Relating CASL with Other Specification Languages: the Institution Level

Till Mossakowski

Bremen Institute of Safe Systems, Department of Computer Science, Universität Bremen, Germany.

Abstract

In this work, we investigate various specification languages and their relation to CASL, the recently developed Common Algebraic Specification Language. In particular, we consider the languages Larch, OBJ3 and functional CafeOBJ, ACT ONE, ASF, and HEP-theories, as well as various sublanguages of CASL. All these languages are translated to an appropriate sublanguage of CASL.

The translation mainly concerns the level of specification in-the-small: the logics underlying the languages are formalized as institutions, and representations among the institutions are developed. However, it is also considered how these translations interact with specification in-the-large.

Thus, we obtain on the one hand translations of any of the abovementioned specification languages to an appropriate sublanguage of CASL. This allows us to take libraries and case studies that have been developed for other languages and re-use them in CASL.

On the other hand, we set up institution representations going from the CASL institution (and some of its subinstitutions) to simpler subinstitutions. Given a theorem proving tool for such a simpler subinstitution, with the help of a representation, it can also be used for a more complex institution. Thus, first-order theorem provers and conditional term rewriting tools become usable for CASL.

Key words: Specification languages, Institutions, Logics, Translations of logics.

Contents

1 Introduction 3

2 Institutions and Institution Representations 5

This research was supported by the ESPRIT-funded CoFI Working Group 29432 and by the DFG project MULTIPLE.
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Institutions</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Specifications Over an Arbitrary Institution</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Free extensions and liberality</td>
<td>11</td>
</tr>
<tr>
<td>2.4</td>
<td>Institution Representations</td>
<td>15</td>
</tr>
<tr>
<td>2.5</td>
<td>Intersections of Substitutions</td>
<td>38</td>
</tr>
<tr>
<td>2.6</td>
<td>Translating language constructs</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>CASL</td>
<td>40</td>
</tr>
<tr>
<td>3.1</td>
<td>Partial First-Order Logic</td>
<td>40</td>
</tr>
<tr>
<td>3.2</td>
<td>Subsorted Partial First-Order Logic</td>
<td>45</td>
</tr>
<tr>
<td>3.3</td>
<td>CASL Language Constructs</td>
<td>47</td>
</tr>
<tr>
<td>3.4</td>
<td>Substitutions of CASL</td>
<td>48</td>
</tr>
<tr>
<td>4</td>
<td>Representations Among Substitutions of CASL</td>
<td>55</td>
</tr>
<tr>
<td>4.1</td>
<td>The First-Order Level</td>
<td>55</td>
</tr>
<tr>
<td>4.2</td>
<td>The Positive Conditional Level</td>
<td>72</td>
</tr>
<tr>
<td>4.3</td>
<td>Theorem proving and liberality</td>
<td>77</td>
</tr>
<tr>
<td>5</td>
<td>Hierarchy Theorems</td>
<td>80</td>
</tr>
<tr>
<td>5.1</td>
<td>Level 5 versus Level 4: partial conditional logic and horn clause logic</td>
<td>82</td>
</tr>
<tr>
<td>5.2</td>
<td>Level 4 versus Level 3: horn clause logic and conditional equational logic</td>
<td>83</td>
</tr>
<tr>
<td>5.3</td>
<td>Level 3 versus Level 2: conditional equational logic and partial equational logic</td>
<td>84</td>
</tr>
<tr>
<td>5.4</td>
<td>Level 2 versus Level 1: partial equational logic and total equational logic</td>
<td>86</td>
</tr>
<tr>
<td>6</td>
<td>The Larch Shared Language</td>
<td>88</td>
</tr>
<tr>
<td>6.1</td>
<td>The LSL Institution and Its Translation to $CFOAlg^= $</td>
<td>88</td>
</tr>
<tr>
<td>6.2</td>
<td>Translating LSL Language Constructs Into CASL Constructs</td>
<td>90</td>
</tr>
<tr>
<td>7</td>
<td>ACT ONE</td>
<td>91</td>
</tr>
</tbody>
</table>
1 Introduction

CASL is a specification language that has been designed by CoFI, the international Common Framework Initiative for algebraic specification and development [65,21], with the goal to subsume many previous algebraic specification languages and to provide a standard language for the specification and development of modular software systems. The design of CASL is explained in another paper in this volume [4], see also [22]. Actually, CASL is a central language in a whole family of languages. CASL concentrates on the specification of abstract data types and (first-order) functional requirements, while some (currently still prototypical) extensions of CASL also consider the specification of higher-order functions [60,72] and of reactive [9,68,69] and object-oriented [3,43] behaviour. Several restrictions of CASL to sublanguages make it possible to use specialized tool support.
The aim of the present paper is to substantiate the claim that CASL subsumes many existing specification languages. We will consider the relation of both specification languages (Larch [40], OBJ3 and functional CafeOBJ [31,26], ACT ONE [20], ASF [11], HEP-theories [70]) and common tool-supported logics (e.g. first-order logic, conditional equational logic) to CASL. As a first step, we deal with specification in-the-small, i.e. unstructured specifications of individual software modules, formulated in a specific logic. We also study how these translations interact with the CASL concepts of specification in-the-large (although we do not study translations among different concepts of specification in-the-large). A clean separation of specification in-the-small and specification in-the-large is possible, because in CASL, both levels are treated separately: the semantics of CASL in-the-large is orthogonal to the underlying logic that is used for specification in-the-small.

For any practically usable specification language, there is a distinction between underlying mathematical concepts on the one hand, and language constructs on the other hand. Typically, the language constructs provide a concise and user-friendly syntax for writing specifications, and the semantics interprets the constructs in terms of the concepts. At the level of specification in-the-small, the essential mathematical concept is that of the logic underlying the language.

Here, we mainly address the problem of relating the underlying logics of the specification languages. To this end, we formalize the logics as institutions in the sense of Goguen and Burstall [33]. For some specification languages, the recognition of the underlying logic is not obvious, and the formalization as an institution is a non-trivial task. Once this has been done, the institutions can be related using institution representations [47,74]. In some cases, there is an obvious subinstitution of the CASL institution that closely corresponds to the institution underlying the specification language in question. We therefore single out a number of subinstitutions of CASL and develop a uniform naming scheme for them. In other cases, the relation is not so obvious, and there are several good choices for the institution representation.

We also address the translation of (subinstitutions of) CASL to other, simpler subinstitutions. Typically, the target of such a translation will be an institution with good tool support, and the translation will have the property that the tool support can be lifted against the translation (this has been called “borrowing” [18]). We also present some new results concerning such lifting properties.

The level of languages constructs is not discussed formally in this paper. This would require detailed language definitions, which would be too lengthy to be presented here (apart from the difficulty that not all of the languages are precisely defined, e.g. w.r.t. their static semantic conditions). Rather, we informally discuss how the institution representations lift to translations of
the corresponding language constructs. In many cases, this is straightforward. But there remain some situations where it is not advisable to first expand a construct into a theory of the underlying institution and then translate this theory. Instead, it is better to use a more direct translation of the constructs.

The structure of this paper is as follows: In Section 2, we recall the notions of institution and institution representation. We study various properties of institution representations that lead to a good interaction of flat and structured specifications with institution representations. In this context, we also develop some new results concerning structured specifications. In Section 3, we introduce the institution underlying CASL and a number of its subinstitutions, for which we develop a uniform naming scheme. Section 4 discusses representations of CASL in some of its subinstitutions such as first-order logic, and also representations among subinstitutions of CASL. We obtain three graphs of institutions and representations: one at the first-order level without sort generation constraints, one at the first-order level with sort generation constraints, and one at the positive conditional level. The graphs contain different kinds of edges corresponding to the different properties of institution representations introduced in Section 2. By combining these graphs with the results about interaction of institution representations with flat and structured specifications, we obtain a number of theorems about re-use of theorem provers and about the existence of free objects (liberality). In particular, we show that a theorem prover for first-order logic plus induction suffices to prove theorems within the CASL institution, even at the level of structured specifications including certain free specifications. Further, conditional term rewriting and paramodulation can be applied to the positive conditional fragment of CASL. Moreover, we show the latter to be liberal. Next, sections 6, 7, 8, 9, 10 are devoted to the definition and translation of the institutions underlying Larch, ACT ONE, ASF, HEP-theories, and OBJ3 and functional CafeOBJ, respectively. In each case, a separate subsection is devoted to an informal discussion of the level of language constructs. Section 11 contains some general remarks about boolean-valued functions versus predicates and about the empty carrier problem. Throughout the paper, only little knowledge about category theory is assumed, except from some parts of Section 2. These parts rely on some purely category theoretic results, which we have relegated to the appendix.

2 Institutions and Institution Representations

In this section, we introduce the notion of institution and different types of institution representations known from the literature. We summarize some known properties of these and prove new results concerning their interaction with specification in-the-large.
2.1 Institutions

A specification formalism is usually based on some notion of signature, model, sentence and satisfaction. These are the usual ingredients of Barwise's abstract model theory [7]. Contrary to Barwise's notions, institutions of Goguen and Burstall [33] do not assume that signatures are algebraic signatures and thus cover a much larger variety of logics. Indeed, the theory of institutions assumes nothing about signatures except that they form a class and that there are signature morphisms, which can be composed in some way. This amounts to stating that signatures form a category.

There is also nothing special assumed about the form of the sentences and models. Given a signature $\Sigma$, the $\Sigma$-sentences form just a set, while the $\Sigma$-models form a category (taking into account that there may be model morphisms).

Signature morphisms lead to translations of sentences and of models (thus, the assignments of sentences and of models to signatures are functors). There is a contravariance between the sentence and the model translation; sentences are translated along signature morphisms, while models are translated against signature morphisms.

Informally, this can be motivated as follows. Forget for a moment the above generality and think of signatures as of sets of certain symbols. Think of sentences over a signature $\Sigma$ as derivation trees over some grammar, decorated at the nodes with the symbols from $\Sigma$. Then sentence translation along a signature morphism $\sigma: \Sigma \to \Sigma'$ keeps the structure of the derivation tree, but replaces the symbols decorating the nodes, using $\sigma$. This explains why sentences are translated along signature morphisms.

Concerning models over a signature: they have to interpret the symbols from the signature somehow. Thus, a $\Sigma$-model can be seen as a map $m$ going from the symbols of $\Sigma$ to some semantical domain. Now given a $\Sigma$-model $m'$ and a signature morphism $\sigma: \Sigma \to \Sigma'$, by composing the interpretation map $m'$ with $\sigma$ we get a new interpretation map $m'|_\sigma$, which is a $\Sigma$-model! ($m'|_\sigma$ is also called the $\sigma$-reduct of $m'$.) This explains why models are translated against signature morphisms.

Of course, these explanations just have motivating purpose: there can be institutions with a completely different view of signatures, models and sentences. However, they shed some light on how many typical institutions work.\footnote{Indeed, the above explanation has been formalized as so-called parchments [55].}

Finally, institutions have a satisfaction relation between models and sentences, which has to be invariant under the simultaneous translation of sentences and
models w.r.t. a given signature morphism.

This leads to the following formal definition. Let $\mathcal{CAT}$ be the category of categories and functors.\(^3\)

**Definition 2.1** An *institution* $I = (\text{Sign}^I, \text{Sen}^I, \text{Mod}^I, \models^I)$ consists of

- a category $\text{Sign}^I$ of signatures,
- a functor $\text{Sen}^I : \text{Sign}^I \to \text{Set}$ giving, for each signature $\Sigma$, the set of *sentences* $\text{Sen}^I(\Sigma)$, and for each signature morphism $\sigma : \Sigma \to \Sigma'$, the *sentence translation map* $\text{Sen}^I(\sigma) : \text{Sen}^I(\Sigma) \to \text{Sen}^I(\Sigma')$, where often $\text{Sen}^I(\sigma)(\varphi)$ is written as $\sigma(\varphi)$,
- a functor $\text{Mod}^I : (\text{Sign}^I)^{\text{op}} \to \mathcal{CAT}$ giving, for each signature $\Sigma$, the category of *models* $\text{Mod}^I(\Sigma)$, and for each signature morphism $\sigma : \Sigma \to \Sigma'$, the *reduct functor* $\text{Mod}^I(\sigma) : \text{Mod}^I(\Sigma') \to \text{Mod}^I(\Sigma)$, where often $\text{Mod}^I(\sigma)(M')$ is written as $M'|_{\sigma}$,
- a satisfaction relation $\models^I_\Sigma \subseteq [\text{Mod}^I(\Sigma)] \times \text{Sen}^I(\Sigma)$ for each $\Sigma \in \text{Sign}^I$,

such that for each $\sigma : \Sigma \to \Sigma'$ in $\text{Sign}^I$ the following *satisfaction condition* holds:

$$M' \models^I_{\Sigma'} \sigma(\varphi) \iff M'|_{\sigma} \models^I_\Sigma \varphi$$

for each $M' \in \text{Mod}^I(\Sigma')$ and $\varphi \in \text{Sen}^I(\Sigma)$. \(\square\)

We will omit the index $I$ when it is clear from the context.

We now informally present some examples. They will be formally introduced in Section 3.

**Example 2.2** The institution $\text{Eq}^-$ of equational logic. Signatures are many-sorted algebraic signatures consisting of a set of sorts and a set of function symbols (where each function symbol has a string of argument sorts and a result sort). Signature morphisms map sorts and function symbols in a compatible way. Models are just many-sorted algebras, i.e. each sort is interpreted as a carrier set, and each function symbol is interpreted as a function between the carrier sets specified by the argument and result sorts. Reducts are constructed as sketched above. Sentences are equations between many-sorted terms, and sentence translation means replacement of the translated symbols. Finally, satisfaction is the usual satisfaction of an equation in an algebra. \(\square\)

**Example 2.3** The institution $\text{FOL}^=$ of many-sorted first-order logic with equality. Signatures are many-sorted first-order signatures, i.e. many-sorted algebraic signatures enriched with predicate symbols. Models are many-sorted first-order structures. Sentences are first-order formulas, and again sentence

\(^3\) Strictly speaking, $\mathcal{CAT}$ is not a category but only a so-called quasicategory, which is a category that lives in a higher set-theoretic universe [42].
translation means replacement of the translated symbols. Satisfaction is the usual satisfaction of a first-order sentence in a first-order structure. □

**Example 2.4** The institution $PFOL^=$ of partial first-order logic with equality. Signatures are many-sorted first-order signatures enriched by partial function symbols. Models are many-sorted partial first-order structures. Sentences are first-order formulas containing existential equations, strong equations, definedness statements and predicate applications as atomic formulas. Satisfaction is defined using total valuations of variables, while valuation of terms is partial due to the existence of partial functions. An existential equation holds if both sides are defined and equal, whereas a strong equation also holds if both sides are undefined. A definedness statement holds if the term is defined. A predicate application holds if the terms contained in it are defined, and the corresponding tuple of values is in the interpretation of the predicate. This is extended to first-order formulas as usual. □

Within an arbitrary but fixed institution, we can easily define the usual notion of logical consequence or semantical entailment: Given a set of $\Sigma$-sentences $\Gamma$ and a $\Sigma$-sentence $\varphi$, we say

$$\Gamma \models_\Sigma \varphi \quad (\varphi \text{ follows from } \Gamma)$$

iff for all $\Sigma$-models $M$, we have

$$M \models_\Sigma \Gamma \text{ implies } M \models_\Sigma \varphi.$$ 

Here, $M \models_\Sigma \Gamma$ means that $M \models_\Sigma \psi$ for each $\psi \in \Gamma$.

We will also freely use other standard logical terminology when working within an arbitrary but fixed institution.

### 2.2 Specifications Over an Arbitrary Institution

This paper mainly concentrates on translating specification languages at the level of specification in-the-small. However, when translating the underlying logics of specification languages, the question arises how this interacts with specification in-the-large, for a given fixed set of structuring operations for specification in-the-large. Therefore, in this section we recall a popular set of institution-independent structuring operations, which seems to be quite universal and which can also be seen as a kernel language for the CASTL structuring constructs. The question how this would relate to other sets of structuring operations is beyond the scope of the present paper.

In the sequel, let us fix an arbitrary institution $I = (\text{Sign}, \text{Sen}, \text{Mod}, \models)$. 

8
The simplest specifications over an arbitrary institution are just theories \( T = \langle \Sigma, \Gamma \rangle \), where \( \Sigma \in \text{Sign} \) and \( \Gamma \subseteq \text{Sen}(\Sigma) \) (we set \( \text{Sig}(T) = \Sigma \) and \( \text{Ax}(T) = \Gamma \)). Theory morphisms \( \sigma: \langle \Sigma, \Gamma \rangle \rightarrow \langle \Sigma', \Gamma' \rangle \) are those signature morphisms \( \sigma: \Sigma \rightarrow \Sigma' \) for which \( \Gamma' \models_{\Sigma'} \sigma(\Gamma) \), that is, axioms are mapped to logical consequences. By inheriting composition and identities from \( \text{Sign} \), we obtain a category \( \text{Th} \) of theories. It is easy to extend \( \text{Sen} \) and \( \text{Mod} \) to start from \( \text{Th} \) by putting \( \text{Sen}(\langle \Sigma, \Gamma \rangle) = \text{Sen}(\Sigma) \) and letting \( \text{Mod}(\langle \Sigma, \Gamma \rangle) \) be the full subcategory of \( \text{Mod}(\Sigma) \) induced by the class of those models \( M \) satisfying \( \Gamma \). The category \( \text{Pres} \) of presentations (also called flat specifications) is just the full subcategory of theories having finite sets of axioms.

Concerning more complex specifications, in [71], the following kernel language for specifications in an arbitrary institution has been proposed. Simultaneously with the notion of specification, we define functions \( \text{Sig} \) and \( \text{Mod} \) yielding the signature and the model class of a specification.

**presentations:** For any signature \( \Sigma \in |\text{Sign}| \) and finite set \( \Gamma \subseteq \text{Sen}(\Sigma) \) of \( \Sigma \)-sentences, the presentation \( \langle \Sigma, \Gamma \rangle \) is a specification with:

\[
\text{Sig}(\langle \Sigma, \Gamma \rangle) := \Sigma \\
\text{Mod}(\langle \Sigma, \Gamma \rangle) := \{ M \in \text{Mod}(\Sigma) \mid M \models \Gamma \}
\]

**union:** For any signature \( \Sigma \in |\text{Sign}| \), given \( \Sigma \)-specifications \( SP_1 \) and \( SP_2 \), their union \( SP_1 \cup SP_2 \) is a specification with:

\[
\text{Sig}(SP_1 \cup SP_2) := \Sigma \\
\text{Mod}(SP_1 \cup SP_2) := \text{Mod}(SP_1) \cap \text{Mod}(SP_2)
\]

**translation:** For any signature morphism \( \sigma: \Sigma \rightarrow \Sigma' \) and \( \Sigma \)-specification \( SP \), translate \( SP \) by \( \sigma \) is a specification with:

\[
\text{Sig}(\text{translate } SP \text{ by } \sigma) := \Sigma' \\
\text{Mod}(\text{translate } SP \text{ by } \sigma) := \{ M' \in \text{Mod}(\Sigma') \mid M'|_\sigma \in \text{Mod}(SP) \}
\]

**hiding:** For any signature morphism \( \sigma: \Sigma \rightarrow \Sigma' \) and \( \Sigma' \)-specification \( SP' \), derive from \( SP' \) by \( \sigma \) is a specification with:

\[
\text{Sig}(\text{derive from } SP' \text{ by } \sigma) := \Sigma \\
\text{Mod}(\text{derive from } SP' \text{ by } \sigma) := \{ M'|_\sigma \mid M' \in \text{Mod}(SP') \}
\]

The above specification-building operations, although extremely simple, already provide flexible mechanisms for expressing basic ways of putting specifications together and thus building specifications in a structured manner. Hence, they can be considered to be a kernel language for structured specification. The specification language CASL provides more sophisticated structuring constructs, but it is possible to translate the CASL constructs (except the free construct, which will be examined in the next section) to the above kernel language, see [59].

A specification \( SP \) is said to be consistent, if \( \text{Mod}(SP) \) is not empty.
Given two structured specifications $SP_1$ and $SP_2$, a specification morphism
\( \sigma: SP_1 \to SP_2 \) is a signature morphism $\sigma: \text{Sig}(SP_1) \to \text{Sig}(SP_2)$ such that $M|_{\sigma} \in \text{Mod}(SP_1)$ for each $M \in \text{Mod}(SP_2)$. By the satisfaction condition, each theory morphism between presentations also is a specification morphism.

A structured $\Sigma$-specification $SP_2$ refines a structured $\Sigma$-specification $SP_1$ (written $SP_1 \equiv\equiv SP_2$), if $\text{Mod}(SP_2) \subseteq \text{Mod}(SP_1)$.

A specification not containing derive is called flatenable. This is because it is easy to normalize flatenable specifications into flat specifications [10]:

**Definition 2.5** Given a flatenable specification $SP$ over an institution $I$, its normal form $NF(SP)$ is inductively defined as follows:

- $NF(\langle \Sigma, \Gamma \rangle) := \langle \Sigma, \Gamma \rangle$
- If $SP = SP_1 \cup SP_2$, let $NF(SP_i) = \langle \Sigma_i, \Gamma_i \rangle \ (i = 1, 2)$. Then $NF(SP) := \langle \Sigma_1, \Gamma_1 \cup \Gamma_2 \rangle$.
- If $SP = \text{translate} SP_1$ by $\sigma: \Sigma_1 \to \Sigma$, let $NF(SP_1) = \langle \Sigma_1, \Gamma_1 \rangle$. Then $NF(SP) := \langle \Sigma, \sigma(\Gamma_1) \rangle$. \( \square \)

**Fact 2.6** If $SP$ is a flatenable specification, then

$$\text{Mod}(SP) = \text{Mod}(NF(SP)). \quad \square$$

To be able to introduce normal forms of non-flatenable specifications, we need the notion of weak amalgamation:

**Definition 2.7** An institution $I = (\text{Sign}, \text{Sen}, \text{Mod}, \models)$ admits weak amalgamation, if $\text{Sign}$ has pushouts and, moreover, given any pushout

any $\Sigma_1$-model $M_1$ and any $\Sigma_2$-model $M_2$ with $M_1|_{\sigma_1} = M_2|_{\sigma_2}$, there exists some $\Sigma'$-model $M'$ with $M'|_{\theta_2} = M_1$ and $M'|_{\theta_1} = M_2$. \( \square \)

It is well known that in institutions with weak amalgamation, there is a normal form for specifications [29,10,12], having the format

**derive from** $\langle \Sigma, \Gamma \rangle$ by $\sigma$.

Since the construction of a normal form involves pushouts and these are only determined up to isomorphism, also the normal form is only determined up to
signature isomorphism. We therefore define a relation “SP" is a normal form of SP”.

**Definition 2.8** The relation “is a normal form of” is the least relation satisfying:

- **derive from** \( \langle \Sigma, \Gamma \rangle \) by \( id \) is a normal form of \( \langle \Sigma, \Gamma \rangle \).
- If \( SP = SP_1 \cup SP_2 \), let **derive from** \( \langle \Sigma_i, \Gamma_i \rangle \) by \( \sigma_i \) be a normal form of \( SP_i \) (\( i = 1, 2 \)), and let

\[
\begin{array}{c}
\Sigma_1 \\
\sigma_1 \\
\sigma_2 \\
\Sigma_2 \\
\theta_2 \\
\theta_1 \\
\Sigma'
\end{array}
\]

be a pushout. Then **derive from** \( \langle \Sigma', \theta_2(\Gamma_1) \cup \theta_1(\Gamma_2) \rangle \) by \( \theta_2 \circ \sigma_1 \) is a normal form of \( SP \).
- If \( SP = \text{translate} SP' \) by \( \sigma_1: \Sigma \rightarrow \Sigma_1 \), let **derive from** \( \langle \Sigma_2, \Gamma_2 \rangle \) by \( \sigma_2 \) be a normal form of \( SP' \), and take the pushout of \( \sigma_1 \) and \( \sigma_2 \) as above. Then **derive from** \( \langle \Sigma', \theta_1(\Gamma_2) \rangle \) by \( \theta_2 \) is a normal form of \( SP \).
- If \( SP = \text{derive from} SP' \) by \( \sigma_1: \Sigma \rightarrow \Sigma_1 \), let **derive from** \( \langle \Sigma_2, \Gamma_2 \rangle \) by \( \sigma_2 \) be a normal form of \( SP' \). Then **derive from** \( \langle \Sigma_2, \Gamma_2 \rangle \) by \( \sigma_2 \circ \sigma_1 \) is a normal form of \( SP \). \( \square \)

**Fact 2.9** Assume that we work with specifications over an institution with weak amalgamation. Let \( SP \) be a normal form of \( SP \). Then

\[
\text{Mod}(SP) = \text{Mod}(SP'). \quad \square
\]

2.3 **Free extensions and liberality**

Another institution independent structuring construct is that of free extensions. We treat it separately because in many structuring languages, it is not included, and moreover, it is not preserved so well along institution representations (see Section 2.4).

Free extensions can be defined w.r.t. an arbitrary functor: Given categories \( A \) and \( B \) and a functor \( G: B \rightarrow A \), an object \( B \in B \) is called \( G \)-free (with unit \( \eta_A: A \rightarrow G(B) \)) over \( A \in A \), if for any object \( B' \in B \) and any morphism \( h: A \rightarrow G(B') \), there is a unique morphism \( h^\#: B \rightarrow B' \) such that \( G(h^\#) \circ \)
\( \eta_A = h. \)

\[
\begin{array}{c}
B \\
\downarrow h \\
\downarrow \eta_A \\
G(h^\#) \\
G(B')
\end{array}
\]

\[
\begin{array}{c}
G(B) \\
f \\
\downarrow \\
G((f \circ \eta_A)^\#) \\
G(B')
\end{array}
\]

In this case, the unit \( \eta_A \) is called a \textit{G-universal arrow}. We will mostly omit the specification of the unit. An object \( B \in \mathbf{B} \) is called \textit{persistently G-free}, if it is \( G \)-free over some \( A \in \mathbf{A} \) with the unit being an isomorphism. It is called \textit{strongly persistently G-free} if it is \( G \)-free with unit \textit{id} over \( G(B) \) (\textit{id} denotes the identity).

**Proposition 2.10** Given a functor \( \mathbf{G}: \mathbf{B} \rightarrow \mathbf{A} \), an object \( B \in \mathbf{B} \) is persistently \( G \)-free if and only if it is strongly persistently \( G \)-free.

\[
\begin{array}{c}
A \\
\downarrow \eta_A \\
\downarrow \text{id} \\
G(B) \\
\downarrow f \\
G((f \circ \eta_A)^\#) \\
G(B')
\end{array}
\]

**Proof.** The “if” direction is clear. For the “only if” direction, let \( \eta_A: A \rightarrow G(B) \) be a \( G \)-universal isomorphism. If \( f: G(B) \rightarrow G(B') \) is a morphism, \( (f \circ \eta_A)^\# \) is the unique morphism \( g: B \rightarrow B' \) with \textit{id} \circ G(g) = f \). Hence, \textit{id}: \( G(B) \rightarrow G(B) \) is \( G \)-universal as well. \( \square \)

We now extend the kernel language of the previous section as follows. For any signature morphism \( \sigma: \Sigma \rightarrow \Sigma' \) and \( \Sigma' \)-specification \( SP' \), \textbf{free} \( SP' \) \textbf{along} \( \sigma \) is a specification with:

\[
\text{Sig(\textbf{free} \ SP' \textbf{ along} \ \sigma)} = \Sigma'
\]

\[
\text{Mod(\textbf{free} \ SP' \textbf{ along} \ \sigma)} = \{ M' \in \text{Mod}(SP') \mid \text{M'} \text{ is strongly persistently (Mod}(\sigma): \text{Mod}(SP') \rightarrow \text{Mod}(\Sigma))\text{-free} \}
\]

These specifications are called \textbf{data} \( SP' \) \textbf{over} \( \sigma \) in [71]. Since the unit has to be the identity, the freeness condition for \( M' \) means that for any model \( N' \in \text{Mod}(SP') \) and any model morphism \( h: M'|_\sigma \rightarrow N'|_\sigma \), there is a unique morphism \( h^\#: M' \rightarrow N' \) such that \( (h^\#)|_\sigma = h \). By Proposition 2.10, we can equally use just persistent freeness instead of strongly persistent freeness.

Free extensions allow one to express certain inductive properties in a concise way. For example, the transitive closure of an arbitrary relation can be specified using the \textbf{free} construct in \textsc{Casl} as follows:

**Example 2.11**
spec BinaryRelation =
    sort Elem
    pred _~_ : Elem × Elem
end

spec TransitiveClosure [BinaryRelation] =
    free
    { pred _~*__ : Elem × Elem
      ∀ x, y, z : Elem
      • x ~ y ⇒ x ~*_ y
      • x ~*_ y ∧ y ~*_ z ⇒ x ~*_ z
    }
end □

The corresponding structured specification is constructed as follows: Let \( \langle \Sigma', \Gamma' \rangle \) be the presentation consisting of all sorts, predicates and axioms declared in either of BinaryRelation and TransitiveClosure, and let \( \Sigma \) be the signature of BinaryRelation. Then as denotation of the above specification, we get

free \( \langle \Sigma', \Gamma' \rangle \) along \( \sigma \)

where \( \sigma \) is the inclusion of \( \Sigma \) into \( \Sigma' \).

Another use of the free construct is in the generation of datatypes. For examples, consider the specification of finite sets over arbitrary elements in CASL:

Example 2.12

spec GenerateFiniteSet [sort Elem] =
    free
    { type FinSet[Elem] ::= {}
      | {\_\_}(Elem)
      | _\_ \_ (FinSet[Elem]; FinSet[Elem])
        assoc, comm, idem, unit {} }
end

This expands to the following:

spec GenerateFiniteSet [sort Elem] =
    free
    { sort FinSet[Elem]
      ops {} : FinSet[Elem];
      {\_\_} : Elem → FinSet[Elem];
      forall x, y, z : Elem
      • x \_\_ (y \_\_ z) = (x \_\_ y) \_\_ z

13
\[ x \cup y = y \cup x \]
\[ x \cup x = x \]
\[ x \cup \{\} = x \]

The question whether free models actually exist leads to the notion of liberality [33]:

**Definition 2.13** Given an institution \( I = (\text{Sign}, \text{Sen}, \text{Mod}, \models) \),

- a theory morphism \( \sigma: T \to T' \) is said to be *liberal* if, for each \( T \)-model \( M \), there is a \( T' \)-model \( M' \) that is \( \text{Mod}(\sigma) \)-free over \( M \); it is called *strongly persistently liberal* if, moreover, \( M' \) is strongly persistently \( \text{Mod}(\sigma) \)-free;
- the institution \( I \) is called liberal if each of its theory morphisms is liberal;
- given a class \( \mathcal{M} \) of theory morphisms in \( I \), \( I \) is called \( \mathcal{M} \)-liberal if each theory morphism in \( \mathcal{M} \) is liberal. \( \Box \)

The notion of (strongly persistent) liberality can easily be extended from theory morphisms to specification morphisms.

One could guess that in a liberal institution, \textbf{free} \( SP \) along \( \sigma \) is consistent whenever \( SP \) is. However, this is not the case, as the following counterexample shows:

**Example 2.14** The institution \( Eq^= \) of equational logic is known to be liberal\(^4\) (see [33] and Theorem 4.16 below). Let \( \Sigma \) consist of one sort \( s \) and one constant \( c : s \), let \( \Sigma' \) be \( \Sigma \) plus one unary function \( f : s \to s \), and let \( \sigma : \Sigma \to \Sigma' \) be the inclusion. Clearly \( \langle \Sigma', \emptyset \rangle \) is consistent. However, \textbf{free} \( \langle \Sigma', \emptyset \rangle \) along \( \sigma \) is inconsistent: any \( \Sigma \)-model is freely extended by adding an \( \omega \)-chain

\[ \{ f(a), f(f(a)), f(f(f(a))), \ldots \} \]

over each of its elements \( a \). Thus, a \( \Sigma' \)-model can never be free over its own \( \sigma \)-reduct. \( \Box \)

To show consistency of \textbf{free} \( SP \) along \( \sigma \), we need strongly persistent liberality:

**Proposition 2.15** If \( \sigma : \langle \Sigma, \emptyset \rangle \to \langle \Sigma', \Gamma' \rangle \) is strongly persistently liberal and \( \langle \Sigma', \Gamma' \rangle \) is consistent, then also \textbf{free} \( \langle \Sigma', \Gamma' \rangle \) along \( \sigma \)

\(^4\) One has to allow empty carrier sets or restrict oneself to strict theory morphisms to get this result, see Section 11.2.
is consistent.

**Proof.** Let $M'_i \in \text{Mod}(\langle \Sigma', \Gamma' \rangle)$ by consistency, and let $M'$ be $\text{Mod}(\sigma)$-free over $M'_i |_\tau$ with $M'$ also being strongly persistently $\text{Mod}(\sigma)$-free. Then $M' \in \text{Mod}(\text{free} \; \langle \Sigma', \Gamma' \rangle \; \text{along} \; \sigma)$, and hence, $\text{free} \; \langle \Sigma', \Gamma' \rangle \; \text{along} \; \sigma$ is consistent. \hfill \Box

For specifications containing **free**, it is not so easy to obtain a normal form. This is because in general **free** does not commute with the other specification building operations.

### 2.4 Institution Representations

In order to relate sublanguages of CASL, we relate their underlying institutions. We therefore use the notion of institution representation (also called simple map of institutions) [47,74]. The idea behind an institution representation is to **encode** an institution $I$ within an institution $J$. A **simple institution representation** from an institution $I$ to an institution $J$ consists of the following components:

- a translation $\Phi$ of $I$-signatures to $J$-presentations. Given an $I$-signature $\Sigma$, the task is to find a $J$-encoding $\Phi(\Sigma)$ of $\Sigma$ in some way. In particular, the model category of $\Phi(\Sigma)$ should approximate the model category of $\Sigma$ somehow. Consider, for example, the translation of $PFOL^= \to FOL^=$ (cf. Examples 2.3 and 2.4). Here, a partial function symbol can be encoded by a total function symbol, assuming that undefinedness is encoded by extra error values in the models. Thus, we would also need definedness predicates (one for each sort) which distinguish the “defined” values from the “undefined” values. Now let us assume that our partial logic is strict. Then we have to assume that in the encoding, the total functions return a “defined” result only for “defined” arguments. This can be expressed by an axiom, which should be included into $\Phi(\Sigma)$. This explains why $\Phi(\Sigma)$ should be a **presentation**, and not just a signature.

- a translation $\alpha$ of $I$-sentences to $J$-sentences. The reason why the sentence translation goes **along** with the signature translation is similar to the reason why the sentence translation within an institution goes along with the signature morphism. Namely, if a signature $\Sigma$ in $I$ is encoded by the presentation $\Phi(\Sigma)$ in $J$, it is expected that each symbol in $\Sigma$ is translated to some corresponding symbol in $\Phi(\Sigma)$. Now if we assume that a $\Sigma$-sentence $\varphi$ is a derivation tree decorated with some symbols from $\Sigma$, the translation $\alpha_\Sigma(\varphi)$ just keeps the structure of the tree and translates the symbols according to the correspondence of symbols in $\Sigma$ and $\Phi(\Sigma)$. In the example above, $\alpha_\Sigma(\varphi)$ would just replace partial function symbols by total function symbols.
a translation $\beta$ of $J$-models to $I$-models, giving the above mentioned relation between $\Sigma$-models in $I$ and $\Phi(\Sigma)$-models in $J$. Here, we again have the contravariance of the model translation, as in the definition of institution. For example, we can extract a $\Sigma$-model $\beta_\Sigma(M')$ with strict partial functions out of a $\Phi(\Sigma)$-model $M'$ with total functions reflecting "defined" values as sketched above: we just take as carriers of the partial model the interpretations of the definedness predicates in the total model, while the total functions are restricted to these new carriers, yielding partial functions. Often it happens that there is also a model translation $\gamma$ in the opposite direction. In the example, one would totalize a partial model by adding one element to each carrier, representing "undefined". However, while $\beta$ can be formalized as a natural transformation, $\gamma$ is not always natural (see [44] for a counterexample). But naturality of $\beta$ is essential if we want to translate structured specifications along the institution representation (see Theorems 2.31, 2.35, 2.36, 2.43, 2.45, 2.49 and 2.51 below). This explains the direction of the model translation.

We also have a condition analogous to the satisfaction condition: we require that a translated model satisfies a sentence iff the original model satisfies the translated sentence. This is called the representation condition.

More formally, given institutions $I$ and $J$, a simple institution representation [74,54] (also called simple map of institutions [47]) $\mu = (\Phi, \alpha, \beta): I \rightarrow J$ consists of

- a functor $\Phi: \text{Sign}' \rightarrow \text{Pres}^I$,
- a natural transformation $\alpha: \text{Sen}' \rightarrow \text{Sen}' \circ \Phi$,
- a natural transformation $\beta: \text{Mod}^J \circ \Phi^{op} \rightarrow \text{Mod}'$

such that the following representation condition is satisfied for all $\Sigma \in \text{Sign}'$, $M' \in \text{Mod}^J(\Phi(\Sigma))$ and $\varphi \in \text{Sen}'(\Sigma)$:

$$M' \models_{\text{Sign}(\Phi(\Sigma))} \alpha_\Sigma(\varphi) \Leftrightarrow \beta_\Sigma(M') \models_{\text{Sign}'} \varphi.$$ 

In more detail, this means that each signature $\Sigma \in \text{Sign}'$ is translated to a presentation $\Phi(\Sigma) \in \text{Pres}^I$, and each signature morphism $\sigma: \Sigma \rightarrow \Sigma' \in \text{Sign}'$ is translated to a presentation morphism $\Phi(\sigma): \Phi(\Sigma) \rightarrow \Phi(\Sigma') \in \text{Pres}^I$.

