *term value* $t_A[v] = v^\#(t)$: may also be determined inductively

Terms

Boringly known . . .

Consider an S-sorted set X of variables.

- *terms* $t \in |T_\Sigma(X)|$ are built using variables X, constants and operations from Ω in the usual way: $|T_\Sigma(X)|$ is the least set such that
  - $X \subseteq |T_\Sigma(X)|$
  - for $f\colon s_1 \times \ldots \times s_n \to s$ and $t_1 \in |T_\Sigma(X)|_{s_1}, \ldots, t_n \in |T_\Sigma(X)|_{s_n}$, $f(t_1, \ldots, t_n) \in |T_\Sigma(X)|_s$
- for any Σ-algebra A and valuation $v : X \to |A|$, *the value* $t_A[v]$ *of a term* $t \in |T_\Sigma(X)|$ *in* A *under* v is determined inductively:
  - $x_A[v] = v_s(x)$, for $x \in X_s$, $s \in S$
  - $(f(t_1, \ldots, t_n))_A[v] = f_A((t_1)_A[v], \ldots, (t_n)_A[v])$, for $f\colon s_1 \times \ldots \times s_n \to s$ and $t_1 \in |T_\Sigma(X)|_{s_1}, \ldots, t_n \in |T_\Sigma(X)|_{s_n}$

Above and in the following: assuming unambiguous "parsing" of terms!

Term algebras

Boringly known . . .

Consider an S-sorted set X of variables.

- The *term algebra* $T_\Sigma(X)$ has the set of terms as the carrier and operations defined "syntactically":
  - for $f\colon s_1 \times \ldots \times s_n \to s$ and $t_1 \in |T_\Sigma(X)|_{s_1}, \ldots, t_n \in |T_\Sigma(X)|_{s_n}$, $f_{T_\Sigma(X)}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$.

Fact: *For any S-sorted set X of variables, Σ-algebra A and valuation $v : X \to |A|$, there is a unique Σ-homomorphism $v^\# : T_\Sigma(X) \to A$ that extends v. Moreover, for $t \in |T_\Sigma(X)|$, $v^\#(t) = t_A[v]$.*

$$\mathbf{Set}^S \qquad \begin{tikzcd} X \arrow[r, "id_{X \hookrightarrow |T_\Sigma(X)|}"] \arrow[dr, "v"'] & {|T_\Sigma(X)|} \arrow[d, "|v^\#|"] \\ & {|A|} \end{tikzcd} \qquad \begin{tikzcd} T_\Sigma(X) \arrow[d, "\exists!\ v^\#"] \\ A \end{tikzcd} \qquad \mathbf{Alg}(\Sigma)$$

Equations

- *Equation*:

$$\boxed{\forall X.t = t'}$$

where:

– X is a set of variables, and

– $t, t' \in |T_\Sigma(X)|_s$ are terms of a common sort.

- *Satisfaction relation*: Σ-algebra A *satisfies* $\forall X.t = t'$

$$\boxed{A \models \forall X.t = t'}$$

when for all $v\colon X \to |A|$, $t_A[v] = t'_A[v]$.

Semantic entailment

$$\boxed{\Phi \models_\Sigma \varphi}$$

Σ-equation φ is a semantic consequence of a set of Σ-equations Φ if φ holds in every Σ-algebra that satisfies Φ.

BTW:

- *Models* of a set of equations: $Mod[\Phi] = \{A \in \mathbf{Alg}(\Sigma) \mid A \models \Phi\}$
- *Theory* of a class of algebras: $Th[\mathcal{C}] = \{\varphi \mid \mathcal{C} \models \varphi\}$
- $\Phi \models \varphi \iff \varphi \in Th[Mod[\Phi]]$
- Mod and Th form a *Galois connection*

Equational calculus

$$\frac{}{\forall X.t = t} \qquad \frac{\forall X.t = t'}{\forall X.t' = t} \qquad \frac{\forall X.t = t' \quad \forall X.t' = t''}{\forall X.t = t''}$$

$$\frac{\forall X.t_1 = t'_1 \quad \ldots \quad \forall X.t_n = t'_n}{\forall X.f(t_1 \ldots t_n) = f(t'_1 \ldots t'_n)} \qquad \frac{\forall X.t = t'}{\forall Y.t[\theta] = t'[\theta]} \text{ for } \theta\colon X \to |T_\Sigma(Y)|$$

Mind the variables!

$a = b$ does *not* follow from $a = f(x)$ and $f(x) = b$, unless...

Proof-theoretic entailment

$$\boxed{\Phi \vdash_\Sigma \varphi}$$

Σ-equation φ is a proof-theoretic consequence of a set of Σ-equations Φ if φ can be derived from Φ by the rules.

How to justify this?

Semantics!

Soundness & completeness

Fact: *The equational calculus is sound and complete:*

$$\boxed{\Phi \models \varphi \iff \Phi \vdash \varphi}$$

- soundness: "all that can be proved, is true" $(\Phi \models \varphi \Longleftarrow \Phi \vdash \varphi)$
- completeness: "all that is true, can be proved" $(\Phi \models \varphi \Longrightarrow \Phi \vdash \varphi)$

Proof (idea):

- soundness: easy!

  Just check for each rule that if premises hold in an algebra then so does the conclusion.

- completeness: not so easy!

  But not too difficult either.

Proving completeness

$$\boxed{\Phi \models \varphi \Longrightarrow \Phi \vdash \varphi}$$

Proof (idea):

- Suppose $\boxed{\Phi \models \forall Y.t_1 = t_2}$
- Consider the term algebra $\boxed{T_\Sigma(Y)}$
- Define $\approx \subseteq |T_\Sigma(Y)| \times |T_\Sigma(Y)|$ by $\boxed{t \approx t' \iff \Phi \vdash \forall Y.t = t'}$
- Check that $\approx$ is a congruence on $T_\Sigma(Y)$; consider the quotient $\boxed{T_\Sigma(Y)/\approx}$
- For any $\theta : X \to |T_\Sigma(Y)|$, define $[\theta]_\approx : X \to |T_\Sigma(Y)/\approx|$ by $\boxed{[\theta]_\approx(x) = [\theta(x)]_\approx}$
- Check that for any $t \in |T_\Sigma(X)|$ and $\theta : X \to |T_\Sigma(Y)|$, $\boxed{t_{T_\Sigma(Y)/\approx}[[\theta]_\approx] = [t[\theta]]_\approx}$
- It follows that $T_\Sigma(Y)/\approx \models \Phi$, and so also $\boxed{T_\Sigma(Y)/\approx \models \forall Y.t_1 = t_2}$
- Conclude from this that $\boxed{t_1 \approx t_2}$ i.e. $\boxed{\Phi \vdash \forall Y.t_1 = t_2}$

Equational specifications

$$\boxed{\langle \Sigma, \Phi \rangle}$$

- signature Σ, to determine the static module interface
- axioms (Σ-equations), to determine required module properties

BUT:

Fact: *A class of Σ-algebras is equationally definable iff it is closed under subalgebras, products and homomorphic images.*

Equational specifications typically admit a lot of undesirable "modules"

Example

$$
\begin{aligned}
\textbf{spec } \textsc{NaiveNat} = \textbf{sort } & Nat \\
\textbf{ops } & 0 : Nat; \\
& \quad succ : Nat \to Nat; \\
& \quad \_+\_ : Nat \times Nat \to Nat \\
\textbf{axioms } & \forall n{:}Nat \bullet n + 0 = n; \\
& \quad \forall n, m{:}Nat \bullet n + succ(m) = succ(n + m)
\end{aligned}
$$

Now:

$$\textsc{NaiveNat} \not\models \forall n, m{:}Nat \bullet n + m = m + n$$

(Nor: $\textsc{NaiveNat} \not\vdash \forall n, m{:}Nat \bullet n + m = m + n$)

How to fix this

- Other (stronger) *logical systems*: conditional equations, first-order logic, higher-order logics, other bells-and-whistles
  – more about this soon. . .

Institutions!

- *Constraints*:
  – *reachability* (and generation): "no junk"
  – *initiality* (and freeness): "no junk" & "no confusion"

  Constraints can be thought of as special (higher-order) formulae.

There has been a population explosion among logical systems. . .

Initial models

Fact: *Every equational specification $\langle \Sigma, \Phi \rangle$ has an initial model: there exists a Σ-algebra $I \in Mod[\Phi]$ such that for every Σ-algebra $M \in Mod[\Phi]$ there exists a unique Σ-homomorphism from I to M.*

Proof (idea): I is the quotient of the algebra of ground Σ-terms by the congruence that glues together all ground terms t, t' such that $\Phi \models \forall\emptyset.t = t'$.

BTW: This can be generalised to the existence of a free model of $\langle \Sigma, \Phi \rangle$ over any (many-sorted) set of data.

BTW: One proof of completeness of equational logic uses the same construction.

Example

$$
\textbf{spec}\ \textsc{Nat} = \textbf{free}\ \{\ \textbf{sort}\ Nat
$$

$$
\textbf{ops}\ 0 : Nat;
$$

$$
succ : Nat \to Nat;
$$

$$
\_ + \_ : Nat \times Nat \to Nat
$$

$$
\textbf{axioms}\ \forall n{:}Nat \bullet n + 0 = n;
$$

$$
\forall n, m{:}Nat \bullet n + succ(m) = succ(n + m)
$$

$$
\}
$$

Now:

$$
\textsc{Nat} \models \forall n, m{:}Nat \bullet n + m = m + n
$$

Example′

$$\textbf{spec } \text{NAT}' = \textbf{free type } Nat ::= 0 \mid succ(Nat)$$
$$\textbf{op } \_ + \_ : Nat \times Nat \to Nat$$
$$\textbf{axioms } \forall n{:}Nat \bullet n + 0 = n;$$
$$\forall n, m{:}Nat \bullet n + succ(m) = succ(n + m)$$

$$\boxed{\text{NAT} \equiv \text{NAT}'}$$

Another example

spec $\textsc{String} =$

generated $\{$ **sort** $String$

ops $nil : String;$

$a, \ldots, z : String;$

$\_ \frown \_ : String \times String \to String \}$

axioms $\forall s{:}String \bullet s \frown nil = s;$

$\forall s{:}String \bullet nil \frown s = s;$

$\forall s, t, v{:}String \bullet s \frown (t \frown v) = (s \frown t) \frown v$

$\}$

Moving between signatures

Let $\Sigma = (S, \Omega)$ and $\Sigma' = (S', \Omega')$

$$\boxed{\sigma\colon \Sigma \to \Sigma'}$$

- *Signature morphism* maps:
  - sorts to sorts: $\sigma\colon S \to S'$
  - operation names to operation names, preserving their profiles: $\sigma\colon \Omega_{w,s} \to \Omega'_{\sigma(w),\sigma(s)}$, for $w \in S^*$, $s \in S$

Let $\sigma\colon \Sigma \to \Sigma'$

Translating syntax

- *translation of variables*: $X \mapsto X'$, where $X'_{s'} = \biguplus_{\sigma(s)=s'} X_s$
- *translation of terms*: $\sigma\colon |T_\Sigma(X)|_s \to |T_{\Sigma'}(X')|_{\sigma(s)}$, for $s \in S$
- *translation of equations*: $\sigma(\forall X.t_1 = t_2)$ yields $\forall X'.\sigma(t_1) = \sigma(t_2)$

. . . and semantics

- *σ-reduct*: $\_|_\sigma\colon \mathbf{Alg}(\Sigma') \to \mathbf{Alg}(\Sigma)$, where for $A' \in \mathbf{Alg}(\Sigma')$
  - $|A'|_\sigma|_s = |A'|_{\sigma(s)}$, for $s \in S$
  - $f_{A'|_\sigma} = \sigma(f)_{A'}$ for $f \in \Omega$

Note the contravariancy!

Satisfaction condition

Fact: *For all signature morphisms $\sigma : \Sigma \to \Sigma'$, Σ'-algebras A' and Σ-equations φ:*

$$\boxed{A'|_\sigma \models_\Sigma \varphi \iff A' \models_{\Sigma'} \sigma(\varphi)}$$

Proof (idea): for $t \in |T_\Sigma(X)|$ and $v : X \to |A'|_\sigma|$, $t_{A'|_\sigma}[v] = \sigma(t)_{A'}[v']$, where $v' : X' \to |A'|$ is given by $v'_{\sigma(s)}(x) = v_s(x)$ for $s \in S$, $x \in X_s$.

TRUTH is preserved (at least) under:

- *change of notation*
- *restriction/extension of irrelevant context*

Crash course II

Category theory

Categories and functors

- A *category* $\mathbf{K}$ consists of:
  - a "set" of *objects*: $|\mathbf{K}|$
  - sets of *morphisms*: $\mathbf{K}(A, B)$, for all $A, B \in |\mathbf{K}|$; $m\colon A \to B$ stands for $m \in \mathbf{K}(A, B)$
  - morphism *composition*: for $m\colon A \to B$ and $m'\colon B \to C$, we have $m;m'\colon A \to C$;

    the composition is associative and has identities.
- A *functor* $\mathbf{F}\colon \mathbf{K} \to \mathbf{K}'$ between two categories maps:
  - $\mathbf{K}$-objects to $\mathbf{K}'$-objects
  - $\mathbf{K}$-morphisms to $\mathbf{K}'$-morphisms, preserving their source and target, composition and identities

Sample categories and functors around

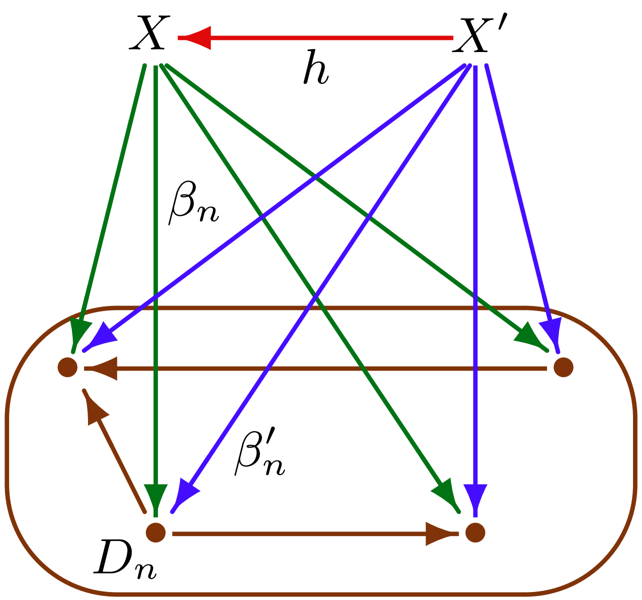
- sets and functions between them form the category $\mathbf{Set}$
- (sm)all categories and functors between them form the category $\mathbf{Cat}$
- Σ-algebras and their homomorphisms form the category $\mathbf{Alg}(\Sigma)$
- algebraic signatures and their morphisms form the category $\mathbf{AlgSig}$
- σ-reduct extends to the functor $\_|_\sigma \colon \mathbf{Alg}(\Sigma') \to \mathbf{Alg}(\Sigma)$
- $\mathbf{Alg} \colon \mathbf{AlgSig}^{op} \to \mathbf{Cat}$ is a (contravariant) functor mapping signature Σ to the category $\mathbf{Alg}(\Sigma)$ and signature morphism $\sigma \colon \Sigma \to \Sigma'$ to the reduct functor $\_|_\sigma \colon \mathbf{Alg}(\Sigma') \to \mathbf{Alg}(\Sigma)$
- $\mathbf{Eq} \colon \mathbf{AlgSig} \to \mathbf{Set}$ is a (covariant) functor mapping signature Σ to the set $\mathbf{Eq}(\Sigma)$ of all Σ-equations and signature morphism $\sigma \colon \Sigma \to \Sigma'$ to the translation function $\sigma \colon \mathbf{Eq}(\Sigma) \to \mathbf{Eq}(\Sigma')$