Moreover, for each signature $\Sigma \in \text{Sign}'$, we have a sentence translation map $\alpha_\Sigma: \text{Sen}'(\Sigma) \rightarrow \text{Sen}'(\Phi(\Sigma))$ and a model translation functor $\beta_\Sigma: \text{Mod}^I(\Phi(\Sigma)) \rightarrow \text{Mod}'(\Sigma)$. Naturality of $\alpha$ and $\beta$ means that for any signature morphism

---

5 Meseguer [47] requires $\Phi: \text{Th}' \rightarrow \text{Th}'$, but since $\mu$ is simple, both formulations are equivalent using Meseguer’s $\alpha$-extension (except for the fact that we use presentations instead of theories).
\( \sigma : \Sigma \rightarrow \Sigma' \in \text{Sign}' \),

\[
\begin{array}{c}
\text{Sen'}(\Sigma) \xrightarrow{\alpha_{\Sigma}} \text{Sen'}(\Phi(\Sigma)) \\
\downarrow \quad \downarrow \\
\text{Sen'}(\sigma) \quad \text{Sen'}(\Phi(\sigma)) \\
\text{Sen'}(\Sigma) \xrightarrow{\alpha_{\Sigma'}} \text{Sen'}(\Phi(\Sigma')) \\
\end{array}
\]

and

\[
\begin{array}{c}
\text{Mod'}(\Sigma) \xleftarrow{\beta_{\Sigma}} \text{Mod'}(\Phi(\Sigma)) \\
\downarrow \quad \downarrow \\
\text{Mod'}(\sigma) \quad \text{Mod'}(\Phi(\sigma)) \\
\text{Mod'}(\Sigma) \xleftarrow{\beta_{\Sigma'}} \text{Mod'}(\Phi(\Sigma')) \\
\end{array}
\]

commute.

We illustrate this definition with some informal examples. They will be formally introduced in Section 4.

**Example 2.16** There is a simple institution representation going from equational logic to first-order logic with equality. An algebraic signature is translated to a first-order signature by just taking the set of predicates to be empty. Sentence translation is just inclusion of equations into first-order sentences. A first-order model with empty set of predicates is translated by just considering it as an algebra. □

**Example 2.17** Institution representations capture the encoding of a richer institution into a poorer one, as the following example (that has already been sketched above) shows: Define a simple institution representation going from partial first-order logic with equality to first-order logic with equality as follows: A partial first-order signature is translated to a total one by encoding each partial function symbol as a total one, plus a (new) unary predicate \( D \) ("definedness") and a (new) function symbol \( \bot \) ("undefined") for each sort (this means that \( \bot \) and \( D \) are heavily overloaded). Furthermore, we add axioms stating that \( D \) does not hold on \( \bot \), and that (encoded) total functions preserve ("totality") and reflect ("strictness") \( D \), while partial functions only reflect \( D \) (and the holding of predicates implies \( D \) to hold on the arguments). Sentence translation is done by replacing all partial function symbols by the total functions symbols encoding them, replacing strong equations by equations, existence equations by conjunctions of the equation and the definedness (using \( D \)) of one of the sides of the equation, replacing definedness with \( D \), and leaving predicate symbols as they are. For a given total model of the translated signature, we just take as carriers of the partial model the interpretations of the definedness predicates in the total model, while the total functions are
restricted to these new carriers, yielding partial functions. □

**Definition 2.18** Given a simple institution representation \(\mu = (\Phi, \alpha, \beta): I \rightarrow J\), we can extend it to a translation \(\hat{\mu}\) of structured specifications, extending the definition in [75,12]:

- If \(SP\) is a \(\Sigma\)-specification of form \(\langle \Sigma, \Gamma \rangle\), then
  \[
  \hat{\mu}(SP) = \langle \Sig(\Phi(\Sigma)), \Ax(\Phi(\Sigma)) \cup \alpha_\Sigma(\Gamma) \rangle.
  \]

- If \(SP\) is a \(\Sigma\)-specification of form \(SP_1 \cup SP_2\), then
  \[
  \hat{\mu}(SP) = \hat{\mu}(SP_1) \cup \hat{\mu}(SP_2).
  \]

- If \(SP\) is a \(\Sigma\)-specification of form **translate** \(SP'\) by \(\sigma: \Sigma \rightarrow \Sigma'\), then
  \[
  \hat{\mu}(SP) = (\text{translate } \hat{\mu}(SP') \text{ by } \Phi(\sigma)) \cup \Phi(\Sigma').
  \]

- If \(SP\) is a \(\Sigma\)-specification of form **derive from** \(SP'\) by \(\sigma\), then
  \[
  \hat{\mu}(SP) = \text{derive from } \hat{\mu}(SP') \text{ by } \Phi(\sigma).
  \]

- If \(SP\) is a \(\Sigma\)-specification of form **free** \(SP'\) along \(\sigma\), then
  \[
  \hat{\mu}(SP') = \text{free } \hat{\mu}(SP') \text{ along } \Phi(\sigma).
  \]

\(\square\)

Note that \(\Phi(\sigma)\) as used above is a morphism between presentations, but as such it is also a signature morphism. Further note that the axioms in \(\Ax(\Phi(\Sigma))\) are added for presentations and also for translations, since the latter lead to an extension of the signature (while union, **derive** and **free** do not extend the signature). This will be crucial for a good interaction with institution representations (Theorem 2.31).

We now come to the interaction of \(\hat{\mu}\) with normal forms.

**Proposition 2.19** Let \(\mu = (\Phi, \alpha, \beta): I \rightarrow J\) be an institution representation and \(SP\) a flattenable specification in \(I\). Then

\[
NF(\hat{\mu}(SP)) = \hat{\mu}(NF(SP)).
\]

**Proof.** By induction over the structure of \(SP\):

- \(SP = \langle \Sigma, \Gamma \rangle:\)
  \[
  NF(\hat{\mu}(SP)) = \hat{\mu}(SP) = \hat{\mu}(NF(SP)).
  \]
• \( SP = SP_1 \cup SP_2 \):  

Let \( NF(SP_i) = \langle \Sigma, \Gamma_i \rangle \) \((i = 1, 2)\). By induction hypothesis, \( NF(\hat{\mu}(SP_i)) = \hat{\mu}(NF(SP_i)) = \langle \text{Sig}(\Phi(\Sigma)), \text{Ax}(\Phi(\Sigma)) \cup \alpha_{\Sigma}(\Gamma_i) \rangle \). Therefore, \( NF(\hat{\mu}(SP)) = \langle \text{Sig}(\Phi(\Sigma)), \text{Ax}(\Phi(\Sigma)) \cup \alpha_{\Sigma}(\Gamma_1) \cup \alpha_{\Sigma}(\Gamma_2) \rangle \). But this is \( \hat{\mu}(NF(SP)) \).

• \( SP = \text{translate } SP_1 \text{ by } \sigma; \Sigma_1 \rightarrow \Sigma \):  

Let \( NF(SP_1) = \langle \Sigma_1, \Gamma_1 \rangle \), then \( NF(SP) = \langle \Sigma, \sigma(\Gamma_1) \rangle \). By induction hypothesis, \( NF(\hat{\mu}(SP_1)) = \hat{\mu}(NF(SP_1)) = \langle \text{Sig}(\Phi(\Sigma_1)), \text{Ax}(\Phi(\Sigma_1)) \cup \alpha_{\Sigma_1}(\Gamma_1) \rangle \).

From this, we get \( NF(\hat{\mu}(SP)) = NF(\text{translate } \Phi(\Sigma) \text{ by } \sigma; \Phi(\Sigma)) = \langle \text{Sig}(\Phi(\Sigma)), \Phi(\sigma)\langle \text{Ax}(\Phi(\Sigma_1)) \cup \alpha_{\Sigma_1}(\Gamma_1) \cup \text{Ax}(\Phi(\Sigma)) \rangle \rangle \). Since \( \Phi(\sigma)\langle \text{Ax}(\Phi(\Sigma_1)) \rangle \subseteq \text{Ax}(\Phi(\Sigma)) \) and \( \alpha \) is natural, this is equal to \( \langle \text{Sig}(\Phi(\Sigma)), \alpha_{\Sigma}(\sigma(\Gamma_1)) \cup \text{Ax}(\Phi(\Sigma)) \rangle \).

But this is just \( \hat{\mu}(NF(SP)) \). \( \square \)

In order to extend this result to non-flattenable specifications, we need a preparatory lemma:

**Lemma 2.20** Given institutions \( I \) and \( J \) and a functor \( \Phi: \text{Sign}^I \rightarrow \text{Sign}^J \) preserving pushouts and a pushout

\[
\begin{array}{ccc}
\Sigma & \xrightarrow{\sigma_1} & \Sigma_1 \\
\downarrow & & \downarrow \theta_2 \\
\Sigma & \xrightarrow{\sigma_2} & \Sigma_2 \\
\downarrow & & \downarrow \theta_1 \\
\Sigma' & & \Sigma'
\end{array}
\]

in \( \text{Sign}^I \), then

\[
\begin{array}{ccc}
\text{Sig}(\Phi(\Sigma)) & \xrightarrow{\Phi(\sigma_1)} & \text{Sig}(\Phi(\Sigma_1)) \\
\downarrow & & \downarrow \Phi(\theta_2) \\
\text{Sig}(\Phi(\Sigma')) & \xrightarrow{\Phi(\sigma_2)} & \text{Sig}(\Phi(\Sigma_2)) \\
\downarrow & & \downarrow \Phi(\theta_1) \\
\text{Sig}(\Phi(\Sigma')) & & \text{Sig}(\Phi(\Sigma'))
\end{array}
\]

is a pushout in \( \text{Sign}^J \), and

\[\text{Ax}(\Phi(\Sigma)) = \Phi(\theta_2)(\text{Ax}(\Phi(\Sigma_1))) \cup \Phi(\theta_1)(\text{Ax}(\Phi(\Sigma_2))).\]

**Proof.** Follows from the construction of colimits of theories, see \([27, 33] \). \( \square \)

We now generalize a result from \([12] \) to the case of institution representations that map signatures to theories (and not just signatures):

**Proposition 2.21** Let \( I \) and \( J \) be two institutions having weak amalgamation, and let \( \mu = (\Phi, \alpha, \beta) \) be an institution representation such that \( \Phi \) pre-
serves pushouts. Furthermore, let $SP$ be a specification not containing free. Then we have:

If $SP'$ is a normal form of $SP$, then $\hat{\mu}(SP')$ is a normal form of $\hat{\mu}(SP)$.

**Proof.** By induction over the structure of $SP$. Since the proof very much resembles the proof of Theorem 5.14 in [12], we only treat the case of translation here.

If $SP = \text{translate } SP_0$ by $\sigma_1: \Sigma \rightarrow \Sigma_1$, let derive from $\langle \Sigma_2, \Gamma_2 \rangle$ by $\sigma_2$ be a normal form of $SP_0$. Now let

$$
\begin{array}{c}
\sigma_1 \\
\Sigma_1 \\
\theta_2 \ \ \ \theta_1
\end{array}
\begin{array}{c}
\sigma_2 \\
\Sigma_2 \\
\Sigma'
\end{array}
$$

be a pushout. Then $SP' = \text{derive from } \langle \Sigma', \theta_1(\Gamma_2) \rangle$ by $\theta_2$ is a normal form of $SP$, and $\hat{\mu}(SP')$ is

$$\text{derive from } \langle \text{Sig}(\Phi(\Sigma')), \text{Ax}(\Phi(\Sigma')) \cup \alpha_{\Sigma'}(\theta_1(\Gamma_2)) \rangle \text{ by } \Phi(\theta_2).$$

By induction hypothesis, $\hat{\mu}(\text{derive from } \langle \Sigma_2, \Gamma_2 \rangle \text{ by } \sigma_2) =

$$\text{derive from } \langle \text{Sig}(\Phi(\Sigma_2)), \text{Ax}(\Phi(\Sigma_2)) \cup \alpha_{\Sigma_2}(\Gamma_2) \rangle \text{ by } \Phi(\sigma_2)$$

is a normal form of $\hat{\mu}(SP_0)$. By Lemma 2.20,

$$\text{derive from } \langle \text{Sig}(\Phi(\Sigma')), \Phi(\theta_1)(\text{Ax}(\Phi(\Sigma_2)) \cup \alpha_{\Sigma_2}(\Gamma_2)) \cup \Phi(\theta_2)(\text{Ax}(\Phi(\Sigma_1))) \rangle \text{ by } \Phi(\theta_2)$$

is a normal form of $(\text{translate } \hat{\mu}(SP_0) \text{ by } \Phi(\sigma_1)) \cup \Phi(\Sigma_1) = \hat{\mu}(SP)$. Since we have $\Phi(\theta_1)(\alpha_{\Sigma_2}(\Gamma_2)) = \alpha_{\Sigma'}(\theta_1(\Gamma_2))$ by naturality of $\alpha$ and $\text{Ax}(\Phi(\Sigma')) = \Phi(\theta_2)(\text{Ax}(\Phi(\Sigma_1)) \cup \Phi(\theta_1)(\text{Ax}(\Phi(\Sigma_2))))$ by Lemma 2.20, this normal form is just $\hat{\mu}(SP')$. \hfill \square

Occasionally, we also need the notion of conjunctive institution representation.

Let $\mathcal{P}_{\text{fin}}: \text{Set} \rightarrow \text{Set}$ be the covariant finite powerset functor, mapping each set to the set of its finite subsets, and each function $f$ to the function taking a finite subset to its image along $f$. Given institutions $I$ and $J$, a conjunctive institution representation [54] $\mu = (\Phi, \alpha, \beta): I \rightarrow J$ consists of

- a functor $\Phi: \text{Sign}' \rightarrow \text{Pres}'$,
• a natural transformation \( \alpha : \text{Sen}^I \to \mathcal{P}_{\text{fin}} \circ \text{Sen}^I \circ \Phi \), and

• a natural transformation \( \beta : \text{Mod}^I \circ \Phi^{op} \to \text{Mod}^I \),

such that the representation condition is satisfied for all \( \Sigma \in \text{Sign}^I \), \( M' \in \text{Mod}^I(\Phi(\Sigma)) \) and \( \varphi \in \text{Sen}^I(\Sigma) \):

\[
M' \models^I_{\text{Sig}(\Phi(\Sigma))} \alpha_\Sigma(\varphi) \iff \beta_\Sigma(M') \models^I \varphi.
\]

It is also possible to combine the sentence-theoretic property of being conjunctive with the other, model-theoretic properties of representations that we are going to introduce in the following subsections.

### 2.4.1 Borrowing

An important use of institution representations is the re-use (also called borrowing) of proof calculi and theorem provers.

**Definition 2.22** Let \( \mu = (\Phi, \alpha, \beta) : I \to J \) be an institution representation and \( \mathcal{SP} \) a class of \( I \)-specifications. We say that \( \mu \) admits borrowing of entailment for \( \mathcal{SP} \), if for any \( \Sigma \)-specification \( SP \in \mathcal{SP} \) and any \( \Sigma \)-sentence \( \varphi \) in \( I \), we have

\[
\models^I_\Sigma \varphi \text{ iff } \mu(SP) \models^I_{\text{Sig}(\Phi(\Sigma))} \alpha_\Sigma(\varphi).
\]

Moreover, we say that \( \mu \) admits borrowing of refinement for \( \mathcal{SP} \), if for any \( \Sigma \)-specifications \( SP_1, SP_2 \in \mathcal{SP} \), we have

\[
SP_1 \approx \approx SP_2 \text{ iff } \mu(SP_1) \approx \approx \mu(SP_2).
\]

The importance of this definition lies in the following: If we have a sound proof calculus for entailment in \( J \), and if we have an institution representation \( \mu : I \to J \) admitting borrowing of entailment for \( \mathcal{SP} \), we can use the proof calculus also for proving entailment concerning \( I \)-specifications in \( \mathcal{SP} \); we just have to translate our proof goals using \( \mu \) and \( \alpha \). If, moreover, the proof calculus is complete for proving entailment in \( J \), then also its re-use for proving entailment in \( I \) is complete. A similar remark holds for proof calculi for refinement.

Since

\[
SP \models^I_\Sigma \varphi \text{ iff } \langle \Sigma, \{ \varphi \} \rangle \approx \approx SP,
\]

\( \mu \) admits borrowing of entailment for \( \mathcal{SP} \) if \( \mu \) admits borrowing of refinement for \( \mathcal{SP} \) (provided that \( \mathcal{SP} \) contains all specifications of form \( \langle \Sigma, \{ \varphi \} \rangle \)).

In the next subsections, we will study conditions under which an institution representation admits borrowing. We therefore introduce various properties of institution representations. A first property is the model expansion property:
Definition 2.23  A simple institution representation \((\Phi, \alpha, \beta)\) admits model expansion if \(\beta\) is pointwise surjective on objects (i.e., each \(\beta_{\Sigma}\) is surjective on objects). \(\Box\)

Example 2.24  The institution representations from Examples 2.16 and 2.17 admit model expansion. For the former one, this is trivial. For the latter one, any partial model can be completed to a total model by adding one element to each carrier (as interpretation of \(\bot\)), representing “undefined”, which is a fixpoint of all functions, while predicates do not hold on it. Then, this one-point completion just generates the original model via the model translation. \(\Box\)

Proposition 2.25  Let \(\mu = (\Phi, \alpha, \beta) : I \to J\) be an institution representation and \(\mathcal{SP}\) a class of \(I\)-specifications.

1. Assume that for each \(SP \in \mathcal{SP}\), \(\text{Mod}^I(SP) = \beta_{\text{Sig}(SP)}(\text{Mod}^J(\mu(SP)))\). Then \(\mu\) admits borrowing of entailment for \(\mathcal{SP}\).
2. Assume that \(\mu\) admits model expansion and that for each \(SP \in \mathcal{SP}\), \(\beta_{\text{Sig}(SP)}^{-1}(\text{Mod}^I(SP)) = \text{Mod}^I(\mu(SP))\). Then \(\mu\) admits borrowing of entailment and of refinement for \(\mathcal{SP}\).
3. The assumption in (1) can be weakened to \(\beta_{\text{Sig}(SP)}(\text{Mod}^I(\mu(SP))) \subseteq \text{Mod}^I(SP)\), if additionally the simultaneous restriction and corestriction \(\beta_{\text{Sig}(SP)} : \text{Mod}^I(\mu(SP)) \to \text{Mod}^I(SP)\) is isomorphism-dense, and satisfaction in \(I\) is closed under isomorphism.
4. The assumption of model expansion in (2) can be replaced by assuming that \(\beta\) is pointwise isomorphism-dense and for each \(SP \in \mathcal{SP}\), \(\text{Mod}^I(SP)\) is isomorphism-closed.

Proof. (1) Let \(SP \in \mathcal{SP}\) be a \(\Sigma\)-specification and \(\varphi\) be a \(\Sigma\)-sentence. Then

\[ SP \models_{\Sigma} \varphi \]

iff (by definition) \(M \in \text{Mod}^I(SP)\) implies \(M \models_{\Sigma} \varphi\).

iff (by the assumption) \(M' \in \text{Mod}^I(\mu(SP))\) implies \(\beta_{\Sigma}(M') \models_{\Sigma} \varphi\).

iff (by the representation condition)

\[ M' \in \text{Mod}^I(\mu(SP)) \]

implies \(M' \models_{\text{Sig}(\Phi(\Sigma))}^J \alpha_{\Sigma}(\varphi)\).

iff (by definition) \(\mu(SP) \models_{\text{Sig}(\Phi(\Sigma))}^J \alpha_{\Sigma}(\varphi)\).

(2) Concerning borrowing of entailment, by surjectivity of \(\beta_{\Sigma}\), we obtain \(\beta_{\Sigma}(\beta_{\Sigma}^{-1}(M)) = M\) for any \(M \subseteq \text{Mod}^I(\Sigma)\). Thus, we obtain that the assumption of (1) is fulfilled, and the result follows.

---

6 This includes the condition that \(\beta_{\text{Sig}(SP)}\) is defined on \(\text{Mod}^I(\mu(SP))\).

7 This precisely means \(\beta_{\text{Sig}(SP)}(M')\) is defined and a member of \(\text{Mod}^I(SP)\) if and only if \(M' \in \text{Mod}^I(\mu(SP))\).
Concerning borrowing of refinement, let $SP_1, SP_2 \in \mathcal{SP}$ be $\Sigma$-specifications. Assume that $SP_1 \rightarrow \rightarrow \rightarrow SP_2$. If now $M' \in \text{Mod}^I(\tilde{\mu}(SP_2))$, by the assumption of the proposition, we get $\beta_\Sigma(M') \in \text{Mod}^I(SP_2)$ and therefore $\beta_\Sigma(M') \in \text{Mod}^I(SP_1)$. Again by the assumption of the proposition, we obtain $M' \in \text{Mod}^I(\tilde{\mu}(SP_1))$. Hence, $\tilde{\mu}(SP_1) \rightarrow \rightarrow \rightarrow \tilde{\mu}(SP_2)$.

Conversely, assume that $\tilde{\mu}(SP_1) \rightarrow \rightarrow \rightarrow \tilde{\mu}(SP_2)$. If now $M \in \text{Mod}^I(SP_2)$, by the assumptions of the proposition, we get some $M' \in \text{Mod}^I(\tilde{\mu}(SP_2))$ with $\beta_\Sigma(M') = M$. Since then also $M' \in \text{Mod}^I(\tilde{\mu}(SP_1))$, by the assumption of the proposition also $\beta_\Sigma(M') = M \in \text{Mod}^I(SP_1)$. Hence, $SP_1 \rightarrow \rightarrow \rightarrow SP_2$.

(3) In the step of the proof of (1) where we use the assumption, we now only get some $M' \in \text{Mod}^I(\tilde{\mu}(SP))$ with $\beta_{\text{Sig}}(M') \cong M$, instead of $\beta_{\text{Sig}}(M') = M$. But this does no harm since satisfaction in $I$ is closed under isomorphism.

(4) Similarly as (3). □

Borrowing of entailment is strictly weaker than borrowing of refinement, see [12] for an example.

2.4.2 Borrowing For Flat and Flattenable Specifications

For representations admitting model expansion, the well-known “Borrowing theorem” [18,74] holds:

**Theorem 2.26** Let $I$ and $J$ be two institutions and $\mu = (\Phi, \alpha, \beta): I \rightarrow J$ be an institution representation admitting model expansion. Then $\mu$ admits borrowing of entailment and refinement for flat specifications.

**Proof.** Let $SP = \langle \Sigma, \Gamma \rangle$ be a flat specification. Then $\beta_\Sigma(M')$ is defined and satisfies $SP$ iff $\beta_\Sigma(M')$ is defined and satisfies $\Gamma$ (by the representation condition) $M'$ satisfies $\text{Ax}(\Phi(\Sigma)) \cup \alpha_\Sigma(\Gamma)$ iff $M'$ satisfies $\tilde{\mu}(SP)$. Thus $\beta_\Sigma^{-1}\text{Mod}^I(SP) = \text{Mod}^I(\tilde{\mu}(SP))$. The result now follows from Proposition 2.25 (2). □

**Corollary 2.27** Let $I$ and $J$ be two institutions and $\mu = (\Phi, \alpha, \beta): I \rightarrow J$ be an institution representation admitting model expansion. Then $\mu$ admits borrowing of entailment and refinement for flattenable specifications.

**Proof.** By Proposition 2.19 and Fact 2.6.

That is, if we have a sound (and complete) theorem prover for flat(tenable) specifications in the target institution of an institution representation admitting model expansion, we can re-use it as a sound (and complete) theorem prover for flat(tenable) specifications in the source institution. In a word:
Institution representations admitting model expansion also admit borrowing of entailment and refinement for flat(lenable) specifications.

2.4.3 Borrowing For Structured Specifications (Excluding free)

If we want to re-use theorem provers not only for flat, but also for structured specifications (excluding free, which will be studied in the next section), we have to assume some additional property:

**Definition 2.28** [12] Let \( \mu = (\Phi, \alpha, \beta): I \rightarrow J \) be a simple institution representation and let \( \mathcal{D} \) be a class of signature morphisms in \( I \). Then \( \mu \) is said to have the weak \( \mathcal{D} \)-amalgamation property, if for each signature morphism \( \sigma: \Sigma_1 \rightarrow \Sigma_2 \in \mathcal{D} \), the diagram

\[
\begin{array}{c}
\text{Mod}^I(\Sigma_2) \xleftarrow{\beta_{\Sigma_2}} \text{Mod}^J(\Phi(\Sigma_2)) \\
\text{Mod}^I(\sigma) \downarrow \quad \downarrow \text{Mod}^J(\Phi(\sigma)) \\
\text{Mod}^I(\Sigma_1) \xleftarrow{\beta_{\Sigma_1}} \text{Mod}^J(\Phi(\Sigma_1))
\end{array}
\]

admits weak amalgamation, i.e. any for any two models \( M_2 \in \text{Mod}^I(\Sigma_2) \) and \( M'_1 \in \text{Mod}^J(\Phi(\Sigma_1)) \) with \( M_2|\sigma = \beta_{\Sigma_1}(M'_1) \), there is some \( M'_2 \in \text{Mod}^J(\Phi(\Sigma_2)) \) with \( \beta_{\Sigma_2}(M'_2) = M_2 \) and \( M'_2|\phi(\sigma) = M'_1 \). \( \square \)

**Example 2.29** The institution representation from Example 2.16 trivially satisfies the weak \( \mathcal{D} \)-amalgamation property, where \( \mathcal{D} \) is the class of all signature morphisms in \( E_{eq} \), since the model translations \( \beta_\Sigma \) are isomorphisms. \( \square \)

**Example 2.30** Let \( \mathcal{D} \) be the class of all injective signature morphisms in \( PFO\!L^\omega \). Weak \( \mathcal{D} \)-amalgamation for the institution representation from Example 2.17 can be seen as follows: Let \( \sigma: \Sigma_1 \rightarrow \Sigma_2 \in \mathcal{D} \), let \( M_2 \) be a \( \Sigma_2 \)-model and \( M'_1 \) be a \( \Phi(\Sigma_1) \)-model such that \( M_2|\sigma = \beta_{\Sigma_1}(M'_1) \). Extend \( M'_1 \) to a \( \Phi(\Sigma_2) \)-model \( M'_2 \) as follows: For any sort \( s \) not in the image of \( \sigma \), let the carrier for sort \( \sigma(s) \) in \( M'_2 \) just be \( (M_2)_s \uplus \{*\} \), and let \( D_{M'_2} \) hold everywhere except on \(*\). \( \bot \) is interpreted as \(* \) in \( M'_2 \). Given a function symbol \( f \) in \( \Sigma \) outside the image of \( \sigma \), \( \sigma(f) \) is interpreted in \( M'_2 \) to be \( g_{M_2} \), except that the interpretation of \( \bot \) is delivered if the argument is outside \( M_2 \) or the result is not defined due to partiality of the function. Given a predicate symbol \( p \) in \( \Sigma \) outside the image of \( \sigma \), \( \sigma(p) \) is interpreted in \( M'_2 \) to be \( M_{p_{M_2}} \), except that it is false if the argument is outside \( M_2 \). Then, we have that \( \beta_{\Sigma_2}(M'_2) = M_2 \) and \( M'_2|\phi(\sigma) = M'_1 \), showing weak \( \mathcal{D} \)-amalgamation. \( \square \)

The following theorem has been proved in [12]:

24
**Theorem 2.31** Let $I$ and $J$ be two institutions, and let $\mu = (\Phi, \alpha, \beta): I \to J$ be an institution representation. Let $\mathcal{SP}$ be the set of structured specifications in $I$ not containing **free**, and containing **derives** only along morphisms in $\mathcal{D}$. Then

1. For any specification $SP$ not containing **free**,
   \[ \beta_{\Sigma_{\text{Sig}(SP)}}(\text{Mod}^I(\tilde{\mu}(SP))) \subseteq \text{Mod}^I(SP). \]
2. If $\mu$ has the weak $\mathcal{D}$-amalgamation property, then for $SP \in \mathcal{SP}$,
   \[ \beta_{\Sigma_{\text{Sig}(SP)}}^{-1}(\text{Mod}^I(SP)) = \text{Mod}^I(\tilde{\mu}(SP)). \]
3. If $\mu$ admits model expansion and has the weak $\mathcal{D}$-amalgamation property, then for $SP \in \mathcal{SP}$,
   \[ \beta_{\Sigma_{\text{Sig}(SP)}}(\text{Mod}^I(\tilde{\mu}(SP))) = \text{Mod}^I(SP). \]
4. If $\mu$ admits model expansion and has the weak $\mathcal{D}$-amalgamation property, then $\mu$ admits borrowing of entailment and refinement for $\mathcal{SP}$. \( \square \)

The combination of (2) and (3) means that model classes (loose semantics) are preserved and reflected. This is especially important since CASL (like many other specification languages) has a model-theoretic semantics; a specification denotes a signature together with a class or category of models.

*Loose semantics for structured specifications without **free** and with hiding only along $\mathcal{D}$-morphisms can be lifted along and against institution representations admitting model expansion and weak $\mathcal{D}$-amalgamation.*

Loose semantics in this context means taking just the class of all models of a specification as its semantics. Of course, whether these are, e.g., all first-order models or just the finitely generated ones, depends on the model functor of the institution.

(4) means that tools for theorem proving within structured specifications in the target institution can be re-used for theorem proving within structured specifications in the source institution. That is:

*Institution representations admitting model expansion and weak $\mathcal{D}$-amalgamation admit borrowing of entailment and refinement for structured specifications without **free** and with hiding only along $\mathcal{D}$-morphisms.*

**Definition 2.32** A simple institution representation $\mu = (\Phi, \alpha, \beta): I \to J$ is said to be model-bijective if for each $\Sigma \in \text{Sign}^I$, $\beta^I_\Sigma$ is a bijection on objects. \( \square \)

**Example 2.33** The institution representation from Example 2.16 is model-bijective, while that from Example 2.17 is not: for a given partial model, it is
possible to add any number of “undefined” elements in a model representing it. Hence, the model translation is not bijective. \( \square \)

**Proposition 2.34** Any model-bijection institution representation both admits model expansion and has the weak \( \mathcal{D} \)-amalgamation property for arbitrary \( \mathcal{D} \).

**Proof.** Admitting of model expansion is immediate. Concerning weak \( \mathcal{D} \)-amalgamation, let \( \sigma : \Sigma_1 \rightarrow \Sigma_2 \) be a signature morphism and let \( M_2 \in \text{Mod}^I(\Sigma_2) \) and \( M'_1 \in \text{Mod}^I(\Phi(\Sigma_1)) \) such that \( M_2|\sigma = \beta_{\Sigma_1}(M'_1) \). Then \( \beta_{\Sigma_1}^{-1}(M_2) \) is the desired model \( M'_2 \in \text{Mod}^I(\Phi(\Sigma_2)) \) with \( \beta_{\Sigma_2}(M'_2) = M_2 \) and \( M'_2|_{\Phi(\sigma)} = M'_1 \). \( \square \)

The above slogans can now be shortened:

*Loose semantics for structured specifications without free can be lifted along and against model-bijection institution representations.*

*Model-bijection institution representations admit borrowing of entailment and refinement for structured specifications without free.*

In [12], it is shown that admitting model expansion and weak \( \mathcal{D} \)-amalgamation are necessary to get Theorem 2.31. However, if we restrict ourselves to borrowing of entailment, we can replace the requirement that derives in the specification are along morphisms in \( \mathcal{D} \)-amalgamation by the corresponding requirement for translates.

**Theorem 2.35** Let \( I \) and \( J \) be two institutions, and let \( \mu = (\Phi, \alpha, \beta) : I \rightarrow J \) be an institution representation admitting both model expansion and weak \( \mathcal{D} \)-amalgamation. Let \( \mathcal{SP} \) be the set of structured specifications \( SP \) in \( I \) satisfying the following properties:

- \( SP \) does not contain free,
- all translates in \( SP \) are either flattenable (i.e. do not contain derive), or the translation is along a morphism in \( \mathcal{D} \), and
- all unions in \( SP \) are flattenable (i.e. do not contain derive).

Then

1. For \( SP \in \mathcal{SP} \),

\[
\text{Mod}^I(\Sigma) = \beta_{\Sigma_2}(\text{Mod}^I(\tilde{\mu}(\Sigma))).
\]

2. \( \mu \) admits borrowing of entailment for \( \mathcal{SP} \).

**Proof.** (1) By induction over the structure of \( SP \). Note that by Theorem 2.31(1), we need only prove the inclusion from left to right.
\( \bullet \) \( SP = \langle \Sigma, \Gamma \rangle \): From the proof of Theorem 2.26, we get \( \beta_{\Sigma}^{-1} \text{Mod}^I(SP) = \text{Mod}^I(\hat{\mu}(SP)) \). By model expansion, we get \( \text{Mod}^I(SP) = \beta_{\Sigma}(\text{Mod}^I(\hat{\mu}(SP))) \).

\( \bullet \) \( SP = SP_1 \cup SP_2 \): Since \( SP \) is flattenable, by Proposition 2.19 and Fact 2.6, we can refer to the case of presentations.

\( \bullet \) \( SP = \text{translate} \ SP_1 \text{ by } \sigma : \Sigma_1 \rightarrow \Sigma \): If \( SP \) is flattenable, by Proposition 2.19 and Fact 2.6, we can refer to the case of presentations.

Otherwise, \( \sigma \in D \). For \( M \in \text{Mod}^I(SP) \), we have \( M|\sigma \in \text{Mod}^I(SP_1) \).

By the induction hypothesis, there is some \( M' \in \text{Mod}^I(\hat{\mu}(SP_1)) \) with \( \beta_{\Sigma_1}(M') = M|\sigma \). By the weak \( D \)-amalgamation property, there is some \( M' \in \text{Mod}^I(\Phi(\Sigma)) \) with \( M'|_{\Phi(\sigma)} = M'_1 \) and \( \beta_{\Sigma}(M') = M \). By \( M'|_{\Phi(\sigma)} = M'_1 \), \( M' \in \text{Mod}^I(\text{translate} \ \hat{\mu}(SP_1) \text{ by } \Phi(\sigma) \cup \Phi(\Sigma)) = \text{Mod}^I(\hat{\mu}(SP)) \). Hence, \( M \in \beta_{\Sigma}(\text{Mod}^I(\hat{\mu}(SP))) \).

\( \bullet \) \( SP = \text{derive from} \ SP_1 \text{ by } \sigma : \Sigma \rightarrow \Sigma_1 \): For \( M \in \text{Mod}^I(SP) \), we have some \( M'_1 \in \text{Mod}^I(SP_1) \) with \( M|_{\sigma} = M \). By induction hypothesis, we get some \( M'_1 \in \text{Mod}^I(\hat{\mu}(SP_1)) \) with \( \beta_{\Sigma_1}(M'_1) = M_1 \). This implies that \( M'_1|_{\Phi(\sigma)} \in \text{Mod}^I(\hat{\mu}(SP)) \). But \( \beta_{\Sigma}(M'_1|_{\Phi(\sigma)}) = \beta_{\Sigma_1}(M'_1)|_{\sigma} = M_1|_{\sigma} = M \). Hence, \( M \in \beta_{\Sigma}(\text{Mod}^I(\hat{\mu}(SP))) \).

(2) Follows from (1) by Proposition 2.25(1). \( \square \)

**Theorem 2.36** Let \( I \) and \( J \) be two institutions having weak amalgamation, and let \( \mu = (\Phi, \alpha, \beta): I \rightarrow J \) be an institution representation admitting model expansion such that \( \Phi \) preserves pushouts. Let \( SP \) be the set of structured specifications in \( I \) not containing \texttt{free}. Then

(1) For \( SP \in SP \),

\[
\text{Mod}^I(SP) = \beta_{\text{Sig}(SP)}(\text{Mod}^J(\hat{\mu}(SP))).
\]

(2) \( \mu \) admits borrowing of entailment for \( SP \).

**Proof**. (1) Let \( SP' \) be a normal form of \( SP \) according to Definition 2.8. By Proposition 2.21, \( \hat{\mu}(SP') \) is a normal form for \( \hat{\mu}(SP) \). Hence by Fact 2.9 it suffices to show \( \text{Mod}^I(SP') = \beta_{\text{Sig}(SP)}(\text{Mod}^J(\hat{\mu}(SP'))) \). Since normal forms neither contain \texttt{translate} nor unions, this follows from Theorem 2.35.

(2) Follows from (1) by Proposition 2.25(1). \( \square \)

### 2.4.4 Borrowing For Structured Specifications (Including \texttt{free})

It is easy to show borrowing for structured specifications including \texttt{free} under the very strong assumptions of a subinstitution representation:

**Definition 2.37** A simple institution representation \( \mu = (\Phi, \alpha, \beta): I \rightarrow J \) is said to be a subinstitution representation if \( \Phi \) is an embedding of categories,
\( \alpha \) is a pointwise injection, and \( \beta \) is a natural isomorphism.

\( I \) is said to be a subinstitution of \( J \) if there is a subinstitution representation from \( I \) to \( J \).  \( \square \)

**Example 2.38** The institution representation from Example 2.16 is a subinstitution representation.  \( \square \)

**Theorem 2.39** Let \( \mu = (\Phi, \alpha, \beta): I \rightarrow J \) be a subinstitution representation. Then

- For any specification \( SP \),
  \[
  \beta_{\text{Sig}(SP)}^{-1}(\text{Mod}^I(SP)) = \text{Mod}^J(\check{\mu}(SP)).
  \]
- \( \mu \) admits borrowing of entailment and refinement for all specifications.

**Proof.** (1) is straightforward. (2) follows with Proposition 2.25(2).  \( \square \)

Thus

Substitution representations admit borrowing of entailment and refinement for all structured specifications.

The substitution property is a rather strong property. Is there a weaker property that suffices to ensure a good interaction with liberality and the free construct? The answer is the following notion, which we have introduced in [44] in a slightly stronger form under the name of categorical retractive simulation:

**Definition 2.40** A simple institution representation \( \mu = (\Phi, \alpha, \beta): I \rightarrow J \) is called persistently liberal if for each \( \Sigma \in \text{Sign}^I \), \( \beta_\Sigma \) has a left adjoint \( \gamma_\Sigma \) such that also \( \beta_\Sigma \circ \gamma_\Sigma \cong \text{id} \). If we have even \( \beta_\Sigma \circ \gamma_\Sigma = \text{id} \), then \( \mu \) is called strongly persistently liberal. We write \( (\mu, \gamma) \) if we want to chose a particular \( \gamma \).  \( \square \)

We here use the term liberal (in accordance with [24]) since it stresses the connection with liberality of institutions. Meseguer [49] has introduced persistently liberal representations under the name of extensions. He additionally requires that the isomorphism \( \text{id} \cong \beta_\Sigma \circ \gamma_\Sigma \) is the unit of the adjunction; however, in the light of Proposition A.1, this requirement is superfluous.