Diagrams, limits, colimits

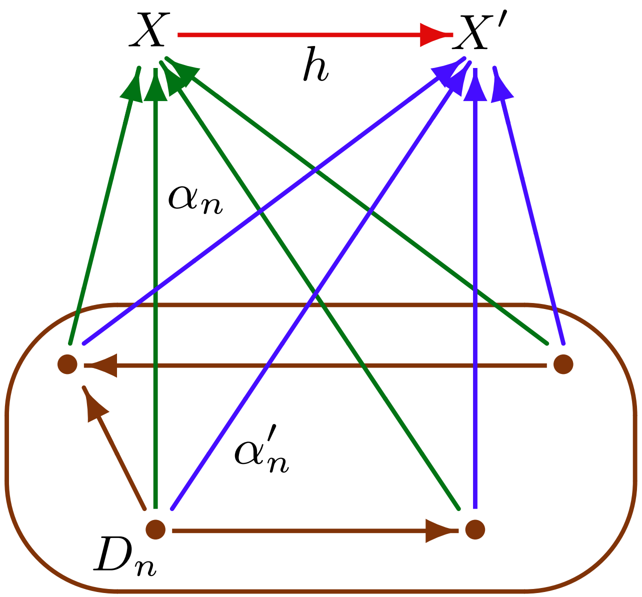
- *Diagram* in $\mathbf{K}$ is a functor $D\colon J \to \mathbf{K}$ from (small) *shape* category J
- *Cocone* $\alpha\colon D \to X$ on diagram D with *vertex* $X \in |\mathbf{K}|$: consists of a family of morphisms $\alpha_n\colon D(n) \to X$, one for each *node* $n \in |J|$, such that $\alpha_n = D(e);\alpha_m$ for each *edge* $e\colon n \to m$ in J
- *Cone* $\beta\colon X \to D$: ... a family of morphisms $\beta_n\colon X \to D(n)$ ... *dually*
- *Colimit* of D is a cocone $colimD\colon D \to |colimD|$ such that for every cocone $\alpha\colon D \to X$ there exists a *unique* $h\colon |colimD| \to X$ such that $(colimD)_n;h = \alpha_n$ for $n \in |J|$
- *Limit* of D is a cone $limD\colon |limD| \to D$ ... *dually*

Limits and colimits (when they exist) are defined up to isomorphism

Limits and colimits



A *limit* of D (in $\mathbf{K}$) is a cone $\langle \beta_n : X \to D_n \rangle_{n \in N}$ on D such that for all cones $\langle \beta'_n : X' \to D_n \rangle_{n \in N}$ on D, for a unique morphism $h : X' \to X$, $h;\beta_n = \beta'_n$ for all $n \in N$.



A *colimit* of D (in $\mathbf{K}$) is a cocone $\langle \alpha_n : D_n \to X \rangle_{n \in N}$ on D such that for all cocones $\langle \alpha'_n : D_n \to X' \rangle_{n \in N}$ on D, for a unique morphism $h : X \to X'$, $\alpha_n;h = \alpha'_n$ for all $n \in N$.

Some limits

diagram	limit	in $\mathbf{Set}$
(empty)	*terminal object*	$\{*\}$
$A \qquad B$	*product*	$A \times B$
$A \overset{f}{\underset{g}{\rightrightarrows}} B$	*equalizer*	$\{a \in A \mid f(a) = g(a)\} \hookrightarrow A$
$A \xrightarrow{f} C \xleftarrow{g} B$	*pullback*	$\{(a,b) \in A \times B \mid f(a) = g(b)\}$

Fact: *All finite limits may be built using terminal object and pullbacks; pullbacks may be built using products and equalizers.*

Give constructions of such limits in $\mathbf{Cat}$
Hint: This is easy!

Some colimits

diagram	colimit	in $\mathbf{Set}$
(empty)	*initial object*	$\emptyset$
$A \qquad B$	*coproduct*	$A \uplus B$
$A \overset{f}{\underset{g}{\rightrightarrows}} B$	*coequalizer*	$B \longrightarrow B/{\equiv}$ where $f(a) \equiv g(a)$ for all $a \in A$
$A \xleftarrow{f} C \xrightarrow{g} B$	*pushout*	$(A \uplus B)/{\equiv}$ where $f(c) \equiv g(c)$ for all $c \in C$

Fact: *All finite colimits may be built using initial object and pushouts; pushouts may be built using coproducts and coequalizers.*

> Give constructions of such colimits in $\mathbf{Cat}$
> Hint: This is not entirely easy!

Example of a pushout

$$
\begin{tikzcd}
\{0,1,2,3\} & & \{1,2,3\} \\
& \{a,b\} \arrow[ul, "{\{a \mapsto 1, b \mapsto 2\}}"] \arrow[ur, "{\{a \mapsto 1, b \mapsto 3\}}"'] &
\end{tikzcd}
$$

Diagrams list objects indicating how they share components

Example of a pushout

$$\begin{tikzcd}
& \{0,[1=1],[2=3],2,3\} & \\
\{0,1,2,3\} \arrow[ur, "{\{0\mapsto 0,1\mapsto[1=1],\ 2\mapsto[2=3],3\mapsto 3\}}"] & & \{1,2,3\} \arrow[ul, "{\{1\mapsto[1=1],\ 2\mapsto 2,3\mapsto[2,3]\}}"'] \\
& \{a,b\} \arrow[ul, "{\{a\mapsto 1,b\mapsto 2\}}"] \arrow[ur, "{\{a\mapsto 1,b\mapsto 3\}}"'] &
\end{tikzcd}$$

Diagrams list objects indicating how they share components

Colimits combine objects taking account of the indicated sharing

A sample pushout in $\mathbf{AlgSig}$

sort $String$
ops $a, \ldots, z : String;$
$\_\frown\_ : String \times String \to String$

sort $Elem$

sorts $Elem, Nat, Array[Elem]$
ops $empty : Array[Elem];$
$put : Nat \times Elem \times Array[Elem] \to Array[Elem];$
$get : Nat \times Array[Elem] \to Elem$

A sample pushout in $\mathbf{AlgSig}$

$$\begin{tikzcd}
\boxed{\begin{array}{l} \textbf{sort}\ \mathit{String} \\ \textbf{ops}\ \ a,\ldots,z : \mathit{String}; \\ _ \frown _ : \mathit{String} \times \mathit{String} \\ \hfill \to \mathit{String} \end{array}} \arrow[r] & \boxed{\begin{array}{l} \textbf{sorts}\ \mathit{String}, \mathit{Nat}, \mathit{Array}[\mathit{String}] \\ \textbf{ops}\ \ a,\ldots,z : \mathit{String}; \\ _ \frown _ : \mathit{String} \times \mathit{String} \to \mathit{String}; \\ \mathit{empty} : \mathit{Array}[\mathit{String}]; \\ \mathit{put} : \mathit{Nat} \times \mathit{String} \times \mathit{Array}[\mathit{String}] \\ \hfill \to \mathit{Array}[\mathit{String}]; \\ \mathit{get} : \mathit{Nat} \times \mathit{Array}[\mathit{String}] \to \mathit{String} \end{array}} \\
\boxed{\textbf{sort}\ \mathit{Elem}} \arrow[u] \arrow[r] & \boxed{\begin{array}{l} \textbf{sorts}\ \mathit{Elem}, \mathit{Nat}, \mathit{Array}[\mathit{Elem}] \\ \textbf{ops}\ \ \mathit{empty} : \mathit{Array}[\mathit{Elem}]; \\ \mathit{put} : \mathit{Nat} \times \mathit{Elem} \times \mathit{Array}[\mathit{Elem}] \\ \hfill \to \mathit{Array}[\mathit{Elem}]; \\ \mathit{get} : \mathit{Nat} \times \mathit{Array}[\mathit{Elem}] \to \mathit{Elem} \end{array}} \arrow[u]
\end{tikzcd}$$

At last...

Institutions

Sample categories and functors around

- sets and functions between them form the category $\mathbf{Set}$
- (sm)all categories and functors between them form the category $\mathbf{Cat}$
- Σ-algebras and their homomorphisms form the category $\mathbf{Alg}(\Sigma)$
- algebraic signatures and their morphisms form the category $\mathbf{AlgSig}$
- σ-reduct extends to the functor $\_|_\sigma \colon \mathbf{Alg}(\Sigma') \to \mathbf{Alg}(\Sigma)$
- $\mathbf{Alg} \colon \mathbf{AlgSig}^{op} \to \mathbf{Cat}$ is a (contravariant) functor mapping signature Σ to the category $\mathbf{Alg}(\Sigma)$ and signature morphism $\sigma \colon \Sigma \to \Sigma'$ to the reduct functor $\_|_\sigma \colon \mathbf{Alg}(\Sigma') \to \mathbf{Alg}(\Sigma)$
- $\mathbf{Eq} \colon \mathbf{AlgSig} \to \mathbf{Set}$ is a (covariant) functor mapping signature Σ to the set $\mathbf{Eq}(\Sigma)$ of all Σ-equations and signature morphism $\sigma \colon \Sigma \to \Sigma'$ to the translation function $\sigma \colon \mathbf{Eq}(\Sigma) \to \mathbf{Eq}(\Sigma')$

Generality and abstraction

There are many choices:

- Software systems: Non-termination allowed? Exceptions? Non-determinism? Higher-order functions? Concurrency? etc.
- Specifications: Logical language to capture basic required properties? Equational? First-order? Higher-order? Temporal formulae? LTL, CTL, CTL*?
- Proofs: Logical calculi for building proofs (of properties, of refinement steps, etc.)

Most of the theory is independent of most of these choices!

We try to make this explicit:

rely only on basic common features

Tuning up the logical system

- various sets of formulae (equations, Horn-clauses, first-order, higher-order, modal formulae, . . .)
- various notions of algebra (partial algebras, relational structures, error algebras, Kripke structures, . . .)
- various notions of signature (order-sorted, error, higher-order signatures, sets of propositional variables, . . .)
- (various notions of signature morphisms)

No best logic for everything

Solution:

Work with an arbitrary logical system

Goguen & Burstall: $1980 \to 1992$

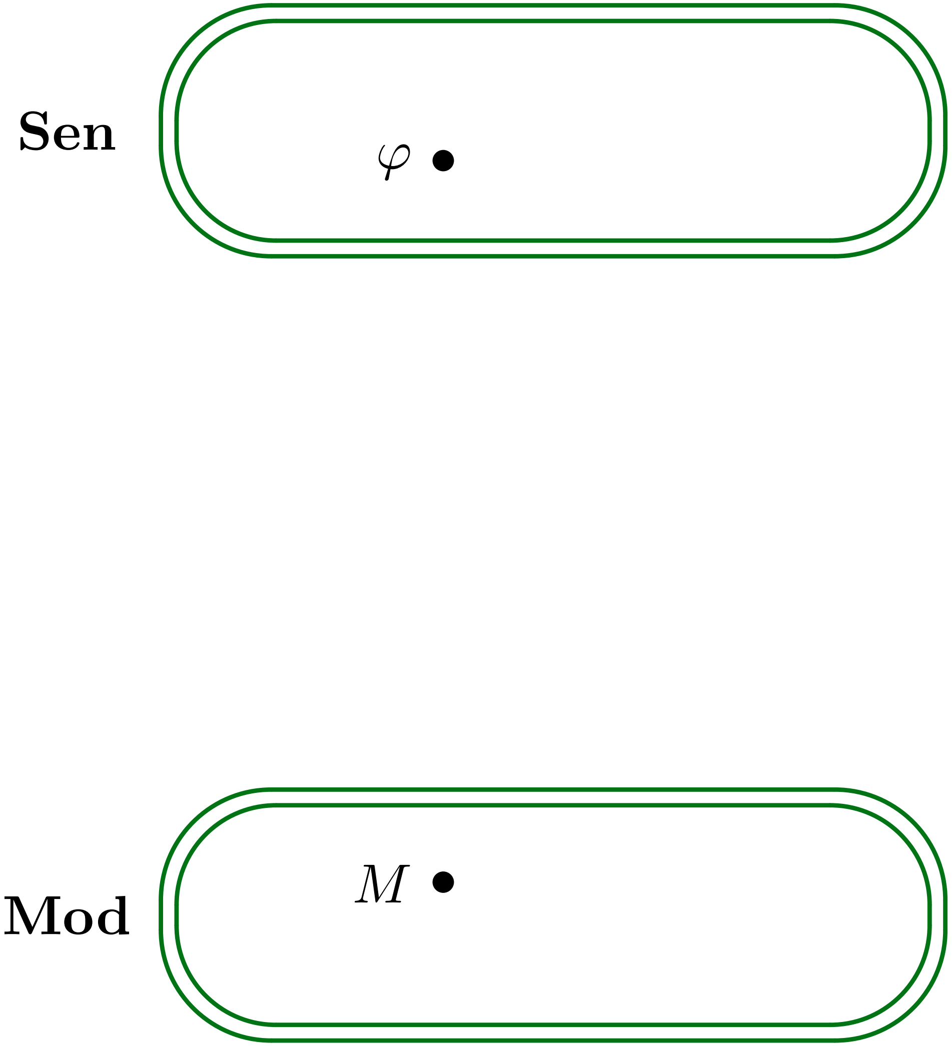
Main tool

Institutions

- a standard formalization of the concept of the underlying logical system for specification formalisms and most work on foundations of software specification and development from algebraic perspective;
- a formalization of the concept of a logical system for foundational studies:
  - truly abstract model theory
  - proof-theoretic considerations
  - heterogeneous logical environments

Abstract model theory
for specification and programming
(using the basics of category theory)