**Example 2.41** The institution representation from Examples 2.16 is strongly

\[ \text{Given a functor } G: B \rightarrow A, \text{ a functor } F: A \rightarrow B \text{ is called left adjoint to } G, \text{ if for each } A \in A, F(A) \text{ is } G\text{-free over } A, \text{ and, moreover, the universal arrows form a natural transformation } \eta: \text{Id} \rightarrow G \circ F. \]
persistently liberal: \( \gamma_\Sigma \) is just the inverse of the isomorphism \( \beta_\Sigma \). \( \square \)

**Example 2.42** The institution representation from Example 2.17 is strongly persistently liberal: \( \gamma_\Sigma \) totalizes a partial model by adding “undefined” values freely (this is the free completion [14,15]). \( \square \)

Let us now study how persistently liberal institution representations interact with liberality, strengthening a result of [44] (we can now drop the assumption of the existence of \( \text{Mod}^f(\sigma) \)-free models).

**Theorem 2.43** Let \( (\mu, \gamma) = ((\Phi, \alpha, \beta), \gamma): I \to J \) be a persistently liberal institution representation such that additionally either satisfaction in \( I \) is closed under isomorphism or \( (\mu, \gamma) \) is even strongly persistently liberal. Then free constructions can be lifted against \( (\mu, \gamma) \) in the following sense:

1. If \( \sigma: \langle \Sigma_1, \Gamma_1 \rangle \to \langle \Sigma_2, \Gamma_2 \rangle \) is a theory morphism in \( I \), \( M_1 \in \text{Mod}^f(\langle \Sigma_1, \Gamma_1 \rangle) \), and if \( M'_2 \in \text{Mod}^f(\langle \Sigma_2, \Gamma_2 \rangle) \) is \( \text{Mod}^f(\Phi(\sigma)) \)-free over \( \gamma_{\Sigma_1}(M_1) \), then \( \beta_{\Sigma_2}(M'_2) \) is \( \text{Mod}^f(\sigma) \)-free over \( M_1 \).
2. If \( \sigma: \langle \Sigma_1, \Gamma_1 \rangle \to \langle \Sigma_2, \Gamma_2 \rangle \) is a theory morphism in \( I \), then \( \sigma \) is liberal if \( \Phi(\sigma) \) is liberal.
3. \( I \) is liberal if \( J \) is liberal.
4. Let \( \mathcal{M} \) be a class of signature morphisms in \( \text{Sign}^f \). \( I \) is \( \mathcal{M} \)-liberal if \( J \) is \( \Phi(\mathcal{M}) \)-liberal.

In a word:

*Free constructions can be lifted against persistently liberal institution representations.*

**Proof.**

\[
\begin{array}{c}
\text{Mod}^f(\langle \Sigma_2, \Gamma_2 \rangle) \xrightarrow{\beta_{\Sigma_2}} \text{Mod}^f(\langle \Sigma_2, \Gamma_2 \rangle)
\end{array}
\]

\[
\begin{array}{c}
\text{Mod}^f(\langle \Sigma_1, \Gamma_1 \rangle) \xrightarrow{\beta_{\Sigma_1}} \text{Mod}^f(\langle \Sigma_1, \Gamma_1 \rangle)
\end{array}
\]

(1) By the representation condition, \( \beta_{\Sigma_i} \) restricts to \( \beta_{\Sigma_i}: \text{Mod}^f(\langle \Sigma_i, \Gamma_i \rangle) \to \text{Mod}^f(\langle \Sigma_i, \Gamma_i \rangle) \) for \( i = 1, 2 \). We now show that similarly, \( \gamma_{\Sigma_i} \) restricts to \( \gamma_{\Sigma_i}: \text{Mod}^f(\langle \Sigma_i, \Gamma_i \rangle) \to \text{Mod}^f(\langle \Sigma_i, \Gamma_i \rangle) \): For \( M \in \text{Mod}^f(\langle \Sigma_i, \Gamma_i \rangle) \), either \( \beta_{\Sigma_i}(\gamma_{\Sigma_i}(M)) = M \), or \( \beta_{\Sigma_i}(\gamma_{\Sigma_i}(M)) \not\cong M \) and satisfaction in \( I \) is closed under isomorphism. In both cases, \( \beta_{\Sigma_i}(\gamma_{\Sigma_i}(M)) \models_{\Sigma_i} \Gamma_i \), and by the representation condition, \( \gamma_{\Sigma_i}(M) \models_{\Sigma_i} \alpha_{\Sigma_i}(\Gamma_i) \), hence \( \gamma_{\Sigma_i}(M) \in \text{Mod}^f(\langle \Sigma_i, \Gamma_i \rangle) \). We now can apply Proposition A.2 (1) with \( U = \text{Mod}^f(\sigma) \), \( V = \text{Mod}^f(\Phi(\sigma)) \), \( R = \beta_{\Sigma_2} \), \( L = \gamma_{\Sigma_2} \), \( R' = \beta_{\Sigma_1} \), and \( L' = \gamma_{\Sigma_1} \), \( B = M'_2 \) and \( X = M_1 \).

29
(2), (3) and (4) directly follow from (1). □

Example 2.55 below shows that the assumption of persistent liberality is needed to get this result.

We now come to borrowing for specifications containing free.

**Definition 2.44** Given institutions \( I \) and \( J \), a persistently liberal institution representation \( (\mu, \gamma) = ((\Phi, \alpha, \beta), \gamma): I \to J \) and a signature morphism \( \sigma: \Sigma_1 \to \Sigma_2 \) in \( I \), \( \gamma \) is called \( \sigma \)-natural, if \( \gamma_{\Sigma_1} \circ \text{Mod}^I(\sigma) = \text{Mod}^J(\Phi(\sigma)) \circ \gamma_{\Sigma_2} \).

**Theorem 2.45** Let \( (\mu, \gamma) = ((\Phi, \alpha, \beta), \gamma): I \to J \) be a persistently liberal institution representation, such that additionally either satisfaction in \( I \) is closed under isomorphism or \((\mu, \gamma)\) is even strongly persistently liberal. Let \( SP \) consist of those \( I \)-specifications \( SP \) for which

- for each free \( SP' \) along \( \sigma: \Sigma \to \Sigma_1 \) occurring in \( SP \), either \( \Phi(\sigma): \Sigma \to SP' \) is strongly persistently liberal or \( \gamma \) is \( \sigma \)-natural, and moreover, \( \gamma_{\Sigma} \) is surjective on objects,
- for each derive from \( SP_1 \) by \( \sigma \) occurring in \( SP \), \( \gamma \) is \( \sigma \)-natural,
- for each translate \( SP_1 \) by \( \sigma \) occurring in \( SP \), either \( \gamma \) is \( \sigma \)-natural, or \( SP_1 \) is flattenable, i.e. contains neither free nor derive.

Then

(1) For \( SP \in SP \),

\[ \beta_{\text{Sig}(SP)}(\text{Mod}^I(\mu(\text{SP}))) \subseteq \text{Mod}^I(\text{SP}). \]

(2) For \( SP \in SP \),

\[ \gamma_{\text{Sig}(SP)}(\text{Mod}^I(\text{SP})) \subseteq \text{Mod}^I(\mu(\text{SP})). \]

(3) \( \mu \) admits borrowing of entailment for \( SP \).

**Proof.** We prove (1) and (2) by simultaneous induction over the structure of \( SP \).

For (1), all cases except from free are treated as in the proof of Lemma 8.6 in [12] (cf. our Theorem 2.31).

- \( SP = (\Sigma, \Gamma) \): (2) Assume that \( M \in \text{Mod}^I(\text{SP}) \), i.e. \( M \models_{\Sigma} \Gamma \). Now either \( M = \beta_{\Sigma}(\gamma_{\Sigma}(M)) \) or \( M \models_{\Sigma} \beta_{\Sigma}(\gamma_{\Sigma}(M)) \) and satisfaction in \( I \) is closed under isomorphism. In both cases, \( \beta_{\Sigma}(\gamma_{\Sigma}(M)) \models_{\Sigma} \Gamma \). By the representation condition, \( \gamma_{\Sigma}(M) \models_{\Sigma} \alpha_{\Sigma}(\Gamma) \). Since \( \gamma_{\Sigma}(M) \in \text{dom} \beta_{\Sigma} \), also \( \gamma_{\Sigma}(M) \models \text{Ax}(\Phi(\Sigma)) \). Hence, \( M \in \text{Mod}^J(\mu(\text{SP})) \).
- \( SP = SP_1 \cup SP_2 \): (2) follows directly from the induction hypothesis.
• **$SP = \text{translate } SP_1 \text{ by } \sigma: \Sigma_1 \rightarrow \Sigma$: (2) In the case that $SP_1$ (and hence $SP$) is flattenable, by Proposition 2.19 and Fact 2.6, we can refer to the case of presentations. Otherwise, let $M \in \text{Mod}^I(SP)_\sigma$, i.e. $M|_\sigma \in \text{Mod}^I(SP_1)$. By induction hypothesis, $\gamma_{\Sigma_1}(M|_\sigma) \in \text{Mod}^I(\mu(SP_1))$. By $\gamma$-naturality of $\sigma$, this is just $\gamma_{\Sigma}(M)|_{\Phi(\sigma)}$. Since also $\gamma_{\Sigma}(M) \in \text{dom } \beta_{\Sigma}$ and therefore $\gamma_{\Sigma}(M) \models \text{Ax}(\Phi(\Sigma))$, altogether $\gamma_{\Sigma}(M) \in \text{Mod}^I(\mu(SP))$.

• **$SP = \text{derive from } SP_1 \text{ by } \sigma: \Sigma \rightarrow \Sigma$: (2) Let $M \in \text{Mod}^I(SP)$. Then there is some $M' \in \text{Mod}^I(SP_1)$ with $M'|_\sigma = M$. By induction hypothesis, $\gamma_{\Sigma_1}(M') \in \text{Mod}^I(\mu(SP_1))$. Since by $\gamma$-naturality of $\sigma$, $\gamma_{\Sigma_1}(M')|_{\Phi(\sigma)} = \gamma_{\Sigma}(M)|_{\Phi(\sigma)}$, $\gamma_{\Sigma}(M) \in \text{Mod}^I(\mu(SP))$.

• **$SP = \text{free } SP' \text{ along } \sigma: \Sigma \rightarrow \Sigma$: By induction hypothesis, $\beta_{\Sigma_1}$ restricts to $\beta_{\Sigma_1}: \text{Mod}^I(\mu(SP')) \rightarrow \text{Mod}^I(SP')$ and $\gamma_{\Sigma_1}$ restricts to $\gamma_{\Sigma_1}: \text{Mod}^I(SP') \rightarrow \text{Mod}^I(\mu(SP'))$. We thus can apply Proposition A.2 in the sequel, with $U = \text{Mod}^I(\sigma)$, $V = \text{Mod}^I(\Phi(\sigma))$, $R = \beta_{\Sigma_1}$, $L = \gamma_{\Sigma_1}$, $R' = \beta_{\Sigma}$, and $L' = \gamma_{\Sigma}$.

  (1) Let $M \in \text{Mod}^I(\mu(SP))$, i.e. $M$ is strongly persistently $\text{Mod}^I(\Phi(\sigma))$-free. By the assumption that $\gamma_{\Sigma}$ is surjective on objects, we can apply Proposition A.2(3) to get that $\beta_{\Sigma_1}(M)$ is strongly persistently $\text{Mod}^I(\sigma)$-free, i.e. $\beta_{\Sigma_1}(M) \in \text{Mod}^I(SP)$.

  (2) Let $M \in \text{Mod}^I(SP)$. From $M$ being $\text{Mod}^I(\sigma)$-free over $M|_\sigma$, by Proposition A.2(2) we get that $\gamma_{\Sigma_1}(M)$ is $\text{Mod}^I(\Phi(\sigma))$-free over $\gamma_{\Sigma}(M)|_\sigma$. According to the assumption, there are two cases: If $\gamma$ is $\sigma$-natural, we can apply Proposition A.2(4) to get that $\gamma_{\Sigma_1}(M)$ is strongly persistently free. If on the other hand, $\Phi(\sigma): \Sigma \rightarrow SP'$ is strongly persistently liberal, then there is some $M'$ that is $\text{Mod}^I(\Phi(\sigma))$-free over $\gamma_{\Sigma}(M)|_\sigma = M'|_{\Phi(\sigma)}$ with the identity as unit. Since free objects are unique up to isomorphism, $\gamma_{\Sigma_1}(M) \cong M'$. Hence, the unit witnessing that $\gamma_{\Sigma_1}(M)$ is free is an isomorphism, and so $\gamma_{\Sigma_1}(M)$ is persistently free. In both cases, by Proposition 2.10, $\gamma_{\Sigma_1}(M) \in \text{Mod}^I(\mu(SP))$.

(3) Follows from (1) and (2) with Proposition 2.25(1) (in case that $(\mu, \gamma)$ is strongly persistently liberal) or Proposition 2.25(3) (if satisfaction in $I$ is closed under isomorphism). \hfill \Box

The following examples shows that the assumption of $\sigma$-naturality of $\gamma$ resp. strongly persistent liberality of $\Phi(\sigma)$ in Theorem 2.45 cannot be dropped.

**Example 2.46** Consider the strongly persistently liberal institution representation going from $PFOL_\Sigma$ to $FOL_\Sigma$ introduced in Section 4.1.4 below. Let $\sigma$ be the signature morphism that is the obvious inclusion of a signature $\Sigma_1$ with just one sort into a signature $\Sigma_2$ with additionally one partial operation symbol. Clearly, $\sigma$ is strongly persistently liberal: the free construction equips any set with the everywhere undefined function. Consequently, a $\Sigma_2$-model is a model of $\text{free } (\Sigma_2, \emptyset)$ along $\sigma$ iff the partial operation is everywhere undefined.

31
Now $\Phi(\sigma)$ is not persistently liberal: similarly to Example 2.14, any $\Phi(\Sigma_1)$-model is freely extended by adding an $\omega$-chain

$$\{ f(a), f(f(a)), f(f(f(a))), \ldots \}$$

over each of its elements $a$. Thus, a $\Phi(\Sigma_2)$-model can never be free over its own $\sigma$-reduct, and $\hat{\mu}(\text{free } (\Sigma_2, \emptyset) \text{ along } \sigma)$ is inconsistent. Since anything is implied by an inconsistent specification, obviously borrowing of entailment fails here. □

The following example shows that surjectivity of $\gamma_\Sigma$ (for $\Sigma$ occurring in “parameter position” in a free specification, i.e. in as in free $SP'$ along $\sigma: \Sigma \rightarrow \Sigma_1$) is really needed in Theorem 2.45.

**Example 2.47** Let $ColouredFOL^\equiv$ be a variant of $FOL^\equiv$ where each symbol in a signature can be coloured either red or blue. Define a strongly persistently liberal institution representation $((\Phi, \alpha, \beta, \gamma): ColouredFOL^\equiv \rightarrow FOL^\equiv$ as follows:

**Signatures** The sorts and operation symbols and non-unary predicate symbols of a $ColouredFOL^\equiv$-signature are left as they are, while unary predicate symbols are translated to binary ones. Moreover, for each blue unary predicate symbol, the corresponding binary relation is axiomatized to be transitive.

**Models** A binary relation encoding a unary one is mapped to its reflexive support (i.e. $\{x \mid R(x, x)\}$), while everything else is left unchanged by $\beta$. Vice versa, $\gamma_\Sigma$ maps a blue unary relation $R$ to $\{(x, x) \mid R(x)\}$. This easily is seen to be left adjoint to $\beta_\Sigma$.

**Sentences** Variables are mapped to pairs of variables (also quantification is over both variables in the pair), and hence, terms have two possible translations: one using the first components of the variable pairs, and the other using the second one. Usually only the first way of translating terms is used, except from the case of binary relations encoding unary ones: here, the binary relation is applied to both the translations.

**Satisfaction** The representation condition can easily be checked.

Now consider the inclusion $\sigma$ of $\Sigma_1$ (consisting of a sort $s$ and a red unary predicate symbol $p : s$) into $SP_2$ (consisting additionally of a blue unary predicate symbol $q : s$, and an axiom $\forall x : s \bullet p(x) \Rightarrow q(x)$). Further consider its translation $\Phi(\sigma): \Phi(\Sigma_1) \rightarrow \Phi(\Sigma_2)$, and consider the $\Phi(\Sigma_1)$-model with carrier $\{1; 2\}$ and relation $\{(1, 2); (2, 1)\}$. Its free extension along $\sigma$ is the $\Phi(\Sigma_2)$-model with the same carrier and interpretation of $p$ as $\{(1, 2); (2, 1)\}$ and of $q$ as $\{(1, 1); (1, 2); (2, 1); (2, 2)\}$. Obviously, this free extension is strongly persistently liberal, i.e. a model of $\hat{\mu}(\text{free } SP_2 \text{ along } \sigma)$. However, this model translated by $\beta$ gives a model with carrier $\{1; 2\}$ and interpretation of $p$ as $\emptyset$ and of $q$ as $\{1; 2\}$. But this is not free over its own $\sigma$-reduct, and therefore
not a model of **free** \( SP_2 \) along \( \sigma \). Still, the (even strongly persistent!) free model over this reduct exists; it interprets both \( p \) and \( q \) as \( \emptyset \). Hence, \( \beta \) does not preserve the semantics of specifications involving **free**. □

The examples show that unfortunately persistent liberality does not suffice for ensuring preservation of all specifications involving **free**\(^9\). We therefore introduce the following stronger notion from [44]:

**Definition 2.48** An institution representation \( \mu = (\Phi, \alpha, \beta) : I \rightarrow J \) is called an **institution embedding** if \( \Phi \) is an embedding and for each \( \Sigma \in \text{Sign}^I \), \( \alpha_\Sigma \) is injective and \( \beta_\Sigma \) is an equivalence of categories. □

Non-trivial examples of institution embeddings (i.e. other than subinstitution representations) are given in Theorems 10.3, 10.8, and 10.9.

**Theorem 2.49** Let \( I \) and \( J \) be institutions such that satisfaction in \( I \) is closed under isomorphisms, and let \( \mu = (\Phi, \alpha, \beta) : I \rightarrow J \) be an institution embedding. Let \( \mathcal{SP} \) consist of those \( I \)-specifications having neither **free** nor **derive** occurring within a **translate** or a union. Then

1. For \( SP \in \mathcal{SP} \),
   \[ \beta_{\text{Sig}(SP)}(\text{Mod}^I(\mu(\text{Sig}(SP)))) \subseteq \text{Mod}^I(SP). \]
2. For \( SP \in \mathcal{SP} \), \( \beta_{\text{Sig}(SP)} : \text{Mod}^I(\mu(S)) \rightarrow \text{Mod}^I(SP) \) is isomorphism-dense.
3. \( \mu \) admits borrowing of entailment for \( \mathcal{SP} \).

**Proof.** We simultaneously prove (1) and (2) by induction over the structure of \( SP \).

- **SP = \langle \Sigma, \Gamma \rangle:** (1) follows from Theorem 2.31(1). Concerning (2), assume \( M \in \text{Mod}^I(SP) \) (i.e. \( M \models_\Sigma \Gamma \)). Since \( \beta_\Sigma : \text{Mod}^J(\Phi(\Sigma)) \rightarrow \text{Mod}^J(\Sigma) \) is isomorphism-dense, there is some \( M' \in \text{Mod}^J(\Phi(\Sigma)) \) with \( \beta_\Sigma(M') \cong M \). Since satisfaction in \( I \) is closed under isomorphism, \( \beta_\Sigma(M') \models \Gamma \). By the satisfaction condition, \( M' \models_{\text{Sig}(\Phi(\Sigma))} \alpha_\Sigma(\Gamma) \). Hence, \( M' \in \text{Mod}^J(\mu(SP)) \).
- **SP = \text{translate} \ SP_1 \cup SP_2:** By assumption, \( SP \) is flattenable. By Proposition 2.19 and Fact 2.6, we can refer to the case of presentations.
- **SP = \text{translate} SP_1 \ by \ \sigma : \Sigma_1 \rightarrow \Sigma:** Here, the same remark as for unions applies.
- **SP = \text{derive from} \ SP_1 \ by \ \sigma : \Sigma \rightarrow \Sigma_1:** (1) is proved by the same inductive argument as used in the proof of Lemma 8.6 in [12] (cf. our Theorem 2.31).\(^9\)

\(^9\) It seems that instead of left adjoint right-inverses to the model translations, we need right adjoint left-inverses. We also have developed a notion of bi-liberality with useful results. The practical examples of this, however, are all equivalences of categories.
Concerning (2), for \( M \in \text{Mod}^f(SP) \), we have some \( M_1 \in \text{Mod}^f(SP_1) \) with \( M_1|_\sigma = M \). By induction hypothesis, we get some \( M'_1 \in \text{Mod}^f(\bar{\mu}(SP_1)) \) with \( \beta_{\Sigma_1}(M'_1) \cong M_1 \). By definition, \( M'_1|_{\phi(\sigma)} \in \text{Mod}^f(\bar{\mu}(SP)) \). Then we get \( \beta_\Sigma(M'_1|_{\phi(\sigma)}) = \beta_{\Sigma_1}(M'_1)|_{\sigma} \cong M_1|_{\sigma} = M \).

- \( SP = \text{free } SP' \) along \( \sigma: \Sigma \to \Sigma_1 \): By induction hypothesis, we have that \( \beta_\Sigma: \text{Mod}^f(\bar{\mu}(SP')) \to \text{Mod}^f(SP') \) is isomorphism-dense. \( \text{Mod}^f(SP') \) is a full subcategory of \( \text{Mod}^f(\Sigma) \), \( \text{Mod}^f(\bar{\mu}(SP')) \) is a full subcategory of \( \text{Mod}^f(\Phi(\Sigma)) \) and \( \beta_\Sigma: \text{Mod}^f(\Phi(\Sigma)) \to \text{Mod}^f(\Sigma) \) is full and faithful. Therefore \( \beta_\Sigma: \text{Mod}^f(\bar{\mu}(SP')) \to \text{Mod}^f(SP') \) is full and faithful as well. Hence, it is an equivalence of categories (and it has an inverse up to natural isomorphism, which also is an equivalence of categories). The result now follows by the fact that universal arrows (and isomorphisms) are preserved by equivalences of categories, together with Proposition 2.10.

(3) Follows from (1) and (2) with Proposition 2.25(3). \( \square \)

The following example shows that the restriction to translates not containing derives is really necessary in Theorem 2.49:

**Example 2.50** Consider the reduction \( \mu \) of the institution \( I \) of order-sorted algebra to the institution \( J \) of many-sorted algebra in Theorem 10.3. While in order-sorted algebra, subsorts are interpreted as set-theoretic inclusions, the coding in many-sorted algebra uses injective functions. The model translation \( \beta \) takes an injection model and uses a colimit of a diagram built up from the injections to generate an inclusion model. This is an institution embedding. Now let \( \Sigma_1 \) be the \( I \)-signature

\begin{align*}
\textbf{sorts} & \quad s1, s2, s3 < t; \\
& \quad s1 < s2; \\
& \quad s2 < s3; \\
\textbf{ops} & \quad f : s2 \to s1; \\
& \quad g : s3 \to s2;
\end{align*}

and \( \Sigma_2 \) be

\begin{align*}
\textbf{sorts} & \quad s1, s2 < t; \\
& \quad s1 < s2; \\
\textbf{ops} & \quad id1 : s1 \to s1; \\
& \quad f : s2 \to s1; \\
& \quad id2 : s2 \to s2;
\end{align*}

There are two obvious signature morphisms \( \sigma_1, \sigma_2: \Sigma_1 \to \Sigma_2 \). \( \sigma_1(t) = \sigma_2(t) = t, \quad \sigma_1(s1) = \sigma_2(s2) = \sigma_1(s2) = s1, \quad \sigma_1(s3) = \sigma_2(s3) = s2, \quad \sigma_1(f) = id1, \quad \sigma_1(g) = \sigma_2(f) = f, \quad \sigma_2(g) = id2. \)

Let \( \Gamma_2 \) consist of the \( \Sigma_2 \)-axioms \( id1(x1) = x1, \quad id2(x2) = x2 \) and \( f(x2) = x2 \) in variables \( x1 : s1 \) and \( x2 : s2 \) (note that for \( f(x2) = x2 \), both sides are
implicitly injected in $t$). Let $SP$ be the specification

$$\text{translate (derive from } \langle \Sigma_2, \Gamma_2 \rangle \text{ by } \sigma_2 \text{) by } \sigma_1.$$

The peculiarity is now that for $M \in \text{Mod}^I(SP)$, we have $M_{s_1} = M_{s_2}$, but due to the use of subsort inclusions, this already holds for any $\langle \Sigma_2, \Gamma_2 \rangle$-model (while it does not hold for $\check{\mu}(\langle \Sigma_2, \Gamma_2 \rangle)$-models). Let $M' \in \text{Mod}^I(\check{\mu}(\langle \Sigma_2, \Gamma_2 \rangle))$ such that $M'_{s_1} \neq M'_{s_2}$. Then $M' \notin \text{Mod}^I(\check{\mu}(SP))$, but $\beta_{\Sigma_2}(M') \in \text{Mod}^I(SP)$, because isomorphic subsorts of the same sort are identified by $\beta$. Thus

$$\beta_{\text{Sig}(SP)}^{-1}(\text{Mod}^I(SP)) \neq \text{Mod}^I(\check{\mu}(SP)),$$

and if we would add sentences that can detect equality of sorts, $\mu$ would not admit borrowing of entailment for $SP$.

Borrowing of refinement does not work either: Let $\Gamma_1$ consist of the $\Sigma_1$-axioms $f(x_2) = x_2$ and $g(x_3) = x_3$ in variables $x_2 : s_2$ and $x_3 : s_3$. Then

$$\text{derive from } \langle \Sigma_2, \Gamma_2 \rangle \text{ by } \sigma_1 \rightsquigarrow \langle \Sigma_1, \Gamma_1 \rangle$$

but

$$\check{\mu}(\text{derive from } \langle \Sigma_2, \Gamma_2 \rangle \text{ by } \sigma_1) \nless \check{\mu}(\langle \Sigma_1, \Gamma_1 \rangle).$$

where the latter can be seen by considering a $\check{\mu}(\langle \Sigma_1, \Gamma_1 \rangle)$-model $M$ with $M_{s_1} \neq M_{s_2}$. □

The counterexample and the rather intricate conditions of Theorem 2.49 show that model translations which are equivalences of categories do not interact so well with structured specification. This has its reason in the fact that model classes need not be closed under isomorphism. Even if this property should hold for flat specifications, the problem still comes in with derive. Namely, in general the image of the reduct functor $\text{Mod}(\sigma)$ is not closed under isomorphism (the reason for this behaviour is that the functor is not transportable [1]). It would make much sense to close up model classes of specifications under isomorphism. (Indeed, this would need to be done only for presentations and hidings — the other structuring operations preserve closedness under isomorphism). Then, Theorem 2.49 would hold for all specifications, and moreover, borrowing of proof calculi also for refinement would become applicable, as it is for subinstitution representations (cf. Theorem 2.39). However, we have refrained from closing up model classes under isomorphism, since this is neither standard in the literature about structured specifications, nor it is implemented in the semantics of CASL [23].

If we omit derive, things behave much better:

**Theorem 2.51** Let $I$ and $J$ be institutions such that satisfaction in $I$ is closed under isomorphisms, and let $\mu = (\Phi, \alpha, \beta): I \rightarrow J$ be an embedding. Let $SP$ consist of those $I$-specifications with no derive. Then
(1) For $SP \in \mathcal{SP}$,

$$\mathsf{Mod}^I(\hat{\mu}(SP)) = \beta_{\mathsf{Sig}(SP)}^{-1}(\mathsf{Mod}^I(SP)).$$

(2) For $SP \in \mathcal{SP}$, $\mathsf{Mod}^I(SP)$ is closed under isomorphism.

(3) $\mu$ admits borrowing of entailment and refinement for $\mathcal{SP}$.

**Proof.** Again, we simultaneously prove (1) and (2) by induction over the structure of $SP$.

- $SP = \langle \Sigma, \Gamma \rangle$: (1) follows from the representation condition. (2) follows from satisfaction in $I$ being closed under isomorphism.
- $SP = SP_1 \cup SP_2$: (1) $M' \in \mathsf{Mod}^I(\hat{\mu}(SP))$ iff $M' \in \mathsf{Mod}^I(\hat{\mu}(SP_1)) \cap \mathsf{Mod}^I(\hat{\mu}(SP_2))$ (by induction hypothesis) $\beta_{\mathsf{Sig}(SP_1)}(M') \in \mathsf{Mod}^I(SP_1) \cap \mathsf{Mod}^I(SP_2)$ iff $\beta_{\mathsf{Sig}(SP_1)}(M') \in \mathsf{Mod}^I(SP_1)$.

(2) The intersection of two isomorphism-closed model classes is isomorphism-closed.
- $SP = \mathsf{translate} SP_1$ by $\sigma: \Sigma \rightarrow \Sigma$: (1) $M' \in \mathsf{Mod}^I(\hat{\mu}(SP))$ iff $M' \in \mathsf{Mod}^I(\Phi(\Sigma))$ and $M'|_{\Phi(\sigma)} \in \mathsf{Mod}^I(\hat{\mu}(SP_1))$ (by induction hypothesis) $\beta_{\Sigma}^I(M')|_{\Phi(\sigma)} = \beta_{\Sigma_1}(M'|_{\Phi(\sigma)}) \in \mathsf{Mod}^I(SP_1)$ iff $\beta_{\Sigma_1}(M') \in \mathsf{Mod}^I(SP_1)$.

(2) If $M \in \mathsf{Mod}^I(SP)$ and $M' \cong M$, then also $M'|_{\sigma} \cong M|_{\sigma}$. By induction hypothesis, $M'|_{\sigma} \in \mathsf{Mod}^I(SP_1)$. Hence, $M' \in \mathsf{Mod}^I(SP)$.
- $SP = \mathsf{free} SP_1$ along $\sigma: \Sigma \rightarrow \Sigma$: (1) To show that $\beta_{\Sigma_1}: \mathsf{Mod}^I(\hat{\mu}(SP_1)) \rightarrow \mathsf{Mod}^I(SP_1)$ is isomorphism-dense, let $M \in \mathsf{Mod}^I(SP_1)$. By isomorphism-denseness of $\beta_{\Sigma_1}: \mathsf{Mod}^I(\Phi(\Sigma_1)) \rightarrow \mathsf{Mod}^I(\Sigma_1)$, there is $M' \in \mathsf{Mod}^I(\Phi(\Sigma_1))$ with $\beta_{\Sigma_1}(M') \cong M$. By induction hypothesis, $\mathsf{Mod}^I(SP_1)$ is closed under isomorphism, hence $\beta_{\Sigma_1}(M') \in \mathsf{Mod}^I(SP)$. Again by induction hypothesis, $M' \in \mathsf{Mod}^I(\hat{\mu}(SP_1))$. The rest of the proof is as in Theorem 2.49.

(2) Freeness is a property that is closed under isomorphism. Together with Proposition 2.10, this shows $\mathsf{Mod}^I(SP)$ to be closed under isomorphism.

(3) Follows from (1) and (2) with Proposition 2.25(4). □

### 2.4.5 Comparison Of Properties Of Institution Representations

We now briefly compare the different notions of representations that have been introduced so far.

**Proposition 2.52** Any strongly persistently liberal institution representation admits model expansion. Any model-bijective institution representation admits model expansion and weak $D$-amalgamation for arbitrary $D$. Any embedding is persistently liberal. Any subinstitution representation is an embedding, strongly persistently liberal and a model-bijective institution represen-
We now present a number of counterexamples showing that the above diagram of implications is optimal.

**Example 2.53** The institution representation from Examples 2.17 and 2.42 admits model expansion and it is strongly persistently liberal, but neither model-bijection nor an embedding: for a given partial model, it is possible to add any number of “undefined” elements in a model representing it. Hence, the model translation is neither bijective nor isomorphism-dense. □

**Example 2.54** Modify the institution representation from Examples 2.17 and 2.42 by omitting the functions \(\bot\) and the axioms involving \(\bot\). Then the resulting institution representation is still strongly persistently liberal, but it does not admit weak \(\mathcal{D}\)-amalgamation, where \(\mathcal{D}\) is the class of all signature morphisms adding a partial function symbol to a signature. Let \(\sigma: \Sigma \rightarrow \Sigma' \in \mathcal{D}\). Take any \(\Sigma\)-model \(M\) containing a truly partial function and such that all functions in \(M|_\sigma\) are total. Then \(M|_\sigma\) can be represented by a model \(M'\) (i.e. \(\beta_\Sigma(M') = M|_\sigma\)) that has no “undefined” elements at all. However, it is not possible to represent \(M\) with an extension of \(M'\), since this would require the presence of some “undefined” element in some carrier of \(M'\) (and hence also \(M\)). □

**Example 2.55** Modify the institution representation from Example 2.17 as follows: Add to \(\Phi(\Sigma)\) a sentence \(D_s(x) : \neg x = \bot_s\) (for each sort \(s\) in \(\Sigma\)). Thus, \(\bot_s\) becomes the unique “undefined” element. Now a two-sided inverse of the model translation can be constructed as follows: For a partial \(\Sigma\)-structure \(M\), form its \emph{one-point completion} by just adding one element, \(*\) (which is the interpretation of \(\bot_s\)), to all carriers, let all functions map \(*\) to itself and behave as in \(M\) otherwise, where undefinedness of partial functions is mapped to \(*\). Predicates are false on \(*\). The interpretation of the predicates \(D\) is fixed by their defining axioms. This shows the representation to be model-bijection (and hence it also admits model expansion).

The above defined inverse to the model translation defines a bijection on model classes, but not an isomorphism of model categories. This is because a \(\Sigma\)-homomorphism need only preserve definedness of partial functions, while
a \(\Phi(\Sigma)\)-homomorphism has to preserve and reflect "definedness" of the representations of partial functions. Thus, the above inverse construction, being the unique inverse construction on models, cannot be extended to a functor. This shows that the above institution representation is neither a substitution representation nor persistently liberal.

Now consider the \(PFOL^+\)-signature \(\Sigma\) with one sort \(s\) and one partial function symbol \(f: s \rightarrow \mathcal{P}s\), and let \(\sigma\) be the inclusion of the signature consisting just of sort \(s\) into \(\Sigma\). Given a set \(X\), the \(\sigma\)-free \(\Sigma\)-model \(M\) over \(X\) is just \(X\) with \(f\) interpreted as the everywhere undefined function. The unique representation \(M'\) of \(M\) is \(X\) with one "undefined" element, and \(f\) yielding everywhere the "undefined" element. Now \(M'\) is not free over \(X\), since any homomorphism starting from \(M'\) has to preserve the undefined element and thus there cannot be a homomorphism from \(M'\) into some model where \(f\) is defined at some point. This shows that the assumption of persistent liberality is really needed in Theorem 2.43. \(\square\)

**Example 2.56** Let \(\mathcal{D}\) be the class of all signature isomorphisms in an institution \(I\). Since functors preserve isomorphisms, any institution representation starting from \(I\) admits weak \(\mathcal{D}\)-amalgamation. Clearly, not every such representation admits model expansion. \(\square\)

**Example 2.57** The institution representation going from \(COS-\text{SubCond}^=\) to \(COSASC\) described in Theorem 10.4 is an embedding, but it does not admit model expansion (just because there are \(COS-\text{SubCond}^=\)-models that have non-inclusions as subsort embeddings). \(\square\)

### 2.5 Intersections of Substitutions

In Section 3.4 below, we want to define substitutions of the CASL institution by removing different features from it. Therefore, we need intersections of substitutions.

Given a family \((\mu_k = (\Phi_k, \alpha^k, \beta^k) : J_k \rightarrow J)_{k \in K}\) of substitutions, with \(J_k = (\text{Sign}^k, \text{Sen}^k, \text{Mod}^k, \models^k)\), their intersection \(I\) is defined as follows:

**Signatures** \(\text{Sign}^I := \bigcap_{k \in K} \Phi^k(\text{Sign}^k)\). This is a category, since the \(\Phi_k\) are embeddings, and categories are closed under taking the image along embeddings and under intersection.

**Models** \(\text{Mod}^I(\Sigma) := \text{Mod}^I(\Sigma)\).

**Sentences** \(\text{Sen}^I(\Sigma) := \bigcap_{k \in K} \alpha^k_{\Phi^k_1(\Sigma)}(\text{Sen}^k(\Phi^k_1(\Sigma)))\), and \(\text{Sen}^I(\sigma: \Sigma \rightarrow \Sigma') : \text{Sen}^I(\Sigma) \rightarrow \text{Sen}^I(\Sigma')\) is the domain-codomain-restriction of \(\text{Sen}^I(\sigma)\), which exists by naturality of \(\alpha\). The injection of \(\text{Sen}^I(\Sigma)\) into \(\text{Sen}^k(\Phi^k_1(\Sigma))\) given
by the appropriate restriction of $(\alpha_k^{\Theta_k^{-1}(\Sigma)})^{-1}$ is denoted by $\ell_k$.

**Satisfaction** $M \models_{\Sigma} \varphi$ iff $M \models_{\Sigma} \varphi$ (i.e., the satisfaction condition is inherited from $J$). □

Note that the intersection is a limit in the category of institutions and simple representations, with limit projections $(\Phi_k^{-1}, \iota, (\beta_k \circ \Phi_k^{-1})^{-1})$: $I \rightarrow J_k$ (well, injections would be a better name here, since they inject the intersection $I$ in the $J_k$).

### 2.6 Translating language constructs

As stated in the introduction, a specification language typically is based on an institution; and the semantics of the language constructs is defined by mapping the language constructs into mathematical concepts defined in terms of the institution. In this paper, we mainly concentrate on the translation of basic specifications, which correspond to presentations at the level of concepts.

Given two specification languages $L_1$ and $L_2$ with underlying institutions $I_1$ and $I_2$, and given an institution representation $\mu: I_1 \rightarrow I_2$, a translation from $L_1$ to $L_2$ can be obtained as follows:

For a basic specification $SP_1$ written in $L_1$, apply the semantics of $L_1$ to obtain a presentation $T_1$ in $I_1$. Then use $\mu$ to obtain a presentation $T_2$ in $I_2$. Finally, find a basic specification $SP_2$ in $L_2$ having semantics $T_2$. This assumes that every presentation can be represented in the language. The language CASL fulfills this assumption. We do not know if other languages fulfill this assumption as well, but we would be surprised if not. (The detailed study of language constructs is beyond the scope if this paper.)

For structured specifications, this translation has to be extended inductively according to Definition 2.18. Note that this assumes that $L_1$ and $L_2$ have the same structuring mechanisms. The translation of different structuring mechanisms into each other is beyond the scope of this paper.

Since the mapping from basic specifications to presentations in the underlying institution is not formally defined for many specification languages (and also because it can become quite complex), we will only informally discuss how an institution representation lifts to a translation of the corresponding language constructs. In some cases, we also indicate how a better translation can be obtained by not going the above way from $SP_1$ to $SP_2$ through the underlying theories $I_1$ and $I_2$, but rather use a direct translation of $SP_1$ to $SP_2$. This is advisable e.g. if $L_1$ and $L_2$ have similar language constructs for expressing complex presentations in a concise way, while these constructs have rather lengthy expansions in terms of the underlying institution.
CASL is an expressive language combining subsorts, partiality, first-order logic and induction (the latter is expressed using so-called sort generation constraints). The institution underlying CASL is introduced in two steps [23,17]: first, we introduce many-sorted partial first-order logic with sort generation constraints and equality (PCFOL$^=$), and then, subsorted partial first-order logic with sort generation constraints and equality (SubPCFOL$^=$) is described in terms of PCFOL$^=$.

3.1 Partial First-Order Logic

**Definition 3.1** The institution PCFOL$^=$. 

**Signatures** A many-sorted signature $\Sigma = (S, TF, PF, P)$ in PCFOL$^=$ consists of
- a set $S$ of sorts,
- two $S^* \times S$-sorted families $TF = (TF_{w,s})_{w \in S^*, s \in S}$ and $PF = (PF_{w,s})_{w \in S^*, s \in S}$ of total function symbols and partial function symbols, respectively, such that $TF_{w,s} \cap PF_{w,s} = \emptyset$, for each $(w, s) \in S^* \times S$ (constants are treated as functions with no arguments), and
- a family $P = (P_w)_{w \in S^*}$ of predicate symbols.

Note that function and predicate symbols may be overloaded, occurring in more than one of the above sets. To ensure that there is no ambiguity in sentences, however, symbols are always qualified by profiles when used. In the CASL language constructs (see Section 3.3), such qualifications may be omitted when these are unambiguously determined by the context.

We now introduce some notation. We write $f : w \rightarrow s \in TF$ for $f \in TF_{w,s}$ (including the special case $f : s$ for empty $w$), $f : w \rightarrow? s \in PF$ for $f \in PF_{w,s}$ (including the special case $f :? s$ for empty $w$) and $p : w \in P$ for $p \in P_w$. For a function symbol $f \in TF_{w,s}$ or $f \in PF_{w,s}$, we call $w \rightarrow s$ or $w \rightarrow? s$, resp., its profile, $w$ its argument sorts and $s$ its result sort. For predicate symbols $p : w$, we call $w$ both its profile and its argument sorts.

Given a function $f : A \rightarrow B$, let $f^* : A^* \rightarrow B^*$ be its extension to finite strings. Given a finite string $w = s_1 \ldots s_n$ and sorts $M_{s_1}, \ldots, M_{s_n}$, we write $M_w$ for the Cartesian product $M_{s_1} \times \ldots \times M_{s_n}$.

Given signatures $\Sigma = (S, TF, PF, P)$ and $\Sigma' = (S', TF', PF', P')$, a signature morphism $\sigma : \Sigma \rightarrow \Sigma'$ consists of
- a map $\sigma^S : S \rightarrow S'$,
- a map $\sigma_{w,s}^F : TF_{w,s} \cup PF_{w,s} \rightarrow TF'_{\sigma^S(w),s} \cup PF'_{\sigma^S(w),s}$ preserving totality, for each $w \in S^*$, $s \in S$, and
- a map $\sigma_{w}^P : P_w \rightarrow P'_{\sigma^S(w)}$ for each $w \in S^*$.
Identities and composition are defined in the obvious way. This gives us a category of $PFOL^\Sigma$-signatures.

Models Given a many-sorted signature $\Sigma = (S, TF, PF, P)$, a many-sorted $\Sigma$-model $M$ consists of:

- a non-empty carrier set $M_s$ for each sort $s \in S$,
- a partial function $(f_{w,s})_M$ (also written just $f_M$) from $M_w$ to $M_s$ for each function symbol $f \in TF_{w,s} \cup PF_{w,s}$, the function being total if $f \in TF_{w,s}$, and
- a predicate $(p_w)_M$ (also written just $p_M$) $\subseteq M_w$ for each predicate symbol $p \in P_w$.

A many-sorted $\Sigma$-homomorphism $h : M \rightarrow N$ consists of a family of functions $(h_s : M_s \rightarrow N_s)_{s \in S}$ with the property that for all $f \in TF_{w,s} \cup PF_{w,s}$ and $(a_1, \ldots, a_n) \in M_w$ with $(f_{w,s})_M(a_1, \ldots, a_n)$ defined, we have

$$h_s((f_{w,s})_M(a_1, \ldots, a_n)) = (f_{w,s})_N(h_{s_1}(a_1), \ldots, h_{s_n}(a_n)),$$

and for all $p \in P_w$ and $(a_1, \ldots, a_n) \in M_w$,

$$(a_1, \ldots, a_n) \in (p_w)_M \text{ implies } (h_{s_1}(a_1), \ldots, h_{s_n}(a_n)) \in (p_w)_N.$$

Identities and composition are defined in the obvious way.

Concerning reducts, if $\sigma : \Sigma \rightarrow \Sigma'$ is a signature morphism with $\Sigma = (S, TF, PF, P)$, and $M'$ is a $\Sigma'$-model, then $M'|_\sigma$ is the $\Sigma$-model $M$ with

- $M_s := M'_{\sigma^S(s)}$ ($s \in S$)
- $(f_{w,s})_M := (\sigma^F_{w,s}(f))_{M'}$ ($f \in TF_{w,s} \cup PF_{w,s}$)
- $(p_w)_M := (\sigma^P_{w}(p))_{M'}$ ($p \in P_w$)

This is well-defined since $\sigma^F_{w,s}$ preserves totality.

Given a $\Sigma'$-homomorphism $h' : M'_1 \rightarrow M'_2$, its reduct $h'|_\sigma : M'_1|_\sigma \rightarrow M'_2|_\sigma$ is the homomorphism defined by

$$(h'|_\sigma)_s := h'_{\sigma^S(s)} \quad (s \in S).$$

It is easy to see that the reduct w.r.t. an identity is the identity, and that the reduct w.r.t. a composition is the composition of the reducts w.r.t. the signature morphisms that are composed. Thus, $\text{Mod}$ is a functor.

Sentences Let a many-sorted signature $\Sigma = (S, TF, PF, P)$ be given. A variable system over $\Sigma$ is an $S$-sorted, pairwise disjoint family of variables $X = (X_s)_{s \in S}$. Let such a variable system be given.

The sets $T_\Sigma(X)_s$ of many-sorted $\Sigma$-terms of sort $s$, $s \in S$, in $X$ are the least sets satisfying the following rules:

1. $x \in T_\Sigma(X)_s$ if $x \in X_s$,
2. $f_{w,s}(t_1, \ldots, t_n) \in T_\Sigma(X)_s$, if $t_i \in T_\Sigma(X)_{s_i} (i = 1, \ldots, n), f \in TF_{w,s} \cup PF_{w,s}$.

Note that each term has a unique sort.

$T_\Sigma(X)$ can be made into a many-sorted $\Sigma$-algebra by putting

- $(f_{w,s})_{T_\Sigma(X)}(t_1, \ldots, t_n) = f_{w,s}(t_1, \ldots, t_n)$ for $f \in TF_{w,s} \cup PF_{w,s}$.

41
• \((p_w)_{\Sigma(X)} = \emptyset\) for \(p \in P_w\).

Variable assignments are total, but the value of a term w.r.t. a variable assignment may be undefined, due to the application of a partial function during the evaluation of the term. The evaluation of a term w.r.t. a variable assignment is defined as follows (see [19]):

Given a variable valuation \(\nu: X \rightarrow M\) for \(X\) in \(M\), the term evaluation \(\nu^\#: T_\Sigma(X) \rightarrow? M\) is inductively defined by:

- \(\nu^*_s(x) = \nu(x)\) for all \(x \in X_s\), and all \(s \in S\),
- \(\nu^*_s(f_{w,s}(t_1, \ldots, t_n)) = \begin{cases} (f_{w,s})_M(\nu^*_{s_1}(t_1), \ldots, \nu^*_{s_n}(t_n)), & \text{if } \nu^*_{s_i}(t_i) \text{ is defined } (i = 1, \ldots, n) \\
\text{undefined, otherwise} & \end{cases}\)

for all \(f \in TF_{w,s} \cup PF_{w,s}\), where \(w = s_1, \ldots, s_n\), and \(t_i \in \Sigma(X)_{s_i}\), for \(i = 1, \ldots, n\).

Given a term \(t\), we say that \(t\) is \(\nu\)-interpretable, if \(t \in \text{dom } \nu^\#\) and in this case we call \(\nu^\#(t) \in M_s\) the value of \(t\) in \(M\) under the valuation \(\nu\).

A many-sorted atomic \(\Sigma\)-formula with variables in \(X\) is either (1) an application of a qualified predicate symbol to terms of appropriate sorts, (2) an existential equation between terms of the same sort, (3) a strong equation between terms of the same sort, or (4) an assertion about definedness of a term:

The set \(AF_\Sigma(X)\) of many-sorted atomic \(\Sigma\)-formulas with variables in \(X\) is the least set satisfying the following rules:

1. \(p_w(t_1, \ldots, t_n) \in AF_\Sigma(X)\), if \(t_i \in \Sigma(X)_{s_i}, p \in P_w, w = s_1 \ldots s_n \in S^*\),
2. \(t = t' \in AF_\Sigma(X)\), if \(t, t' \in \Sigma(X)_{s}, s \in S\) (existential equations),
3. \(t = t' \in AF_\Sigma(X)\), if \(t, t' \in \Sigma(X)_{s}, s \in S\) (strong equations),
4. \(\text{def } t \in AF_\Sigma(X)\), if \(t \in \Sigma(X)_{s}, s \in S\) (definedness assertions).

The set \(FO_\Sigma(X)\) of many-sorted first-order \(\Sigma\)-formulas with variables in \(X\) is the least set satisfying the following rules:

1. \(AF_\Sigma(X) \subseteq FO_\Sigma(X)\),
2. \(F \in FO_\Sigma(X)\) (read: false),
3. \((\varphi \land \psi) \in FO_\Sigma(X)\) and \((\varphi \Rightarrow \psi) \in FO_\Sigma(X)\) for \(\varphi, \psi \in FO_\Sigma(X)\),
4. \(\forall x : s \cdot \varphi \in FO_\Sigma(X)\) for \(\varphi \in FO_\Sigma(X \cup \{x : s\})\), \(s \in S\).

We omit brackets whenever this is unambiguous and use the usual abbreviations: \(\neg \varphi\) for \(\varphi \Rightarrow F\), \(\varphi \lor \psi\) for \(\neg(\neg \varphi \land \neg \psi)\), \(T\) for \(\neg F\) and \(\exists x : s \cdot \varphi\) for \(\neg \forall x : s \cdot \neg \varphi\).

A sort generation constraint states that some set of sorts is generated by some set of functions. Technically, sort generation constraints also contain a signature morphism component; this is needed to be able to translate them along signature morphisms without sacrificing the satisfaction condition.

Formally, a sort generation constraint over a signature \(\Sigma\) is a triple \((\hat{S}, \hat{F}, \theta)\), where \(\theta: \hat{\Sigma} \rightarrow \Sigma, \hat{\Sigma} = (\hat{S}, \hat{T}, \hat{F}, \bar{F}), \hat{S} \subseteq S\) and \(\hat{F} \subseteq TF \cup \bar{F}\).

Now a \(\Sigma\)-sentence is a closed many-sorted first-order \(\Sigma\)-formula (i.e. a many-sorted first-order \(\Sigma\)-formula in the empty set of variables), or a sort
generation constraint over $\Sigma$.

Given a signature morphism $\sigma: \Sigma \to \Sigma'$ and variable system $X$ over $\Sigma$, we can get a variable system $\sigma(X)$ over $\Sigma'$ by putting

$$\sigma(X)_{\sigma'} = \bigcup_{\sigma[S] = \sigma'} X_S$$

Since the term algebra is total, the inclusion $\zeta_{\sigma, X}: X \to T_{\Sigma'}(\sigma(X))$ (constructed as a variable valuation) leads to a term evaluation function

$$\zeta_{\sigma, X}^*: T_\Sigma(X) \to T_{\Sigma'}(\sigma(X))$$

that is total as well. This can be inductively extended to a translation of $\Sigma$-first order formulas along $\sigma$:

- $\sigma(t) = \zeta_{\sigma, X}^*(t)$, if $t$ is a $\Sigma$-term,
- $\sigma(p_w(t_1, \ldots, t_n)) = \sigma^*_{\Sigma}(p)_{\sigma^*_{\Sigma}(w)}(\sigma(t_1), \ldots, \sigma(t_n))$,
- $\sigma(t \equiv t') = \sigma(t) \equiv \sigma(t')$,
- $\sigma(t = t') = \sigma(t) = \sigma(t')$,
- $\sigma(\text{def } t) = \text{def } \sigma(t)$,
- $\sigma(F') = F'$,
- $\sigma(\varphi \land \psi) = \sigma(\varphi) \land \sigma(\varphi)$,
- $\sigma(\forall x : s \cdot \varphi) = \forall x : \sigma^*(s) \cdot \sigma(\varphi)$.

The translation of a $\Sigma$-constraint $(\bar{S}, \bar{F}, \theta)$ along $\sigma$ is the $\Sigma'$-constraint $(\bar{S}, \bar{F}, \sigma \circ \theta)$.

It is easy to see that the sentence translation along the identity signature morphism is the identity, and that the sentence translation along a composition of two signature morphisms is is the composition of the sentence translations along the individual signature morphisms. Hence, sentence translation functorial.

**Satisfaction Relation** Even though the evaluation of a term w.r.t. a variable assignment may be undefined, the evaluation of a formula is always defined (and it is either true or false). That is, we have a two-valued logic. The application of a predicate symbol $p$ to a sequence of argument terms holds w.r.t. a valuation $\nu: X \to M$ iff the values of all the terms are defined under $\nu^*$ and give a tuple belonging to $p_M$. A definedness assertion concerning a term holds iff the value of the term is defined. An existential equation holds iff the values of both terms are defined and identical, whereas a strong equation holds also when the values of both terms are undefined; thus both notions of equation coincide for defined terms.

More formally, satisfaction of a formula $\varphi \in FO_\Sigma(X)$ by a variable valuation $\nu: X \to M$ is defined inductively over the structure of $\varphi$:

- $\nu \models_\Sigma p_w(t_1, \ldots, t_n)$ iff $\nu^*(t_i)$ is defined ($i = 1, \ldots, n$) and, moreover, $(\nu^*(t_1), \ldots, \nu^*(t_n)) \in (p_w)_M$,
- $\nu \models_\Sigma t_1 \equiv t_2$ iff $\nu^*(t_1)$ and $\nu^*(t_2)$ are both defined and equal,
- $\nu \models_\Sigma t_1 = t_2$ iff $\nu^*(t_1)$ and $\nu^*(t_2)$ are either both defined and equal, or
both undefined,
- $\nu \vdash_\Sigma \text{def } t$ iff $\nu^\#(t)$ is defined,
- not $\nu \vdash_\Sigma F$
- $\nu \vdash_\Sigma (\varphi \land \psi)$ iff $\nu \vdash_\Sigma \varphi$ and $\nu \vdash_\Sigma \psi$
- $\nu \vdash_\Sigma (\varphi \Rightarrow \psi)$ iff $\nu \vdash_\Sigma \varphi$ implies $\nu \vdash_\Sigma \psi$
- $\nu \vdash_\Sigma (\forall x : s \bullet \varphi)$ iff for all valuations $\xi : X \cup \{x : s\} \rightarrow M$ which extend $\nu$ on $X \setminus \{x : s\}$ (i.e., $\xi(x) = \nu(x)$ for $x \in X \setminus \{x : s\}$), we have $\xi \vdash_\Sigma \varphi$.

A formula is satisfied in a model iff it is satisfied w.r.t. all variable valuations into that model.

A $\Sigma$-constraint $(\hat{S}, \hat{F}, \theta)$ satisfied in a $\Sigma$-model $M$, if the carriers of $M|_{\theta}$ of the sorts in $\hat{S}$ are generated by the function symbols in $\hat{F}$, i.e. for every sort $s \in \hat{S}$ and every value $a \in (M|_{\theta})_s$, there is a $\Sigma$-term $t$ containing only function symbols from $\hat{F}$ and variables of sorts not in $\hat{S}$ such that $\nu^\#(t) = a$ for some assignment $\nu$ into $M|_{\theta}$.

For a sort generation constraint $(\hat{S}, \hat{F}, \theta)$ we can assume without loss of generality that all the result sorts of function symbols in $\hat{F}$ occur in $\hat{S}$. If not, we can just leave out from $\hat{F}$ those function symbols not satisfying this requirement. The satisfaction of the sort generation constraint in any model will not be affected by this: in the $\Sigma$-term $t$ witnessing the satisfaction of the constraint, any application of a function symbol with result sort outside $\hat{S}$ can just be replaced by a variable of that sort, which gets then as assigned value the evaluation of the function application.

Concerning the satisfaction condition, we need the following two lemmas:

**Lemma 3.2** Let a signature morphism $\sigma : \Sigma \rightarrow \Sigma'$, a $\Sigma'$-model $M'$, and a variable system $X$ over $\Sigma$ be given. Then for each valuation $\nu : \sigma(X) \rightarrow M'$, there is valuation $\nu|_{\sigma} : X \rightarrow M'|_{\sigma}$ such that $\nu^\#|_{\sigma} \circ \zeta^\#_{\sigma,X} = (\nu|_{\sigma})^\#$. Moreover, this is a one-one correspondence.

![Diagram](image)

**Proof.** We put

$$(\nu|_{\sigma})_s(x) := \nu_{\sigma(s)}(x) \text{ for } x \in X_s.$$ 

The inverse is given by

$$(\nu|_{\sigma})^{-1}(x) := \nu_s(x),$$

where $s$ is the unique $s \in S$ with $x \in X_s$ and $\sigma(s) = s'$.

The property $\nu^\#|_{\sigma} \circ \zeta^\#_{\sigma,X} = (\nu|_{\sigma})^\#$ follows by induction over $T_{\Sigma}(X)$. \qed
Lemma 3.3  Given a signature morphism $\sigma: \Sigma \rightarrow \Sigma'$, a $\Sigma'$-model $M$, a variable system $X$ over $\Sigma$, and a formula $\varphi \in FO_\Sigma(X)$, we have

$$\nu|_\sigma \models \varphi \text{ iff } \nu \models \sigma(\varphi)$$

Proof. Induction over $\varphi$.

E.g. for strong equations, we have $\nu|_\sigma \models t_1 = t_2$

iff $(\nu|_\sigma)^\#(t_1) = (\nu|_\sigma)^\#(t_2)$ (or both sides are undefined)

iff (by Lemma 3.2) $\nu^\# \circ \zeta^\#_\sigma(x)(t_1) = \nu^\# \circ \zeta^\#_\sigma(x)(t_2)$ (or both sides are undefined)

iff (by definition) $\nu^\#(\sigma(t_1)) = \nu^\#(\sigma(t_2))$ (or both sides are undefined)

iff $\nu \models \sigma(t_1 = t_2)$.

For quantifications, we have $\nu|_\sigma \models \forall x : s \bullet \psi$ iff for all $\xi : X \cup \{x : s\} \rightarrow M'_{\sigma}$ extending $\nu|_\sigma$ on $X \setminus \{x : s\}$, $\xi \models \psi$ iff (by induction hypothesis) for all $\xi : X \cup \{x : s\} \rightarrow M'_{\sigma}$ extending $\nu|_\sigma$ on $X \setminus \{x : s\}$, $(\downarrow_{\sigma})^{-1}(\xi) \models \sigma(\psi)$

iff (since $\downarrow_{\sigma}$ is one-one) for all $\rho : \sigma(X \cup \{x : s\}) \rightarrow M'$ extending $\nu$ on $\sigma(X \setminus \{x : \sigma(s)\})$, $\rho \models \sigma(\psi)$ iff $\nu \models \forall x : \sigma(s) \bullet \sigma(\psi)$.

The other cases are treated similarly. □

The satisfaction condition for first-order formulas now follows easily from Lemma 3.3 (noting that by Lemma 3.2 $\sigma$ is surjective on valuations).

The satisfaction condition for sort generation constraints is obvious (indeed, the extra signature morphism component in the sort generation constraints has been introduced to make it work).

This completes the definition of the institution $PCFOL^\equiv$. □

3.2 Subsorted Partial First-Order Logic

Subsorted partial first-order logic is defined in terms of partial first-order logic. The basic idea is to reduce subsorting to injections between sorts. While in the subsorted institution, these injections have to occur explicitly in the sentences, in the CASL language, they may be left implicit. Apart from the injections, one also has partial projection functions (one-sided inverses of the injections) and membership predicates.

Definition 3.4  The institution $SubPCFOL^\equiv$.

Signatures  The notion of subsorted signatures extends the notion of ordered-sorted signatures as given by Goguen and Meseguer [36], by allowing not only total function symbols, but also partial function symbols and predicate symbols:

A subsorted signature $\Sigma = (S, TF, PF, P, \leq_S)$ consists of a many-sorted signature $(S, TF, PF, P)$ together with a reflexive transitive subsort relation $\leq_S$ on the set $S$ of sorts. Note that $\leq_S$ is not required to be antisymmetric; this allows to declare isomorphic sorts, where the injections correspond to

45
change of representations.

The relation \( \leq_S \) extends pointwise to sequences of sorts. We drop the subscript \( S \) when obvious from the context.

For a subsorted signature, \( \Sigma = (S, TF, PF, P, \leq_S) \), we define overloading relations (also called monotonicity orderings), \( \sim_F \) and \( \sim_P \), for function and predicate symbols, respectively:

Let \( f : w_1 \rightarrow s_1, f : w_2 \rightarrow s_2 \in TF \cup PF \), then

\[
f : w_1 \rightarrow s_1 \sim_F f : w_2 \rightarrow s_2
\]

iff there exist \( w \in S^* \) with \( w \leq w_1 \) and \( w \leq w_2 \) and \( s \in S \) with \( s_1 \leq s \) and \( s_2 \leq s \).

Let \( p : w_1, p : w_2 \in P \), then \( p : w_1 \sim_P \ p : w_2 \) iff there exists \( w \in S^* \) with \( w \leq w_1 \) and \( w \leq w_2 \).

A signature morphism \( \sigma : \Sigma \rightarrow \Sigma' \) is a many-sorted signature morphism that preserves the subsort relation and the overloading relations. Note that, due to preservation of subsorting, the preservation of the overloading relations can be simplified to:

\[
f : w_1 \rightarrow s_1 \sim_F f : w_2 \rightarrow s_2 \quad \text{implies} \quad \sigma^F_{w_1, s_1} (f) = \sigma^F_{w_2, s_2} (f)
\]

\[
p : w_1 \sim_P p : w_2 \quad \text{implies} \quad \sigma^P_{w_1} (p) = \sigma^P_{w_2} (p)
\]

With each subsorted signature \( \Sigma = (S, TF, PF, P, \leq_S) \) we associate a many-sorted signature \( \hat{\Sigma} \), which is the extension of the underlying many-sorted signature \( (S, TF, PF, P) \) with

- a total \textit{injection} function symbol \( \text{inj} : s \rightarrow s' \), for each pair of sorts \( s \leq_S s' \),
- a partial \textit{projection} function symbol \( \text{pr} : s' \rightarrow s \), for each pair of sorts \( s \leq_S s' \), and
- a unary \textit{membership} predicate symbol \( \in^s : s' \), for each pair of sorts \( s \leq_S s' \).

We assume that the symbols used for injection, projection and membership are not used otherwise in \( \Sigma \). In formulas, we also write \( t \in s \) instead of \( \in^s (t) \) if \( s' \) is clear from the context.

Given a signature morphism \( \sigma : \Sigma \rightarrow \Sigma' \), we can extend it to a signature morphism \( \hat{\sigma} : \hat{\Sigma} \rightarrow \hat{\Sigma}' \) by just mapping the injections, projections and memberships in \( \Sigma \) to the corresponding injections, projections and memberships in \( \hat{\Sigma} \). This turns \( \hat{\_} \) into a functor \( \hat{\_} : \text{Sign}_{\text{SubPCFOL}^a} \rightarrow \text{Sign}_{\text{PCFOL}^a} \).

Models Subsorted \( \Sigma \)-models are ordinary many-sorted \( \Sigma \)-models satisfying the following set of axioms \( J(\Sigma) \) (where the variables are all universally quantified):

\[
\text{inj}_{(s, s)} (x) \equiv x \quad \text{(identity)}
\]

\[
\text{inj}_{(s, s')} (x) \equiv \text{inj}_{(s, s')} (y) \Rightarrow x \equiv y \quad \text{for} \ s \leq_S s' \quad \text{(embedding-injectivity)}
\]

\[
\text{inj}_{(s', s)} (\text{inj}_{(s, s')} (x)) \equiv \text{inj}_{(s, s')} (x) \quad \text{for} \ s \leq_S s' \leq_S s'' \quad \text{(transitivity)}
\]

\[
\text{pr}_{(s', s)} (\text{inj}_{(s, s')} (x)) \equiv x \quad \text{for} \ s \leq_S s' \quad \text{(projection)}
\]

\[
\text{pr}_{(s', s)} (x) \equiv \text{pr}_{(s, s)} (y) \Rightarrow x \equiv y \quad \text{for} \ s \leq_S s' \quad \text{(projection-injectivity)}
\]

46
$\in^s_s(x) \iff \text{def pr}_{\{s, s\}}(x)$ for $s \leq s'$ (membership)
\[ \text{inj}_{\{s, s\}}(f_{w, s'}(\text{inj}_{\{s_1, s_1\}}(x_1), \ldots, \text{inj}_{\{s_n, s_n\}}(x_n))) = \]
\[ = \text{inj}_{\{s', s'\}}(f_{w, s'}(\text{inj}_{\{s_1, s_1\}}(x_1), \ldots, \text{inj}_{\{s_n, s_n\}}(x_n))) \]
for $f_{w, s'} \sim_F f_{w', s'}$, where $w \leq w', \ w = s_1 \ldots s_n, \ w' = s'_1 \ldots s'_n, \ w'' = s''_1 \ldots s''_n$ and $s', s'' \leq s$ (function-monotonicity)
\[ p_{w'}(\text{inj}_{\{s_1, s_1\}}(x_1), \ldots, \text{inj}_{\{s_n, s_n\}}(x_n)) \iff p_{w''}(\text{inj}_{\{s_1, s_1\}}(x_1), \ldots, \text{inj}_{\{s_n, s_n\}}(x_n)) \]
for $p_{w'} \sim_P p_{w''}$, where $w \leq w', \ w'' = s_1 \ldots s_n, \ w' = s'_1 \ldots s'_n$, and $w'' = s''_1 \ldots s''_n$ (predicate-monotonicity)

$\Sigma$-homomorphisms are $\Sigma$-homomorphisms.

**Lemma 3.5** Let $\sigma: \Sigma \rightarrow \Sigma'$ be a subsorted signature morphism. Then

\[ \text{Sen}^{PCFOL^=}(\hat{\sigma})(\hat{J}(\Sigma)) \subseteq \hat{J}(\Sigma') \]

**Proof** $\hat{\sigma}$ preserves subsorting, injections, projections, membership and the overloading relations $\sim_F$ and $\sim_P$. Now the sentences in $\hat{J}(\Sigma)$ and $\hat{J}(\Sigma')$ just correspond to these. $\square$

To obtain a reduct of a subsorted $\Sigma$-model $M'$ along a subsorted signature morphism $\sigma: \Sigma \rightarrow \Sigma'$, take the the many-sorted reduct $M'|_{\sigma} = \text{Mod}^{PCFOL^=}(\hat{\sigma})(M')$ of $M'$ along $\hat{\sigma}: \hat{\Sigma} \rightarrow \hat{\Sigma'}$. By definition of subsorted model, $M' \models^{PCFOL^=}_{\Sigma} \hat{J}(\Sigma')$. By the lemma,

\[ M' \models^{PCFOL^=}_{\Sigma} \text{Sen}^{PCFOL^=}(\hat{\sigma})(\hat{J}(\Sigma)) \]

From this, we get by the satisfaction condition for $PCFOL^=\]

\[ \text{Mod}^{PCFOL^=}(\hat{\sigma})(M') \models^{PCFOL^=}_{\hat{\Sigma}} \hat{J}(\Sigma). \]

Thus, $\text{Mod}^{PCFOL^=}(\hat{\sigma})(M')$ is a subsorted $\Sigma$-model, and hence, we can define $\text{Mod}^{SubPCFOL^=}(\sigma)(M')$ to be $\text{Mod}^{PCFOL^=}(\hat{\sigma})(M')$.

**Sentences** Subsorted $\Sigma$-sentences are ordinary many-sorted $\hat{\Sigma}$-sentences. Sentence translation along a subsorted signature morphism $\sigma$ is just sentence translation along the many-sorted signature morphism $\hat{\sigma}$.

**Satisfaction** Since models and sentences are taken from $PCFOL^=\]$, satisfaction, as well as the satisfaction condition, can also be inherited from $PCFOL^=\]$. This completes the definition of the institution $SubPCFOL^=\]$. $\square$

### 3.3 Casl Language Constructs

Since the level of constructs will be treated only informally in this work, we just give a brief overview of the constructs for writing basic specifications (i.e. specifications in-the-small) in Casl. A detailed description can be found in the Casl Language Summary [22] and the Casl semantics [23].

47
The CASL language provides constructs for writing sort, subsort, operation\textsuperscript{10} and predicate declarations that contribute to the signature in the obvious way. Operations, predicates and subsorts can also be defined; this leads to a corresponding declaration plus a defining axiom.

Operation and predicate symbols may be overloaded; this may lead to ambiguities in the formulas. A formula is well-formed only if it has a unique fully-qualified expansion up to equivalence w.r.t. the overloading relations $\sim_F$ and $\sim_P$.

For operations and predicates, a mixfix syntax is provided. Precedence and associativity annotations may help to disambiguate terms containing mixfix symbols. There is also a syntax for literals such as numbers and strings, which allows the specification of the usual datatypes purely in CASL, without the need of magic built-in modules.

Binary operations can be declared to be associative, commutative, idempotent, or to have a unit. This leads to a corresponding axiom, and, in the case of associativity, to an associativity annotation.

The type, free type and generated type constructs allow the concise description of datatypes. They are expanded into the declaration of the corresponding constructor and selector operations and axioms relating the selectors and constructors. In the case of generated and free datatypes, also a sort generation constraint is produced. Free datatypes additionally lead to axioms that state the injectivity of the constructors and the disjointness of their images.

A typical CASL specification is shown in Fig. 1. Its translation to a presentation in $SubPCFOL^n$, the institution underlying CASL, is shown in Fig. 2. The translation has been generated with the CASL tool set [51] and uses the CASL notation to display theories in $SubPCFOL^n$. However, the notations are so close to each other that it should be easy to understand Fig. 2. The translation of CASL constructs to the underlying mathematical concepts is formally defined in the CASL semantics [23]. Since such a formal semantics is missing for most other languages, we discuss language constructs only informally in this paper.

\textit{3.4 Substitutions of CASL}

Despite being first-order, CASL is a quite rich and complex language. When relating CASL to other languages, it is quite useful to single out sublanguages of CASL. Here, we define a number of subinstitutions of the CASL institution

\textsuperscript{10}At the level of constructs, functions are called operations.
\%list \[\_, \_::\_\]
\%prec \{\_::\_\} < \{\_++\_\}

**spec** \text{l}ist [sort \text{Elem}] =

free type \text{List}[\text{Elem}] ::= \_nil | \_::(\text{head} :: ? \text{Elem}; \text{tail} :: ? \text{List}[\text{Elem}]);

sort \text{NList}[\text{Elem}] = \{\text{L} : \text{List}[\text{Elem}]\} \bullet -L = \_nil\};

op \_++\_ : \text{List}[\text{Elem}] \times \text{List}[\text{Elem}] \rightarrow \text{List}[\text{Elem}];

forall \_e : \text{Elem}; K, L : \text{List}[\text{Elem}]

\begin{align*}
&\cdot \_nil ++L = L & (\%\text{concat} _\_nil) \\
&\cdot (\_e :: K) ++L = e :: K ++L & (\%\text{concat} _\_cons)
\end{align*}

end

Fig. 1. Specification of lists over an arbitrary element sort in \text{CASL}.

\text{SubPCFOL}^=\text{.} The corresponding restriction of the \text{CASL} language is then straightforward, see [56].

Below, we define subinstitutions of \text{SubPCFOL}^= by just imposing restrictions on the signatures and/or axioms. This implicitly means that we take the full subcategory of signatures satisfying the restriction, and restrict the model and sentence functors and the satisfaction relation to this signature subcategory. A restriction on sentences further leads to the replacement of the sentence functor by a subfunctor. In each case, it is quite obvious to construct the corresponding subinstitution representation into \text{SubPCFOL}^=.

3.4.1 A Number of Features of \text{CASL}

In this section, we describe a number of \text{CASL}'s features \emph{negatively} by specifying, for each feature, the subinstitution of \text{CASL} that leaves out exactly that feature. This is possible since \text{CASL} is already the combination of all its features. A combination of only some of \text{CASL}'s features can then be obtained by intersecting all those subinstitutions that exclude exactly one of the undesired features. (Note that the combination of features from scratch is far more complicated [62].)

3.4.1.1 Partiality The institution \text{SubCFOL}^= is the restriction of the institution \text{SubPCFOL}^= to those signatures with an empty set of partial function symbols and those sentences that do not involve partial projection symbols. Note that \text{SubCFOL}^=, like \text{SubPCFOL}^=, is still defined via a reduction to \text{PCFOL}^=, which involves signatures $\Sigma$ containing partial projection symbols. However, these symbols are not used in the sentences, and they are redundant in the models (meaning that leaving them out leads to isomorphic model categories). Thus, it is justified to call \text{SubCFOL}^= an institution of \emph{total} algebras.
spec List|sort Elem| =
sorts NEList[Elem] < List[Elem]
ops _ + + _ : List[Elem] × List[Elem] → List[Elem];
    _ :: _ : Elem × List[Elem] → List[Elem];
    head : List[Elem] →? Elem;
    nil : List[Elem];
generated
{ sort List[Elem]
    ops _ :: _ : Elem × List[Elem] → List[Elem];
    nil : List[Elem] }
forall X0 : Elem; X1 : List[Elem]
    X0 =
            (op _ :: _ : Elem × List[Elem] → List[Elem])(X0, X1))
        % (selector_head)%
forall X0 : Elem; X1 : List[Elem]
    X1 : List[Elem] =
        (op tail : List[Elem] →? List[Elem])(
            (op _ :: _ : Elem × List[Elem] → List[Elem])(X0, X1))
        % (selector_tail)%
forall Y0 : Elem; Y1 : List[Elem]
    \ ~ (op nil : List[Elem]) =
        (op _ :: _ : Elem × List[Elem] → List[Elem])(Y0, Y1)
        % (disjoint_nil :: _)%
forall X0 : Elem; X1 : List[Elem]; Y0 : Elem; Y1 : List[Elem]
    (op _ :: _ : Elem × List[Elem] → List[Elem])(X0, X1) =
        (op _ :: _ : Elem × List[Elem] → List[Elem])(Y0, Y1)
        ⇒ X0 = Y0 ∧ X1 = Y1
        % (injective _ :: _)%
forall L : List[Elem]
    L ∈ NEList[Elem] ⇔ \ ~ L = (op nil : List[Elem])
        % (subsort_defn_NEList[Elem])%
forall K : List[Elem]
    (op _ + + _ : List[Elem] × List[Elem] → List[Elem])(
        (op nil : List[Elem]), L) = L
        % (concat_nil)%
forall e : Elem; K : List[Elem]; L : List[Elem]
    (op _ + + _ : List[Elem] × List[Elem] → List[Elem])(
        (op _ :: _ : Elem × List[Elem] → List[Elem])(e, K), L) =
        (op _ :: _ : Elem × List[Elem] → List[Elem])(e,
        % (concat_cons)%
end

Fig. 2. Translation of the specification List to a presentation in SubPCFOL^=.
3.4.1.2 Predicates  The institution $\text{SubPCFOL}^=\,$ is the restriction of $\text{SubPCFOL}^\leq$ to those signatures with an empty set of predicate symbols. Note that the signatures $\Sigma$ and therefore the sentences in $\text{SubPCFOL}^=\,$ do involve membership predicate symbols. In the OJ community, these are called “sort constraints”. That is, $\text{SubPCFOL}^=\,$ includes subsorting with sort constraints. In Section 10, we will also consider subsorting without sort constraints. (Note that sort constraints should not be confused with sort generation constraints.)

3.4.1.3 Subsorting  The institution $\text{PCFOL}^=\,$ has already been defined. It can be made into a subinstitution of $\text{SubPCFOL}^=\,$ by extending each signature with the trivial subsort relation (i.e., the subsort relation which is the identity relation on the set of sorts).