Institution: abstraction



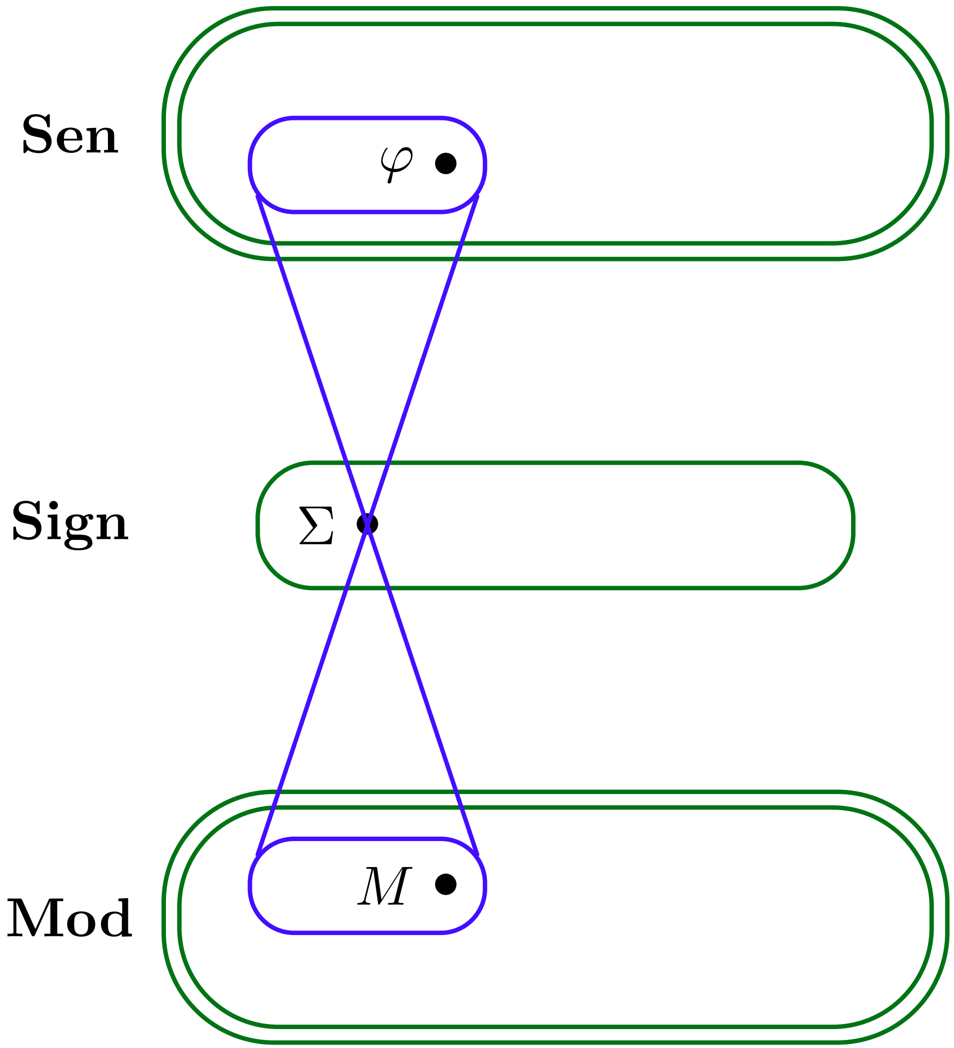
plus *satisfaction relation:*

$$\boxed{M \models \varphi}$$

and so the usual Galois connection between classes of models and sets of sentences, with the standard notions induced ($Mod[\Phi]$, $Th[\mathcal{M}]$, $Th[\Phi]$, $\Phi \models \varphi$, etc).

- Also, possibly adding (sound) consequence: $\Phi \vdash \varphi$ (implying $\Phi \models \varphi$) to deal with proof-theoretic aspects.

Institution: first insight



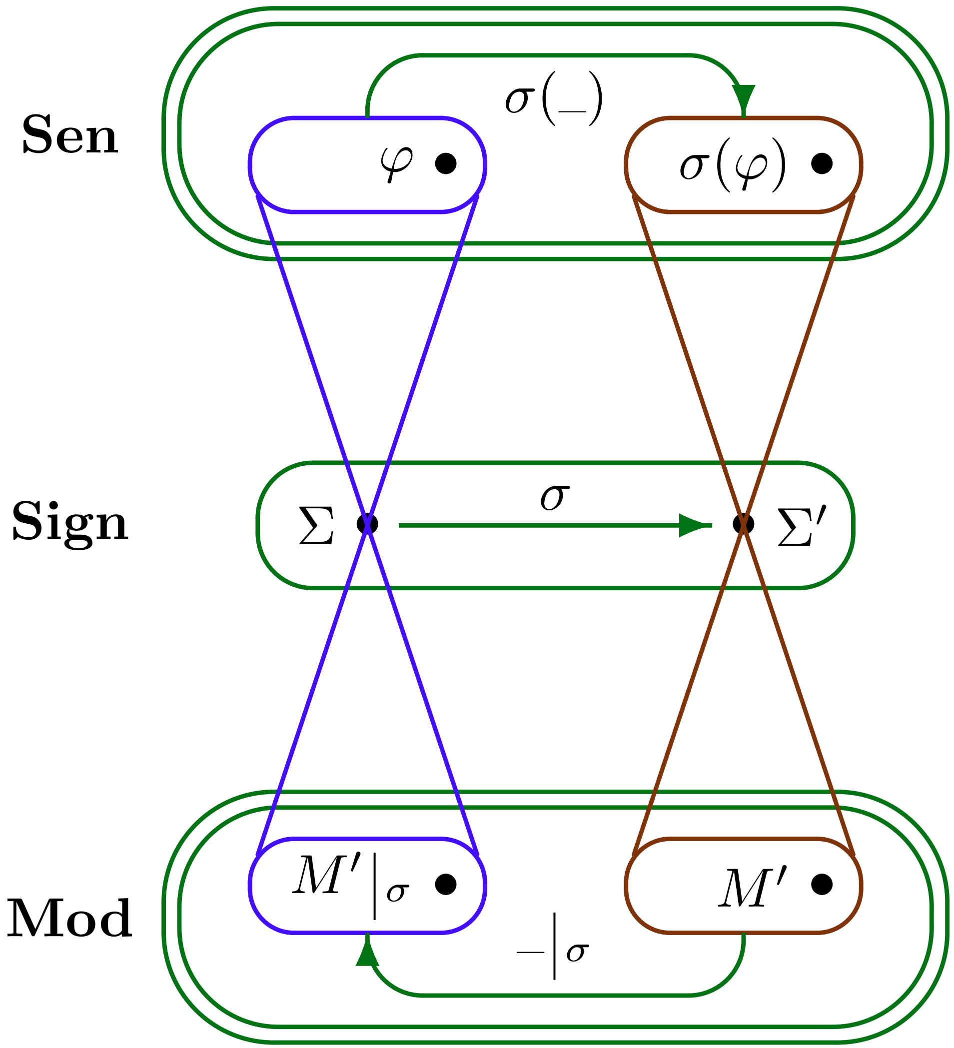
plus *satisfaction relation:*

$$\boxed{M \models_\Sigma \varphi}$$

and so, for each signature, the usual Galois connection between classes of models and sets of sentences, with the standard notions induced ($Mod_\Sigma[\Phi]$, $Th_\Sigma[\mathcal{M}]$, $Th_\Sigma[\Phi]$, $\Phi \models_\Sigma \varphi$, etc).

- Also, possibly adding (sound) consequence: $\Phi \vdash_\Sigma \varphi$ (implying $\Phi \models_\Sigma \varphi$) to deal with proof-theoretic aspects.

Institution: key insight



imposing the *satisfaction condition:*

$$\boxed{M' \models_{\Sigma'} \sigma(\varphi) \text{ iff } M'|_\sigma \models_\Sigma \varphi}$$

Truth is invariant
under change of notation

and independent of
any additional symbols around

Institution

- a category $\mathbf{Sign}$ of *signatures*
- a functor $\mathbf{Sen}\colon \mathbf{Sign} \to \mathbf{Set}$
  - $\mathbf{Sen}(\Sigma)$ is the set of Σ-*sentences*, for $\Sigma \in |\mathbf{Sign}|$
- a functor $\mathbf{Mod}\colon \mathbf{Sign}^{op} \to \mathbf{Cat}$
  - $\mathbf{Mod}(\Sigma)$ is the category of Σ-*models*, for $\Sigma \in |\mathbf{Sign}|$
- for each $\Sigma \in |\mathbf{Sign}|$, Σ-*satisfaction relation* $\models_\Sigma \subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$

subject to the *satisfaction condition*:

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

where $\sigma\colon \Sigma \to \Sigma'$ in $\mathbf{Sign}$, $M' \in |\mathbf{Mod}(\Sigma')|$, $\varphi \in \mathbf{Sen}(\Sigma)$,
$M'|_\sigma$ stands for $\mathbf{Mod}(\sigma)(M')$, and $\sigma(\varphi)$ for $\mathbf{Sen}(\sigma)(\varphi)$.

Typical institutions

- **EQ** — equational logic
- **FOEQ** — first-order logic (with predicates and equality)
- **PEQ**, **PFOEQ** — as above, but with partial operations
- **HOL** — higher-order logic
- logics of constraints (fitted via signature morphisms)
- **CASL** — the logic of CASL: partial first-order logic with equality, predicates, generation constraints, and subsorting

  CASL subsorting: the sets of sorts in signatures are *pre-ordered*; in every model M, $s \leq s'$ yields an injective *subsort embedding* (*coercion*) $em_M^{s\leq s'} : |M|_s \to |M|_{s'}$ such that $em_M^{s\leq s} = id_{|M|_s}$ for each sort s, and $em_M^{s\leq s'};em_M^{s'\leq s''} = em_M^{s\leq s''}$, for $s \leq s' \leq s''$; plus partial projections and subsort membership predicates derived from the embeddings.

Somewhat less typical institutions:

- modal logics
- three-valued logics
- programming language semantics:
  - **IMP**: imperative programming language with sets of computations as models and procedure declararions as sentences
  - **FPL**: functional programming language with partial algebras as models and the usual axioms with extended term syntax allowing for local recursive function definitions

Temporal logic

extremely simplified version
and oversimplified presentation

Institution $\mathbf{TL}$:

- signatures $\mathcal{A}$: (finite) sets of *actions*;
- models $\mathcal{R}$: sets of *runs*, finite or infinite sequences of (sets of) actions;
- sentences φ: built from atomic statements a (action $a \in \mathcal{A}$ happens) using the usual propositional and temporal connectives, including $\mathbf{X}\varphi$ (an action happens and then φ holds) and $\varphi\mathbf{U}\psi$ (φ holds until ψ holds)
- satisfaction $\mathcal{R} \models \varphi$: φ holds at the beginning of every run in $\mathcal{R}$

WATCH OUT!

Under some formalisations, satisfaction condition may fail!

Care is needed in the exact choice of sentences considered, morphisms (between sets of actions) allowed, and reduct definitions.

Perhaps unexpected examples:

- no sentences
- no models
- no signatures
- trivial satisfaction relations

- sets of sentences as sentences
- sets of sentences as signatures
- classes of models as sentences
- sets of sentences as models
- ...

typical, not so typical and perhaps unexpected examples abound

WORK IN AN ARBITRARY INSTITUTION

...adding extra structure and assumptions only if really needed ...

Let's fix an institution $\mathbf{I} = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|})$ for a while.

Semantic entailment

$$\boxed{\Phi \models_\Sigma \varphi}$$

Σ-*sentence* φ *is a semantic consequence of a set of* Σ-*sentences* Φ *if* φ *holds in every* Σ-*models that satisfies* Φ.

BTW:

- *Models* of a set of sentences: $Mod[\Phi] = \{M \in |\mathbf{Mod}(\Sigma)| \mid M \models \Phi\}$
- *Theory* of a class of models: $Th[\mathcal{C}] = \{\varphi \mid \mathcal{C} \models \varphi\}$
- $\Phi \models \varphi \iff \varphi \in Th[Mod[\Phi]]$
- Mod and Th form a *Galois connection*

Semantic equivalences

Equivalence of sentences: for $\Sigma \in |\mathbf{Sign}|$, $\varphi, \psi \in \mathbf{Sen}(\Sigma)$ and $\mathcal{M} \subseteq |\mathbf{Mod}(\Sigma)|$,

$$\boxed{\varphi \equiv_{\mathcal{M}} \psi}$$

if for all Σ-models $M \in \mathcal{M}$, $M \models \varphi$ iff $M \models \psi$. For $\varphi \equiv_{|\mathbf{Mod}(\Sigma)|} \psi$ we write:

$$\boxed{\varphi \equiv \psi}$$

Semantic equivalence

Equivalence of models: for $\Sigma \in |\mathbf{Sign}|$, $M, N \in |\mathbf{Mod}(\Sigma)|$, and $\Phi \subseteq \mathbf{Sen}(\Sigma)$,

$$\boxed{M \equiv_{\Phi} N}$$

if for all $\varphi \in \Phi$, $M \models \varphi$ iff $N \models \varphi$. For $M \equiv_{\mathbf{Sen}(\Sigma)} N$ we write:

$$\boxed{M \equiv N}$$

Elementary equivalence

Compactness, consistency, completeness...

- Institution $\mathbf{I}$ is *compact* if for each signature $\Sigma \in |\mathbf{Sign}|$, set of Σ-sentences $\Phi \subseteq \mathbf{Sen}(\Sigma)$, and Σ-sentences $\varphi \in \mathbf{Sen}(\Sigma)$,

$$\boxed{\text{if } \Phi \models \varphi \text{ then } \Phi_{\mathit{fin}} \models \varphi \text{ for some finite } \Phi_{\mathit{fin}} \subseteq \Phi}$$

- A set of Σ-sentences $\Phi \subseteq \mathbf{Sen}(\Sigma)$ is *consistent* if it has a model, i.e.,

$$\boxed{\mathit{Mod}[\Phi] \neq \emptyset}$$

- A set of Σ-sentences $\Phi \subseteq \mathbf{Sen}(\Sigma)$ is *complete* if it is a maximal consistent set of Σ-sentences, i.e., Φ is consistent and

$$\boxed{\text{for } \Phi \subseteq \Phi' \subseteq \mathbf{Sen}(\Sigma), \text{ if } \Phi' \text{ is consistent then } \Phi = \Phi'}$$

Fact: *Any complete set of Σ-sentences $\Phi \subseteq \mathbf{Sen}(\Sigma)$ is a theory:* $\Phi = \mathit{Th}[\mathit{Mod}[\Phi]]$.

Preservation of entailment

Fact:

$$\boxed{\Phi \models_\Sigma \varphi \implies \sigma(\Phi) \models_{\Sigma'} \sigma(\varphi)}$$

for $\sigma\colon \Sigma \to \Sigma'$, $\Phi \subseteq \mathbf{Sen}(\Sigma)$, $\varphi \in \mathbf{Sen}(\Sigma)$.