3.4.1.4 Sort Generation Constraints  The institution $\text{SubPFOL}^=\,$ is the restriction of the institution $\text{SubPCFOL}^=\,$ to those sentences that are not sort generation constraints.

3.4.1.5 Equality  The institution $\text{SubPCFOL}$ is the restriction of the institution $\text{SubPCFOL}^=\,$ to those sentences that involve neither strong nor existential equality.

3.4.2 A Number of Levels of Axiom Expressiveness

In the sequel, we introduce a number of subinstitutions of $\text{SubPCFOL}^=\,$ that correspond to different levels of expressiveness of the axioms. In contrast to the previous subsection, these are not orthogonal features, but rather we get a hierarchy of expressiveness.

3.4.2.1 First-order Logic  This is given by $\text{SubPCFOL}^=\,$, which trivially is a subinstitution of itself.

3.4.2.2 Positive Conditional Logic  Positive conditional logic more precisely means: universally quantified positive conditional logic. Usually this means that formulas are restricted to universally quantified implications that consist of a premise that is a conjunction of atoms, and a conclusion that is an atom:

$$\forall x_1 : s_1 \ldots \forall x_k : s_k \bullet \varphi_1 \land \ldots \land \varphi_m \Rightarrow \varphi$$
Positive conditional means that the $\varphi_i$ must not implicitly contain negative parts. Usually, this condition is satisfied by atomic formulas. However, strong equations are implicit implications (if one side is defined, then so is the other, and they are equal), and the premise of an implication is a negative part of a formula. Hence, strong equations may not occur in the premises of positive conditional axioms (they are harmless in the conclusion, since the implicit premise can be thought of as an additional premise of the whole implication). The main motivation for this is that we want to use proof techniques such as conditional term rewriting and paramodulation [67] and semantical constructions such as initial models. These work only if strong equations are not allowed in the premises (see [16,5]).

Let $SubPC\text{Horn}^=$ be the restriction of $SubPCFOL^=$ to sentences of form

$$\forall x_1 : s_1 \ldots \forall x_k : s_k \bullet \varphi_1 \land \ldots \land \varphi_m \Rightarrow \varphi$$

where the $\varphi_i$ and $\varphi$ are atomic formulas such that none of the $\varphi_i$ is a strong equation.

### 3.4.2.3 Generalized Positive Conditional Logic

In the following, we generalize the above form of positive conditional formulas. Each formula of this more general kind is equivalent to a set of formulas of the standard conditional kind. Thus, there is an easy transformation from generalized positive conditional logic to plain positive conditional logic.

Within generalized positive conditional formulas, we also allow

- conjunctions of atoms in the conclusion (they can be removed by writing, for each conjunct, an implication with the original premise and the conjunct as conclusion), and
- equivalences instead of implications (an equivalence is equivalent to two implications).

Thus, let $SubPCGHorn^=$ be the restriction of $SubPCFOL^=$ to sentences of form

$$\forall x_1 : s_1 \ldots \forall x_k : s_k \bullet \varphi_1 \land \ldots \land \varphi_m \Rightarrow \psi_1 \land \ldots \land \psi_n$$

where the $\varphi_i$ and $\psi_j$ are atomic formulas such that none of the $\varphi_i$ is a strong equation, or of form

$$\forall x_1 : s_1 \ldots \forall x_k : s_k \bullet \varphi_1 \land \ldots \land \varphi_m \Leftrightarrow \psi_1 \land \ldots \land \psi_n$$

where the $\varphi_i$ and $\psi_j$ are atomic formulas that are not strong equations.
3.4.2.4 Atomic Logic Let $SubPCAtom^=$ be the restriction of $SubPCFOL^=$ to sentences of form

$$\forall x_1 : s_1 \ldots \forall x_n : s_n \bullet \varphi$$

where $\varphi$ is an atomic formula not being a strong equation.

This is the restriction of conditional logic to unconditional formulas. Strong equations are removed due to their conditional nature: in [53] it is proved that strong equations can simulate positive conditional formulas.

3.4.3 A Terminology for Naming CASL Substitutions

We now give a two-component name to the various substitutions that can be obtained by combining CASL’s features. The first component is a vector of tokens. The presence (or absence) of a token denotes the presence (or absence) of a corresponding feature (cf. Section 3.4.1). The second component determines the level of expressiveness due to Section 3.4.2.

We assign the following tokens to the features:

- $Sub$ stands for subsorting,
- $P$ stands for partiality,
- $C$ stands for sort generation constraints, and
- an equality symbol (=) stands for equality.

There is a naming problem with the predicate feature. Firstly, the letter $P$ already stands for partiality. Secondly, $FOL$ for first-order logic or $Horn$ for Horn clause logic have become quite standard, but do not contain a token corresponding to predicates. Therefore, we deal with the predicate feature together with the levels of expressiveness from Section 3.4.2:

With predicates, we have the following endings:

- $FOL$ stands for the unrestricted form of axioms (first-order logic),
- $G\text{Horn}$ stands for the restriction to generalized positive conditional logic,
- $H\text{orn}$ stands for the restriction to positive conditional logic,
- $A\text{tom}$ stands for the restriction to atomic logic.

Without predicates, we have the following endings:

- $FA\text{l}g$ stands for the unrestricted form of axioms (first-order logic),
- $G\text{Con}d$ stands for the restriction to generalized positive conditional logic,
- $C\text{ond}$ stands for the restriction to positive conditional logic,
- $Eq$ stands for the restriction to atomic logic.

Any subset of the set of the four tokens $Sub, P, C$ and $=$, followed by any of
the eight above introduced endings now denotes the subinstitution obtained by intersecting

- the subinstitution of $\text{SubPCFOL}^=$ corresponding to the ending with
- the intersection of all the substitutions of $\text{SubPCFOL}^=$ associated to those letters not occurring in the set of tokens.

We finally adopt the convention that the equality sign $=$ is always put at the end, as a superscript.

### 3.4.4 Some Interesting Substitutions of Casl

This section shall help to understand what the above naming scheme means in practice.

$\text{SubPCFOL}^=$ (read: sorted partial constraint first-order logic with equality). This is the logic of CAsl itself!

$\text{SubPFOL}^=$ (read: subsorted partial first-order logic with equality). CAsl without sort generation constraints. This is described in [17].

$\text{FOL}^=$ Standard many-sorted first-order logic with equality.

$\text{PFOL}^=$ Partial many-sorted first-order logic with equality.

$\text{FOAlg}^=$ First-order algebra (i.e., no predicates).

$\text{SubPHorn}^=$ This is the positive conditional fragment of CAsl. It has two important properties:

1. Initial models and free extensions exist (see Theorem 4.16).
2. Using a suitable encoding of subsorting and partiality, one can use conditional term rewriting or paramodulation [67] for theorem proving.

$\text{SubPCHorn}^=$ The positive conditional fragment plus sort generation constraints. Compared with $\text{SubPHorn}^=$, one has to add induction techniques to the theorem proving tools.

$\text{PCond}^=$ These are Burmeister’s partial quasi-varieties [14] modulo the fact the Burmeister does not have total function symbols. But total function symbols can be easily simulated by partial ones, using totality axioms, as in the partly total algebras of [15]. A suitable restriction leads to Reichel’s HEP-theories [70]. Meseguer’s Rewriting Logic [48] can be embedded into $\text{PCond}^=$.

$\text{Horn}^=$ This is Eqlog [34,67]. By further restricting this we get Membership Equational Logic $\text{MEqtl}^=$ [49], Equational Type Logic [46] and Unified Algebras [63]. Of course, Membership Equational Logic, Equational Type Logic and Unified Algebras are not just restrictions of $\text{Horn}^=$, but all have been invented in order to represent more complex logics within a subset of $\text{Horn}^=$.

$\text{Horn}^=$ Logic Programming (Pure Prolog) [45].

$\text{SubCond}^=$ Subsorted conditional logic. This is similar but not equal to OBJ [38], see Section 10.
Cond$^=$ This is many-sorted conditional equational logic [76].

SubPAtom The atomic subset of CASL. Unconditional term rewriting becomes applicable.

SubPCAtom The atomic subset plus sort generation constraints.

Eq$^=$ This is the classical equational logic [37].

CEq$^=$ Equational logic plus sort generation constraints.

In the literature, some of the above institutions are typically defined in a way allowing empty carrier sets, while CASL excludes empty carriers. This problem is discussed in Section 11.2.

4 Representations Among Subinstitutions of CASL

In order to relate subinstitutions of CASL, we relate their underlying institutions. We therefore use the notion of institution representations (also called simple maps of institutions) [47,74] introduced in Section 2.

4.1 The First-Order Level

The diagram in Fig. 3 shows that the first-order subinstitutions of CASL have all the same expressiveness, except from FOL, which is a bit weaker (FOL$^=$ can be represented in FOL only with a representation admitting model expansion, but not as a subinstitution). Some of the arrows are labeled with numbers in brackets; this refers to later subsections where the corresponding representations are described in detail. Obvious subinstitution representations are not labeled.

Another diagram, shown in Fig. 4, can be obtained by adding a “C” to each institution in Fig. 3. We also have added the institution SOL$^=$ (second-order logic with equality), a superinstitution of FOL$^=$, which is strong enough to express sort generation constraints directly. The institutions in Fig. 4 are strictly more expressive than those in Fig. 3, since sort generation constraints cannot be expressed within first-order logic (since sort generation constraints can be used to specify the natural numbers up to isomorphism, this follows from Gödel’s incompleteness theorem).

The institution SOL$^=$ of second-order logic can be described as follows:

Signatures Signatures and signature morphisms are those of FOL$^=.$

Models Models, model homomorphisms and reducts are that of FOL$^=.$

Sentences Σ-sentences may contain variables that may be typed not only with sorts, but also with function types $w \rightarrow s$ or predicate types $\text{pred}(w)$,
Index for arrow types:

1. Admits model expansion: Borrowing of entailment and refinement for flat specifications
2. Strongly persistently liberal: Lifting of free constructions, implies 1.
3. Embedding: Borrowing of entailment for certain structured specifications including free
5. Substitution: Borrowing of entailment and refinement for all structured specifications

where \( w \in S^* \), \( s \in S \). Quantification within sentences over variables of these higher types is also allowed. Variables of function or predicate types may be applied to arguments, like function and predicate symbols in \( FOL^= \).

**Satisfaction** The satisfaction is defined much as in \( FOL^= \), with the exception that valuations map variables of a function type to functions of that type, and variables of a predicate type to predicates of that type.  

Fig. 3. The first-order level

4.1.1 (1): Mapping \( FOL^= \) to \( FOL \)

The idea here is simply to replace equality by a congruence relation.

**Signatures** A \( FOL^= \)-signature \( \Sigma \) is mapped to the \( FOL \)-presentation \( \Phi(\Sigma) \) that extends \( \Sigma \) by adding predicate symbols \( \equiv: s \times s \) (overloaded for each sort \( s \)) that are axiomatized to be a congruence (also w.r.t. the predicates).

**Models** A model \( M \) is translated by factoring its carriers w.r.t. \( \equiv \). The functions and predicates can act on the equivalence classes because \( \equiv \) is a congru-
Fig. 4. The first-order level with sort generation constraints (index: see Fig. 3).

ence. A homomorphism \( h : M \rightarrow M' \) is translated to \( \bar{h} \) with \( \bar{h}([a]) = [h(a)] \) (where \([a]\) is the congruence class of \( a \)). This is well-defined since \( a \equiv_M a' \) implies \( h(a) \equiv_M h(a') \) (predicates are preserved by homomorphisms).

**Sentences** Sentence translation is done by replacing \( = \) by \( \equiv \).

**Satisfaction** To prove the representation condition, define a mapping on valuations as follows: Given a \( \Phi(\Sigma) \)-model \( M' \) and a valuation \( \nu : X \rightarrow M' \), define \( \beta(\nu) : X \rightarrow \beta_{\Sigma}(M') \) by

\[
\beta(\nu)_s(x) := [\nu(x)] \quad \text{for} \quad x \in X_s.
\]

We then have

\[
\beta(\nu)^\#(t) = [\nu^\#(t)],
\]

which can be easily proved by induction over \( t \), using the congruence axioms.

By induction over \( \varphi \in \text{Sen}(\Sigma) \), we can now show that

\[
\nu \vdash \alpha_{\Sigma}(\varphi) \iff \beta(\nu) \vdash \varphi.
\]

Concerning e.g. strong equations, we have \( \nu \vdash \alpha_{\Sigma}(t_1 = t_2) \iff \nu \vdash t_1 \equiv t_2 \)
iff \( [\nu^\#(t_1)] = [\nu^\#(t_2)] \iff \beta(\nu)^\#(t_1) = \beta(\nu)^\#(t_2) \iff \beta(\nu) \vdash t_1 = t_2 \).

Concerning predicate applications, \( \nu \vdash \alpha_{\Sigma}(p(t_1, \ldots, t_n)) \iff \nu \vdash p(t_1, \ldots, t_n) \)
iff \( (\nu^\#(t_1), \ldots, \nu^\#(t_n)) \in P_M \iff (\beta(\nu))^\#(t_1), \ldots, (\beta(\nu))^\#(t_n)) \in P_{\beta_{\Sigma}(M')} \iff \beta(\nu) \vdash p(t_1, \ldots, t_n) \).

Concerning quantification, \( \nu \vdash \forall x : s \bullet \alpha_{\Sigma}(\psi) \iff \text{for all } \xi : X \cup \{x : s\} \rightarrow M' \text{ extending } \nu \text{ on } X \setminus \{x : s\}, \xi \vdash \alpha_{\Sigma}(\psi) \iff \text{(by induction hypothesis)} \)
for all $\xi: X \cup \{ x : s \} \rightarrow M'$ extending $\nu$ on $X \setminus \{ x : s \}$, $\beta(\xi) \vdash \varphi$ iff for all $\rho: X \cup \{ x : s \} \rightarrow \beta_\Sigma(M')$ extending $\beta(\nu)$ on $X \setminus \{ x : s \}$, $\rho \vdash \varphi$ iff $\beta(\nu) \vdash \forall x : s \bullet \varphi$.

The other cases are treated similarly. The representation condition now follows by noting that $\beta$ is surjective on valuations. □

The representation can be seen to admit model expansion as follows: Given a $\Sigma$-model $M$ in $FOL^=\Sigma$, turn it into a $\Phi(\Sigma)$-model by letting $\equiv$ be interpreted as the identity relation. This model is a pre-image of $M$ under the model translation. However, the representation is not persistently liberal: for example, let $\Sigma$ consist of just one sort, let $M$ be the $\Sigma$-model consisting of two points, and let $M'$ be the $\Phi(\Sigma)$-model consisting of two equivalent points. Then there is just one homomorphism from $M$ to $\beta_\Sigma(M')$. If the representation were persistently liberal, there would have to be just one homomorphism from $\gamma_\Sigma(M)$ to $M'$. But this is impossible, since then $\gamma_\Sigma(M)$ would have to be empty (even if empty carriers were allowed, this still would contradict the requirement $\beta_\Sigma(\gamma_\Sigma(M)) = M$).

Note that the representation cannot be generalized to the level of sort generation constraints in a straightforward way, because these are not preserved along taking quotients: Let $Nat$ be the signature consisting of a sort $Nat$ and two total function symbols $0: Nat$ and $suc: Nat \rightarrow Nat$. Let $M'$ be the $\Phi(Nat)$-model consisting of two copies of the natural numbers, which are identified by the congruence relation. Since the constant $0$ yields the zero only of one copy of the naturals, $M$ is not term-generated. However $\beta_\Sigma(M)$ consist of just one copy of the naturals and therefore is term-generated.

4.1.2 (2) and (2') Mapping $(C)FOL^=\Sigma$ to $(C)FOAlg^=$

Here, the idea is to replace predicates by Boolean-valued functions. Note that the Booleans can be axiomatized monomorphically in $FOAlg^=$.

Signatures A signature is mapped by adding the presentation $Bool$ and replacing each predicate symbol $p : w$ by a total function symbol $f_p: w \rightarrow Bool$

**spec** $Bool =$

**sort** $Bool$

**ops** $True, False : Bool$

**forall** $b : Bool$

- $b = True \lor b = False$
- $\neg True = False$

**end**

Strictly speaking, $BOOL$ has to be renamed in order to become disjoint with the signature.
**Models** A model is translated by replacing each $\text{Bool}$-valued function by the corresponding predicate.

**Sentences** Sentence translation is done by replacing atomic formulas

$$p_w(t_1, \ldots, t_n)$$

by

$$(f_p)_{w, \text{Bool}}(t_1, \ldots, t_n) = \text{True_{Bool}}.$$ 

Sort generation constraints are left unchanged.

**Satisfaction** The representation condition is proved in a straightforward way. \(\Box\)

It is also straightforward to show that this gives a model-bijective institution representation. Note, however, that we do not get a subinstitution representation. This is because homomorphisms in $\text{FOL}^\approx$ need to preserve just truth (but not falsehood) of predicates, while homomorphisms in the representation in $\text{FOAlg}^\approx$ need to preserve both truth and falsehood of (represented) predicates.

Indeed, we conjecture that there cannot be a subinstitution representation from $\text{FOL}^\approx$ to $\text{FOAlg}^\approx$ since $\text{FOL}^\approx$ is strictly more expressive than $\text{FOAlg}^\approx$ w.r.t. a construct that actually exploits the homomorphism: with the **free** construct, it is possible to express the transitive closure of an arbitrary (parameter) relation, while we conjecture that this is not possible in $\text{FOAlg}^\approx$.

4.1.3 (3) and (3'): Mapping $\text{SubP}(C)\text{FOL}^\approx$ to $P(C)\text{FOL}^\approx$

The translation of $\text{SubP}(C)\text{FOL}^\approx = P(C)\text{FOL}^\approx$ is trivial, since $\text{SubPCFOL}^\approx$ is defined in terms of $\text{PCFOL}^\approx$:

**Signatures** A signature $\Sigma$ is mapped to the presentation $(\hat{\Sigma}, \hat{J}(\Sigma))$.

**Models** Model translation is the identity.

**Sentences** Sentence translation is the identity.

**Satisfaction** The representation condition follows immediately. \(\Box\)

This trivially gives a subinstitution representation.

4.1.4 (4) and (4a): Mapping $\text{PFO} \approx$ to $\text{FOL}^\approx$

A translation of $\text{PFO} \approx$ into $\text{FOL}^\approx$ is described in [19]. We refine this translation here. The main idea is to use a definedness predicate to divide each carrier into “defined” and “undefined” elements. The “defined” elements represent ordinary values, while the “undefined” elements all represent the undefined.
Partial functions thus can be totalized; they possibly yield an “undefined” element. We specify that there is at least one “undefined” element \( \bot \); however, it may be not the only one.

**Signatures** A \( PFOL^- \)-signature \( \Sigma = (S, TF, PF, P) \) is translated to a \( FOL^- \)-presentation having the signature

\[
\text{Sig}(\Phi(\Sigma)) = (S, TF \cup PF \cup \{ \bot : s \in S \}, P \cup \{ D : s \in S \})
\]

and the set of axioms \( \text{Ax}(\Phi(\Sigma)):\)

\[
\begin{align*}
\exists x : s \cdot D_s(x) & \quad s \in S \\
\neg D_s(\bot) & \\
D_s(f(x_1, \ldots, x_n)) & \iff \bigwedge_{i=1..n} D_{s_i}(x_i) \quad f : s_1, \ldots, s_n \to s \in TF \\
D_s(g(x_1, \ldots, x_n)) & \to \bigwedge_{i=1..n} D_{s_i}(x_i) \quad g : s_1 \ldots s_n \to s \in PF \\
p(x_1, \ldots, x_n) & \to \bigwedge_{i=1..n} D_{s_i}(x_i) \quad p : s_1 \ldots s_n \in P
\end{align*}
\]

\( D \) plays the role of a definedness predicate; the elements inside \( D \) are called “defined”, those outside \( D \) are called “undefined”. The axioms in the signature translation state that there is at least one “defined” element (1), that \( \bot \) is an “undefined” element (2), total functions are indeed total (3) and all functions ((3), (4)) and predicates (5) are strict.

A signature morphism \( \sigma \) is translated to a presentation morphism \( \Phi(\sigma) \) which acts as \( \sigma \) on those parts of \( \Phi(\Sigma) \) being included from \( \Sigma \), while it maps the added structure for a signature component to the added structure for a mapped component.

**Models** A \( \Phi(\Sigma) \)-structure \( M \) (in \( FOL^- \)) is translated to the partial \( \Sigma \)-structure \( \beta_\Sigma(M) = M' \) (in \( PFOL^- \)) with

\[
\begin{align*}
\bullet & \quad M'_s = (D_s)_M \text{ for } s \in S, \\
\bullet & \quad f_M' \text{ is } f_M \text{ restricted to } M'_w \text{ for } f : w \to s \in TF \text{ (this is a total function by (3)),} \\
\bullet & \quad g_M'(a_1, \ldots, a_n) = \begin{cases} g_M(a_1, \ldots, a_n), & \text{if } g_M(a_1, \ldots, a_n) \in M'_s \text{ for } g : w \to s \in PF, \\
\text{undefined,} & \text{otherwise}
\end{cases} \\
\bullet & \quad p_M' = p_M \cap M'_w \text{ for } p : w \in P.
\end{align*}
\]

Homomorphisms are translated by restricting them to the carriers of \( M' \).

This is well-defined since predicates are preserved by homomorphisms.

**Sentences** The sentence translation keeps the structure of the sentences and maps strong and existential equality to appropriate circumscriptions using the definedness predicate \( D \). Definedness is mapped to \( D \), and quantifiers are relativized to the set of all defined elements.

Formally, a \( \Sigma \)-sentence \( \varphi \) (in \( PFOL^- \)) is translated to the \( \Phi(\Sigma) \)-sentence

\[
\begin{align*}
\alpha_\Sigma(\text{def}(t)) & = D_s(t) \\
\alpha_\Sigma(\varphi \land \psi) & = \alpha_\Sigma(\varphi) \land \alpha_\Sigma(\psi) \\
\alpha_\Sigma(p(t_1, \ldots, t_n)) & = p(t_1, \ldots, t_n) \\
\alpha_\Sigma(F) & = F \\
\alpha_\Sigma(\varphi \Rightarrow \psi) & = \alpha_\Sigma(\varphi) \Rightarrow \alpha_\Sigma(\psi) \\
\alpha_\Sigma(\forall x : s \cdot \varphi) & = \forall x : s \cdot (D_s(x) \Rightarrow \alpha_\Sigma(\varphi))
\end{align*}
\]

60
**Satisfaction** To prove the representation condition, we need to talk about partial variable valuations. This is necessary since a valuation in a $\Phi(\Sigma)$-model $M'$ may map an “undefined” element to a variable. In the $\Sigma$-model $M = \beta_{\Sigma}(M')$, this should correspond to a truly undefined variable. Thus, we consider partial variable valuations $\nu: X \rightarrow M$. The evaluation $\nu^\#$ of terms and the satisfaction $\nu \vdash_\Sigma$ of formulas is defined as before (see Section 3.1), with the following two exceptions:

- $\nu^\#_s(x) = \begin{cases} 
\nu(x), & \text{if this is defined} \\
\text{undefined, otherwise} & \text{for all } x \in X_s \text{ and all } s \in S;
\end{cases}$

- $\nu \vdash_\Sigma (\forall x : s \bullet \varphi)$ iff for all valuations $\xi: X \cup \{x : s\} \rightarrow M$ which extend $\nu$ on $X \setminus \{x : s\}$ and are defined on $x : s$, we have $\xi \vdash_\Sigma \varphi$.

Note that the new definitions of $\nu^\#$ and $\vdash$ coincide with the old ones for total valuations. To achieve this, it is crucial to require the extended valuation $\xi$ to be defined on $x$ in the clause for satisfaction of quantified formulas above.

We now have the following lemma:

**Lemma 4.1** For each $\Sigma$-formula $\varphi$ and each (total) valuation $\nu: X \rightarrow M'$ into a $\Phi(\Sigma)$-model $M'$, define the partial valuation $\rho: X \rightarrow \beta_{\Sigma}(M')$ to be

$$
\rho(x) = \begin{cases} 
\nu(x), & \text{if } \nu(x) \in (D_s)_{M'} \\
\text{undefined, otherwise} & (x \in X_s)
\end{cases}
$$

Then we have

(a) $\rho^\#(t)$ is defined iff $\nu^\#(t) \in (D_s)_{M'}$ for all $\Sigma$-terms $t$ of sort $s$.

(b) In the case that one of the sides of (1) holds, we have

$$
\rho^\#(t) = \nu^\#(t).
$$

(c) $\rho \vdash_{\Sigma}^{P\text{FOL}} \varphi$ iff $\nu \vdash_{\text{Sig}(\Phi(\Sigma))}^{\text{FOL}} \alpha_\Sigma(\varphi)$.

**Proof.** (a) and (b): By induction over the structure of $t$. For variables, we need just use the definition of $\rho$. For applications of total or partial function symbols, use axioms (3) or (4) in $\Phi(\Sigma)$.

(c): By induction over the structure of $\varphi$. Considering existence equations, we have

- $\rho \vdash_{\Sigma}^{P\text{FOL}} t_1 \equiv t_2$ if $\rho^\#(t_1)$ and $\rho^\#(t_2)$ are defined and equal
- $\nu^\#(t_1) \in (D_s)_M$ and $\nu^\#(t_2) \in (D_s)_M$ and $\nu^\#(t_1) = \nu^\#(t_2)$
- $\nu \vdash_{\text{Sig}(\Phi(\Sigma))} t_1 = t_2 \land D_s(t_1)$
- $\nu \vdash_{\text{Sig}(\Phi(\Sigma))} \alpha_\Sigma(t_1 \equiv t_2)$

Considering universally quantified formulas, we have

61
$\rho \vdash_{\Sigma}^{P\text{FOL}} \forall x : s \bullet \varphi$

iff for all $\xi : X \cup \{x : s\} \longrightarrow \beta_{\Sigma}(M')$ extending $\rho$ on $X \setminus \{x : s\}$ and being defined on $\{x : s\}$, $\xi \vdash_{\Sigma}^{P\text{FOL}} \varphi$

iff for all $\tau : X \cup \{x : s\} \longrightarrow M'$ extending $\nu$ on $X \setminus \{x : s\}$ for which

$x \in (D_\tau)_{M'}$, $\tau : \vdash_{\text{Sig}(\Phi(\Sigma))}^{\text{FOL}} \alpha_\Sigma(\varphi)$

iff $\nu \vdash_{\text{Sig}(\Phi(\Sigma))}^{\text{FOL}} \alpha_\Sigma(\forall x : s \bullet \varphi)$

The other cases are treated similarly. $\square$

The representation condition can now be shown as follows. Let $\varphi$ be a $\Sigma$-sentence and $\nu : \emptyset \longrightarrow M'$ and $\rho : \emptyset \longrightarrow \beta_{\Sigma}(M')$ be the unique empty valuations. Then

$\beta_{\Sigma}(M') \models_{\text{PFOL}}^{\Sigma} \varphi$

iff $\rho \vdash_{\Sigma}^{P\text{FOL}} \varphi$

iff $\nu \vdash_{\text{Sig}(\Phi(\Sigma))}^{\text{FOL}} \alpha_\Sigma(\varphi)$

iff $M' \models_{\text{Sig}(\Phi(\Sigma))}^{\text{FOL}} \alpha_\Sigma(\varphi)$ $\square$

The model translation can be easily shown to be surjective: For a partial $\Sigma$-structure $M$, form its one-point completion by just adding one element, $\ast$, to all carriers, which is the interpretation of $\bot$. The functions map $\ast$ to itself and behave as in $M$ otherwise, where undefinedness of partial functions is mapped to $\ast$. Predicates are false on $\ast$. The predicate $D$ is true everywhere except on $\ast$. Thus, our institution representation admits model expansion. Moreover, with the argument of Example 2.30, the representation also has the weak $D$-amalgamation property, where $D$ is the class of all injective signature morphisms.

**Proposition 4.2** The above constructed institution representation is strongly persistently liberal.

**Proof.** Define $\gamma$ as follows. Put $(\gamma_{\Sigma}(M))_s := M_s \cup \widetilde{M}_s$, where

\begin{itemize}
  \item $\widetilde{M} \subseteq T_{\text{Sig}(\Phi(\Sigma))}([M])$ is the least sorted set satisfying
    \begin{itemize}
      \item $\bot \in \widetilde{M}_s$
      \item $f(a_1, \ldots, a_n) \in \widetilde{M}_s$ for $f \in TF_{w,s} \cup PF_{w,s}$, $w = s_1 \ldots s_n$, $(a_1 \ldots a_n) \in (M_w \cup \widetilde{M}_w) \setminus dom(f_M)$
    \end{itemize}
  \item $(D_s)_{\gamma_{\Sigma}(M)} := M_s$
  \item $(\bot_s)_{\gamma_{\Sigma}(M)} := \bot$
  \item $f_{\gamma_{\Sigma}(M)}(a_1, \ldots, a_n) := \begin{cases} f_M(a_1, \ldots, a_n), & \text{if } (a_1, \ldots, a_n) \in dom(f_M) \\ f(a_1, \ldots a_n), & \text{otherwise} \end{cases}$
    for $f : w \longrightarrow s \in TF \cup PF$,
  \item $p_{\gamma_{\Sigma}(M)} = p_M$ for $p : w \in P$.
\end{itemize}

Now (1) of $\Phi(\Sigma)$ above is satisfied since the carriers of $M$ are non-empty. (2) holds by definition of $(D_{\gamma_{\Sigma}(M)})_s$, and (3) and (4) hold by definition of $f_{\gamma_{\Sigma}(M)}$. 
(5) holds by definition of $p_{\Sigma(M)}$.

Clearly, we have $\beta_{\Sigma}(\gamma_{\Sigma}(M)) = M$. To show the universal property of

$$M \overset{id}{\longrightarrow} \beta_{\Sigma}(\gamma_{\Sigma}(M)),$$

let $h: M \longrightarrow \beta_{\Sigma}(N)$ be a homomorphism. Since $\beta_{\Sigma}(N) \subseteq N_s$, we can define $h^\#: \gamma_{\Sigma}(M) \longrightarrow N$ by

$$h^\#(a) = \begin{cases} h(a), & \text{if } a \in M_s \\ \nu^\#(a), & \text{if } a \in \tilde{M}_s \end{cases}$$

where $\nu: |M| \longrightarrow N$ is defined by $\nu(a) = h(a)$. The first case is determined by the requirement that $\beta_{\Sigma}(h^\#) = h$, while the second case is determined by the requirement that $h^\#$ is a homomorphism. Thus, $h^\#$ is the unique homomorphism from $\gamma_{\Sigma}(M)$ to $N$ that is mapped to $h$ under $\beta_{\Sigma}$. \(\square\)

Concerning model-bijectivity, by adding to $\Phi(\Sigma)$ the sentences $\neg x = \bot_s \Rightarrow D_s(x)$ (for each $s \in S$), $\bot$ becomes the unique “undefined” element. With this, we get an institution representation (4a) that is model-bijective. Moreover, the translation of strong equations can be simplified by translating them just to ordinary equations. However, the representation then is no longer persistently liberal.

4.1.5 (4b): Mapping $\text{PFO}_L^=$ to $FOL_-$ via functions as graphs

The representations of the previous section are quite good. However, in some cases, one needs a representation with stronger properties (like being an embedding or a substitution representation). Indeed, it is possible to represent $\text{PFO}_L^=$ within $FOL_-$ as a substitution. This is achieved by representing functions by their graphs. The benefit of obtaining a substitution representation comes at a high price, however: the translation of sentences will turn out to be rather clumsy.

**Signatures** A $\text{PFO}_L^=$-signature $(S, TF, PF, P)$ is translated to the $FOL_-$-presentation $\langle (S, TF, P \cup G), \Gamma \rangle$, where $G$ contains a predicate symbol

$$G^f : s_1 \ldots s_n$$

and $\Gamma$ contains an axiom

$$\forall x_1 : s_1, \ldots, x_n : s_n; y, z : s \bullet G^f(x_1, \ldots, x_n, y) \land G^f(x_1, \ldots, x_n, z) \Rightarrow y = z$$

for each $f: s_1, \ldots, s_n \longrightarrow s \in PF$. The predicate $G^f$ shall hold the graph of the partial operation $f$, and the axiom states that $G^f$ is right-unique, that is, the graph of a partial operation.
**Models** \( \beta_{\Sigma} \) takes a \( \Phi(\Sigma) \)-model \( M \) and replaces each relation \( G^f_M \) by the partial operation with graph \( G^f_M \).

**Sentences** Following Burmeister [13, p. 325], we translate sentences into a relational form. \( \alpha_{\Sigma} \) is defined inductively as follows:

- \( \alpha_{\Sigma}(f(t_1, \ldots, t_n) = t_0) = \)
  \[ \exists y_0 : s_0 \ldots y_n : s_n \bullet G^f(y_1, \ldots, y_n, y_0) \land \alpha_{\Sigma}(t_0 = y_0) \land \cdots \land \alpha_{\Sigma}(t_n = y_n) \]
  for \( f : s_1 \ldots s_n \rightarrow s_0 \in PF \), where \( y_0, \ldots, y_n \) are new variables,

- \( \alpha_{\Sigma}(f(t_1, \ldots, t_n) = t_0) = \)
  \[ \exists y_0 : s_0 \ldots y_n : s_n \bullet f(y_1, \ldots, y_n) = y_0 \land \alpha_{\Sigma}(t_0 = y_0) \land \cdots \land \alpha_{\Sigma}(t_n = y_n) \]
  for \( f : s_1 \ldots s_n \rightarrow s_0 \in TF \), where \( y_0, \ldots, y_n \) are new variables,

- \( \alpha_{\Sigma}(x = t) = \alpha_{\Sigma}(t = x) \), if \( t \) is not a variable,

- \( \alpha_{\Sigma}(x = y) = x = y \),

- \( \alpha_{\Sigma}(t_1 = t_2) = (\alpha_{\Sigma}(x = t_1) \Rightarrow \alpha_{\Sigma}(x = t_2)) \land (\alpha_{\Sigma}(x = t_1) \Rightarrow \alpha_{\Sigma}(x = t_2)) \),

  where \( x \) is a new variable,

- \( \alpha_{\Sigma}(p(t_1, \ldots, t_n)) = \)
  \[ \exists y_0 : s_0 \ldots y_n : s_n \bullet p(y_1, \ldots, y_n) \land (t_1 = y_1) \land \cdots \land \alpha_{\Sigma}(t_n = y_n) \]
  for \( p : s_1 \ldots s_n \in P \), where \( y_1, \ldots, y_n \) are new variables,

- \( \alpha_{\Sigma}(\text{def}(t)) = \exists x : s \bullet \alpha_{\Sigma}(t = x) \), where \( x \) is a new variable and \( t \) is of sort \( s \),

- \( \alpha_{\Sigma}(F) = F \),

- \( \alpha_{\Sigma}(\varphi \land \psi) = \alpha_{\Sigma}(\varphi) \land \alpha_{\Sigma}(\psi) \),

- \( \alpha_{\Sigma}(\varphi \Rightarrow \psi) = \alpha_{\Sigma}(\varphi) \Rightarrow \alpha_{\Sigma}(\psi) \),

- \( \alpha_{\Sigma}(\forall x : s \bullet \varphi) = \forall x : s \bullet \alpha_{\Sigma}(\varphi) \).

**Satisfaction** The representation condition essentially is the proposition on page 326 of [13].

### 4.1.6 (4a', 4a') and (4b') Mapping PCFOL\( ^= \) to CFOL\( ^= \)

The representation (4a') extends the model-bijective institution representation (4a) described above. Due to the need to translate sort generation constraints, we must be able to generate also the “undefined” elements in the CFOL\( ^= \)-models. In order to be able to correctly model generatedness w.r.t. to partial functions, we have to ensure generatedness of the “undefined” elements. This is ensured by the axioms \( \neg x = s \models D_s(x) \) (for each \( s \in S \)). Since then there is just one “undefined” element, namely \( \bot \), the “undefined” elements are now term generated. However, this method leads to a loss of the persistent liberality of the representation.

**Signatures** Signatures are translated as in (4a).

---

64
Models Models are translated as in (4a).

Sentences First-order sentences are translated as in (4a). A $\Sigma$-sort generation constraint $(\beta, F, \theta : \Sigma \rightarrow \Sigma)$ is translated to $(\beta, F, \cup \{ s | s \in S \}, \theta')$, where $\theta' : \Phi(\Sigma) \rightarrow \Phi(\Sigma)$ is the extension of $\theta$ to $\Phi(\Sigma)$ that is defined by mapping the added symbols in $\Phi(\Sigma)$ to the corresponding added symbols in $\Phi(\Sigma)$.

Satisfaction The representation condition for formulas is proved as in (4a). Concerning sort generation constraints, note that model translation just removes the interpretation of $\bot$ from the carriers. Since $\bot$ cannot occur in $\beta$ and moreover all functions are strict, generatedness w.r.t. $\beta$ in the model $\beta(\Sigma')$ (where interpretations of $\bot_s$ have been removed) is the same as generatedness w.r.t. $\beta \cup \{ s | s \in S \}$ in the model $\Sigma'$ (where interpretations of $\bot_s$ are not removed). □

If one wants to have a strongly persistently liberal representation (call it (4′), since it extends (4)) also translating sort generation constraints, one can restrict sort generation constraints in $PCFOL^=$ to those including total function symbols only; in this case, the introduction of a unique undefined element is not necessary.

The same restriction of sort generation constraints in $PCFOL^=$ also has to be done if we want to extend (4b) to (4b′): sort generation constraints can just be left as they are in this case.

4.1.7 (5), (5a) and (5b): Mapping $SubPFOL^=$ to $SubFOL^=$

The idea here is again to add new elements (among them $\bot$) that represent “undefined”, but to retain the old sorts without the undefined elements. That is, the sort set is doubled: each sort $s$ now gets a companion supersort $s'$, which contains all values of $s$ plus undefined elements. Since we keep the old sorts, the definedness predicate is just membership in the old sorts. Moreover, we add the companion supersort only to those sorts for which it is necessary, i.e. for those where a term involving partial function symbols exists.