If the reduct $\_|_\sigma\colon |\mathbf{Mod}(\Sigma')| \to |\mathbf{Mod}(\Sigma)|$ *is surjective, then*

$$\boxed{\Phi \models_\Sigma \varphi \iff \sigma(\Phi) \models_{\Sigma'} \sigma(\varphi)}$$

Adding provability

Add to institution:

- *proof-theoretic entailment*:

$$\vdash_\Sigma \subseteq \mathcal{P}(\mathbf{Sen}(\Sigma)) \times \mathbf{Sen}(\Sigma)$$

  for each signature $\Sigma \in |\mathbf{Sign}|$, closed under

  – weakening, reflexivity, transitivity (cut)

  – translation along signature morphisms

Require:

- *soundness*: $\Phi \vdash_\Sigma \varphi \implies \Phi \models_\Sigma \varphi$

(?) *completeness*: $\Phi \models_\Sigma \varphi \implies \Phi \vdash_\Sigma \varphi$

Presentations
(basic specifications)

$$\boxed{\langle \Sigma, \Phi \rangle}$$

- signature Σ, to determine the static module interface
- axioms (Σ-sentences) $\Phi \subseteq \mathbf{Sen}(\Sigma)$, to determine required module properties

Use strong enough logic to capture the "right" class of models,
excluding undesirable "modules"

Presentation morphisms

Presentation morphism:

$$\boxed{\sigma\colon \langle \Sigma, \Phi \rangle \to \langle \Sigma', \Phi' \rangle}$$

is a signature morphism $\sigma\colon \Sigma \to \Sigma'$ such that for all $M' \in \mathbf{Mod}(\Sigma')$:

$$\boxed{M' \in Mod[\Phi'] \implies M'|_\sigma \in Mod[\Phi]}$$

Then $\_|_\sigma \colon Mod[\Phi'] \to Mod[\Phi]$

Fact: *A signature morphism* $\sigma\colon \Sigma \to \Sigma'$ *is a presentation morphism* $\sigma\colon \langle \Sigma, \Phi \rangle \to \langle \Sigma', \Phi' \rangle$ *if and only if* $\Phi' \models \sigma(\Phi)$ *.*

BTW: for all presentation morphisms $\Phi \models_\Sigma \varphi \implies \Phi' \models_{\Sigma'} \sigma(\varphi)$

Conservativity

A presentation morphism:

$$\boxed{\sigma\colon \langle\Sigma,\Phi\rangle \to \langle\Sigma',\Phi'\rangle}$$

is *conservative* if for all Σ-sentences φ: $\boxed{\Phi' \models_{\Sigma'} \sigma(\varphi) \implies \Phi \models_\Sigma \varphi}$

A presentation morphism $\sigma\colon \langle\Sigma,\Phi\rangle \to \langle\Sigma',\Phi'\rangle$ *admits model expansion* if for each $M \in Mod[\Phi]$ there exists $M' \in Mod[\Phi']$ such that $M'|_\sigma = M$

(i.e., $\_|_\sigma\colon Mod[\Phi'] \to Mod[\Phi]$ is surjective).

Fact: *If $\sigma\colon \langle\Sigma,\Phi\rangle \to \langle\Sigma',\Phi'\rangle$ admits model expansion then it is conservative.*

In general, the equivalence does not hold!

Fact: *If $\langle\Sigma,\Phi\rangle$ is complete and $\langle\Sigma',\Phi'\rangle$ is consistent then any presentation morphism $\sigma : \langle\Sigma,\Phi\rangle \to \langle\Sigma',\Phi'\rangle$ is conservative.*

Categories of presentations & of theories

- $\mathbf{Pres}$: the *category of presentations* in $\mathbf{I}$ has presentations as objects and presentation morphisms as morphisms, with identities and composition inherited from $\mathbf{Sign}$, the category of signatures.
- $\mathbf{TH}$: the *category of theories* in $\mathbf{I}$ is the full subcateogry of $\mathbf{Pres}$ with theories (presentations with sets of sentences closed under consequence) as objects.

$\mathbf{Pres}$ and $\mathbf{TH}$ are equivalent: $id_\Sigma \colon \langle \Sigma, \Phi \rangle \to \langle \Sigma, Th[Mod[\Phi]] \rangle$ is an isomorphism in $\mathbf{Pres}$

Fact: *The forgetful functors from* $\mathbf{Pres}$ *and* $\mathbf{TH}$*, respectively, to* $\mathbf{Sign}$ *preserve and create colimits.*

Fact: *If the category* $\mathbf{Sign}$ *of signatures is cocomplete, so are the categories* $\mathbf{Pres}$ *of presentations and* $\mathbf{TH}$ *of theories.*

Proof hint

in **Sign**:

$$\begin{tikzcd} & \Sigma' & \\ \Sigma_1 \arrow[ur, "\sigma_2'"] & & \Sigma_2 \arrow[ul, "\sigma_1'"'] \\ & \Sigma \arrow[ul, "\sigma_1"] \arrow[ur, "\sigma_2"'] & \end{tikzcd}$$

in **Pres**:

$$\begin{tikzcd} & \langle \Sigma', \sigma_2'(\Phi_1) \cup \sigma_1'(\Phi_2) \rangle & \\ \langle \Sigma_1, \Phi_1 \rangle \arrow[ur, "\sigma_2'"] & & \langle \Sigma_2, \Phi_2 \rangle \arrow[ul, "\sigma_1'"'] \\ & \langle \Sigma, \Phi \rangle \arrow[ul, "\sigma_1"] \arrow[ur, "\sigma_2"'] & \end{tikzcd}$$

Logical connectives

- $\mathbf{I}$ *has negation* if for every signature $\Sigma \in |\mathbf{Sign}|$ and Σ-sentence $\varphi \in \mathbf{Sen}(\Sigma)$, there is a Σ-sentence "$\neg\varphi$" $\in \mathbf{Sen}(\Sigma)$ such that for all Σ-models $M \in |\mathbf{Mod}(\Sigma)|$, $M \models$ "$\neg\varphi$" iff $M \not\models \varphi$.
- $\mathbf{I}$ *has conjunction* if for every signature $\Sigma \in |\mathbf{Sign}|$ and Σ-sentences $\varphi, \psi \in \mathbf{Sen}(\Sigma)$, there is a Σ-sentence "$\varphi \wedge \psi$" $\in \mathbf{Sen}(\Sigma)$ such that for all Σ-models $M \in |\mathbf{Mod}(\Sigma)|$, $M \models$ "$\varphi \wedge \psi$" iff $M \models \varphi$ and $M \models \psi$.
- ... *implication*, *disjunction*, *falsity*, *truth* ...

Fact: *For any signature morphism $\sigma : \Sigma \to \Sigma'$ and Σ-sentence $\varphi \in \mathbf{Sen}(\Sigma)$ $\sigma($"$\neg\varphi$"$)$ and "$\neg\sigma(\varphi)$" are equivalent.*

Similarly, for Σ-sentences $\varphi, \psi \in \mathbf{Sen}(\Sigma)$), $\sigma($"$\varphi \wedge \psi$"$)$ and "$\sigma(\varphi) \wedge \sigma(\psi)$" are equivalent.

Similarly for other connectives...

For any institution $\mathbf{I}$, define its closures: under negation $\mathbf{I}^{\neg}$, under conjunction $\mathbf{I}^{\wedge}$, etc.

... as well as under all boolean connectives $\mathbf{I}^{bool}$

Some "institutional" topics

- **Institutions: intuitions and motivations**
  Goguen & Burstall $\sim 1980 \to 1992$
- **Very abstract model theory**
  Tarlecki ~ 1986, Diaconescu *et al* $\sim 2003 \to \ldots$
- **Structured specifications**
  CLEAR ~ 1980, Sannella & Tarlecki $\sim 1984 \to \ldots$, CASL ~ 2004
- **Moving between institutions**
  Goguen & Burstall $\sim 1983 \to 1992$, Tarlecki $\sim 1986, 1996$, Goguen & Rosu ~ 2002
- **Heterogeneous specifications**
  Sannella & Tarlecki ~ 1988, Tarlecki $\sim 2000 \to \ldots$, Diaconescu $\sim 2002 \to \ldots$,
  Mossakowski $\sim 2002 \to \ldots$ (HETS)

...apologies for missing some names and for inaccurate years...

Institutional (Abstract) Model Theory

An institution $\mathbf{I} = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|})$ remains fixed for another while.

Abstract abstract model theory

Providing new insights and abstract formulations for classical model-theoretic concepts and results

- amalgamation over pushouts
- the method of elementary diagrams
- existence of free extensions
- Birkhoff variety theorem(s)
- interpolation results
- Beth definability theorem
- logical connectives, free variables, quantification
- completeness for *any* first-order logic
- ...

in *any* institution
with various bits of extra structure,
under some technical assumptions...

Interpolation

Classical Craig interpolation

In first-order logic:

Fact: *Any sentences* $\varphi_1 \in \mathbf{Sen}(\Sigma_1)$ *and* $\varphi_2 \in \mathbf{Sen}(\Sigma_2)$ *such that* $\varphi_1 \models_{\Sigma_1 \cup \Sigma_2} \varphi_2$, *have an interpolant* $\theta \in \mathbf{Sen}(\Sigma_1 \cap \Sigma_2)$ *such that* $\varphi_1 \models_{\Sigma_1} \theta$ *and* $\theta \models_{\Sigma_2} \varphi_2$.

$$
\begin{tikzcd}
& \boxed{\varphi_1 \models \varphi_2} & \\
& \Sigma_1 \cup \Sigma_2 & \\
\boxed{\varphi_1 \models \theta} \;\; \Sigma_1 \arrow[ur] & & \Sigma_2 \;\; \boxed{\theta \models \varphi_2} \arrow[ul] \\
& \Sigma_1 \cap \Sigma_2 \arrow[ul] \arrow[ur] & \\
& \boxed{\theta} &
\end{tikzcd}
$$

In general though:

intersection-union squares

are not enough!

Example

$$
\begin{tikzcd}
\boxed{\begin{array}{l}
\textbf{sort } String \\
\textbf{ops } a, \ldots, z : String; \\
\_\,\widehat{}\,\_ : String \times String \\
\qquad\qquad \to String
\end{array}}
\arrow[r]
&
\boxed{\begin{array}{l}
\textbf{sorts } String, Nat, Array[String] \\
\textbf{ops } a, \ldots, z : String; \\
\_\,\widehat{}\,\_ : String \times String \to String; \\
empty : Array[String]; \\
put : Nat \times String \times Array[String] \\
\qquad\qquad \to Array[String]; \\
get : Nat \times Array[String] \to String
\end{array}}
\\
\boxed{\textbf{sort } Elem}
\arrow[u]
\arrow[r]
\arrow[ur, phantom, "\mathsf{PO}"]
&
\boxed{\begin{array}{l}
\textbf{sorts } Elem, Nat, Array[Elem] \\
\textbf{ops } empty : Array[Elem]; \\
put : Nat \times Elem \times Array[Elem] \\
\qquad\qquad \to Array[Elem]; \\
get : Nat \times Array[Elem] \to Elem
\end{array}}
\arrow[u]
\end{tikzcd}
$$

Craig interpolation, take #1

In $\mathbf{I}$, *interpolation property* holds for a signature pushout below, if any sentences $\varphi_1 \in \mathbf{Sen}(\Sigma_1)$ and $\varphi_2 \in \mathbf{Sen}(\Sigma_2)$ such that $\sigma_2'(\varphi_1) \models_{\Sigma'} \sigma_1'(\varphi_2)$, have an *interpolant* $\theta \in \mathbf{Sen}(\Sigma)$ such that $\varphi_1 \models_{\Sigma_1} \sigma_1(\theta)$ and $\sigma_2(\theta) \models_{\Sigma_2} \varphi_2$.

$$
\begin{tikzcd}
 & & \boxed{\sigma_2'(\varphi_1) \models \sigma_1'(\varphi_2)} & & \\
 & & \Sigma' & & \\
\boxed{\varphi_1 \models \sigma_1(\theta)} & \Sigma_1 \arrow[ur, "\sigma_2'"] & \text{PO} & \Sigma_2 \arrow[ul, "\sigma_1'"'] & \boxed{\sigma_2(\theta) \models \varphi_2} \\
 & & \Sigma \arrow[ul, "\sigma_1"] \arrow[ur, "\sigma_2"'] & & \\
 & & \boxed{\theta} & &
\end{tikzcd}
$$

In general though:

single sentences

are not enough!

Institutional Craig interpolation

In $\mathbf{I}$, *Craig interpolation property* holds for a pushout in $\mathbf{Sign}$

$$
\begin{tikzcd}
 & \Sigma' & \\
\Sigma_1 \arrow[ur, "\sigma'_2"] & \text{PO} & \Sigma_2 \arrow[ul, "\sigma'_1"'] \\
 & \Sigma \arrow[ul, "\sigma_1"] \arrow[ur, "\sigma_2"'] &
\end{tikzcd}
$$

if for all $\Phi_1 \subseteq \mathbf{Sen}(\Sigma_1)$ and $\varphi_2 \in \mathbf{Sen}(\Sigma_2)$ such that $\sigma'_2(\Phi_1) \models_{\Sigma'} \sigma'_1(\varphi_2)$ there is $\Theta \subseteq \mathbf{Sen}(\Sigma)$ such that $\Phi_1 \models_{\Sigma_1} \sigma_1(\Theta)$ and $\sigma_2(\Theta) \models_{\Sigma_2} \varphi_2$.

Fact: *Many-sorted first-order logic has the interpolation property for the pushout as above provided that at least one of the two morphisms σ_1, σ_2 is injective on sorts.*

Fact: *Many-sorted equational logic has the interpolation property for the pushout as above provided that all sorts are non-void and σ_2 is injective.*

Institutional Craig-Robinson interpolation

In $\mathbf{I}$, *Craig-Robinson interpolation property* holds for a pushout in $\mathbf{Sign}$ if for all $\Phi_1 \subseteq \mathbf{Sen}(\Sigma_1)$, $\Gamma_2 \subseteq \mathbf{Sen}(\Sigma_2)$ and $\varphi_2 \in \mathbf{Sen}(\Sigma_2)$ such that $\sigma_2'(\Phi_1) \cup \sigma_1'(\Gamma_2) \models_{\Sigma'} \sigma_1'(\varphi_2)$ there is $\Theta \subseteq \mathbf{Sen}(\Sigma)$ such that $\Phi_1 \models_{\Sigma_1} \sigma_1(\Theta)$ and $\sigma_2(\Theta) \cup \Gamma_2 \models_{\Sigma_2} \varphi_2$.

$$
\begin{tikzcd}
 & & \boxed{\sigma_2'(\Phi_1) \cup \sigma_1'(\Gamma_2) \models \sigma_1'(\varphi_2)} & & \\
 & & \Sigma' & & \\
\boxed{\Phi_1 \models \sigma_1(\Theta)} & \Sigma_1 \arrow[ur, "\sigma_2'"] & \mathsf{PO} & \Sigma_2 \arrow[ul, "\sigma_1'"'] & \boxed{\sigma_2(\Theta) \cup \Gamma_2 \models \varphi_2} \\
 & & \Sigma \arrow[ul, "\sigma_1"] \arrow[ur, "\sigma_2"'] & & \\
 & & \boxed{\Theta} & &
\end{tikzcd}
$$

Fact: *Craig-Robinson interpolation implies Craig interpolation.*

Fact: *In compact institutions with conjunction and implication, Craig interpolation and Craig-Robinson interpolation are equivalent.*

BTW:

Consistency theorem

$\mathbf{I}$ has the *consistency property* for a pushout in $\mathbf{Sign}$

$$\begin{tikzcd}[column sep=small] & \Sigma' & \\ \Sigma_1 \arrow[ur, "\sigma_2'"] & & \Sigma_2 \arrow[ul, "\sigma_1'"'] \\ & \Sigma \arrow[ul, "\sigma_1"] \arrow[ur, "\sigma_2"'] & \end{tikzcd} \qquad \begin{tikzcd}[column sep=small] & \langle \Sigma', \sigma_2'(\Phi_1) \cup \sigma_1'(\Phi_2) \rangle & \\ \langle \Sigma_1, \Phi_1 \rangle \arrow[ur, "\sigma_2'"] & & \langle \Sigma_2, \Phi_2 \rangle \arrow[ul, "\sigma_1'"'] \\ & \langle \Sigma, \Phi \rangle \arrow[ul, "\sigma_1"] \arrow[ur, "\sigma_2"'] & \end{tikzcd}$$

Robinson consistency theorem
(for first-order logic)

if for all sets of sentences $\Phi \subseteq \mathbf{Sen}(\Sigma)$, $\Phi_1 \subseteq \mathbf{Sen}(\Sigma_1)$ and $\Phi_2 \subseteq \mathbf{Sen}(\Sigma_2)$ and presentation morphisms $\sigma_1 : \langle \Sigma, \Phi \rangle \to \langle \Sigma_1, \Phi_1 \rangle$ and $\sigma_2 : \langle \Sigma, \Phi \rangle \to \langle \Sigma_2, \Phi_2 \rangle$ such that Φ_1 and Φ_2 are consistent and σ_1 is conservative, $\langle \Sigma', \sigma_2'(\Phi_1) \cup \sigma_1'(\Phi_2) \rangle$ is consistent.

Fact: *In any compact institution with falsity, negation and conjunction, Craig interpolation, Craig-Robinson interpolation and Robinson consistency properties are equivalent.*

Amalgamation for algebras taken for granted

$$
\begin{tikzcd}
 & & A' \arrow[ddll] \arrow[ddrr] & & \\
 & & \Sigma_1 \cup \Sigma_2 & & \\
A_1 = A'|_{\Sigma_1} \arrow[ddrr] & \Sigma_1 \arrow[ur] & & \Sigma_2 \arrow[ul] & A'|_{\Sigma_2} = A_2 \arrow[ddll] \\
 & & \Sigma_1 \cap \Sigma_2 \arrow[ul] \arrow[ur] & & \\
 & & A_1|_{\Sigma_1 \cap \Sigma_2} = A_2|_{\Sigma_1 \cap \Sigma_2} & &
\end{tikzcd}
$$

Fact: *For any algebras $A_1 \in |\mathbf{Alg}(\Sigma_1)|$ and $A_2 \in |\mathbf{Alg}(\Sigma_2)|$ with common interpretation of common symbols $A_1|_{\Sigma_1 \cap \Sigma_2} = A_2|_{\Sigma_1 \cap \Sigma_2}$, there is a unique "union" of A_1 and A_2, $A' \in |\mathbf{Alg}(\Sigma_1 \cup \Sigma_2)|$ with $A'|_{\Sigma_1} = A_1$ and $A'|_{\Sigma_2} = A_2$.*

Amalgamation

$$
\begin{tikzcd}
 & & M' \arrow[ddll] \arrow[ddrr] & & \\
 & & \Sigma' & & \\
M_1 = M'|_{\sigma_2'} \arrow[ddrr] & \Sigma_1 \arrow[ur, "\sigma_2'"] & \text{PO} & \Sigma_2 \arrow[ul, "\sigma_1'"'] & M'|_{\sigma_1'} = M_2 \arrow[ddll] \\
 & & \Sigma \arrow[ul, "\sigma_1"] \arrow[ur, "\sigma_2"'] & & \\
 & & M_1|_{\sigma_1} = M_2|_{\sigma_2} & &
\end{tikzcd}
$$

May be sensibly stated for any commuting square of morphisms

In $\mathbf{I}$, *amalgamation property* holds for the pushout above if for all $M_1 \in |\mathbf{Mod}(\Sigma_1)|$ and $M_2 \in |\mathbf{Mod}(\Sigma_2)|$ with $M_1|_{\sigma_1} = M_2|_{\sigma_2}$, there is a unique $M' \in |\mathbf{Mod}(\Sigma')|$ with $M'|_{\sigma_1'} = M_2$ and $M'|_{\sigma_2'} = M_1$.

Fact: *Many-sorted first-order and equational logics admit amalgamation.*

Adding amalgamation

Assume:

- the model functor $\mathbf{Mod}\colon \mathbf{Sign}^{op} \to \mathbf{Cat}$ is *continuous* (maps colimits of signatures to limits of model categories)

Fact: $\mathbf{Alg}\colon \mathbf{AlgSig}^{op} \to \mathbf{Cat}$ *is continuous.*

Amalgamation property: *Amalgamation property follows for a pushout in* $\mathbf{Sign}$ *if* $\mathbf{Mod}$ *maps it to a pullback in* $\mathbf{Cat}$*:*

$$\begin{tikzcd} \Sigma_1 \arrow[r, "\sigma_2'"] \arrow[dr, phantom, "\text{PO}"] & \Sigma' \\ \Sigma \arrow[u, "\sigma_1"] \arrow[r, "\sigma_2"'] & \Sigma_2 \arrow[u, "\sigma_1'"'] \end{tikzcd} \quad \xmapsto{\mathbf{Mod}} \quad \begin{tikzcd} \mathbf{Mod}(\Sigma_1) \arrow[d, "-|_{\sigma_1}"'] \arrow[dr, phantom, "\text{PB}"] & \mathbf{Mod}(\Sigma') \arrow[l, "-|_{\sigma_2'}"'] \arrow[d, "-|_{\sigma_1'}"] \\ \mathbf{Mod}(\Sigma) & \mathbf{Mod}(\Sigma_2) \arrow[l, "-|_{\sigma_2}"] \end{tikzcd}$$

Birkhoff-style results

Fact: *In (many-sorted) equational logic, for any class of Σ-algebras $\mathcal{A} \subseteq |\mathbf{Alg}(\Sigma)|$, $Mod[Th[\mathcal{A}]] = \mathcal{HSP}(\mathcal{A})$.*

General scheme:

$\mathbf{I}$ is a *Birkhoff institution* with $\mathcal{F}$ and $\mathcal{B}$, if for any signature $\Sigma \in |\mathbf{Sign}|$ and class of Σ-models $\mathcal{M} \subseteq |\mathbf{Mod}(\Sigma)|$

$$\boxed{Mod[Th[\mathcal{M}]] = \mathcal{B}_\Sigma(\mathcal{F}(\mathcal{M}))}$$

where:

- $\mathcal{F}$ is a family of filters with $\{\{*\}\} \in \mathcal{F}$, all model categories have F-filtered products for all $F \in \mathcal{F}$; then $\mathcal{F}(\mathcal{M})$ is the class of all F-filtered products of models in $\mathcal{M}$, for all $F \in \mathcal{F}$, and
- $\mathcal{B} = \langle \mathcal{B}_\Sigma \subseteq |\mathbf{Mod}(\Sigma)| \times |\mathbf{Mod}(\Sigma)| \rangle_{\Sigma \in |\mathbf{Sign}|}$ is a family of relations closed under isomorphism; then $\mathcal{B}_\Sigma(\mathcal{F}(\mathcal{M}))$ is the image of $\mathcal{F}(\mathcal{M})$ under relation $\mathcal{B}_\Sigma$.

Diaconescu $\sim$2004

Interpolation from axiomatisability

Fact: *Let $\mathbf{I}$ be a Birkhoff institution with $\mathcal{F}$ and $\mathcal{B}$. Consider a pushout in $\mathbf{Sign}$, for which $\mathbf{I}$ admits (weak) amalgamation, and such that reducts w.r.t. σ_1 and σ_2 preserve $\mathcal{F}$-filtered products.*
Then for this pushout $\mathbf{I}$ has

$$\begin{tikzcd} & \Sigma' & \\ \Sigma_1 \arrow[ur, "\sigma'_2"] & \mathit{PO} & \Sigma_2 \arrow[ul, "\sigma'_1"'] \\ & \Sigma \arrow[ul, "\sigma_1"] \arrow[ur, "\sigma_2"'] & \end{tikzcd}$$

- *Craig interpolation property if the reduct w.r.t. σ_2 lifts $\mathcal{B}^{-1}$,*

As in Rodenburg's proof for equational interpolation

- *Craig-Robinson interpolation property if the reduct w.r.t. σ_1 lifts $\mathcal{B}$.*

Quite a few examples, both known and new

Free variables

Standard algebra	Institution $\mathbf{I}$
algebraic signature $\Sigma = \langle S, \Omega \rangle$	signature $\Sigma \in \lvert\mathbf{Sign}\rvert$
S-sorted set of variables X	signature extension $\iota : \Sigma \to \Sigma(X)$ ($\Sigma(X)$ expands Σ by variables X as constants)
open Σ-formula φ with variables X	$\Sigma(X)$-sentence φ
Σ-algebra M	Σ-model $M \in \lvert\mathbf{Mod}(\Sigma)\rvert$
valuation of variables $v : X \to \lvert M\rvert$ in M	ι-expansion M^v of M, i.e., $M^v \in \lvert\mathbf{Mod}(\Sigma(X)\rvert)$, $M^v\rvert_\iota = M$ ($x_{M^v}{=}v(x)$ for each variable/constant $x \in X$)
satisfaction of formula φ in M under v: $M \models^v_\Sigma \varphi$	satisfaction of "open formula" φ $M^v \models_{\Sigma(X)} \varphi$

Quantification

Let $\mathcal{I}$ be a class of signature morphisms. For decency, assume that it forms a subcategory of $\mathbf{Sign}$ and is closed under pushouts with arbitrary signature morphisms.

- $\mathbf{I}$ *has universal quantification along* $\mathcal{I}$ if for every signature morphism $\iota : \Sigma \to \Sigma'$ in $\mathcal{I}$ and Σ'-sentence $\psi \in \mathbf{Sen}(\Sigma')$, there is a Σ-sentence "$\forall\iota.\psi$" $\in \mathbf{Sen}(\Sigma)$ such that for all Σ-models $M \in |\mathbf{Mod}(\Sigma)|$, $M \models$ "$\forall\iota.\psi$" iff for all Σ'-models with $M'|_\iota = M$, $M' \in |\mathbf{Mod}(\Sigma')|$, $M' \models \psi$.
- $\mathbf{I}$ *has existential quantification along* $\mathcal{I}$ if for $\iota : \Sigma \to \Sigma'$ in $\mathcal{I}$ and Σ'-sentence $\psi \in \mathbf{Sen}(\Sigma')$, there is a Σ-sentence "$\exists\iota.\psi$" $\in \mathbf{Sen}(\Sigma)$ such that for all Σ-models $M \in |\mathbf{Mod}(\Sigma)|$, $M \models$ "$\exists\iota.\psi$" iff for some Σ'-model $M' \in |\mathbf{Mod}(\Sigma')|$ with $M'|_\iota = M$, $M' \models \psi$.

Fact: *For any* $\sigma : \Sigma \to \Sigma_1$, $\sigma($"$\forall\iota.\psi$"$)$ *and* "$\forall\iota'.\sigma'(\psi)$" *are equivalent, where the following is a pushout in* $\mathbf{Sign}$ *with* $\iota' \in \mathcal{I}$:

$$\begin{tikzcd} \Sigma' \arrow[r, "\sigma'"] \arrow[dr, phantom, "PO"] & \Sigma_1' \\ \Sigma \arrow[u, "\iota"] \arrow[r, "\sigma"'] & \Sigma_1 \arrow[u, "\iota'"'] \end{tikzcd}$$

Similarly for existential quantification.

AMALGAMATION NEEDED

Define $\mathbf{I}^{FO}$, *"first-order closure" of* $\mathbf{I}$

The method of diagrams

Institution $\mathbf{I}$	Standard algebra
Given a signature Σ and Σ-model M,	(algebraic signature Σ and Σ-algebra M)
build signature extension $\iota : \Sigma \to \Sigma(M)$	(adding elements of $\lvert M \rvert$ as constants)
and a $\Sigma(M)$-presentation $\mathcal{E}_M$	(all ground atoms true in M^{id_M}, the natural ι-expansion of M)
so that the reduct by ι yields isomorphism $\mathbf{Mod}_{\Sigma(M)}[\mathcal{E}_M] \to (M/\mathbf{Mod}(\Sigma))$	(then the reduct by ι yields isomorphism $\mathbf{Alg}_{\Sigma(M)}[\mathcal{E}_M] \to (M/\mathbf{Alg}(\Sigma))$)
... and everything is natural ...	(everything is natural)
Now: M has a "canonical" ι-expansion which is initial in $\mathbf{Mod}_{\Sigma(M)}[\mathcal{E}_M]$	(M^{id_M}, reachable ι-expansion of M, is initial in $\mathbf{Alg}_{\Sigma(M)}[\mathcal{E}_M]$)

Equipped with the method of diagrams, one can do a lot!

Institutional very abstract model theory

Providing new insights and abstract formulations
for classical model-theoretic concepts and results

- amalgamation over pushouts
- the method of elementary diagrams
- existence of free extensions
- Birkhoff variety theorem(s)
- interpolation results
- Beth definability theorem
- logical connectives, free variables, quantification
- completeness for *any* first-order logic
- ...

in *any* institution
with various bits of extra structure,
under some technical assumptions...

Foundations of Software Specification and Development

Keeping an institution $\mathbf{I} = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|})$ fixed for yet another while.