In the signature translation, we introduce two versions for each function and predicate symbol: the original one and its (overloaded) strict extension that just propagates the undefined element. The strict extension is necessary if we want to apply a function or predicate symbol to a term involving partial function symbols, since this term can be translated only to a term of sort $s'$.

Since all partial functions with result sort $s$ are totalized to functions with result sort $s'$, also the partial projections have to be totalized. For them, we introduce new special function symbols $\text{proj}$. We do not need an axiom like $\exists x : s \bullet D_s(x)$ in (4), because subsorts are interpreted with non-empty carriers.

65
Signatures Let \( \Sigma = (S, TF, PF, P, \leq_S) \) be a \( \text{SubPFOL}^= \)-signature. Let
- \( S^2 = \{ s \in S \mid \text{there exists a variable system } X \text{ and a } \Sigma(X) \text{-term of sort } s \text{ built using at least one partial function symbol or projection cast} \} \),
- \( S' = S \cup \{ s^2 \mid s \in S^2 \} \) and \( Q(s) = \begin{cases} s^2, & \text{if } s \in S^2 \\ s, & \text{otherwise} \end{cases} \)

Then \( \Sigma \) is translated to the \( \text{SubFOL}^= \)-signature \( \text{Sig}(\Phi(\Sigma)) \) consisting of the sort set \( S' \), ordered by
- \( s \leq s' \) iff \( s \leq s' (s, s' \in S) \);
- \( s \leq s'' \) iff \( s \leq s' (s, s' \in S) \);
- \( s^2 \not\leq s' \) \( s, s' \in S \), and
- \( s^2 \leq s' \) iff \( s \leq s', (s, s' \in S) \);

augmented by the following total function symbols:
- \( \bot : s^2 \) for \( s \in S^2 \)
- \( f : s_1, \ldots, s_n \to s \) for \( f : s_1, \ldots, s_n \to s \in TF \)
- \( f : s_1, \ldots, s_n \to (s^2) \) for \( f : s_1, \ldots, s_n \to ? s \in PF \),
- \( f : Q(s_1), \ldots, Q(s_n) \to (s^2) \) for \( f : s_1, \ldots, s_n \to s \in TF \cup PF, \{s_1, \ldots, s_n\} \cap S' \neq \emptyset \)
- \( \text{proj}: s \to (s^2) \) for \( s^2 \leq s \)
- \( \text{proj}: s \to (s^2) \) for \( s^2 \leq s \)

the following predicate symbols:
- \( p : s_1, \ldots, s_n \) for \( p : s_1, \ldots, s_n \in P, \)
- \( p : Q(s_1), \ldots, Q(s_n) \) for \( p : s_1, \ldots, s_n \in P, \{s_1, \ldots, s_n\} \cap S' \neq \emptyset \)

and the following axioms \( \text{Ax}(\Phi(\Sigma)) \):
- \( \neg \bot \in s \) for \( s \in S^2 \) (1)
- \( \forall x : s' \bullet \text{proj}_{(s', s^2)}(x) \in s \iff x \in s \) for \( s \leq s' \) (2)
- \( \forall x : s \bullet \text{proj}_{(s', s^2)}(\text{inj}_{(s, s^2)}(x)) = \text{inj}_{(s, s^2)}(x) \) for \( s \leq s' \) (3)
- \( \forall x, y : s' \bullet \text{proj}_{(s', s^2)}(x) = \text{proj}_{(s', s^2)}(y) \land \text{proj}_{(s', s^2)}(x) \in s \Rightarrow x = y \) for \( s \leq s' \) (4)

\( \forall x_1 : Q(s_1), \ldots, x_n : Q(s_n) \bullet f(x_1, \ldots, x_n) \in s \iff \bigwedge_{s \in s} x_i \in s_i \)
- \( f : s_1, \ldots, s_n \to s \in TF \) (5)
- \( \forall x_1 : Q(s_1), \ldots, x_n : Q(s_n) \bullet f(x_1, \ldots, x_n) \in s \Rightarrow \bigwedge_{s \in s} x_i \in s_i \)
- \( f : s_1, \ldots, s_n \to ? s \in PF \) (6)
- \( \forall x_1 : Q(s_1), \ldots, x_n : Q(s_n) \bullet p(x_1, \ldots, x_n) \Rightarrow \bigwedge_{s \in s} x_i \in s_i \)
- \( p : s_1, \ldots, s_n \in P \) (7)

The axioms of form (1) state that \( \bot \) is an “undefined” element. (2), (3) and (4) express the membership, projection and projection-injectivity axioms – these are those of the axioms used in the definition of \( \text{SubPCFOL}^= \).

\( ^\text{11} \text{We assume that } S \text{ does not contain sorts of the form } s^2. \text{ In a higher-order extension of \text{CASL}, there could be a type constructor } "\tau" \text{ behaving like our } "\ell". \)
(see Section 3.2) that refer to the partial projections. (5), (6) and (7) state
strictness of the functions and predicates, while (5) additionally expresses
totality of the total functions.

Signature morphisms are translated in a way analogous to that of Sec-
tion 4.1.4.

Models A \( \Phi(\Sigma) \)-structure \( M \) (in \( \mathcal{S}_{\text{SubFOL}}^\infty \)) is translated to the partial \( \Sigma \-
structure \( \beta(\Sigma)(M) \) (in \( \mathcal{S}_{\text{SubFOL}}^\infty \)) having carrier sets \( M_s \) for \( s \in S \) and total
functions and predicates inherited from \( M \), while partial functions \( g_{\beta(\Sigma)(M)} \)
are the corestrictions of \( g_M \) to \( M_s \) (where \( s \) is the result sort of \( g \)).

Again, we have a straightforward inverse: a \( \Sigma \)-structure \( M \) is extended
by adding new carriers \( M_s \) \( = M_s \oplus \{ * \} \), interpreting \( \bot \) as \( * \) and adding
the strict extensions that map \( * \) to itself. Thus, we have an institution
representation that admits model expansion.

Sentences A \( \Sigma \)-sentence in \( \mathcal{S}_{\text{SubFOL}}^\infty \) is translated to a \( \Phi(\Sigma) \)-sentence in
\( \mathcal{S}_{\text{SubFOL}}^\infty \) inductively as follows:

- \( \alpha_{\Sigma}(x : s) = x : s \)
- \( \alpha_{\Sigma}(f_{s_1, s_n,t_1, \ldots, t_n})(t_1, \ldots, t_n) = \begin{cases} f : s_1, \ldots, s_n \rightarrow s \in \text{TF} \\ \{ f_{s_1, \ldots, s_n, t_1}((\alpha_{\Sigma}(t_1), \ldots, \alpha_{\Sigma}(t_n))) \text{ if for } i = 1 \ldots n, \alpha_{\Sigma}(t_i) \text{ is of sort } s_i \\ \{ q_{s_1, \ldots, s_n, t_1}((\alpha_{\Sigma}(t_1), \ldots, \alpha_{\Sigma}(t_n))) \text{ otherwise} \end{cases} \)
- \( \alpha_{\Sigma}(p_{s_1, \ldots, s_n, t_1, \ldots, t_n}) = \begin{cases} p : s_1, \ldots, s_n \in P \\ \{ p_{s_1, \ldots, s_n, \alpha_{\Sigma}(t_1), \ldots, \alpha_{\Sigma}(t_n)} \text{ if for } i = 1 \ldots n, \alpha_{\Sigma}(t_i) \text{ is of sort } s_i \\ \{ q_{s_1, \ldots, s_n, \alpha_{\Sigma}(t_1), \ldots, \alpha_{\Sigma}(t_n)} \text{ otherwise} \end{cases} \)
- \( \alpha_{\Sigma}(\text{pr}_{s_1, s_2}(t)) = \begin{cases} \text{proj}_{s_1, s_2}((\alpha_{\Sigma}(t)), \text{ if } \alpha_{\Sigma}(t) \text{ is of sort } s_1 \wedge s_2, s_1 \leq s_2 \\ \text{proj}_{s_1, s_2}((\alpha_{\Sigma}(t)), \text{ otherwise} \end{cases} \)
- \( \alpha_{\Sigma}(\text{def}(t)) = \begin{cases} \alpha_{\Sigma}(t) \in s, \text{ if } \alpha_{\Sigma}(t) \text{ is of sort } s', s \in S' \\ T, \text{ otherwise} \end{cases} \)
- \( \alpha_{\Sigma}(t \in s_1) = \begin{cases} t \in s_1 \wedge \alpha_{\Sigma}(t) \text{ is of sort } s_1, \text{ if } \alpha_{\Sigma}(t) \text{ is of sort } s_1 \\ t \in s_1, \text{ otherwise} \end{cases} \)
- \( \alpha_{\Sigma}(t_1 = t_2) = \begin{cases} \alpha_{\Sigma}(t_1) \in s \wedge \alpha_{\Sigma}(t_2) \in s \Rightarrow \alpha_{\Sigma}(t_1) = \alpha_{\Sigma}(t_2), \text{ if } \alpha_{\Sigma}(t_1), \alpha_{\Sigma}(t_2) \text{ are of sort } s' \wedge \alpha_{\Sigma}(t_1) = \alpha_{\Sigma}(t_2), \text{ otherwise} \\ \alpha_{\Sigma}(t_1) = \alpha_{\Sigma}(t_2), \text{ if } \alpha_{\Sigma}(t_1), \alpha_{\Sigma}(t_2) \text{ are of sort } s' \wedge \alpha_{\Sigma}(t_1) \in s, \text{ otherwise} \\ \alpha_{\Sigma}(t_1) = \alpha_{\Sigma}(t_2), \text{ if } \alpha_{\Sigma}(t_1), \alpha_{\Sigma}(t_2) \text{ are of sort } s' \wedge \alpha_{\Sigma}(t_1) \in s, \text{ otherwise} \end{cases} \)

Note that in the last two cases, if the translation of only one term is of a
"question mark" sort, the translation of the other one has to be made to be
of "question mark" sort as well (by inserting an appropriate injection).
The translation can easily be extended from atomic to all formulas.

Apart from mapping definedness and projection to the special symbols
introduced above, the translation mainly checks whether the original version
of a function or predicate symbol can be used, or if we have to use its strict
extension that propagates the undefined element. The strict extension is necessary whenever an argument term involves a partial function symbol. **Satisfaction** The representation condition can be proved in a way similar to the proof of that of (4). □

With the argument of Example 2.30, the representation has the weak $\mathcal{D}$-amalgamation property, where $\mathcal{D}$ is the class of all injective signature morphisms.

**Proposition 4.3** The above constructed institution representation is strongly persistently liberal.

**Proof.** For a SubPFOL~-signature $\Sigma = (S, TF, PF, P, \leq)$, let $\Phi(\Sigma)$ be the SubPFOL~-presentation resulting from removing the axioms of form (1) from $\Phi(\Sigma)$, while adding the partial function symbols from $PF$ and the axioms

$$\forall x_1 : s_1, \ldots, x_n : s_n \bullet f_{s_1, \ldots, s_n, s}(x_1, \ldots, x_n) \in s \iff \text{def } f_{s_1, \ldots, s_n, s}(x_1, \ldots, x_n)$$

(8)

$$\forall x_1 : s_1, \ldots, x_n : s_n \bullet \text{def } f_{s_1, \ldots, s_n, s}(x_1, \ldots, x_n)$$

$$\implies \text{inj}_{(s, s)}(f_{s_1, \ldots, s_n, s}(x_1, \ldots, x_n)) = f_{s_1, \ldots, s_n, s}(x_1, \ldots, x_n)$$

(9)

for each $f : s_1, \ldots, s_n \rightarrow s, s \in PF$. The axioms relate the partial functions with their total encoding, in such a way that a partial function is uniquely determined by its total encoding, and moreover, the way in which it is determined corresponds to the action of $\beta_\Sigma$.

In this way, $\Phi(\Sigma)$ includes both $\Sigma$ and $\Phi(\Sigma)$ (except from the axioms of form (1)). Let $\iota_\Sigma$ and $\kappa_\Sigma$ be the inclusions of $\Sigma$ and $\Phi(\Sigma)$ into $\Phi(\Sigma)$. By Theorem 4.16, the free functor $F_{\iota_\Sigma} : \text{Mod}(\Sigma) \rightarrow \text{Mod}(\Phi(\Sigma))$ exists. Moreover, since $\bot \in s$ does not follow from $\Phi(\Sigma)$, models constructed by this free functor also satisfy the axioms of form (1).

We can now define $\gamma_\Sigma$ to act as follows: Given a $\Sigma$-model $M$,

$$\gamma_\Sigma(M) := F_{\iota_\Sigma}(M)|_{\kappa_\Sigma}.$$  

It is tedious but not difficult to check that $\iota_\Sigma$ is sufficiently complete, i.e.

$$M \cong F_{\iota_\Sigma}(M)|_{\kappa_\Sigma}$$

via the unit of the adjunction. By Lemma 8.14 of [28], we can choose $F_{\iota_\Sigma}$ such that the identity is the unit of the adjunction, i.e. $M = F_{\iota_\Sigma}(M)|_{\kappa_\Sigma}$. But then, since the unique determination of a partial function through (8) and (9) is the

\[\text{Theorem 4.16 depends on several subinstitution representations presented in Section 4.2, but not on the institution representation currently being considered.}\]
same as the construction of the partial function through $\beta_\Sigma$, we get

$$\beta_\Sigma(\gamma_\Sigma(M)) = M.$$ 

To show the universal property of $M \xrightarrow{id} \beta_\Sigma(\gamma_\Sigma(M))$, let $h: M \rightarrow \beta_\Sigma(N)$ be a $\Sigma$-homomorphism. Let $N' \in \text{Mod}(\Phi(\Sigma))$ be the expansion of $N$ with new partial functions (uniquely determined by (8) and (9)). We thus have $N'|_{\kappa_\Sigma} = N$ and, since the expansion acts like $\beta$, $\beta_\Sigma(N) = N'|_{\kappa_\Sigma}$. By the universal property of $F_{\kappa_\Sigma}$, we get $h^\#: M \rightarrow N'|_{\kappa_\Sigma}$ with $h^\#|_{\kappa_\Sigma} = h$. Hence, $h^\#|_{\kappa_\Sigma}: \gamma_\Sigma(M) \rightarrow N$ is the required extension of $h$ with $\beta_\Sigma(h^\#|_{\kappa_\Sigma}) = h^\#|_{\kappa_\Sigma} = h$. □

Also, by modifying (5) (similar to the passage from (4) to (4a)), we get a representation (5a) that is model-bijective. The exact axioms are described in (5a') below.

Further, we can extend the representation (4b) to get a representation (5b) of $\text{SubPFOL}^=$ in $\text{SubFOL}^=$ that is even a subinstitution representation. For signatures and models, the extension of (4b) is quite straightforward: the sub-sorting structure is just kept. Sentence translation is extended as follows:

- $\alpha_\Sigma(\text{pr}_{\langle \xi, \delta \rangle}(t)) \overset{\circ}{=} t_0 = \exists x: s \cdot \alpha_\Sigma(\text{inj}_{\langle \xi, \delta \rangle}(x) \overset{\circ}{=} t) \land \alpha_\Sigma(x \overset{\circ}{=} t_0)$, where $x$ is a new variable,
- $\alpha_\Sigma(t \in s) = \exists x: s \cdot \alpha_\Sigma(t \overset{\circ}{=} x) \land x \in s$, where $x$ is a new variable.

Due to the presence of sub-sorting, we also can replace the representation of partial functions as graphs by another one: we can, for each partial function, introduce a new (sub)sort holding its domain, and thus making the function total. However, note that sentence translation will become equally clumsy as for (5b) above, since it will involve existential quantifiers over the domains. Moreover, since the domain generally is a subsort of a product and $\text{SubFOL}^=$ lacks products, we have to introduce products explicitly.

4.1.8 (5'), (5a') and (5b'): Mapping $\text{SubPCFOL}^=$ to $\text{SubCFOL}^=$

Again, the translation (5a') is very similar to that of the previous section. As in (4a'), due to the need to translate sort generation constraints, we must be able to generate also the “undefined” elements in the $\text{SubCFOL}^=$-models. One could think of using the sorts without a question mark for the translation of sort generation constraints. However, this does not work for sort generation constraints involving partial function symbols: partial function symbols are always translated to a function symbol with result sort of form $s^?$. Also, the projection symbols do not help to get back to $s$: the also have result sort $s^?$. This forces us to consider generation of the $s^?$ sorts.
In order to be able to do this, we again restrict the models to those with exactly one “undefined” element. This leads to a loss of persistent liberality of the representation.

**Signatures** A signature is translated as in (5), with the addition that the sentences \( x \in s \Rightarrow \neg x = \bot \) (for each \( s \in S \)) are added to the presentation.

Thus, \( \bot_s \) becomes the unique “undefined” element.

**Models** Models are translated exactly as in (5).

**Sentences** First-order sentences are translated as in (5). A \( \Sigma \)-sort generation constraint \( \langle \bar{S}, \bar{F}, \bar{\theta}; \bar{\Sigma} \rightarrow \bar{\Sigma} \rangle \) is translated to \( \langle \bar{S}, \bar{F}', \bar{\theta}' \cup \{ \bot_s : s \in S \}, \bar{\theta}' \rangle \), where \( \bar{F}' \) is \( \bar{F} \) with all function symbols of form \( \text{pr}_{(s,s')} \) replaced by \( \text{proj}_{(s,s')} \)

and \( \bar{\theta}' : \Phi(\bar{\Sigma}) \rightarrow \Phi(\bar{\Sigma}) \) is the extension of \( \bar{\theta} \) to \( \Phi(\bar{\Sigma}) \) that is defined by mapping the added symbols in \( \Phi(\bar{\Sigma}) \) to the corresponding added symbols in \( \Phi(\bar{\Sigma}) \).

**Satisfaction** The representation condition for formulas is proved as in (4a').

As in (4'), we can restore strongly persistent liberality by a restriction of sort generation constraints in \( \text{SubPCFOL}^= \) to those not involving partial function symbols: in this case, the extra axioms for the \( \bot \) functions are not needed. We thus get a representation \( (5') \).

Moreover, with the same restriction of \( \text{SubPCFOL}^= \), we also can extend \( (5b) \) to \( (5b') \) by just leaving sort generation constraints unchanged.

4.1.9 (6) and (6') Mapping \( \text{Sub}(C)\text{FOL}^= \) to \( (C)\text{FOL}^= \)

Here we can use a restriction of the translation of \( \text{Sub}(C)\text{FOL}^= \) to \( (C)\text{FOL}^= \) described in Section 4.1.3 above, which leaves out the partial projection symbols and those axioms from \( \bar{J}(\bar{\Sigma}) \) involving these. There is one problem connected with this: the membership predicate was originally axiomatized using partial projections and definedness:

\[
\forall x : s' \bullet \in_{s'}^{\bar{\Sigma}}(x) \iff \text{pr}_{(s',s)}(x)
\]

for \( s \leq s' \). Now we cannot just leave out these axioms, since then the membership predicates could be interpreted in an unintended way. Therefore, these axioms have to be replaced by

\[
\forall x : s' \bullet \in_{s'}^{\bar{\Sigma}}(x) \iff \exists y \bullet \text{inj}_{(s,s')}(y) = x
\]

4.1.10 Comparison of (4)\( \circ \) (3) with (6)\( \circ \) (5)

Comparing both ways of encoding \( \text{SubPFOL}^= \) into \( \text{FOL}^= \), we can see that the way via \( \text{SubFOL}^= \) has the disadvantage that many new sorts are introduced,
but the advantage than the original sort system and its interpretation are kept. Moreover, many-sorted total specifications are left unchanged (but the encoding via $PFOL^=\ $may also be adapted to have this property).

\subsection{4.1.11 Mapping $CFOL^=\ $to $SOL^=\ $}

$CFOL^=\ $adds sort generation constraints to $FOL^=\ $. These cannot be expressed within $FOL^=\ $, but can be translated to induction schemes, which can be expressed in second-order logic with equality ($SOL^=\ $) by second-order quantification over predicates.

**Signatures** Signatures and signature morphisms are translated identically.

**Models** Models, model homomorphisms and reducts are translated identically.

**Sentences** First-order sentences are translated identically. For a sort generation constraint
\begin{equation*}
(S, F; \theta; \Sigma \to \Sigma)
\end{equation*}
we assume without loss of generality that all the result sorts of function symbols in $F$ occur in $S$ (see the corresponding remark in Section 3.1 where sort generation constraints have been introduced). Let
\begin{align*}
\mathcal{S} &= \{s_1; \ldots; s_n\}, \\
\overline{F} &= \{f_1 : s_1^1 \ldots s_{m_1}^1 \to s_1; \ldots; f_k : s_1^k \ldots s_{m_k}^k \to s_k\}
\end{align*}
The sort generation constraint is now translated to the $SOL^=\ $-sentence
\begin{equation*}
\forall P_{s_1} : \text{pred}(\theta(s_1)) \ldots \forall P_{s_n} : \text{pred}(\theta(s_n)) \\
\bullet (\varphi_1 \land \cdots \land \varphi_k) \Rightarrow \bigwedge_{j=1,\ldots,n} \forall x : \theta(s_j) \bullet P_{s_j}(x)
\end{equation*}
where
\begin{equation*}
\varphi_j = \forall x_1 : \theta(s_1^j), \ldots, x_{m_j} : \theta(s_{m_j}^j) \\
\bullet \left( \bigwedge_{i=1,\ldots,m_j} P_{s_i}(x_i) \right) \Rightarrow P_{s_j}(\theta(f_j)(x_1,\ldots,x_{m_j}))
\end{equation*}

**Satisfaction** To prove the representation condition, let a $\Sigma$-model $M$ and a $\Sigma$-sort generation constraint $(S, F; \theta; \Sigma \to \Sigma)$ be given. Call an $\overline{S}$-sorted set $\overline{\mathcal{P}} \subseteq |M|_{\theta(\cdot)}$ $(\overline{S}, F; \theta; \Sigma \to \Sigma)$-closed iff it is closed under the application of functions $\theta(f)$ with $f \in \overline{F}$ and the filling in of arbitrary values of $M_{\theta(s)}$ for $s \not\in \overline{S}$.

**Lemma 4.4** Let $X$ be the variable system $\{P_{s_1} : \text{pred}(\theta(s_1))], \ldots, P_{s_n} : \text{pred}(\theta(s_n))\}$. Given an assignment $\xi : X \to M$, consider the $\overline{S}$-sorted set $\overline{\mathcal{P}}(\xi)$ consisting of $\xi(P_s)$ for $s \in \overline{S}$ and of $M_{\theta(s)}$ for $s \not\in \overline{S}$.

Then we have
\begin{equation*}
\xi \models \varphi_1 \land \cdots \land \varphi_k \text{ iff } \overline{\mathcal{P}}(\xi) \text{ is } (\overline{S}, F; \theta; \Sigma \to \Sigma)-closed.
\end{equation*}

**Proof** This directly follows from the form of the $\varphi_j$. Note that the filling in of arbitrary values of $M_{\theta(s)}$ for $s \not\in \overline{S}$ is captured by the condition $s_1^j \in \overline{S}$ in the conjunction in the premise of $\varphi_j$ for $s_1^j \not\in \overline{S}$, nothing is required. \qed

71
Now $M$ satisfies the sort generation constraint $(\mathcal{S}, \mathcal{F}, \theta; \Sigma \rightarrow \Sigma)$
 iff the smallest $(\mathcal{S}, \mathcal{F}, \theta)$-closed set is $|M|_{\theta}$
 iff for all $\xi : X \rightarrow M$, $\tilde{\mathcal{P}}(\xi)$ is $(\mathcal{S}, \mathcal{F}, \theta)$-closed implies $\tilde{\mathcal{P}}(\xi) = |M|_{\theta}$
 iff for all $\xi : X \rightarrow M$,
    $\xi \vdash \forall \varphi \land \cdots \land \varphi_k$ implies $\xi \vdash \bigwedge_{j=1, \ldots, n} \forall x : \theta(s_j) \cdot P_{s_j}(x)$
 iff $M$ satisfies the translation of the sort generation constraint. $\square$

4.2 The Positive Conditional Level

At the positive conditional level (see Fig. 5), we have used generalized conditional logic ($GCond^=, GHorn^= \text{ etc.}$). The reason for this is that we want to be able to use equivalences (and not just implications) in the translations of sentences, which would not be possible within ordinary conditional logic ($Cond^=, Horn^= \text{ etc.}$). However, note that there is a conjunctive substitution representation from any of the generalized conditional logics to the corresponding ordinary conditional logic. Note also that in most cases, the diagram remains the same if the $G$ is deleted everywhere (only a few arrows become conjunctive representations).

$MEqtl$ denotes Meseguer’s Membership Equational Logic, see [49], where also the arrows to and from $MEqtl$ are described. Also, Equational Type Logic [46] and Unified Algebras [63] and similar frameworks could be added at the same corner of the diagram (however, the arrow from $GHorn^= \text{ will not be persistently liberal in all these cases}$).

At the positive conditional level, we have the following levels of expressiveness:

(1) The institutions $SubPGHorn^=, PGHorn^=, SubPGCond^= \text{ and } PGC\text{ond}^= \text{ (all involving partiality) are all substitutions of each other. They form the strongest level of expressiveness. By using more complicated representations, one also can count } SubGHorn^= \text{ and } SubGCond^= \text{ to belong to this level, see [53]. We have not included these representations here, since they are based on a weaker notion of embedding, which is too weak to be practically useful for borrowing.}$

(2) The next level of expressiveness is $GHorn^=$. Here, we lose the ability of specifying conditional generation of data using partial functions within free specifications, as shown in [52].

(3) Then comes $GCond^=$. As shown in [52], due to lack of predicates, we cannot specify conditional generation of relations using free, as in the transitive closure example (see Example 2.11).

(4) The lowest level is $Eq^=, \text{ where we have no conditional axioms.}$

All these levels are separated from each other in Theorem 5.1.
Index for arrow types:

- --- 1. Admits model expansion: Borrowing of entailment and refinement for flat specifications

- - - - 2. Strongly persistently liberal: Lifting of free constructions, implies 1.

--- 3. Embedding: Borrowing of entailment for certain structured specifications including \textit{free}


\rightarrow 5. Substitution: Borrowing of entailment and refinement for all structured specifications

Fig. 5. The positive conditional level

We now come the the description of the representations.

4.2.1 (1): Mapping $SubGHorn^=$ to $SubPCond^=$

Here, we cannot directly use the idea of representation (2) from the first-order level of replacing predicates by Boolean functions, since the set of Boolean
cannot be axiomatized monomorphically in positive conditional logic. However, what we can do is to axiomatize a single-valued set \( \text{Bool} \) of truth-values and map predicates to partial functions into \( \text{Bool} \).

**Signatures** A signature is translated by adding disjointly to it the presentation

\[
\begin{align*}
\text{spec} \ & \text{Bool} = \\
\text{sort} & \text{Bool} \\
\text{ops} & \text{True}_1 : \text{Bool} \\
\text{axiom} & \forall x, y : \text{Bool} \bullet x = y
\end{align*}
\]

and replacing each predicate symbol \( p : w \) by a partial function symbol \( p : w \rightarrow \text{Bool} \).

**Models** A model \( M \) is translated by replacing each partial function \( (p_{w, \text{Bool}})_M \) by its domain, considered as a predicate. Note that model homomorphisms can be translated identically, since they behave in the same way for domains of partial functions and for predicates.

**Sentences** A sentence is translated by replacing each occurrence of a predicate symbol application \( p_w(t_1, \ldots, t_n) \) by the equation \( p_{w, \text{Bool}}(t_1, \ldots, t_n) = \text{True}_1 \).

**Satisfaction** The representation condition is straightforward to show. 

It is easy to show this representation is a subinstitution representation.

### 4.2.2 (2): Mapping \( P\text{GHorn}^= \) to \( P\text{GCond}^= \)

Here, we can use the same representation as in (1), just restricted to signatures with trivial subsort relation.

### 4.2.3 (3): Mapping \( \text{SubGHorn}^= \) to \( \text{SubGCond}^= \)

Here, we can use a representation similar to that in (1). The only difference is that predicate symbols \( p : w \) are translated to total function symbols \( p : w \rightarrow \text{Bool} \). (Note that we cannot represent predicates by subsorts, since predicates may be empty, while subsorts may not.) This leads to a loss of the subinstitution property, but we still have a strongly persistently liberal representation, which can be seen as follows: Let \( \Sigma = (S, TF, P, \preceq) \) be a signature in \( \text{SubGHorn}^= \). Given a \( \Sigma \)-model \( M \), \( \gamma_{\Sigma}(M) \) extends \( M \) by interpreting \( \text{Bool} \)

\[13\] Actually, since positive conditional logic is closed under products by the well-known Mal’cev theorem, any theory having a model with a two-valued Boolean carrier set has also models where the Boolean carrier set has any given cardinality.
as
\[
\{ \text{true} \} \uplus \{ p_w(a_1, \ldots, a_n) \mid p : w \in P, a_1, \ldots, a_n \in M_w \setminus p_M \}
\]

True1 is interpreted with \text{true}, and
\[
(p_w)_{\gamma_{\Sigma}(M)}(a_1, \ldots, a_n) = \begin{cases} 
\text{true}, & \text{if } (a_1, \ldots, a_n) \in p_M \\
 p_w(a_1, \ldots, a_n), & \text{otherwise}
\end{cases}
\]

Clearly, \( \beta_{\Sigma}(\gamma_{\Sigma}(M)) = M \). Now let \( h : M \rightarrow \beta_{\Sigma}(N) \) be a \( \Sigma \)-homomorphism. Then \( h \) can be uniquely extended to a \( \Phi(\Sigma) \)-homomorphism \( h^\# : \gamma(M) \rightarrow N \) with \( \beta_{\Sigma}(h^\#) = h \) by putting
\[
(h^\#)_{\text{Bool1}}(\text{true}) = \text{True1}_N \\
(h^\#)_{\text{Bool1}}(p_w(a_1, \ldots, a_n)) = p_N(h(a_1), \ldots, h(a_n))
\]

4.2.4 \( (4) \): Mapping \( \text{GHorn}^= \) to \( \text{GCond}^= \)

Here, we can use the same representation as in (3), just restricted to signatures with trivial subsort relation. Again, the representation is strongly persistently liberal.

4.2.5 \( (5) \): Mapping \( \text{SubPGHorn}^= \) to \( \text{SubGHorn}^= \)

This strongly persistently liberal representation works like the representation (5) from the first-order level, except that the constants \( \perp \) and their axiomatizations are omitted; then, all axioms in \( \Phi(\Sigma) \) are in Horn form. The omission of \( \perp \) leads to a loss of the weak (injective)-amalgamation property.

4.2.6 \( (6) \): Mapping \( \text{PGHorn}^= \) to \( \text{GHorn}^= \)

This representation works similar to the representation (4) from the first-order level. As in (5), we have to remove the constants \( \perp \) and their axiomatizations which are not in Horn form (thereby losing the weak (injective)-amalgamation property). Moreover, the axioms \( \exists x : s \cdot D_s(x) \) are not in Horn form either. These axioms have to be replaced by the introduction of new constants \( c_s : s \) (for each \( s \in S \)). The new constant has essentially the same effect as the existential axiom, except that homomorphisms have to preserve it, which leads to a loss of persistent liberality (but the representation still admits model expansion).

4.2.7 \( (7) \): Mapping \( \text{SubPGCond}^= \) to \( \text{SubGCond}^= \)

This works like the (strongly persistently liberal) representation (5) above.
4.2.8 (8): Mapping $PGCon d^= \rightarrow GC on d^=$

Here, we cannot restrict (4) from the first-order level, since we need to eliminate the definedness predicates. Therefore, we use the composition

$$PGCon d^= \rightarrow P GHorn^= \overset{(6)}{\rightarrow} G Horn^= \overset{(4)}{\rightarrow} GC on d^= $$

which is a representation admitting model expansion.

4.2.9 (9): Mapping $SubPGHorn^= \rightarrow PGHorn^=$

This substitution representation works exactly as the representation (3) from the first-order level: all axioms in $\Phi(\Sigma)$ are already in Horn form.

4.2.10 (10): Mapping $SubPGCon d^= \rightarrow PGCon d^=$

This representation works similar to (9). The difference it that the membership predicates have to be deleted. Now when looking at the axioms in $\bar{J}$, one can see that the membership predicates are just the domains of the partial projections. Thus, all occurrences of

$$t \in s$$

in sentences have to be translated to

$$\text{def } pr_{\{s', s\}}(t).$$

4.2.11 (11): Mapping $SubGHorn^= \rightarrow G Horn^=$

When setting up this representation, we encounter the difficulty to axiomatize the membership predicates. In $\bar{J}$, they are axiomatized as the domains of the partial projections. Now we do not have partial functions at hand. In the representation (6) at the first-order level this problem is solved by the axiomatization

$$\forall x : s' \in s' (x) \Leftrightarrow \exists y \bullet \text{inj}_{\{s, s'\}}(y) = x$$

But this axiom is not in Horn form. The best that we can do at the moment instead of this is just to use the composition

$$SubGHorn^= \overset{(9)}{\rightarrow} SubPGHorn^= \overset{(6)}{\rightarrow} PGHorn^= \overset{(4)}{\rightarrow} G Horn^= $$

which is a representation admitting model expansion.
4.2.12 \((12)\): Mapping \(\mathit{SubGC}ond^=\) to \(\mathit{GC}ond^=\)

For the same reasons as for \((11)\) above, we use the composition

\[
\begin{align*}
\mathit{SubGC}ond^= & \xrightarrow{\text{\((10)\)}} \mathit{SubPGC}ond^= \\
& \xrightarrow{\text{\((8)\)}} \mathit{PGC}ond^= \xrightarrow{\text{\((8)\)}} \mathit{GC}ond^= 
\end{align*}
\]

which again is a representation admitting model expansion.

4.3 Theorem proving and liberality

We can now combine the results of the previous subsections with the general results about institution representations in Section 2 to obtain specific results about theorem provers and liberality.

**Fact 4.5** \(\mathit{PC}ond^=\) allows conditional term rewriting and paramodulation as sound and complete proof systems.

**Proof.** See e.g. [67]. \(\Box\)

**Theorem 4.6** Each of the institutions shown in Fig. 5 allows the re-use of conditional term rewriting and paramodulation as sound and complete proof systems for flat specifications.

**Proof.** By chasing the diagram in Fig. 5 and using the remark at the beginning of Section 4.2 about the representation of \(\mathit{PGC}ond^=\) in \(\mathit{PC}ond^=\), one can see that each of the institutions in Fig. 5 can be represented in \(\mathit{PC}ond^=\) with a representation admitting model expansion. The result now follows from Fact 4.5 and Theorem 2.26. \(\Box\)

**Theorem 4.7** Each of the institutions shown in Fig. 3 (except \(\mathit{SOL}^=\)) allows the re-use of a first-order theorem prover for flat specifications.

**Proof.** By chasing the diagram in Fig. 3, one can see that each of the institutions in Fig. 3 except \(\mathit{SOL}^=\) can be represented in \(\mathit{FOL}^=\) with a representation admitting model expansion. The result now follows from Theorem 2.26. \(\Box\)

**Theorem 4.8** Each of the institutions shown in Fig. 4 (except \(\mathit{SOL}^=\)) allows the re-use of a first-order theorem prover with induction (or second-order, or higher-order prover) for flat specifications.

**Proof.** By chasing the diagram in Fig. 4, one can see that each of the institutions in Fig. 4 can be represented in \(\mathit{CFOL}^=\) with a representation admitting model expansion. The result now follows from Theorem 2.26. \(\Box\)

**Theorem 4.9** Each of the institutions shown in Fig. 3, except from \(\mathit{SOL}^=\),
**COSASC\textsuperscript{\textregistered}-Bool**, and **COSASC\textsubscript{\textregistered}-Bool** allows the re-use of first-order logic theorem provers for both entailments and refinements concerning structured specifications without **free**.

**Proof.** By chasing the diagram in Fig. 3, one can see that the above-mentioned institutions have a model-bijective representation in \(FOL^=\). The result now follows from Proposition 2.52 and Theorem 2.31. \(\Box\)

**Theorem 4.10** Each of the institutions shown in Fig. 4 except from \(SOL^=\) and \(LSL\text{-}Bool\) allows the re-use of first-order theorem provers with induction (or second-order, or higher-order provers) for both entailments and refinements concerning structured specifications without **free**.

**Proof.** By chasing the diagram in Fig. 4, one can see that each of the above mentioned institutions has a model-bijective representation in \(CFOL^=\). The result now follows from Propositions 2.52 and Theorem 2.31. \(\Box\)

**Theorem 4.11** For each of the institutions shown in Fig. 4 except \(LSL\text{-}Bool\) and \(SOL^=\), take the subinstitution obtained by forbidding sort generation constraints with partial function symbols. Each of the thus obtained institutions allows

- the re-use of a checker for \(CFOL^=\) that determines whether presentation morphisms are liberal, and
- the re-use of first-order theorem provers with induction (or second-order, or higher-order provers) for entailments concerning structured specifications \(SP\) (including **free**), such that for the strongly persistently liberal representation \((\Phi, \alpha, \beta, \gamma)\) into \(CFOL^=\),
  - for each **free** \(SP'\) along \(\sigma; \Sigma \rightarrow \Sigma_1\) occurring in \(SP\), either \(\Phi(\sigma); \Sigma \rightarrow SP'\) is strongly persistently liberal or \(\gamma\) is \(\sigma\)-natural, and moreover, \(\gamma_{\Sigma}\) is surjective on objects,
  - for each **derive** from \(SP_1\) by \(\sigma\) occurring in \(SP\), \(\gamma\) is \(\sigma\)-natural,
  - for each **translate** \(SP_1\) by \(\sigma\) occurring in \(SP\), either \(\gamma\) is \(\sigma\)-natural, or \(SP_1\) is flattenable, i.e. contains neither **free** nor **derive**.