Revised rough analogy

module interface	$\rightsquigarrow$	$\mathbf{I}$-signature
module	$\rightsquigarrow$	$\mathbf{I}$-model
module specification	$\rightsquigarrow$	class of $\mathbf{I}$-models

Structured specifications and their consequences

Example:
(double/linear) hash table

spec $\textsc{Nat} = \ldots$

spec $\textsc{String} = \ldots$

spec $\textsc{Elem} =$ **sort** Elem

spec $\textsc{StringKey} = \textsc{String}$ **and** $\textsc{Nat}$
  then op $\mathit{hash} : \mathit{String} \to \mathit{Nat}$

spec $\textsc{StringKey0} = \textsc{StringKey}$ **with** $\mathit{hash} \mapsto \mathit{hash0}$

spec $\textsc{Array\_of\_Elem} = \textsc{Elem}$ **and** $\textsc{Nat}$

then sort $Array[Elem]$

ops $empty : Array[Elem];$

$put : Nat \times Elem \times Array[Elem] \to Array[Elem];$

$get : Nat \times Array[Elem] \to Elem$

pred $used : Nat \times Array[Elem]$

$\forall\ i, j : Nat; e : Elem; a : Array[Elem]$

- $\neg used(i, empty)$
- $used(i, put(i, e, a))$
- $i \neq j \implies (\ used(i, put(j, e, a)) \iff used(i, a)\ )$
- $get(i, put(i, e, a)) = e$
- $i \neq j \implies get(i, put(j, e, a)) = get(i, a)$

Parametrized specification

spec $\textsc{Array}[\textsc{Elem}] = \textsc{Array\_of\_Elem}$

$\textsc{Array}[SP$ **fit** $Elem \mapsto Asort]$

stands for

$\{\textsc{Array}[\textsc{Elem}]$ **with** $Elem \mapsto Asort\}$ **and** SP

spec $\textsc{Bucket} = \textsc{Array}[\textsc{String}$ **fit** $Elem \mapsto String]$
with $Array[String] \mapsto Bucket$

spec $\textsc{Table} = \textsc{Array}[\textsc{Bucket}$ **fit** $Elem \mapsto Bucket]$
with $Array[Bucket] \mapsto Table$

$\textsc{StringHashTable0} = \textsc{StringKey0}$ **and** $\textsc{Bucket}$

then ops $add : String \times Bucket \to Bucket;$

$putnear : Nat \times String \times Bucket \to Bucket$

preds $present : String \times Bucket$

$isnear : Nat \times String \times Bucket$

$\forall\, i : Nat; s : String; b : Bucket$

- $add(s, b) = putnear(hash0(s), s, b)$
- $\neg used(i, b) \implies putnear(i, s, b) = put(i, s, b)$
- $used(i, b) \wedge get(i, b) = s \implies putnear(i, s, b) = b$
- $used(i, b) \wedge get(i, b) \neq s \implies$
  $putnear(i, s, b) = putnear(succ(i), s, b)$
- $present(s, b) \iff isnear(hash0(s), s, b)$
- $\neg used(i, b) \implies \neg isnear(i, s, b)$
- $used(i, b) \wedge get(i, b) = s \implies isnear(i, s, b)$
- $used(i, b) \wedge get(i, b) \neq s \implies (\cdots)$

$$
\begin{array}{l}
\textsc{StringHashTable} = \\
\quad \textsc{StringHashTable0}\ \textbf{and}\ \textsc{StringKey}\ \textbf{and}\ \textsc{Table} \\
\qquad \textbf{then op}\ add : String \times Table \to Table \\
\qquad\qquad \textbf{pred}\ present : String \times Table \\
\qquad\qquad \forall\, i : Nat; s : String; t : Table \\
\qquad\qquad\quad \bullet\ hash(s) = i \wedge used(i,t) \implies \\
\qquad\qquad\qquad\qquad add(s,t) = put(i, add(s, get(i,t)), t) \\
\qquad\qquad\quad \bullet\ hash(s) = i \wedge \neg used(i,t) \implies \\
\qquad\qquad\qquad\qquad add(s,t) = put(i, add(s, empty), t) \\
\qquad\qquad\quad \bullet\ hash(s) = i \wedge used(i,t) \implies \\
\qquad\qquad\qquad\qquad (present(s,t) \iff present(s, get(i,t))) \\
\qquad\qquad\quad \bullet\ hash(s) = i \wedge \neg used(i,t) \implies \neg present(s,t)
\end{array}
$$

spec $\textsc{UserStringHashTable} =$

$\quad \textsc{StringHashTable}$

$\qquad$ **reveal** $String, nil, a, \ldots, z, \_\hat{}\_, Table, empty : Table,$

$\qquad\qquad add : String \times Table \to Table, present : String \times Table$

Specification structure

- This is a (nicely) structured specification
- The specification structure can guide, for instance, proof search
- The specification structure does not prescribe the structure of programs that implement the specification

spec $\textsc{SimpleUserStringHashTable} = \textsc{String}$

then sort $Table$

ops $empty : Table;$

$add : String \times Table \rightarrow Table$

pred $present : String \times Table$

$\forall\ s, s' : String, t : Table$

- $\neg present(s, empty)$
- $present(s, add(s, t))$
- $s \neq s' \implies (present(s, put(s', t)) \iff present(s, t))$

Specifications

$$\boxed{SP \in \mathit{Spec}}$$

Adopting the model-theoretic view of specifications

The meaning of any specification $SP \in \mathit{Spec}$ built over $\mathbf{I}$ is given by:

- its *signature* $\mathit{Sig}[SP] \in |\mathbf{Sign}|$, and
- a class of its *models* $\mathit{Mod}[SP] \subseteq |\mathbf{Mod}(\mathit{Sig}[SP])|$.

This yields the usual notions:

- *semantic equivalence*: $SP \equiv SP'$ iff $\mathit{Sig}[SP]{=}\mathit{Sig}[SP']$ and $\mathit{Mod}[SP]{=}\mathit{Mod}[SP']$;
- *semantic consequence*: $SP \models \varphi$ iff $\mathit{Mod}[SP] \models \varphi$;
- *theory of a specification*: $\mathit{Th}[SP] = \{\varphi \mid SP \models \varphi\}$; etc

Standard structured specifications

Basic specification: $\boxed{\langle \Sigma, \Phi \rangle}$ — for $\Sigma \in |\mathbf{Sign}|$ and $\Phi \subseteq \mathbf{Sen}(\Sigma)$:

$Sig[\langle \Sigma, \Phi \rangle] = \Sigma$

$Mod[\langle \Sigma, \Phi \rangle] = Mod[\Phi]$

captures basic properties

Union: $\boxed{SP_1 \cup SP_2}$ — for SP_1 and SP_2 with $Sig[SP_1] = Sig[SP_2]$:

$Sig[SP_1 \cup SP_2] = Sig[SP_1]$

$Mod[SP_1 \cup SP_2] = Mod[SP_1] \cap Mod[SP_2]$

combines the constraints imposed

Translation: $\boxed{\sigma(SP)}$ — for any SP and $\sigma\colon Sig[SP] \to \Sigma'$:

$Sig[\sigma(SP)] = \Sigma'$

$Mod[\sigma(SP)] = \{M' \in |\mathbf{Mod}(\Sigma')| \mid M'|_\sigma \in Mod[SP]\}$

renames and introduces new components

Hiding: $\boxed{SP'|_\sigma}$ — for any SP' and $\sigma\colon \Sigma \to Sig[SP']$:

$Sig[SP'|_\sigma] = \Sigma$

$Mod[SP'|_\sigma] = \{M'|_\sigma \mid M' \in Mod[SP']\}$

hides auxiliary components

Normal forms

Fact: *Any specification built out of basic specifications using union and translation only is equivalent to a basic specification.*

Fact: *If the category of signatures has pushouts and the institution admits amalgamation, then any specification SP built out of basic specifications using union, translation and hiding may be equivalently transformed to its normal form:*

$$\boxed{\mathbf{nf}(SP) = \langle \Sigma_{all}, \Phi_{all} \rangle \big|_{\sigma_{res}}}$$

such that

$$\boxed{SP \equiv \mathbf{nf}(SP)}$$

Proof: by induction on the structure of SP.

Know about them — use them for meta-results — never use them for applications

Proving semantic consequence

$$\frac{\mathbf{nf}(SP) = \langle \Sigma_{all}, \Phi_{all} \rangle|_{\sigma_{res}} \qquad \Phi_{all} \models_{\Sigma_{all}} \sigma_{res}(\varphi)}{SP \vdash \varphi}$$

This is sound and complete for semantic consequence when the category of signatures has pushouts, the institution admits amalgamation (then the normal forms as above can be constructed), but:

This is a bad way!

- lack of compositionality
- no use of the specification structure
- typically, Φ_{all} is HUGE
- no help in proof search

Standard compositional proof system

$$\frac{\varphi \in \Phi}{\langle \Sigma, \Phi \rangle \vdash \varphi} \qquad \frac{SP_1 \vdash \varphi}{SP_1 \cup SP_2 \vdash \varphi} \qquad \frac{SP_2 \vdash \varphi}{SP_1 \cup SP_2 \vdash \varphi}$$

$$\frac{SP \vdash \varphi}{\sigma(SP) \vdash \sigma(\varphi)} \qquad \frac{SP' \vdash \sigma(\varphi)}{SP'|_{\sigma} \vdash \varphi}$$

Plus a *structural rule*:

$$\frac{\text{for } i \in J, SP \vdash \varphi_i \qquad \{\varphi_i\}_{i \in J} \models \varphi}{SP \vdash \varphi}$$

Soundness & completeness

$$\boxed{SP \vdash \varphi \implies SP \models \varphi}$$

Fact: *If the category of signatures has pushouts, the institution admits (weak) amalgamation and Craig-Robinson interpolation then*

$$\boxed{SP \vdash \varphi \iff SP \models \varphi}$$

Proof (idea):

- soundness: easy! Check for each rule that if premises hold so does the conclusion.
- completeness: not so easy! By induction on the structures of specification: for each specification-building operation, assume completeness of consequences for its arguments, and use their normal forms to show that the premises of the rule needed to prove the consequence for the result specification hold.

Can we do better?

In fact: given the other assumptions on the institution, Craig-Robinson interpolation is a necessarry condition for completeness of the above standard proof system.

In general: there is *no* sound and complete *compositional* proof system for semantic consequence for structured specifications *because:*

Claim: *The best sound and compositional proof system one can have is given above.*

Really ?

The only better proof systems are incidental

Fact: *The standard proof system is at least as strong as any other sound, compositional, non-absent-minded and theory-oriented proof system for consequences of structured specifications built from basic specifications using union, translation and hiding.*

Fact: *The standard proof system is at least as strong as any other sound, monotone, non-absent-minded proof system for consequences of structured specifications built from basic specifications using union, translation and hiding.*

Fact: *The standard proof system is at least as strong as any other persistently sound and compositional proof system for consequences of structured specifications built from basic specifications using union, translation and hiding.*

These also hold for proof systems based on a sound *entailment* for $\mathbf{I}$

The extra assumptions cannot be dropped

Program development

Verification

Given specification SP *and program* P*, prove that* $[\![P]\!] \in Mod[SP]$

BUT:

Proofs of software correctness are notoriously difficult

SO:

Build software together with a proof of its correctness

Programmer's task

Informally:

Given a requirements specification
produce a module that correctly implements it

Semantically:

Given a requirements specification SP
build a model $M \in |\mathbf{Mod}(Sig[SP])|$ such that
$M \in Mod[SP]$

Program development

May be easy:

Casl specification

spec NAT =
  free type $Nat ::= 0 \mid succ(Nat)$
  op $\_+\_ : Nat \times Nat \rightarrow Nat$
  axioms $\forall n{:}Nat \bullet n + 0 = n;$
    $\forall n, m{:}Nat \bullet n + succ(m) = succ(n + m)$

Standard ML code

```
structure NAT =
    struct
        datatype Nat = 0 | succ of Nat
        fun add(n,0) = n
          | add(n,succ(m)) = succ(add(n,m))
    end
```

. . . but this is rare

Key idea

$$\boxed{SP \rightsquigarrow M}$$

Never in a single jump!

Rather: proceed step by step, adding gradually more and more detail and incorporating more and more design and implementation decisions, until a specification is obtained that is easy to implement directly

$$\boxed{SP_0 \leadsto SP_1 \leadsto \cdots \leadsto SP_n}$$

Refinement step

$$\boxed{SP' \rightsquigarrow SP}$$

Means:

$$Sig[SP'] = Sig[SP] \text{ and } Mod[SP] \subseteq Mod[SP']$$

So:

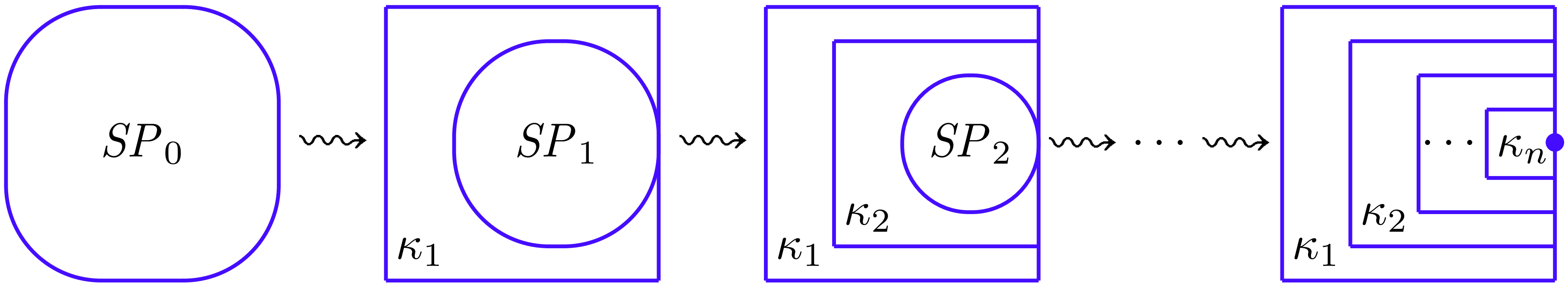
- preserve the static interface (preserving the signature)
- incorporate further details (narrowing the class of models)

Fact:

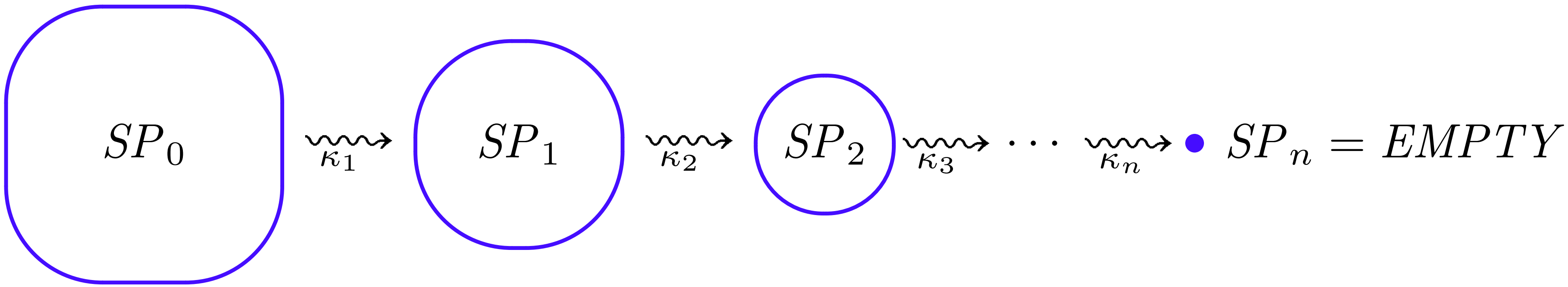
$$\frac{SP_0 \rightsquigarrow SP_1 \rightsquigarrow \cdots \rightsquigarrow SP_n \qquad M \in Mod[SP_n]}{M \in Mod[SP_0]}$$

Satisfaction guaranteed!

In practice, some parts will get fixed:



Keep them apart from whatever is really left for implementation:



Constructor refinement step

$$\boxed{SP' \underset{\kappa}{\rightsquigarrow} SP}$$

Means:

$$\kappa(Mod[SP]) \subseteq Mod[SP']$$

where

$$\kappa\colon |\mathbf{Mod}(Sig[SP])| \to |\mathbf{Mod}(Sig[SP'])|$$

is a *constructor*:

Intuitively: *parametrised module* (*functor* of STANDARD ML)

Semantically: function between model classes

Trivial example

CASL specifications

spec $\textsc{NatAdd} = \textsc{Nat}$ **with** $\{+ \mapsto add\}$

spec $\textsc{NatAddMult} = \textsc{NatAdd}$ **then**

op $mult : Nat \times Nat \to Nat$

axiom $\forall n{:}Nat \bullet mult(n, succ(0)) = n;$

$\forall n, m{:}Nat \bullet mult(n, m) = mult(m, n)$

```
functor MULT ( X: NAT_ADD_SIG ): NAT_ADD_MULT_SIG =
  struct open X
         fun mult(n,0) = 0 | mult(n,succ(m)) = n + mult(n,m)
  end
```

STANDARD ML code

$$\textsc{NatAddMult} \underset{\texttt{MULT}}{\rightsquigarrow} \textsc{NatAdd}$$

Development process

Fact:

$$\frac{SP_0 \underset{\kappa_1}{\rightsquigarrow} SP_1 \underset{\kappa_2}{\rightsquigarrow} \cdots \underset{\kappa_n}{\rightsquigarrow} SP_n = \mathit{EMPTY}}{\kappa_1(\kappa_2(\ldots \kappa_n(\mathit{empty})\ldots)) \in \mathit{Mod}[SP_0]}$$

Methodological issues:

- *top-down* vs. *bottom-up* vs. *middle-out* development?
- *modular decomposition* (designing modular structure)

Satisfaction guaranteed!