**Proof.** By chasing the diagram in Fig. 4, one can see that each of the above mentioned institutions is a subinstitution of \(SubPCFOL^=\) with sort generation constraints restricted to total functions. The representation \((4')\) of the first-order level describes a strongly persistently liberal institution representation from the thus restricted \(SubPCFOL^=\) to \(CFOL^=\). The result now follows from Theorems 2.43 and 2.45. \(\Box\)

**Theorem 4.12** Consider the restriction of the CASL institution \(SubPCFOL^=\) to sort generation constraints only with total function symbols. This institution can be represented in \(CFOL^=\) in such a way that it is possible to
- re-use of first-order theorem provers with induction (or second-order, or higher-order provers) for both entailments and refinements concerning structured specifications without **free**, and with **derive** only along injective signature morphisms,
- re-use of a checker for $CFOL^=\text{ }$ that determines whether presentation morphisms are liberal, and
- re-use first-order theorem provers with induction (or second-order, or higher-order provers) for entailments concerning structured specifications $SP$ (including **free**), where $SP$ obeys the restrictions listed in Theorem 4.11 above.

**Proof.** Compose representations $(6')$ and $(5')$ in Fig. 4, thus obtaining a strongly persistently liberal institution representation admitting weak (injective)-amalgamation going from (the restricted) $SubPCFOL^=\text{ }$ to $CFOL^=\text{ }$. The result now follows from Proposition 2.52 and Theorems 2.31, 4.10 and and 4.11. □

Of course, the question arises whether the above semantic restrictions on the specifications that include **free** can be reformulated in more syntactic terms.

Concerning surjectivity of $\gamma_\Sigma$ on objects, this can be ensured by simply letting the parameter signature $\Sigma$ have no partial operations and no subsorts\textsuperscript{14}, since then both $\beta_\Sigma$ and $\gamma_\Sigma$ are isomorphisms.

Concerning strongly persistent liberality of $\Phi(\sigma)\colon \Sigma \longrightarrow SP'$, this can be ensured if $SP'$ is a presentation and only introduces new operation symbols that are specified in a sufficiently complete way (i.e. such that each new term can be reduced to an old term). The crucial point is here that this must also hold (in the encoding) for terms containing partial operation symbols. That is, we have to specify their behaviour also in the undefined case (with $\neg \text{def } f(t_1, \ldots, t_n)$). However, in general this would destroy the existence of free models. But we can use a trick: namely, we can make sentences of form $\neg \text{def } t$ parts of signatures, and as parts of signatures, we can translate them to $t = \bot$ (while doing so at the sentence level would destroy the representation condition). $\beta$ and $\gamma$ can be easily adapted to work also with these new signatures.

Concerning $\sigma$-naturality of $\gamma$: this always holds if $\sigma$ is an isomorphism. Hence, bijective renamings are covered. If we want to hide something, with each operation symbol, we also have to hide its result sort: this also implies $\sigma$-naturality of $\gamma$, but it is of course a rather strong restriction.

If we want to overcome these restrictions, we can use the following:

**Theorem 4.13** Each of the institutions shown in Fig. 4 except from $SOL^=\text{ }$

\textsuperscript{14} Since subsorts lead to partial projection functions, we have to exclude them as well. An alternative is to allow subsorts without the possibility to use partial projections.
and LSL-Bool allows the re-use of first-order theorem provers with induction (or second-order, or higher-order provers) for both entailments and refinements concerning structured specifications including free.

Proof. By chasing the diagram in Fig. 4, one can see that each of the above mentioned institutions has a subinstitution representation in CFOLE. The result now follows from Theorem 2.39. □

Note, however, that the sentence translation involved in this result is rather clumsy. Therefore, in practice, one will use the weaker results (Theorems 4.10, 4.11) above in order to get a better sentence translation, and fall back on the clumsier sentence translation only if dealing with specifications involving uses of free that are not covered by Theorem 4.11.

Definition 4.14 In any of the institutions introduced so far, a signature Σ is called strict, if for each sort, there is a total ground (i.e. variable-free) Σ-term of that sort. A theory is called strict if its signature is. A theory morphism is called strict if its target theory is. □

Fact 4.15 In PCond, every strict theory morphism is liberal, i.e. PCond is (strict)-liberal.

Proof. For a modification of PCond with empty carriers in the models allowed, this is quite standard and follows e.g. from the very general results in [73]. The restriction to strict theory morphisms ensures that the free model has non-empty carriers, thus it is always contained in the full subcategory of models without empty carriers. Since free objects in a full subcategory are again free, the result follows. □

Theorem 4.16 Each of the institutions shown in Fig. 5 is (strict)-liberal.

Proof. By chasing the diagram in Fig. 5 and using the remark at the beginning of Section 4.2 about the representation of PGCond in PCond, one can see that each of the institutions in Fig. 5 can be represented strongly persistently liberally (even as a subinstitution) in PCond. By Fact 4.15, PCond is (strict)-liberal, and by Theorem 2.43, this is lifted against strongly persistently liberal institution representations (noting that all representations in Fig. 5 map strict theory morphisms to strict theory morphisms). □

5 Hierarchy Theorems

In this section, at the positive level, some negative results will be proved. This yields a strict hierarchy of institutions and shows that the different levels introduced in the previous section really are different levels of expressiveness.

80
Hierarchy theorems are usually proved by finding some property which holds for all objects in some class, but which does not hold for some objects in a more expressive class, which then turns out to be strictly more expressive. Here, we use properties of the model categories of institutions, to show that some model category which is definable in a more expressive institution is not definable in a less expressive institution, showing that the more expressive institution is not a subinstitution of (and even cannot be embedded into) the less expressive institution. In [52], we also use properties of parameterized abstract data types specifiable with free and derive to separate the same levels of expressiveness.

Both ways lead to a hierarchy of five levels.

**Theorem 5.1** The five institutions we consider form a proper hierarchy:

<table>
<thead>
<tr>
<th>Institution</th>
<th>Properties of model categories</th>
<th>Separating example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$E_{eq}^=$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$At_{orn}^=$ subobjects commute with coequalizers</td>
<td>graphs = binary relations</td>
</tr>
<tr>
<td>3</td>
<td>$GC_{ond}^=$ forgetful functors preserve regular epis</td>
<td>some special theory</td>
</tr>
<tr>
<td>4</td>
<td>$GH_{orn}^=$ (regular epi, mono)-factorizable</td>
<td>transitive binary relations</td>
</tr>
<tr>
<td>5</td>
<td>$SubPG_{Horn}^=$ locally finitely presentable</td>
<td>transitive multi-graphs or small categories</td>
</tr>
</tbody>
</table>

The proof is split into four parts, each of which is presented in one of the subsequent subsections. Before coming to that, we quote the following characterization of model categories at level 5:

**Theorem 5.2** [53]

1. All model categories specifiable in $SubPG_{Horn}^=$ (or any of its subinstitutions) are locally finitely presentable.
2. If we extend $SubPG_{Horn}^=$ to infinite signatures (consisting of infinite sets of sorts, infinite sets of (finitary) operation and predicate symbols), then exactly the locally finitely presentable categories can be specified as model categories of theories. □
5.1 Level 5 versus Level 4: partial conditional logic and horn clause logic

The essential difference between level 4 and level 5 is that in the former, the theorem of homomorphisms holds, while in the latter, it fails. Categorically, the theorem of homomorphisms means that there exist (regular epi, mono)-factorizations.

**Proposition 5.3** There is no institution embedding from \( PGHorn^e \) to \( GHorn^e \). Thus, \( PGHorn^e \) (and therefore also \( SubPGHorn^e \)) is strictly more expressive than \( GHorn^e \).

**Proof.** For a theory \( T = \langle \Sigma, \Gamma \rangle \) in \( PGHorn^e \) and a model morphism \( f: A \rightarrow B \) in \( \text{Mod}(T) \), we can try to get a factorization of \( f \) through its image by the following procedure: first, take the kernel of \( f \), that is, the pullback

\[
\begin{array}{c}
\ker f \\
\downarrow e_1 \\
A \\
\downarrow f \\
B
\end{array} \quad \begin{array}{c}
\downarrow e_2 \\
\downarrow f \\
B
\end{array}
\]

(By Theorem 5.2 and Proposition B.3, \( \text{Mod}(T) \) is complete and cocomplete.) Then, take the coequalizer of \( e_1 \) and \( e_2 \) to get the image \( f[A] \):

\[
\begin{array}{c}
\ker f \\
\downarrow e_1 \\
A \\
\downarrow f \\
f[A]
\end{array} \quad \begin{array}{c}
\downarrow e_2 \\
\downarrow f \\
B
\end{array}
\]

By the universal property of the coequalizer, there exists a unique \( m: f[A] \rightarrow B \) with \( f = m \circ e \).

By remarks 3.4(2) and 5.13(2) of [2], if \( T \) belongs to \( GHorn^e \), \( e \) is surjective, which implies that \( m \) is injective, hence a monomorphism, and \( (e, m) \) is a (regular epi,mono)-factorization of \( f \). This is the well-known theorem of homomorphisms, generalized to \( GHorn^e \).

In contrast to this, in \( SubPGHorn^e \) and even in \( PGHorn^e \), \( e \) may be not surjective and \( m \) no monomorphism, that is, there is no (regular epi,mono)-factorization of \( f \). In [70, 3.4], a counterexample very similar to Example 5.11 is discussed in detail. For factorizations in \( PGCond^e \) and \( PGHorn^e \), we have to iterate the construction of kernels and coequalizers (taking \( e \) instead of \( f \) and so on) possibly infinitely often.

82
Now (regular epi,mono)-factorizability clearly is preserved by equivalences of categories. So there cannot be an institution representation from $PGHorn^=$ to $GHorn^=$ with model translation being a pointwise equivalence of categories. □

**Corollary 5.4** $SubPGHorn^=$ and $PGHorn^=$ are strictly more expressive than $GHorn^=.$

### 5.2 Level 4 versus Level 3: horn clause logic and conditional equational logic

The difference between level 3 and level 4 is that in the latter, there are quotient homomorphisms which are not closed, while in the former all homomorphisms are closed. Categorically this means that forgetful functors preserve regular epis.

**Proposition 5.5** $GHorn^=$ cannot be embedded into $GCond^=.$ Therefore, $GHorn^=$ is strictly more expressive than $GCond^=.$

**Proof.** A difference between $GHorn^=$ and $GCond^=$ is that in $GHorn^=,$ there are quotients which are not closed. This cannot happen in $GCond^=,$ where all homomorphisms are closed. Consider the theories $BinaryRelation$ and $TransitiveClosure$ from Example 2.11.

Now in $TransitiveClosure,$ if we identify the elements $b$ and $c$ in

$$A = \{a \leq b; \; c \leq d\}$$

we get the quotient

$$Q = \{[a] \leq [b, c] \leq [d]\}$$

with $[a] \leq [d].$ The quotient homomorphism $q:A \rightarrow Q$ with $q(x) = [x]$ is not closed: we have $q(a) \leq q(d),$ but not $a \leq d.$ On the other hand, in $BinaryRelation$ the quotient $q':A \rightarrow Q'$ with $Q' = \{[a] \leq [b, c] \leq [d]\}$ is closed, since we do not have $q(a) \leq q(d)$ in $Q'.$

To be able to use this for the theory of institution representations, we have to switch to a categorical formulation. We observe that $q$ is a regular epimorphism in $TransitiveClosure,$ but not in $BinaryRelation$ (since it does not factor through $q'$). So the forgetful functor $\text{Mod} \sigma$ associated to the inclusion $\sigma:BinaryRelation \hookrightarrow TransitiveClosure$ does not preserve regular epis.

On the other hand, in $GCond^=,$ the regular epis are precisely the surjective homomorphisms (this follows from [1], 7.72(1), 23.39 and 24.9). Since all forgetful functors in $GCond^=$ preserve surjectivity, they also preserve regular

83
epis.

Now, preservation of regular epis by reduct functors is a categorical property inherited against embeddings (this follows from the model translations being categorical equivalences and being natural). So $G\text{Horn}^\infty$ cannot be embedded into $GC\text{ond}^\infty$.

Other categorical properties used to separate $GC\text{ond}^\infty$ from $G\text{Horn}^\infty$ use essentially the same intuition. In $GC\text{ond}^\infty$, all model categories have a dense set of regular projectives, whereas $G\text{Horn}^\infty$ lacks this property (see [2, 3.19, 3.21]; see [6] for a similar property).

5.3 Level 3 versus Level 2: conditional equational logic and partial equational logic

The difference of level 3 and level 2 lies in the presence of conditional axioms at level 3. These imply that the construction of quotients (categorically: coequalizers) is more complex: We cannot take just the algebraic quotient, but have to factor it by new equations which may be derived using the conditional axioms from the elements of the congruence relation. In contrast to this, at level 2, quotients are just algebraic quotients.

To be able to formulate this categorically, we need to compare two different quotient constructions, one of which is done on a subalgebra. Now at level 2, quotients commute with subalgebras in the following sense: Given a congruence relation $R$ on an algebra $A$ and a subalgebra $A'$ of $A$, then $R$ induces a congruence relation $R'$ on $A'$. Then $A'/R'$ is a subalgebra of $A/R$ in a natural way, while at level 3, this is not the case.

**Definition 5.6** For a category having coequalizers and pullbacks, we say that *subobjects commute with coequalizers* in this category, if the following holds: Given an equivalence relation $R \xrightarrow{p,q} A$ on an object $A$ and a subobject $m:A' \hookrightarrow A$ of $A$ (i.e. $m$ is a monomorphism), taking the pullback

\[
\begin{array}{ccc}
R & \xrightarrow{(p,q)} & A \times A \\
m_R & & m \times m \\
R' & \xrightarrow{(p',q')} & A' \times A'
\end{array}
\]

and then taking the coequalizers $R \xrightarrow{p,q} A \xrightarrow{e} Q$ and $R' \xrightarrow{p',q'} A' \xrightarrow{e'} Q'$ has the result that the unique factorization $mQ:Q \xrightarrow{e \circ m} Q'$ through $e'$ (which
exists by the universal property of the coequalizer) is a monomorphism.

\[
\begin{array}{ccc}
R & \xrightarrow{p} & A \\
\downarrow m_R & & \downarrow m \\
R' & \xrightarrow{p'} & A'
\end{array}
\quad
\begin{array}{ccc}
A & \xrightarrow{e} & Q \\
\downarrow m_Q & & \downarrow m_Q \\
A' & \xrightarrow{e'} & Q'
\end{array}
\]

Lemma 5.7 For all theories \( T = (\Sigma, \Gamma) \) in \( PEq^\sim \), coequalizers in \( \text{Mod}(T) \) are just quotients.

**Proof.** Surjective homomorphisms preserve existence equations by Table 8.1 of [14]. Therefore, the quotient taken in \( \text{Mod}(\Sigma) \) satisfies \( \Gamma \). The coequalizing property now follows from the diagram completion lemma (Lemma 2.7.1 of [14]). \( \square \)

**Proposition 5.8** For all theories \( T = (\Sigma, \Gamma) \) in \( PEq^\sim \), categorical intersections in \( \text{Mod}(T) \) are constructed by taking the relative subalgebra on the intersection of the carriers.

**Proof.** By 4.2.4 of [14], pullbacks in \( \text{Mod}(\Sigma) \) are constructed upon the pullbacks of the underlying sets. In particular, intersections are constructed by taking the relative subalgebra on the intersection of the carriers. Now if \( t_1 = t_2 \in \Gamma \) and \( \nu: X \rightarrow A_1 \cap A_2 \) is a valuation into the intersection,

\[
\begin{array}{ccc}
X & \xrightarrow{\nu} & A_1 \cap A_2 \\
\downarrow i_1 & & \downarrow m_1 \\
A_1 & & A \\
\downarrow i_2 & & \downarrow m_2 \\
A_2 & \xrightarrow{\nu} & A_2
\end{array}
\]

then we know that \((i_1 \circ \nu)^\#(t_1)\) and \((i_1 \circ \nu)^\#(t_2)\) are defined and equal, and similarly for \( i_2 \). Thus, assuming w. l. o. g. that \( i_1 \) and \( i_2 \) are inclusions, we have that \((i_1 \circ \nu)^\#(t_1) = (i_2 \circ \nu)^\#(t_1) = \nu^\#(t_1)\), and similarly for \( t_2 \). Thus \( \nu^\#(t_1) = \nu^\#(t_2) \). \( \square \)

**Proposition 5.9** For all theories \( T = (\Sigma, \Gamma) \) in \( PEq^\sim \), for \( \text{Mod}(T) \), subobjects commute with coequalizers.

**Proof.** Let \( R \xrightarrow{p} A \) be a relation on \( A \) and \( m: A \rightarrow A' \) be a subobject of \( A \). By Lemma 5.7, coequalizers are just quotients. Thus \( m_Q \) is given by

\[
m_Q([a]_{p',q'}) = [m(a)]_{[p,q]}
\]

85
for \( a \in A' \) (here, \([a]_\equiv\) is the \( \equiv \)-congruence class of \( a \)). By Proposition 5.8, 
\[ R' \xrightarrow{p'} A' \] 
is \( R \xrightarrow{p} A \) restricted to \( A' \). Thus, \( m_Q \) is injective. \( \Box \)

A counterexample showing that in \( GC_{\text{and}}^{\cong} \), not for all model categories subobjects commute with coequalizers, is the following. Let \( T = \langle \Sigma, \Gamma \rangle \) be the presentation

\[ \text{spec } T = \]

\[ \text{sort } s \]

\[ \text{ops } f : s \to s; \]

\[ a, b : s; \]

\[ \text{forall } x : s \]

\[ \cdot \quad f(x) = x \Rightarrow a = b \]

end

Let \( A \) be the \( T \)-algebra with \( A_s = \{ 1, 2, 3, 4 \} \), \( a_A = 1 \), \( b_A = 2 \), \( f_A(1) = 2 \), \( f_A(2) = 1 \), \( f_A(3) = 4 \), and \( f_A(4) = 3 \). Let \( A' \) be the subalgebra on the set \( \{ 1, 2 \} \). Now let \( \equiv \) be the congruence generated by \( 3 \equiv 4 \). Then \( A/\equiv \) does not satisfy the \( T \)-axioms: in order to do enforce this, we have to identify 1 and 2. This then is the corresponding coequalizer, having carrier set \( \{ [1, 2], [3, 4] \} \). \( \equiv \) restricted to \( A' \) is the diagonal relation \( \Delta A' \), and \( A'/\Delta A' \cong A' \) is the corresponding coequalizer. The unique homomorphism \( m_Q : A'/\Delta A' \to F_{\langle \Sigma, \emptyset \rangle \to \langle \Sigma, \Gamma \rangle}(A/\equiv) \) is not injective, since it identifies 1 and 2. \( \Box \)

5.4 Level 2 versus Level 1: partial equational logic and total equational logic

The difference of level 2 and level 1 concerns the role of congruence relations: while at level 1, the algebraic structure of a congruence relation (considered as an algebra) is determined already by the set-theoretical relation, at level 2, there is more flexibility, i.e. there are several algebras with the same congruence relation as carrier, but only one of them is “effectively” generated as a kernel of some homomorphism.

**Proposition 5.10** In \( Eq^\cong \), each model category has effective equivalence relations (for the category-theoretic definition of effective equivalence relation, see Definition C.1).

**Proof.** See Remarks 3.4(8) and 3.6(7) in [2]. \( \Box \)

**Example 5.11** Consider the signature \( \text{Graph} \) consisting of one sort symbol and one binary relation symbol. This is clearly a signature in \( \text{Atom}^\cong \). In the sequel, we identify the models of \( \text{Graph} \) with graphs. Now \( \text{Mod}(\text{Graph}) \), the
category of graphs, does not have effective equivalence relations: Consider the graph $G$

Then $G \times G$ is

and $\Delta G \subseteq G \times G$, the diagonal $(\text{id}_{G}, \text{id}_{G}): G \to G \times G$ is

Now consider the relation $R$ on $G$:

It is given by the inclusion $m: R \to G \times G$, and is represented by the pair $(\pi_1 \circ m, \pi_2 \circ m)$. Now $\Delta G$ is contained in $R$, so $R \xrightarrow{m} G$ is reflexive.

Define $\text{flip}: R \to R$ by just flipping $(1, 0)$ and $(0, 1)$. Then $\text{flip}$ is an isomorphism from the subobject of $G$ represented by $(\pi_1 \circ m, \pi_2 \circ m)$ to the subobject represented by $(\pi_2 \circ m, \pi_1 \circ m)$. Thus $R \xrightarrow{m} G$ is symmetric.

Composing $R \xrightarrow{m} G$ with itself by pulling back yields the diagonal relation (as a subobject of $A \times A \times A$, but this plays no role), which is contained in $R \xrightarrow{m} G$ by reflexivity above. Thus $R \xrightarrow{m} G$ is transitive.

Now suppose that for some graph homomorphism $f: G \to G'$ the inner square in
is a pullback. Since \( m \) is the identity on the underlying sets, the outer diagram commutes as well. By the pullback property, there has to exists a graph homomorphism \( h: G \times G \to R \) with \( \pi_1 \circ m \circ h = \pi_1 \) and \( \pi_2 \circ m \circ h = \pi_2 \). By the uniqueness property of the product, \( m \circ h = \text{id} \), so \( h \) has, like \( m \), to be the identity on the underlying sets. But this violates the graph homomorphism property.

Therefore, \( R \xrightarrow{m} G \) is an equivalence relation which cannot be given by a kernel pair. Thus \( \text{Mod} (\text{graph}) \) does not have effective equivalence relations. \( \square \)

**Corollary 5.12** \( \text{Atom}^= \) cannot be embedded into \( \text{Eq}^= \).

**Proof.** Clearly, effective equivalence relations are preserved by equivalences of categories. \( \square \)

A very similar counterexample can be constructed in the category of unary relations, but we found the above example more instructive. In general, predicates introduce some kind of flexibility: information which is not represented by data elements. This kind of flexibility is present in \( \text{Atom}^= \), but not in \( \text{Eq}^= \).

After these more theoretical studies, we now come to the examination of different specification languages.

### 6 The Larch Shared Language

Larch \([40]\) consists of a family of languages supporting a *two-tiered* approach of specification. A specification consists of programming language-independent parts, formulated in the *Larch Shard Language* (LSL), and of programming language-specific parts, formulated in one of the *Larch interface languages*. Since the relation of CASL to programming languages has not been worked out yet, we here can treat the Larch Shared Language only.

#### 6.1 The LSL Institution and Its Translation to \( \text{CFOAlg}^= \)

There is no explicit logic given in the Larch report \([39]\). Therefore, we here introduce an institution \( \text{LSL} \) that fits to interpret the LSL constructs, mostly following \([8]\). Most of \( \text{LSL} \) will be inherited from \( \text{Eq}^= \).

**Signatures** \( \text{LSL} \)-signatures and signature morphisms are those of \( \text{Eq}^= \).

**Models** \( \text{LSL} \)-models and reducts are those of \( \text{Eq}^= \).

**Sentences** Let \( \Sigma = (S, TF) \) be an \( \text{LSL} \)-signature. A \( \Sigma \)-sentence is one of the following:
(1) a $\Sigma$-sentence in $Eq^-$,
(2) a sentence
\[ s \text{ generated by } TF' \text{ wrt } \theta \]
where $\theta : \Sigma \rightarrow \Sigma$ is a signature morphism, $\Sigma = (S, TF)$, $s \in S$ and $TF' \subseteq TF$
(3) a sentence
\[ s \text{ partitioned by } TF' \]
where $s \in S$ and $TF' \subseteq TF$ such that for that each function symbol in $TF'$, $s$ occurs exactly once as an argument sort. Concerning sentence translation,
\[ s \text{ generated by } TF' \text{ wrt } \theta \]
is translated along a signature morphism $\sigma$ to
\[ s \text{ generated by } TF' \text{ wrt } \sigma \circ \theta, \]
and
\[ s \text{ partitioned by } TF' \]
is translated to
\[ \sigma^S(s) \text{ partitioned by } \sigma(TF'), \]
where
\[ \sigma(TF')_{w', s'} := \bigcup_{w \in S^*, s \in S, \sigma^S(w) = w', \sigma^S(s) = s'} \sigma^F_{w, s}(TF'_{w, s}). \]

**Satisfaction** Satisfaction of an equation from $Eq^-$ is defined as in $Eq^-$. Satisfaction of
\[ s \text{ generated by } TF' \text{ wrt } \theta \]
is the same as satisfaction of $\{s\}, TF', \theta$ in $CEq^-$. Finally, satisfaction of
\[ s \text{ partitioned by } TF' \]
is the same as satisfaction of
\[ \forall x_1, x_2 : s \bullet \]
\[ (\bigwedge_{i=1}^{n} \forall y_i : s_i^1 \bullet y_1 : s_1^1 \rightarrow x_1, \ldots, y_{m_i} : s_{m_i}^i \rightarrow x_1, y_{m_i+1} : s_{m_i+1}^i \rightarrow x_2, \ldots, y_{n_i} : s_{n_i}^i \rightarrow x_2) \]
\[ \Rightarrow x_1 = x_2 \]
where $TF' = \{op_1 : s_1^1 \rightarrow s_1^1, \ldots, s_{m_1-1}^1 \rightarrow s_1^1, \ldots, s_{n_1}^1 \rightarrow s_1^1, \ldots, op_h : s_h^k \rightarrow s_h^k, \ldots, s_{m_h}^k \rightarrow s_h^k, \ldots, s_{n_h}^k \rightarrow s_h^k\} \quad \square$

LSL has no logical connectives in the standard sense, but rather a sort $Boolean$ with the usual connectives is built-in, and one can use equations over $Boolean$ to simulate the logical connectives. We do not introduce this built-in $Boolean$
into the institution, since we think that this is better dealt with at the level of structured specification: One can think of an LSL-specification Boolean that is automatically included into each user-defined specification.

There is an obvious institution representation from LSL to $CFOAlg^-$:

**Signatures** Signatures are translated identically.  
**Models** Models are translated identically as well.  
**Sentences** Equations are translated identically. Sentences of form

\[ s \text{ generated by } TF' \text{wrt } \theta \]

or

\[ s \text{ partitioned by } TF' \]

are translated into the corresponding $CFOAlg^-$-sentences that were used above for the definition of their satisfaction.  
**Satisfaction** The representation condition is obvious.  

\[ \square \]

6.2 Translating LSL Language Constructs Into CASL Constructs

Overloading, disambiguation of terms and operation symbols, and mixfix operation symbols are treated in the same way in both LSL and CASL. Thus, there should be no difficulty with the translation. LSL’s built-in precedences can be specified using CASL’s precedence annotations (note, however, that it is necessary for if\_then\_else\_ to override the standard CASL precedence scheme between infix and prefix operations).

Enumeration types are expanded to LSL-theories in the Larch report. However, when translating LSL to CASL, it would be more advisable to translate them directly for CASL’s free types. That is,

\[ s \text{ enumeration of } c_1 \ldots c_n \]

is translated to

\[
\text{free type}
\]

\[ s ::= c_1 \mid \ldots \mid c_n; \]

\[
\text{op succ : } s \rightarrow s; \]

\[
\text{axioms } \quad succ(c_1) = c_2; \]

\[ \ldots \]

\[ \quad succ(c_{n-1}) = c_n; \]

An LSL union type like

\[ s \text{ union of } f_1 : s_1 \ldots f_n : s_n \]

90
is translated to
\[
\text{generated type} \\
\quad s := f_1(-f_1: s_1) \mid \ldots \mid f_n(-f_n: s_n);
\]

Both of the above translations directly correspond to the LSL expansion scheme. However, note that both the selectors \(-f_i\) as well as the successor operation \(\text{succ}\) are total, which generally leads to unwanted error elements. If one does not want to have this rather strange behaviour of LSL, one also could use partial operations here. This would lead to the translations
\[
\text{free type} \\
\quad s := c_1 \mid \ldots \mid c_n; \\
\quad \text{op succ : } s \rightarrow ? s; \\
\quad \text{axioms } \text{succ}(c_1) = c_2; \\
\quad \ldots \\
\quad \text{succ}(c_{n-1}) = c_n; \\
\quad -\text{def succ}(c_n);
\]

and
\[
\text{generated type} \\
\quad s := f_1(-f_1:?s_1) \mid \ldots \mid f_n(-f_n:?s_n);
\]

This would, however, require to extend the target institution to \(\text{PCFOAlg}^e\).

\section{ACT ONE}

ACT ONE originally has been based on equational logic [28] and later was extended to conditional equational logic [20]. Thus, the underlying institution is \(\text{Cond}^e\), modulo the problem of empty carriers, which is treated in Section 11.

The ACT ONE languages constructs can be translated very easily to \(\text{CASL}\). Overloading in ACT ONE is a special case of overloading in \(\text{CASL}\). ACT ONE’s \texttt{let} definitions can be translated to \(\text{CASL}\)’s operation definitions. Note that strictly speaking, one has to hide the locally defined operation afterwards; this would require an in-the-large construct of \(\text{CASL}\).

ACT ONE distinguishes \textit{constructor} and \textit{function} declarations. Both lead to declarations of operations in \(\text{CASL}\). However, the constructors additionally lead to special axioms concerning the equality predicate \(eq\). These axioms also have to be included in the \(\text{CASL}\) translation, of course. An alternative way of dealing with constructors is described in Section 11.1.
The language ASF [11] is based on conditional equational logic. Thus, again the underlying institution is \textit{Cond}¹, modulo the problem of empty carriers, which is treated in Section 11. ASF has an initial semantics for the modules. Initial semantics can be chosen in \textit{Casl} by writing a \textbf{free \{} ... \}\textbf{ around a specification.}

Concerning the language constructs of ASF, the prefix and infix syntax is a special case of \textit{Casl}'s mixfix syntax. Now there is a syntax definition formalism (called SDF) built on top of ASF, which is much more flexible. SDF definitions of context free syntax can be translated to mixfix syntax declarations in \textit{Casl}. SDF's priorities and associativity attributes are available as annotations in \textit{Casl} as well (in fact, the \textit{Casl} design followed the SDF design at this point). Concerning lexical syntax, \textit{Casl} is not as flexible as SDF: it is not possible to redefine the lexical syntax in \textit{Casl}: one has to use the fixed built-in lexical syntax. However, \textit{Casl} provides a means to define special literal syntaxes for numbers, strings and list-like structures. This should already cover many applications of redefinition of lexical syntax in SDF.

ASF allows product types as result types of functions and provides a tupling notation for terms. In [11], a translation of this into extra product sorts and explicit tupling functions is described; this translation has to be applied when translating ASF to \textit{Casl}.

Overloading in ASF is a special case of overloading in \textit{Casl}. ASF's if function can be translated to \textit{Casl}'s \texttt{when else construct} (indeed, the expansion of these two constructs to formulas of the underlying institution is essentially the same in ASF and in \textit{Casl}).

A further informal comparison of \textit{Casl} and ASF can be found in [64].

9 \textbf{HEP Theories}

\textit{HEP} theories (for Hierarchical Equationally Partial Theories) have been introduced by Reichel [70]. The idea was to introduce the domains of partial functions in a structured way.

\textbf{Signatures} \textit{HEP}-signatures are of form \( \Sigma = (S, TF, PF, \preceq, \text{Def}) \) such that

- \((S, TF, PF, \emptyset)\) is a signature in \textit{PFOL}⁻,
- \(\preceq\) is a well-founded partial order on \(TF\), and
- \text{Def} is a mapping that assigns to each \(f: s₁, \ldots, sₙ \rightarrow s \in TF\) a finite set
of existence equations over the variables \( x_1 : s_1, \ldots, x_n : s_n \), such that all partial function symbols occurring in \( \text{Def}(f) \) are strictly less than \( f \) w.r.t. \( \preceq \).

Signature morphisms are those from \((S, TF, PF, \emptyset)\) with the additional requirement that

\[
\text{Def}(\sigma(f)) \subseteq \sigma(\text{Def}(f))
\]

**Models** \( \Sigma \)-models are \((S, TF, PF, \emptyset)\)-models \( A \) in \( \text{PFOL}^- \) such that \( \text{dom} f_A \) equals

\[
\{(\nu(x_1), \ldots, \nu(x_n)) \in A_w \mid \nu : X \rightarrow A \text{ with } \nu \models t_1 \equiv t_2 \text{ for all } t_1, t_2 \in \text{Def}(f)\}
\]

for each \( f : w \rightarrow s \in PF, w = s_1 \ldots s_n \).

Reducts are defined as \( \text{PFOL}^- \).

**Sentences** \( \Sigma \)-sentences are \((S, TF, PF, \emptyset)\)-sentences in \( \text{PFOL}^- \).

**Satisfaction** Satisfaction is defined as \( \text{PFOL}^- \).

Note that Reichel’s theory morphisms defined in [70] are slightly more general than theory morphisms in our \( \text{HEP} \). However, Reichel does not define an institution. Indeed, from the definitions in [70], one can only extract a specification frame (i.e., a category of theories \( \text{Th} \) and a contravariant model functor \( \text{Mod} : \text{Th}^a \rightarrow \text{CAT} \) into the quasicategory of categories). If signatures and sentences have to be separated, our slight restriction has to be made.

Now there is an easy institution representation from \( \text{HEP} \) to \( \text{PCond}^- \):  

**Signatures** A \( \text{HEP} \)-signature \( \Sigma = (S, TF, PF, \preceq, \text{Def}) \) is sent to the \( \text{PCond}^- \)-presentation consisting of the signature \((S, TF, PF, \emptyset)\) and of an axiom

\[
\forall x_1 : s_1, \ldots, x_n : s_n \bullet \text{def } f(x_1, \ldots, x_n) \iff \bigwedge_{t_1 \equiv t_2 \in \text{Def}(f)} t_1 \equiv t_2
\]

for each partial function \( f : s_1, \ldots, s_n \rightarrow s \in TF \).

**Models** The model translation is just the identity.

**Sentences** The sentence translation is the identity as well.

**Satisfaction** The representation condition is obvious. \( \Box \)

In [53], we also have set up a so-called weak institution representation in the other direction, i.e. from \( \text{PCond}^- \) to \( \text{HEP} \). It uses an encoding of partial function as sorts and of axioms as partial functions. This leads to rather complex translated theories. We therefore do not go into the details here.

Reichel also defines canons that can be used to restrict subtheories of a given theory to an initial interpretation. This can be easily translated to the free \{ \ldots \} construct in \( \text{CASL} \). However, we do not show this translation here: Since the free \{ \ldots \} construct belongs to the level of specification in-the-large, this is beyond the scope of this paper.

93
OBJ3 [31] is a specification and high-level programming language supporting sub-sorting. The logic is similar to SubCond, but has some differences. Apart from the empty carrier problem, OBJ has a different way of overload resolution. Furthermore, casts from a superset to a subset, which are partial functions in CaSL, are total functions called retracts in OBJ. Moreover, membership formulas are not present in OBJ, though there are attempts to extend OBJ with sort constraints [77], which are nothing but CaSL’s membership formulas (do not confuse them with CaSL’s sort generation constraints).

CafeOBJ [26] is a successor of OBJ3 extending OBJ3 with hidden algebra (for modeling object orientation) and rewriting logic (for modeling concurrency). Now the comparison of these features with some corresponding (object-oriented resp. concurrent) extension of CaSL is beyond the scope of this paper. We therefore concentrate on the functional part of CafeOBJ, called fun-CafeOBJ to be short. With respect to sub-sorting, the CafeOBJ report [26] leaves open the details, and states only some very general constraints on the logic. However, it is expected [25] that the sub-sorting in fun-CafeOBJ will be some kind of Meseguer’s Membership Equational Logic MEqtl [49] (it seems that also sorts in fun-CafeOBJ always have error superset; but this is more a question of language constructs). Since MEqtl already has been treated in Section 4.2, we here concentrate on OBJ3. See [66] for further comparison of CaSL and CafeOBJ (also at the in-the-large level) and more examples.

10.1 The Institution of Order-Sorted Algebra

The OBJ3 manual [31] claims that OBJ3 is based on order-sorted algebra as developed in [36]. We here describe the institution COSASC of coherent order-sorted algebra enriched with sort constraints, as introduced by Goguen and Meseguer [36, 50, 30], in some detail, since it appears that there is no complete detailed description of it in the literature.

Definition 10.1 The institution COSASC.

Signatures Order signatures are triples \( \Sigma = (S, TF, \leq) \), where \( \leq \) is a partial order on \( S \) and \( (S, TF) \) is a many-sorted signature, such that the following conditions are satisfied:

- \( f \in TF_{w,s} \cap TF_{w',s'} \) and \( w \leq w' \) imply \( s \leq s' \) (signature monotonicity)

\(^{15}\) Han Yan [77] comes closest to describing such an institution, but [77] is not publicly available.
• for each \( f \in TF_{w,s} \) and \( w_0 \leq w \in S^n \), there is a least arity for \( f \) that is greater than or equal to \( w_0 \) (regularity). This condition implies that there also is a least rank (i.e. arity and coarity) for \( f \) with arity greater than or equal to \( w_0 \), see [36].
• each connected component of \((S, \leq)\) is filtered, i.e. any two elements (of the connected component) have a common upper bound (local filtration).
    (The conjunction of regularity and local filtration is called coherence.)
We further assume that each signature is strongly locally filtered [57], which means that any two elements of a connected component have a least upper bound. This assumption is needed for setting up the institution representations, but it might be omitted when translating the language constructs.

Given two signatures \( \Sigma = (S, TF, \leq) \) and \( \Sigma' = (S', TF', \leq') \), an OSA-signature morphism \( \sigma: \Sigma \rightarrow \Sigma' \) consists of
• a map \( \sigma^S: S \rightarrow S' \),
• a family of maps \( \sigma^F_{w,s}: TF_{w,s} \rightarrow TF'_{\sigma^S(w),\sigma^S(s)} \) for each \( w \in S^*, s \in S \) such that
    • \( \sigma^S(s) \leq' \sigma^S(s') \) whenever \( s \leq s' \) (morphism monotonicity),
    • for \( w \leq w' \), \( f \in TF_{w,s} \cap TF_{w',s'} \), we have \( \sigma^F_{w,s}(f) = \sigma^F_{w',s'}(f) \) (preservation of overloading)\(^{16}\),
    • for each \( f \in TF_{w,s} \) and \( w_0 \leq w, (\sigma^S)^n \) applied to the least arity for \( f \) which is \( \geq w_0 \) yields the least arity for \( \sigma^F_{w,s}(f) \) which is \( \geq (\sigma^S)^n(w_0) \) (preservation of regularity)\(^{17}\).
We further assume that least upper bounds of connected sorts are preserved by signature morphisms.