Branching refinement steps

$$SP \rightsquigarrow \boxed{BR \begin{cases} SP_1 \\ \vdots \\ SP_n \end{cases}}$$

Branching step BR involves a "*linking procedure*" (n-argument *constructor*)

$$\kappa_{BR} \colon |\mathbf{Mod}(Sig[SP_1])| \times \cdots \times |\mathbf{Mod}(Sig[SP_n])| \to |\mathbf{Mod}(Sig[SP])|$$

and we require

$$\frac{M_1 \in Mod[SP_1] \quad \cdots \quad M_n \in Mod[SP_n]}{\kappa_{BR}(M_1, \ldots, M_n) \in Mod[SP]}$$

Further development proceeds for each SP_i *separately*

Architectural specifications

CASL provides an explicit way to write down the organisational specification such a branching amounts to:

$$\textbf{arch spec } BR = \boxed{\begin{array}{ll} \textbf{units} & U_1 : SP_1 \\ & \ldots \\ & U_n : SP_n \\ \textbf{result} & \kappa_{BR}(U_1, \ldots, U_n) \end{array}}$$

Moreover:

- units my be generic (parametrised modules, STANDARD ML functors), but *always* are declared with their specifications
- CASL provides a rich collection of combinators to define κ_{BR} and various additional ways to *define* units

$$
\begin{array}{l}
\textbf{arch spec } \textsc{StringHashTableDesign} = \\
\quad \textbf{units } N : \textsc{Nat}; \\
\qquad\quad S : \textsc{String}; \\
\qquad\quad SK : \textsc{StringKey} \textbf{ given } S, N; \\
\qquad\quad SK0 : \textsc{StringKey0} \textbf{ given } S, N; \\
\qquad\quad A : \textsc{Elem} \rightarrow \textsc{Array\_of\_Elem} \textbf{ given } N; \\
\qquad\quad A0 : \textsc{Elem} \rightarrow \textsc{Array\_of\_Elem} \textbf{ given } N; \\
\qquad\quad B = A0[S \textbf{ fit } Elem \mapsto String] \textbf{ with } Array[String] \mapsto Bucket; \\
\qquad\quad T = A[B \textbf{ fit } Elem \mapsto Bucket] \textbf{ with } Array[Bucket] \mapsto Table; \\
\qquad\quad HT0 : \textsc{StringHashTable0} \textbf{ given } SK0, B; \\
\qquad\quad HT : \textsc{StringHashTable} \textbf{ given } HT0, SK, T \\
\quad \textbf{result } HT \textbf{ reveal } String, nil, a, \ldots, z, \_\widehat{\ }\_, Table, empty, add, present
\end{array}
$$

Further development

IF

$$SP \rightsquigarrow \boxed{\begin{array}{l} \textbf{units}\ \ U_1 : SP_1 \quad \ldots \quad U_n : SP_n \\ \textbf{result}\ \ \kappa(U_1, \ldots, U_n) \end{array}}$$

$$SP_1 \underset{\kappa_1}{\rightsquigarrow} SP'_1 \qquad \cdots \qquad SP_n \underset{\kappa_n}{\rightsquigarrow} SP'_n$$

THEN

$$SP \rightsquigarrow \boxed{\begin{array}{l} \textbf{units}\ \ U'_1 : SP'_1 \quad \ldots \quad U'_n : SP'_n \\ \textbf{result}\ \ \kappa(\kappa_1(U'_1), \ldots, \kappa_n(U'_n)) \end{array}}$$

Better still, keep the development tree within architectural specifications, as proposed for CASL

Local constructions / parametrized units

Local construction:

$$F\colon |\mathbf{Mod}(\Sigma)| \to |\mathbf{Mod}(\Sigma')|$$

Assume *persistency*:

$$\iota\colon \Sigma \to \Sigma' \text{ and } F(M)\big|_\iota = M, \text{ for all } M \in |\mathbf{Mod}(\Sigma)|$$

Local constructions are meant to be applied in a *global context* Σ_G via a *fitting morphism* $\gamma\colon \Sigma \to \Sigma_G$

From local to global constructions

$$
\begin{tikzcd}
\mathcal{G} \arrow[r, mapsto, "F_G"] & F_G(\mathcal{G}) \\
\Sigma_G \arrow[r, "\iota'"] & \Sigma'_G \\
\Sigma \arrow[u, "\gamma"] \arrow[r, "\iota"'] \arrow[ur, phantom, "\mathrm{PO}"] & \Sigma' \arrow[u, "\gamma'"'] \\
\mathcal{G}\big|_{\gamma} \arrow[r, mapsto, "F"'] & F(\mathcal{G}\big|_{\gamma})
\end{tikzcd}
$$

Given $F\colon |\mathbf{Mod}(\Sigma)| \to |\mathbf{Mod}(\Sigma')|$ persistent along $\iota\colon \Sigma \to \Sigma'$ and fitting morphism $\gamma\colon \Sigma \to \Sigma_G$ we obtain

$$F_G\colon |\mathbf{Mod}(\Sigma_G)| \to |\mathbf{Mod}(\Sigma'_G)|$$

as defined by the pushout in **Sign** and the following condition:

$$F_G(\mathcal{G})\big|_{\iota'} = \mathcal{G} \text{ and } F_G(\mathcal{G})\big|_{\gamma'} = F(\mathcal{G}\big|_{\gamma})$$

CASL syntax for $F_G(\mathcal{G})$: $F[\mathcal{G}$ **fit** $\sigma]$

Amalgamation required!

Specifications for local constructions

$$\boxed{SP \xrightarrow{\iota} SP'}$$

Do not confuse:
parametrized (program specification)
$\not\equiv$
(parametrized program) specification

Strict correctness

$$\boxed{Mod[SP \xrightarrow{\iota} SP']}$$

the class of all local constructions

$F\colon |\mathbf{Mod}(Sig[SP])| \to |\mathbf{Mod}(Sig[SP'])|$ that are

- persistent along $\iota\colon Sig[SP] \to Sig[SP']$
- *strictly correct* w.r.t. parameter specification SP and result specification SP':

$$\boxed{F(M) \in Mod[SP'] \text{ for all } M \in Mod[SP]}$$

Correctness of global implementations

Take (as before)

- $\iota\colon \Sigma \to \Sigma'$, $\gamma\colon \Sigma \to \Sigma_G$
- SP with $Sig[SP] = \Sigma$, SP' with $Sig[SP'] = \Sigma'$
- SP_G with $Sig[SP_G] = \Sigma_G$, SP'_G with $Sig[SP'_G] = \Sigma'_G$

Fact: *If*

- $F \in Mod[SP \xrightarrow{\iota} SP']$
- $Mod[SP_G] \subseteq Mod[\gamma(SP)]$
- $Mod[\gamma'(SP') \cup \iota'(SP_G)] \subseteq Mod[SP'_G]$

then $F_G(Mod[SP_G]) \subseteq Mod[SP'_G]$, *i.e.:*

$$\begin{tikzcd} SP_G \arrow[r, "\iota'"] & SP'_G \\ SP \arrow[u, "\gamma"] \arrow[r, "\iota"'] & SP' \arrow[u, "\gamma'"'] \end{tikzcd}$$

$$\boxed{SP'_G \underset{F_G}{\rightsquigarrow} SP_G}$$

Correctness of global implementations

$$
\begin{tikzcd}
SP'_G \arrow[r, squiggly] & \gamma'(SP') \cup \iota'(SP_G) \arrow[r, squiggly, "F_G"'] & SP_G \arrow[l, bend left, "\iota'"] \\
& SP' \arrow[u, "\gamma'"] \arrow[ur, phantom, "\mathsf{PO}"] & SP \arrow[u, "\gamma"'] \arrow[l, "\iota"]
\end{tikzcd}
$$

Program development

- Start with a SPECIFICATION
- Develop software via a SEQUENCE of *refinement steps*
- Each step is small enough that a PROOF OF CORRECTNESS is possible
- Correct refinement steps can be COMPOSED
- Some refinement steps involve DECOMPOSITION into SEPARATE TASKS

RESULT:

Well-designed, well-structured, well-documented

correct and highly modular software

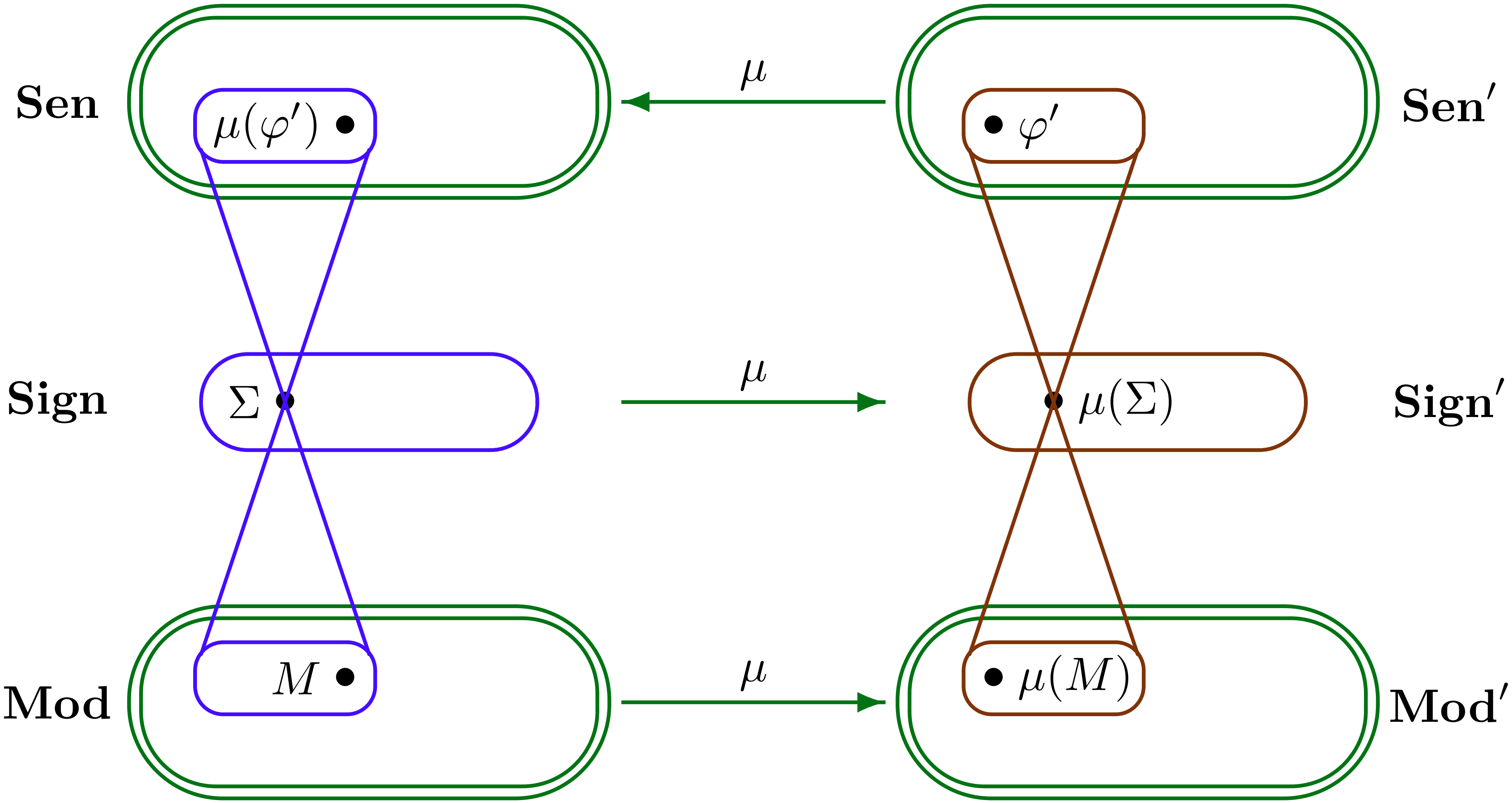
Toward heterogeneous specifications

Linking institutions with each other

. . . various maps between institutions. . .

Categories of institutions

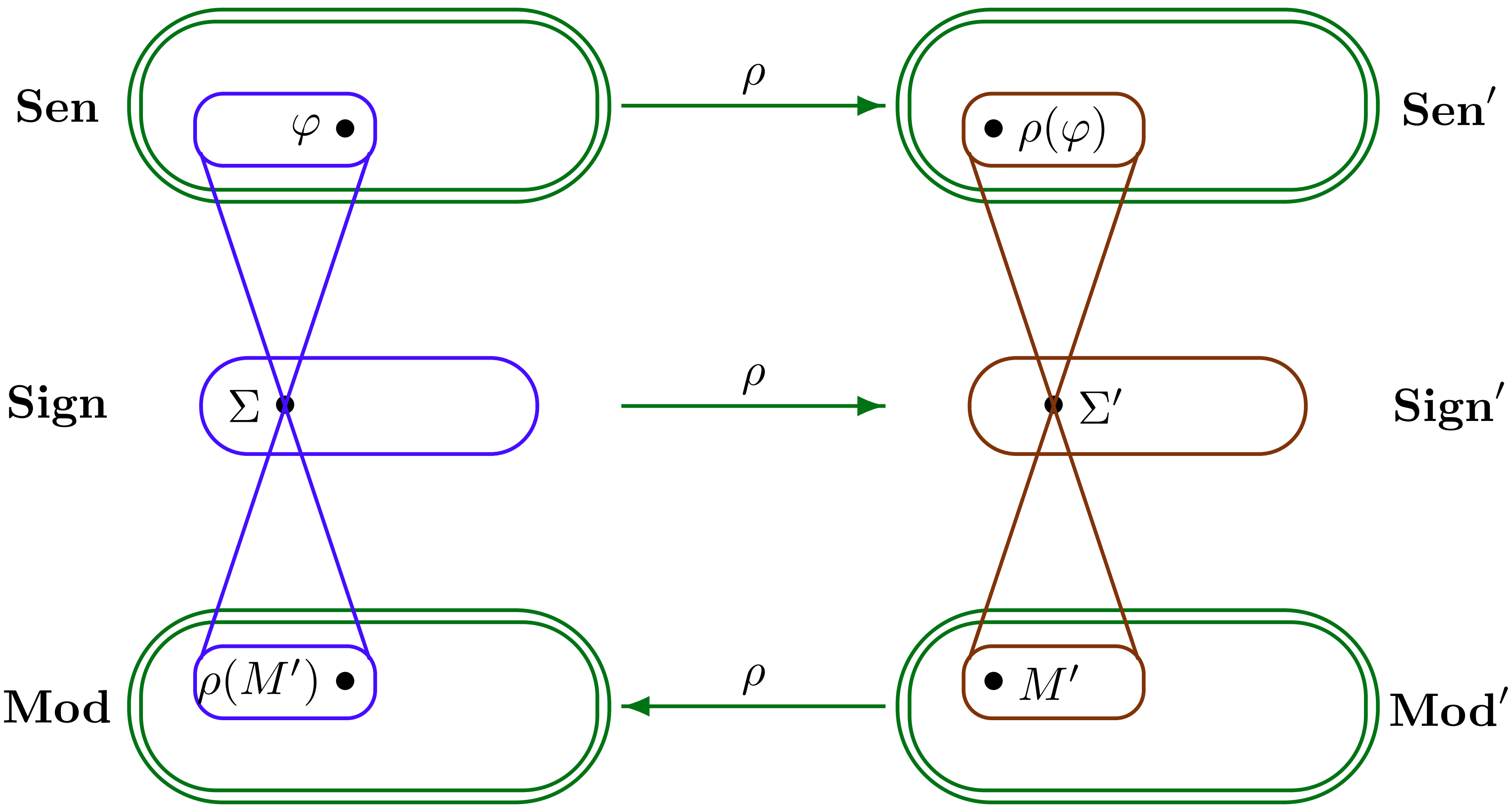
Institution morphism: $\mu\colon \mathbf{I} \longrightarrow \mathbf{I}'$



with the *satisfaction condition* lurking again:

$$M \models \mu(\varphi) \;\text{ iff }\; \mu(M) \models' \varphi'$$

Institution comorphism: $\rho\colon \mathbf{I} \longrightarrow \mathbf{I}'$



with the *satisfaction condition* lurking again:

$$\rho(M') \models \varphi \quad \text{iff} \quad M' \models' \rho(\varphi)$$

Moving between institutions: a taxonomy of maps

morphisms μ	$\mathbf{Sen} \longleftarrow \mathbf{Sen}'$ $\mathbf{Sign} \longrightarrow \mathbf{Sign}'$ $\mathbf{Mod} \longrightarrow \mathbf{Mod}'$	*semi-morphisms* μ	$\mathbf{Sen} \qquad \mathbf{Sen}'$ $\mathbf{Sign} \longrightarrow \mathbf{Sign}'$ $\mathbf{Mod} \longrightarrow \mathbf{Mod}'$
comorphisms ρ	$\mathbf{Sen} \longrightarrow \mathbf{Sen}'$ $\mathbf{Sign} \longrightarrow \mathbf{Sign}'$ $\mathbf{Mod} \longleftarrow \mathbf{Mod}'$	*semi-comorphisms* ρ	$\mathbf{Sen} \qquad \mathbf{Sen}'$ $\mathbf{Sign} \longrightarrow \mathbf{Sign}'$ $\mathbf{Mod} \longleftarrow \mathbf{Mod}'$
forward morphisms	$\mathbf{Sen} \longrightarrow \mathbf{Sen}'$ $\mathbf{Sign} \longrightarrow \mathbf{Sign}'$ $\mathbf{Mod} \longrightarrow \mathbf{Mod}'$		
forward comorphisms	$\mathbf{Sen} \longleftarrow \mathbf{Sen}'$ $\mathbf{Sign} \longrightarrow \mathbf{Sign}'$ $\mathbf{Mod} \longleftarrow \mathbf{Mod}'$		

plus *theoroidal* versions,

plus *weak* versions, plus . . .

Mastering the diversity

Morphism

$$
\begin{tikzcd}
& \mathbf{Set} \arrow[d, phantom, "\overset{\alpha}{\longleftarrow}"] & \\
\mathbf{Sign}^{(op)} \arrow[ur, "\mathbf{Sen}"] \arrow[dr, "\mathbf{Mod}"'] \arrow[rr, "\Phi"] & & \mathbf{Sign}'^{(op)} \arrow[ul, "\mathbf{Sen}'"'] \arrow[dl, "\mathbf{Mod}'"] \\
& \mathbf{Cat} \arrow[u, phantom, "\overset{\beta}{\longrightarrow}"] &
\end{tikzcd}
$$

Span of comorphisms

$$
\begin{tikzcd}
& \mathbf{Set} & \\
& \bullet \arrow[u, "\mathbf{Sen}'"] \arrow[l, phantom, "\overset{\alpha}{\longleftarrow}"] \arrow[r, phantom, "\overset{=}{\longrightarrow}"] & \\
\mathbf{Sign}^{(op)} \arrow[uur, "\mathbf{Sen}"] \arrow[ddr, "\mathbf{Mod}"'] & \mathbf{Sign}^{(op)} \arrow[l, "="'] \arrow[r, "\Phi"] \arrow[u, "\Phi"] \arrow[d, "\Phi^{op}"] & \mathbf{Sign}'^{(op)} \arrow[uul, "\mathbf{Sen}'"'] \arrow[ddl, "\mathbf{Mod}'"] \\
& \bullet \arrow[d, "\mathbf{Mod}'"] \arrow[l, phantom, "\overset{\beta}{\longleftarrow}"] \arrow[r, phantom, "\overset{=}{\longrightarrow}"] & \\
& \mathbf{Cat} &
\end{tikzcd}
$$

Putting institutions together

Fact: *The category $\mathcal{INS}$ of institutions and institution morphisms is complete and (nearly) cocomplete. So is the category $co\mathcal{INS}$, the category of institutions and institution comorphisms.*

- Limits in $\mathcal{INS}$: a rudimentary way of combining institutions linked by institution morphisms to capture how one institution is built over another.
- This is in contrast with the *Grothendieck institution* built over the same diagram, which just puts the institutions involved next to each other, with additional signature morphisms induced by institution morphisms.

Limits of limits

- In general, limits in $\mathcal{INS}$ *do not* preserve cocompleteness of the category of signatures, amalgamability, interpolation, etc.
- Nothing comes for free in $co\mathcal{INS}$ either (though some things might be easier).

Systematically building complex logical systems

- Logic presentations: *parchments*. . .
- Putting parchments together — (co)completeness of parchments categories. . .
- Parchment constructions, extensions, modifications. . .
- Preserving and combining proof systems. . .

EXAMPLE: CafeOBJ cube of logics

Heterogeneous environment

A collection of institutions
linked by
(forward) (semi-) (co-) morphisms

A collection of institutions
linked by (semi-)comorphisms

A diagram $\mathcal{HIE}$ in the category $co\mathcal{INS}$
(of institutions and institution comorphisms)

EXAMPLES:

- a dozen of logics, one for each kind of UML diagrams
- the HETS family of institutions
- CafeOBJ cube of logics
- Mossakowski's diagram of algebraic and other institutions
- . . .

Heterogeneous specifications

- Move to a **universal institution** $\mathbf{UI}$
  (encode institutions in $\mathcal{HIE}$ using comorphisms into $\mathbf{UI}$, compatible with maps within $\mathcal{HIE}$; then work in $\mathbf{UI}$)
- **Focussed heterogeneous specifications**
  (specifications that reside in an institution, but may involve specifications from other institutions in $\mathcal{HIE}$)
- **Distributed heterogeneous specifications**
  (specification diagrams over $\mathcal{HIE}$)

Focused heterogeneous specifications

In a heterogeneous environment $\mathcal{HIE}$:

Translation: introduces new structure to specification models, following an institution semi-comorphism $\rho\colon \mathbf{I} \to \mathbf{I}'$; for any $\mathbf{I}$-specification SP,

$$\boxed{\rho(SP)}$$

is an $\mathbf{I}'$-specification with $Sig[\rho(SP)] = \rho(Sig[SP])$ and $Mod[\rho(SP)] = \{M' \in |\mathbf{Mod}'(\rho(Sig[SP]))| \mid \rho(M') \in Mod[SP]\}$.

Hiding: hides extra structure of specification models, following an institution semi-morphism $\mu\colon \mathbf{I}' \to \mathbf{I}$; for any $\mathbf{I}'$-specification SP',

$$\boxed{SP'|_{\mu}}$$

is an $\mathbf{I}$-specification with $Sig[SP'|_{\mu}] = \mu(Sig[SP'])$ and $Mod[SP'|_{\mu}] = \{\mu(M') \mid M' \in Mod[SP']\}$.

Some topics to repeat for focused heterogeneous specifications

- structured specifications
- proving semantic consequence of, and between specifications
  - institution (co)morphisms in use
- soundness and completeness of (compositional) proof systems
- stepwise software development
- constructor and abstractor implementations
  - inter-institutional constructors needed: the model component of institution semi-(co)morphisms
- branching implementations and architectural specifications
  - developments of individual units may proceed independently within different institutions, given inter-institutional constructors to join them

Distributed heterogeneous specifications

. . . some preliminary ideas . . .

Heterogeneous specification morphisms

Recall: a *specification morphism* $\sigma : SP \to SP'$ in an institution $\mathbf{I}$ is a signature morphism $\sigma : Sig[SP] \to Sig[SP']$ such that for all models $M' \in Mod[SP']$, $M'|_{\sigma} \in Mod[SP]$.

Define: a *heterogeneous specification morphism* from $\mathbf{I}$-specification SP to $\mathbf{I}'$-specification SP' is a pair $\langle \rho, \sigma' \rangle : SP \to SP'$, where $\rho : \mathbf{I} \to \mathbf{I}'$ is an institution (semi-)comorphism, and $\sigma' : \rho(Sig[SP]) \to Sig[SP']$ is an $\mathbf{I}'$-signature morphism such that for all models $M' \in Mod[SP']$, $\rho(M'|_{\sigma'}) \in Mod[SP]$.

This yields a category $\mathcal{HSPEC}$ of heterogeneous specifications over $\mathcal{HIE}$.

. . . Grothendieck construction. . .

Distributed heterogeneous specifications

- A *distributed heterogeneous specification* $\mathcal{HSP}$ is a diagram of heterogeneous specifications in $\mathcal{HSPEC}$, $\mathcal{HSP}: \mathcal{J} \to \mathcal{HSPEC}$.

  Notation:
  - for $i \in |\mathcal{J}|$, $\mathcal{HSP}_i$ is the specification $\mathcal{HSP}(i)$
  - for $e: i \to j$ in $\mathcal{J}$, $\mathcal{HSP}_e = \langle \rho_e, \sigma_e \rangle : \mathcal{HSP}_i \to \mathcal{HSP}_j$ is the heterogeneous specification morphism $\mathcal{HSP}(e)$.

- A *distributed heterogeneous model* of $\mathcal{HSP}$ is a family $\mathcal{M} = \langle M_i \rangle_{i \in |\mathcal{J}|}$ of models *compatible with* $\mathcal{HSP}$.

  That is, such that
  - for $i \in |\mathcal{J}|$, $M_i \in Mod[\mathcal{HSP}_i]$
  - for $e: i \to j$ in $\mathcal{J}$, $M_i = \rho_e(M_j|_{\sigma_e})$.

$\mathcal{HSP}$ is (globally) consistent if it has a (distributed) model

Moving to the limit

Fact: *If* $\mathbf{I}$ *is the colimit of* $\mathcal{HIE}$ *in* $co\mathcal{INS}$ *then for any distributed heterogeneous specification* $\mathcal{HSP}$ *over* $\mathcal{HIE}$ *there is a (focussed heterogeneous)* $\mathbf{I}$*-specification* SP *with models corresponding exactly to distributed heterogeneous models of* $\mathcal{HSP}$.

. . . given enough assumptions. . .

So what?

Typically, the limit institution $\mathbf{I}$ is not "natural" — hence it is better to work with distributed specifications, dealing with various views of the system separately.

Work with local views, local understanding, and local compatibility

. . . but do not forget about global consistency and emerging properties

Implementing distributed specifications

To implement $\mathcal{HSP} : \mathcal{J} \to \mathcal{HSPEC}$ by $\mathcal{HSP}' : \mathcal{J}' \to \mathcal{HSPEC}$, provide:

- a *covering function* $f : |\mathcal{J}| \to |\mathcal{J}'|$, and
- a *distributed constructor* $\kappa = \langle \kappa_i : Mod[\mathcal{HSP}'_{f(i)}] \to Mod[\mathcal{HSP}_i] \rangle_{i \in |\mathcal{J}|}$.

So that for each $i \in |\mathcal{J}|$, we have $\mathcal{HSP}_i \underset{\kappa_i}{\rightsquigarrow} \mathcal{HSP}'_{f(i)}$.

THEN:

$$\boxed{\mathcal{HSP} \underset{\langle \kappa, f \rangle}{\rightsquigarrow} \mathcal{HSP}'}$$

if for each distributed heterogeneous model $\mathcal{M}' = \langle M'_{i'} \rangle_{i' \in |\mathcal{J}'|}$ of $\mathcal{HSP}'$, $\kappa_f(\mathcal{M}') = \langle \kappa_i(M'_{f(i)}) \rangle_{i \in |\mathcal{J}|}$ is a distributed heterogeneous model of $\mathcal{HSP}$.

STRUCTURE MAY CHANGE! INSTITUTIONS MAY CHANGE!

WE NEED TO ARRIVE AT A SINGLE “IMPLEMENTATION” INSTITUTION

One standard way

Fact: *For any* $\mathcal{HSP} : \mathcal{J} \to \mathcal{HSPEC}$ *and* $\mathcal{HSP}' : \mathcal{J}' \to \mathcal{HSPEC}$, *given*

- *a functor* $F : \mathcal{J} \to \mathcal{J}'$
- *a natural transformation* $\tau : \mathcal{HSP} \to F;\mathcal{HSP}'$ *with* $\tau_i = \langle \rho_i, \sigma_i \rangle : \mathcal{HSP}_i \to \mathcal{HSP}_{F(i)}$ *for* $i \in |\mathcal{J}|$

we have

$$\boxed{\mathcal{HSP} \underset{\langle \kappa, f \rangle}{\rightsquigarrow} \mathcal{HSP}'}$$

where

- $f = |F| : |\mathcal{J}| \to |\mathcal{J}'|$
- $\kappa = \langle \rho_i(\_|_{\sigma_i}) : Mod[\mathcal{HSP}'_{F(i)}] \to Mod[\mathcal{HSP}_i] \rangle_{i \in |\mathcal{J}|}$

Key idea

A semantic view of heterogeneous logical environment for software specification and programming emerges: a diagram of institutions

Sample further work:

- keep building up the environment of relevant institutions and (forward) (semi-)(co)morphisms between them;
- expected results and methods for distributed heterogeneous specifications;
- proof theoretic links between institutions linked semantically;
- programming links between "programming" institutions linked semantically.

Summing up

- Standard underlying logical and preliminaries: basic algebraic framework, equational logic; category theory
- Institutions: motivation, abstraction, generality; formalization of the concept of a logical system
- Institutional model theory: numerous bits and pieces of classical model theory reformulated, clarified and sharpened
- Foundations of software specification and development:
  - Specifications: basic and structured specifications; proof systems for specifications
  - Program development: (constructor) refinements; architectural specifications
  - (Observational approach)
- Heterogeneous logical frameworks: maps between institutions; heterogeneous specifications and development; building complex logical systems

Conclusion

A small dose of

mathematics (universal algebra, logic, category theory)

helps to clarify, sharpen, expand and develop

the concepts, methods and results we want