**Models** Given \( \Sigma = (S, TF, \leq) \), an order-sorted \( \Sigma \)-algebra is a many sorted \((S, TF)\)-algebra \( M \) such that
1. \( s \leq s' \) implies \( M_s \subseteq M_{s'} \) (subsort inclusion),
2. \( f \in TF_{w,s} \cap TF_{w',s'} \) and \( w \leq w' \) imply \( f_M: M_w \rightarrow M_s \) equals \( f_M: M_{w'} \rightarrow M_{s'} \) (algebra monotonicity).

Homomorphisms \( h: M \rightarrow N \) are many-sorted homomorphisms \( h: M \rightarrow N \) such that \( s \leq s' \) and \( a \in M_s \) imply \( h_s(a) = h_{s'}(a) \). Reducts are defined as for many-sorted algebras. Since many-sorted reducts preserve inclusions and signature morphisms preserve overloading, the resulting many-sorted algebra is again an order-sorted algebra.

**Sentences** An variable system \( X \) over \( \Sigma = (S, TF, \leq) \) is a family of sets \( X = (X_s)_{s \in S} \) with the \( X_s \) pairwise disjoint. Given such a variable system, the set \( T_{\Sigma}(X) \) of \( \Sigma \)-terms of sort \( s \) is inductively defined:
1. \( x \in T_{\Sigma}(X) \) for \( x \in X_s \)
2. \( f(t_1, \ldots, t_n) \in T_{\Sigma}(X) \) for \( f: s_1, \ldots, s_n \rightarrow s \in TF \) and \( t_i \in T_{\Sigma}(X)_{s_i} \),

\(^{16}\)This condition is needed to ensure that reduct functors preserve monotonicity of models (cf. Lemma 3.5, [41]). In [77], this condition is missing, while the condition of [32] is too weak.

\(^{17}\)This condition is needed to ensure that the least sort parse algorithm described below is compatible with signature morphisms.
\( \hat{i} = 1, \ldots, n. \)

(3) \( T_\Sigma(X)_{s} \subseteq T_\Sigma(X)_{s'} \) for \( s \leq s' \).

It is straightforward to make \( T_\Sigma(X) \) into an order-sorted \( \Sigma \)-algebra. Moreover we have the following Lemma from [36], holding for regular signatures.

**Lemma 10.2** Each valuation \( \nu: X \rightarrow M \) has a unique homomorphic extension \( \nu^\#: T_\Sigma(X) \rightarrow M. \)

**Proof.** Given a variable valuation (i.e. an \( S \)-sorted map) \( \nu: X \rightarrow M \) into a \( \Sigma \)-algebra \( M \), the homomorphism \( \nu^\#: T_\Sigma(X) \rightarrow M \) is defined by

- \( \nu^\#_s(x) = \nu_s(x) \) for \( x \in X_s, s \in S, \)
- \( \nu^\#_s(\langle f(t_1, \ldots, t_n) \rangle_s) = f_M(\nu^\#_{s_1}(t_1), \ldots, \nu^\#_{s_n}(t_n)) \) for \( f \in TF_{s_1, \ldots, s_n}, t_i \in T_\Sigma(X)_{s_i} \),
- \( \nu^\#_{s'}(t) = \nu^\#_s(t) \) for \( s' \leq s, t \in T_\Sigma(X)_{s'} \).

By regularity, this is well-defined, and moreover, \( \nu^\# \) is the unique homomorphism from \( T_\Sigma(X) \) to \( M \) extending \( \nu \). By the last equation, we can omit the sort index and write just \( \nu^\#(t) \) instead of \( \nu^\#_{s}(t) \) whenever \( t \in T_\Sigma(X) \). \( \square \)

A \( \Sigma \)-sentence in OSA is a conditional formula

\[ \forall X \bullet e_1 \land \ldots \land e_n \Rightarrow e \]

where \( X \) is a variable system over \( \Sigma \) and the atomic formulas \( e \) and \( e_i \) are of two kinds: Either equations

\[ t_1 = t_2 \]

where \( t_1 \in T_\Sigma(X)_{s_1}, t_2 \in T_\Sigma(X)_{s_2} \) and \( s_1, s_2 \) are in the same connected component of \( (S, \leq) \), or sort constraints

\[ t : s \]

where \( t \in T_\Sigma(X)_{s'} \) and \( s \) is a sort in the same connected component as \( s' \).

Given a signature morphism \( \sigma: \Sigma \rightarrow \Sigma' \) and a variable system \( X \) over \( \Sigma \), we can get a variable system \( \sigma(X) \) over \( \Sigma' \) by putting

\[ \sigma(X)_{s'} = \bigcup_{\sigma(s') \rightarrow s} X_s \]

The inclusion \( \zeta_{\sigma,X}: X \rightarrow T_{\Sigma'}(\sigma(X)) \mid_\sigma \) has a unique homomorphic extension \( \zeta_{\sigma,X}^\#: T_{\Sigma'}(\sigma(X)) \rightarrow T_{\Sigma'}(\sigma(X)) \mid_\sigma \). Now a \( \Sigma \)-atomic formula \( t_1 = t_2 \) is translated to \( \zeta_{\sigma,X}^\#(t_1) = \zeta_{\sigma,X}^\#(t_2) \), and a \( \Sigma \)-atomic formula \( t : s \) is translated to \( \zeta_{\sigma,X}^\#(t_1) : \sigma^S(s) \). This easily extends to conditional formulas.

**Satisfaction** A \( \Sigma \)-algebra \( M \) satisfies a conditional axiom, if all valuations which satisfy the premises also satisfy the conclusion. A valuation \( \nu: X \rightarrow M \) satisfies an equation \( t_1 = t_2 \) if \( \nu^\#(t_1) = \nu^\#(t_2) \), and it satisfies a sort constraint \( t : s \) if \( \nu^\#(t) \in M_s \). The satisfaction condition can be proved by noting that (similarly as in Lemma 3.2) valuations \( \nu: \sigma(X) \rightarrow M \) are in a one-one-correspondence to valuations \( \nu': X \rightarrow M \mid_\sigma \) with the property that \( \zeta_{\sigma,X}^\# \circ \nu^\# \mid_\sigma = (\nu')^\# \). The satisfaction condition for sort constraints is proved in [77]. \( \square \)

96
We will also need the reduction theorem from [36], which translates order-sorted to many-sorted algebra.

In COSASC, terms are not disambiguated; they can have many different sorts. In contrast to this, terms in $\text{SubPFOL}^-$ are always fully qualified.

Recall from Section 3.2 that the many-sorted signature $\Sigma$ is obtained from a subsorted signature $\Sigma$ by adding injection, projection and membership symbols. Following Goguen and Meseguer in [36], let $\Sigma^\#$ be the subsignature of $\Sigma$ without projection and membership symbols, and let $J(\Sigma)$ be the subset of $J(\Sigma)$ consisting of the identity, embedding-injectivity and transitivity axioms and of those function monotonicity axioms for which $w = w' \leq w''$ in the condition. Further, let $\mu_\Sigma^*: \Sigma^\# \rightarrow \Sigma$ be the inclusion. We will write $J$ for $J(\Sigma)$ and $\hat{J}$ for $J(\Sigma)$ if $\Sigma$ is clear from the context.

**Theorem 10.3** A reduction from order-sorted to many-sorted algebra, can be formalized as a simple institution representation from COSA (coherent order-sorted algebra as introduced above, but without sort constraints) to $\text{Cond}^-$ (many-sorted algebra), as follows.

**Signatures** A COSA-signature $\Sigma$ is mapped to the ($\text{Cond}^-$)-presentation $\langle \Sigma^\#, J \rangle$. A signature morphism $\sigma: \Sigma \rightarrow \Sigma'$ in COSA is mapped to its obvious extension $\sigma^\#: \Sigma^\# \rightarrow \Sigma'^\#$.

**Models** By Theorem 4.2 of [36], there is an equivalence of categories ($\_\cdot$ between $\text{Mod}_{\text{Cond}^-}(\Sigma^\#, J)$ and $\text{Mod}^{\text{COSA}}(\Sigma)$. Basically, given a $\langle \Sigma^\#, J \rangle$-model $N$, the subsort injections are replaced by set inclusions in $N^\bullet$ using colimits of filtered diagrams built up by the injections. Vice versa, any $\Sigma$-algebra $M$ can be mapped to a $\langle \Sigma^\#, J \rangle$-algebra $M^\#$ by just taking the inclusions as injections. Since viewing the inclusions as injections and then going back to the injections is the identity, the representation is strongly persistently liberal. Since we have an equivalence of categories, it is also an embedding.

**Sentences** Each COSA-term $t \in T_\Sigma(X)$ has a least sort $LS_\Sigma(t)$ [36, 2.10] and a least sort parse $LP_\Sigma(t) \in T_{\Sigma^\#}(X)$ [36, p. 252]. They are defined inductively as follows: If $t = x$ is a variable, it has a unique sort $LS_\Sigma(x)$ due to the disjointness condition for variable systems, and $LP_\Sigma(x) = x$. If $t = f(t_1, \ldots, t_s) \in T_\Sigma(X)_s$, then by induction hypothesis, $t_i$ has least sort $s_i$. Take $w_0 = s_1 \ldots s_n$. Then by the term formation rules, $f \in TF_{w', s'}$ for some $w', s'$ with $s' \leq s$ and $w_0 \leq w'$. By regularity, there is a least rank $w', s'$ for $f$ such that $w' \geq w_0$. This least $s'$ is the desired least sort of $t$, while $\text{inj}_{s'}(f_{w', s'}(\text{inj}_{s_1, s_1}(LP_\Sigma(t_1)), \ldots, \text{inj}_{s_n, s_n}(LP_\Sigma(t_n))))$ is the least sort parse of $t$.

Given an equation $t_1 = t_2$, let $s_1 = LS(t_1)$, $s_2 = LS(t_2)$ and $s = \text{lub}(s_1, s_2)$, the least upper bound of $s_1$ and $s_2$ which exists by strong lo-
cal filtration. Then \( LP_\Sigma(t_1 = t_2) \) is

\[
\text{inj}_{(s_1, s)}(LP_\Sigma(t_1)) = \text{inj}_{(s_2, s)}(LP_\Sigma(t_2))
\]

This can easily be extended to conditional equations, giving translations

\( LP_\Sigma : \text{Sen}^{COSA}(\Sigma) \to \text{Sen}^{Cond^*}(\Sigma^\#) \). Since signature morphisms preserve regularity and least upper bounds, the construction is natural in \( \Sigma \).

**Satisfaction** The representation condition follows from Theorem 4.4 of [36]. \( \Box \)

We also have a representation in the converse direction:

**Theorem 10.4** Let \( COS-\text{SubCond}^* \) be the restriction of \( \text{SubCond}^* \) to coherient signatures. There is a model-bijective institution representation from \( COS-\text{SubCond}^* \) to \( COSASC \).

**Proof.** The representation works follows:

**Signatures** A coherent \( \text{CAsL} \) signature (morphism) obviously is a \( COSASC \)-signature (morphism).

**Models** Since the membership predicates in \( \tilde{\Sigma} \) are fully determined by the axioms in \( \tilde{J}(\Sigma) \), we can think of \( \text{CAsL} \) \( \Sigma \)-models as \( \langle \Sigma^\#, J \rangle \)-models. The model translation now works as follows: Just take inclusions in \( COSASC \)-models to be the injections in \( COS-\text{SubCond}^* \). Obviously, this is not bijective. However, it is a (pointwise) equivalence of categories, as shown in Theorem 10.3. (Note that

**Sentences** Just delete the injections. Applications of membership are translated into sort constraints.

**Satisfaction** As in Theorem 10.3.

\( \Box \)

The institution \( COSASC \) is not yet the right institution to give a precise meaning to all the constructs of OBJ3. Most importantly, retracts are missing. The OBJ3 system automatically extends an OSA-presentation \( \langle \Sigma, \Gamma \rangle \) with \( \Sigma = (S, TF, \leq) \) to an OSA-presentation \( \langle \Sigma^\circ, \Gamma^\circ \rangle \) which is \( \langle \Sigma, \Gamma \rangle \) extended by functions

\[
(r : s') : s' \to s
\]

and axioms

\[
\forall \{ x : s \} \bullet r : s' > s(x) = x
\]

for \( s, s' \) in the same connected component of \( (S, \leq) \). Let \( \kappa_{\Sigma, \Gamma} : \langle \Sigma, \Gamma \rangle \to \langle \Sigma^\circ, \Gamma^\circ \rangle \) be the inclusion, and \( \kappa_{\Sigma, \Gamma} = \kappa_{\Sigma, \emptyset} \). Signature morphisms \( \sigma : \Sigma \to \Sigma \) can also be extended to signature morphisms \( \sigma^\circ : \Sigma^\circ \to \Sigma^\circ \). The following lemma is important for the semantics of retracts:

**Lemma 10.5** Let \( F_{\kappa_{\Sigma, \Gamma}} : \text{Mod}(\langle \Sigma, \Gamma \rangle) \to \text{Mod}(\langle \Sigma^\circ, \Gamma^\circ \rangle) \) be the free construction (left adjoint) w.r.t. \( \text{Mod}(\kappa_{\Sigma, \Gamma}) \). Then the unit \( \iota_M : M \to F_{\kappa_{\Sigma, \Gamma}}(M) |_{\kappa_{\Sigma, \Gamma}} \)
is injective.

**Proof.** The lemma follows from Theorem 3.5 of [36] by using the diagram method for free extensions [73]. □

### 10.2 The Institutions Underlying OBJ3

Joseph Goguen claims in [32] that the semantics of a theory \( \langle \Sigma, \Gamma \rangle \) is neither \( \text{Mod}(\langle \Sigma, \Gamma \rangle) \) nor \( \text{Mod}(\langle \Sigma^0, \Gamma^0 \rangle) \), but rather \( \iota: \text{Id}_{\text{Mod}(\langle \Sigma, \Gamma \rangle)} \rightarrow \text{Mod}(\kappa_{\Sigma, \Gamma}) \circ F_{\kappa_{\Sigma, \Gamma}} \) (actually, he considers initial semantics only, and the above is the straightforward generalization to loose semantics). But there is no explicit description of an institution that uses \( \iota \) as a semantics in the literature. Perhaps the OBJ community does not feel a need of such an institution. But when translating OBJ3 to other languages, we need a precise semantics of all the underlying concepts. Therefore, in this section, we introduce two institutions that can serve as underlying institutions of OBJ3.

The following institution captures OBJ3’s retracts and also allows to deal with error recovery by treating error values as first-class citizens.

**Definition 10.6** The institution \( \text{COSASC}^\circ \).

**Signatures** A \( \text{COSASC}^\circ \)-signature (resp. signature morphism) is just a signature (resp. signature morphism) in \( \text{COSASC} \).

**Models** A \( \Sigma \)-algebra in \( \text{COSASC}^\circ \) is a \( \langle \Sigma^0, \emptyset^0 \rangle \)-algebra in \( \text{COSASC} \). Much in the same way, reducts are inherited.

**Sentences** A \( \Sigma \)-sentence in \( \text{COSASC}^\circ \) is a \( \Sigma^0 \)-sentence in \( \text{COSASC} \). Sentence translation is inherited in the same way.

**Satisfaction** A \( \Sigma \)-algebra \( M \) satisfies a \( \Sigma \)-sentence \( \varphi \) in \( \text{COSASC}^\circ \) if we have in \( \text{COSASC} \) that the \( \Sigma^0 \)-algebra \( M \) satisfies the \( \Sigma^0 \)-sentence \( \varphi \). Thus the satisfaction condition for \( \text{COSASC}^\circ \) follows from that for \( \text{COSASC} \). □

Treating error values as first-class citizens in \( \text{COSASC}^\circ \) has one main disadvantage: equations involving retracts may lead to confusion among ordinary values, which generally is undesirable. So one has to prove that retract equations do not interfere with ordinary values. More importantly, in \( \text{COSASC}^\circ \) we entirely lose the distinction between ordinary and error values, since ordinary functions and retracts can both map ordinary values to error values and vice versa. Also CafeOBJ’s error supersorts do not help for separating retract-generated error values from ordinary values. This is because retract functions

\(^{18}\) Notice that here we need the assumption that all carriers are non-empty, which implies faithfulness of \( \langle \Sigma, \Gamma \rangle \) in the sense of [36]. This will be further discussed in Section 11.2.
have to go to ordinary sorts, since they serve for the purpose of coercing a value to an (ordinary) subsort of a sort.

This justifies the introduction of a more complex institution that keeps error values and ordinary values distinct. It also comes closer to Joseph Goguen’s claim that the injection \( \iota \) is the semantics, meaning that “the ‘new error elements’ reside in a shadowy penumbra, distinctly set off from the bright central region of pure elements by the injection, as well as from the dark outer region of error terms by a subsort relation” [32]. The best approximation of this “shadowy penumbra” seems to be the institution \( COSASC_{\circ} \) introduced below. In the presence of CafeOBJ’s error supersorts, satisfaction in this institution is closely related to existential super satisfaction in [32].

**Definition 10.7** The institution \( COSASC_{\circ} \).

**Signatures** A \( COSASC_{\circ} \)-signature (resp. signature morphism) is just a signature (resp. signature morphism) in \( COSASC \).

**Models** A \( \Sigma \)-algebra in \( COSASC_{\circ} \) is a \( \Sigma \)-algebra in \( COSASC \).

**Sentences** A \( \Sigma \)-sentence in \( COSASC_{\circ} \) is a \( \Sigma^{\circ} \)-sentence in \( COSASC \).

**Satisfaction** A valuation \( \nu: X \rightarrow M \) satisfies a \( \Sigma \)-equation \( t_1 = t_2 \) with \( s = \operatorname{lub}(LS(t_1), LS(t_2)) \) if

\[
(\iota_M \circ \nu)^\#(t_1) = (\iota_M \circ \nu)^\#(t_2) \in \iota_M(M_s)
\]

where \( \iota_M \circ \nu \) is considered to be a valuation from \( X \) to \( F^{\Sigma \circ}_s(M) \). A valuation \( \nu: X \rightarrow M \) satisfies a \( \Sigma \)-sort constraint \( t : s \) if

\[
(\iota_M \circ \nu)^\#(t) \in \iota_M(M_s)
\]

This is extended to sentences in the same way as for \( COSASC \).

In order to prove the satisfaction condition, consider a signature morphism \( \sigma: \Sigma \rightarrow \Sigma' \) and a \( \Sigma' \)-algebra \( M' \). By the universal property of the free extension \( F^{\Sigma \circ}_s(M'\mid \sigma) \), the homomorphism \( (\iota_{M'})\mid \sigma \) has a unique extension

\[
(\iota_{M'})\mid \sigma^-: F^{\Sigma \circ}_s(M'\mid \sigma) \rightarrow F^{\Sigma \circ}_s(M')\mid \sigma^-\circ
\]

making the right triangle in the diagram below commutative. Now we use the fact stated in the satisfaction part of Definition 10.1 that valuations \( \nu: \sigma(X) \rightarrow M' \) are in a one-one-correspondence to valuations \( \nu\mid \sigma: X \rightarrow M'\mid \sigma \) with the property that \( \zeta_{X,\sigma}^\#: \nu\mid \sigma \rightarrow (\nu\mid \sigma)^\# \). The remaining subdiagrams in the following diagram commute by freeness of the respective term.
algebras; thus, the outer square commutes as well.

\[
\begin{array}{c}
|T_{\Sigma \oplus}(X)| \\
\zeta^\oplus_{\odot,X} \downarrow \\
|T_{\Sigma}(X)| \\
\zeta^\oplus_{\odot} \downarrow \nu^\oplus_{\odot} \downarrow |M'|_{\sigma} \\
|T_{\Sigma}(\sigma(X))|_{\sigma} \downarrow \nu^\oplus_{\odot} \downarrow (\iota_{M'})_{|\sigma} \downarrow |F_{\kappa_{\Sigma'}}(M')|_{\sigma \odot} \downarrow \\
|T_{\Sigma}(X)| \\
\zeta^\oplus_{\odot,X} \downarrow \\
|T_{\Sigma}(X)| \\
\zeta^\oplus_{\odot} \downarrow \nu^\oplus_{\odot} \downarrow |M'|_{\sigma} \\
|T_{\Sigma}(\sigma(X))|_{\sigma} \downarrow \nu^\oplus_{\odot} \downarrow (\iota_{M'})_{|\sigma} \downarrow |F_{\kappa_{\Sigma'}}(M')|_{\sigma \odot} \downarrow \\
\end{array}
\]

Since \((\iota_{M'})_{|\sigma} \circ \iota_{M'}_{|\sigma} = (\iota_{M'})_{|\sigma}\) exactly to the image of \(\iota_{M'}_{|\sigma}\) is injective, since \((\iota_{M'})_{|\sigma} \circ \iota_{M'}_{|\sigma} = (\iota_{M'})_{|\sigma}\) is injective by Lemma 10.5. Therefore, there exists some \(a \in (M')_{|\sigma}\) with

\[(\iota_{M'}_{|\sigma} \circ \nu_{|\sigma})(t_1) = (\iota_{M'}_{|\sigma} \circ \nu_{|\sigma})(t_2)\]

if and only if there exists some \(a \in M'_{\sigma\delta\iota(s)}\) with

\[(\iota_{M'} \circ \sigma(\nu))^\#(\zeta^\#_{\odot,X}(t_1)) = \iota_{M'}(a) = (\iota_{M'} \circ \sigma(\nu))^\#(\zeta^\#_{\odot,X}(t_2))\]

Since \(\zeta^\#_{\odot,X}\) is used for sentence translation, the satisfaction condition follows easily. \(\square\)

In \textit{COSASC}_\odot, error values generated by retracts only have a virtual existence, since retracts are always interpreted freely. This means that equations involving retracts are false in all models if they do not follow already from the built-in retract equations in \(\Gamma^{\odot}\). In the case that all user-defined retract equations involve only one retract, and this retract occurs at the utmost position of a term, this works fine, and \textit{COSASC}_\odot corresponds to the operational semantics of OBJ3.

However, \textit{COSASC}_\odot does not work for error recovery, where error values should be treated as first-class values (such that functions can compute with them). In this case, we have to use \textit{COSASC}^\odot, though we then lose the distinction between ordinary and error values.

Let us illustrate this with an example. Consider the following OBJ3 specification of paths over an arbitrary graph, which later on is instantiated to a particular graph with four nodes and three edges (the specification is also a
CafeOBJ specification, the only difference being that CafeOBJ generates the super-sort Path_Err automatically):

\[ \text{th GRAPH is} \]
\[ \text{sorts Nodes Edges} \]
\[ \text{ops source target : Edges -> Nodes} \]
\[ \text{end} \]

\[ \text{obj PATH[G :: GRAPH] is} \]
\[ \text{sorts Edges < Path} \]
\[ \text{sorts Path < Path_Err} \]
\[ \text{ops source target : Path -> Nodes} \]
\[ \text{op } _ :: _ : \text{Path Path -> Path_Err assoc} \]
\[ \text{vars p,q : Path} \]
\[ \text{ceq source((p :: q) as Path) = source(p) if p :: q : Path} \]
\[ \text{ceq target((p :: q) as Path) = target(q) if p :: q : Path} \]
\[ \text{ceq target(p)=source(q) if p :: q : Path} \]
\[ \text{ceq p :: q in Path if target(p)=source(q)} \]
\[ \text{end} \]

\[ \text{obj MY-GRAPH is} \]
\[ \text{using GRAPH} \]
\[ \text{ops a b c d : -> Nodes} \]
\[ \text{ops 1 2 3 : -> Edges} \]
\[ \text{eq source(1) = a} \]
\[ \text{eq target(1) = b} \]
\[ \text{eq source(2) = b} \]
\[ \text{eq target(2) = c} \]
\[ \text{eq source(3) = c} \]
\[ \text{eq target(3) = d} \]
\[ \text{end} \]

\[ \text{obj MY-PATH is} \]
\[ \text{protecting PATH[MY-GRAPH]} \]
\[ \text{end} \]

In \textit{COSASC$^\circledR$}, the initial Path-model contains as values of the sort Path not only values like 1, 2, 3, 1::2, 2::3, but also values like r:Path_Err>Path(2::1). Thus, not only the sort Path_Err contains “error elements” like

\[ r:Path_Err>Path(2::1), \]

also the sort Path contains them!

Now in \textit{COSASC$^@$}, the specification behaves as expected: the sort Path of
the initial **MY-PATH**-model indeed is just interpreted (up to change of representation) by the set

\[ \{1, 2, 3, 1:2, 2:3, 1:2:3 \}, \]

while the sort **Path_Err** additionally contains “error elements”, for example \( r: \text{Path_Err} > \text{Path}(2:1) \).

### 10.3 Translating COSASC\(^\circledast\) to SubCond\(^-\)

In this and the following subsection, we describe two translations of institutions underlying OBJ3 to (fragments of) the CASL institution, serving different methodological purposes. The present section describes the translation of COSASC\(^\circledast\) to subsorted conditional equational logic (SubCond\(^-\)). It translates retracto to **total** functions, and thus retracts usually generate new values in the carrier sets (which are called error values).

**Theorem 10.8** The following describes a strongly persistently liberal simple institution representation from COSASC\(^\circledast\) to SubCond\(^-\) that is also an embedding.

**Signatures** A COSASC\(^\circledast\)-signature \( \Sigma \) is mapped to \( \langle \Sigma^\otimes, \emptyset^\otimes \rangle \) considered as an SubCond\(^-\)-presentation (which means that the retracts in the retract equations have to be explicitly qualified with their profiles). A signature morphism \( \sigma: \Sigma \rightarrow \Sigma' \) is mapped to the signature morphism \( \sigma^\otimes: \Sigma^\otimes \rightarrow \Sigma'^\otimes \).

Now \( \sigma^\otimes \) is monotone because \( \sigma \) is. It remains to show that \( \sigma^\otimes \) preserves the overloading relation \( \sim_F \). Let \( f : w_1 \rightarrow s_1 \sim_F f : w_2 \rightarrow s_2 \) in \( \Sigma \), i.e. there exist \( w_0 \in S^* \) with \( w_0 \leq w_1 \) and \( w_0 \leq w_2 \) and a common supersort \( s_0 \) of \( s_1 \) and \( s_2 \). By regularity of \( \Sigma \), there is some \( w \in S^* \), \( s \in S \) with \( f : w \rightarrow s \) and \( w_0 \leq w \leq w_1, w_2 \). Since \( \sigma \) preserves overloading, \( \sigma^\otimes_{w_1,s_1}(f) = \sigma^\otimes_{w_2,s_2}(f) = \sigma^\otimes_{w,s_0}(f) \).

**Models** Given a COSASC\(^\circledast\)-signature \( \Sigma \), the model translation is the fol-
owing composite ([–• has been introduced in Theorem 10.3): 19

\[
\begin{align*}
\text{Mod}^\text{COSA}\oplus(\Sigma) & \quad \text{Mod}^\text{SubCond}^-((\Sigma^\oplus, \emptyset^\oplus)) \\
\text{Mod}^\text{COSA}(\Sigma^\oplus, \emptyset^\oplus) & \xrightarrow{[\cdot]} \text{Mod}^\text{Cond}^\sim(\Sigma^\oplus#, J^\oplus) & \text{Mod}^\text{Cond}^\sim(\Sigma^\oplus, J^\oplus)
\end{align*}
\]

Concerning strongly persistent liberality and the embedding property, ([–• has a left adjoint right inverse and is an equivalence of categories due to Theorem 10.3. \text{Mod}(\mu_{\Sigma^\oplus}) just forgets membership predicates, which are fully determined by the axioms in \(J^\oplus\). Therefore, it is even an isomorphism. (The additional function-overloading axioms in \(J^\oplus\) already follow from those in \(J^\oplus\) with an argument similar to the above argument showing preservation of overloading by signature morphisms.)

**Sentences** Given a \text{COSASC}\oplus-signature \(\Sigma\), for sentences without sort constraints, the sentence translation is the following composite (\(LP\) has been introduced in Theorem 10.3):

\[
\begin{align*}
\text{Sen}^\text{COSA}\oplus(\Sigma) & \quad \text{Sen}^\text{SubCond}^-((\Sigma^\oplus)) \\
\text{Sen}^\text{COSA}(\Sigma^\oplus) & \xrightarrow{[\cdot]} \text{Sen}^\text{Cond}^\sim(\Sigma^\oplus#, \text{Sen}^\text{LP}(\Sigma^\oplus)) & \text{Sen}^\text{Cond}^\sim(\Sigma^\oplus)
\end{align*}
\]

Sort constraints within sentences are translated as follows: Given \(t : s\) as an atomic constituent of a sentence in \text{Sen}^\text{COSA}\oplus(\Sigma), \text{let } s' = \text{lub}(s, \text{LS}(t)) \text{ and translate it to}

\[
\text{inj}_{(\text{LS}(t), s')}(\text{Sen}(\mu_{\Sigma^\oplus})(\text{LP}_{\Sigma^\oplus}(t))) \in s
\]

as an atomic constituent in \text{Sen}^\text{SubCond}^-((\Sigma^\oplus)).

**Satisfaction** The representation condition follows from that of the representation in Theorem 10.3. For sort constraints, we also need the axioms for membership in \(J\). \(\square\)

When we translate the OBJ3 specification \text{MY-PATH} given above with this translation, we get the CASL specification

**spec**

GRAPH =

sorts Nodes, Edges

ops source, target : Edges \rightarrow Nodes;

a, b, c, d : Nodes;

1, 2, 3 : Edges

19 Since \text{SubCond}^- is a total framework, we have to leave out the partial projections from \(\Sigma\) as said in the paragraph about partiality in Section 3.4.1. Otherwise, the target of the translation would be \text{PHorn}^-.
axioms source(1) = a;
    target(1) = b;
    source(2) = b;
    target(2) = c;
    source(3) = c;
    target(3) = d
end

spec Path[GRAPH] =
sorts Edges < Path;
Path < Path_Err
ops r.Path_Err.Path : Path_Err → Path;
    source, target : Path → Nodes;
    _::*_ : Path × Path → Path_Err, assoc
forall p, q : Path
    • p :: q ∈ Path ⇒ source(r.Path_Err.Path(p :: q)) = source(p)
    • p :: q ∈ Path ⇒ target(r.Path_Err.Path(p :: q)) = target(q)
    • p :: q ∈ Path ⇔ target(p) = source(q)
end

spec MY_GRAPH =
    GRAPH
then
    ops a, b, c, d : Nodes;
    1, 2, 3 : Edges
axioms source(1) = a;
    target(1) = b;
    source(2) = b;
    target(2) = c;
    source(3) = c;
    target(3) = d
end

spec MY_PATH =
    PATH[MY_GRAPH]
end

10.4 Translating COSASC to SubPCond−

The translation of COSASC to SubPCond− maps retracts to partial projections. This means that retracts do not generate new values in the carrier sets.

Theorem 10.9 The following describes a strongly persistently liberal simple institution representation from COSASC to SubPCond− that is also an
embedding.

**Signatures** Any \( \text{COSASC}_\oplus \)-signature is a \( \text{SubPC}d^\rightarrow \)-signature by letting the set of partial function symbols be empty.

**Models** Given a \( \text{COSASC}_\oplus \)-signature \( \Sigma \), the model translation is the following composite:

\[
\begin{array}{c}
\text{Mod}^{\text{COSASC}_\oplus}(\Sigma) \\
\Downarrow \\
\text{Mod}^{\text{COSASC}}(\Sigma) \overset{(\cdot)^*}{\longrightarrow} \text{Mod}^{\text{COSASC}}(\langle \Sigma^\#, J \rangle) \\
\Downarrow \\
\text{Mod}^{\text{PC}d^\rightarrow}(\langle \hat{\Sigma}, \hat{J} \rangle)
\end{array}
\]

Concerning strongly persistent liberality and the embedding property, \((\cdot)^* \) has a left adjoint right inverse and is a natural equivalence due to Theorem 10.3. \( \text{Mod}^{(\mu_S)} \) just forgets membership predicates and partial projections, which are fully determined by the axioms in \( J^\oplus \). Therefore, using a similar argument as in the proof of Theorem 10.8, it is even an isomorphism.

**Sentences** Given a \( \text{COSASC}_\oplus \)-signature \( \Sigma \), sentences without sort constraints are translated with the following composite:

\[
\begin{array}{c}
\text{Sen}^{\text{COSASC}_\oplus}(\Sigma) \\
\Downarrow \\
\text{Sen}^{\text{COSA}}(\Sigma^\oplus) \overset{\text{LP}^{\text{COSA}}}{\longrightarrow} \text{Sen}^{\text{Cond}^\rightarrow}(\Sigma^\oplus) \\
\Downarrow \\
\text{Sen}^{\text{PC}d^\rightarrow}(\hat{\Sigma})
\end{array}
\]

Here, \([\cdot] \) is defined inductively as follows:

- \([x] = x \)
- \([\text{inj}_{(s,s')}^\oplus(t)] = \text{inj}_{(s,s')}([t]) \)
- \([r : s' > s(t)] = \text{pr}_{(s',s)}([t]) \)
- \([\text{inj}_s(t_1, \ldots, t_n)] = \text{inj}_{(s,\text{inj}_s(t_1), \ldots, \text{inj}_s(t_n))} \)
- \([t_1 = t_2] = [t_1] = [t_2] \)
- \(\forall X \bullet e_1 \land \ldots \land e_n \Rightarrow e = \forall X \bullet [e_1] \land \ldots \land [e_n] \Rightarrow [e] \)

A sort constraint \( t : s \) occurring in a sentence is translated to

\[
\text{inj}_{(\text{LS}(t), s')}(\text{LP}_{\Sigma^\oplus}(t)) \in s,
\]

where \( s' = \text{lub}(s, \text{LS}(t)) \).

**Satisfaction** Given a \( M \in \text{Mod}^{\text{PC}d^\rightarrow}(\hat{\Sigma}) \), by Theorem 10.3 without loss of generality we can assume that \( M \) consists of inclusions as injections already, and \( (M|_{\mu_S})^* = M|_{\mu_S} \) (note that also \( (M|_{\mu_S})_s = M_s \)). The key to prove the representation condition is the following lemma:

**Lemma 10.10** Given \( M \in \text{Mod}^{\text{PC}d^\rightarrow}(\hat{\Sigma}) \) with \( (M|_{\mu_S})^* = M|_{\mu_S} \), valuations \( \nu : X \to M \) are the same as valuations \( \tilde{\nu} : X \to (M|_{\mu_S}) \). For \( t \in T_{\Sigma^\oplus}(X) \), the following properties hold:

1. \( \nu^\#([\text{LP}_{\Sigma^\oplus}(t)]) \) is defined iff \( (\iota_{(M|_{\mu_S})} \circ \tilde{\nu})^\#(t) \in (\iota_{(M|_{\mu_S})})(M|_{\mu_S}) \)
(2) In case of (1), \( \iota(M|_{\rho_{\Sigma}})(\nu^\#([LP_{\Sigma}(t)])) = (\iota(M|_{\rho_{\Sigma}} \circ \tilde{\nu})^\#(t) \)

**Proof.** The proof proceeds by induction over the structure of \( t \). The only critical case is that of retracts. We have \( \nu^\#([LP_{\Sigma}(r : s' > s(t))]) \) is defined iff \( \nu^\#([pr_{(s',s)}(LP_{\Sigma}(t))]) \) is defined and \( \in M_s \) iff \( (\iota(M|_{\rho_{\Sigma}} \circ \tilde{\nu})^\#(t) \in \iota(M|_{\rho_{\Sigma}})((M|_{\rho_{\Sigma}})_s) \) iff (since retracts are interpreted freely outside \( M_s \) \( (\iota(M|_{\rho_{\Sigma}} \circ \tilde{\nu})^\#(r : s' > s(t)) \in \iota(M|_{\rho_{\Sigma}})((M|_{\rho_{\Sigma}})_s) \). \)

**Proposition 10.11** The model translation component of the representation from \( COSASC_{\otimes} \) to \( SubPCond^- \) is a natural equivalence.

**Proof.** \((_.^\#)*\) is shown to be a natural equivalence in [36], and \( \text{Mod}(\mu_{\Sigma}) \) is a natural equivalence since \( \langle \Sigma, \mathcal{J} \rangle \) is a definitional extension of \( \langle \Sigma^#, \mathcal{J} \rangle \). \)

With this translation, the specification \( \text{Path} \) is translated to \( \text{CASL} \) as follows:

**spec** Graph =

**sorts** Nodes, Edges

**ops** source, target : Edges \( \rightarrow \) Nodes;

\( a, b, c, d : \) Nodes;

\( 1, 2, 3 : \) Edges;

**axioms** source(1) = a;

target(1) = b;

source(2) = b;

target(2) = c;

source(3) = c;

target(3) = d

end

**spec** Path [Graph] =

**sorts** Edges < Path;

Path < Path_Err

**ops** source, target : Path \( \rightarrow \) Nodes;

\( \_ :: \_ : \) Path \( \times \) Path \( \rightarrow \) Path_Err

**forall** p, q : Path

\( p :: q \in \) Path \( \Rightarrow \) source((p :: q) as Path) = source(p)

\( p :: q \in \) Path \( \Rightarrow \) target((p :: q) as Path) = target(q)

\( p :: q \in \) Path \( \Leftrightarrow \) target(p) = source(q)

end

10.5 Translating OBJ3's Constructs into CASL Constructs

We now briefly describe how any of the institution representations defined in the previous section can be lifted to a translation of language constructs.

107
One difference between OBJ3 and Casl concerns disambiguation of terms. Sort disambiguation in OBJ3 inserts subsort inclusions and retracts, while in Casl, only subsort injections are inserted. Note that Casl's disambiguation [61] corresponds to the least sort parse for regular signatures. Thus, at the construct level, the translation from OBJ3 to Casl has just to insert retracts. (It does not matter here that at the concepts level, Casl terms are disambiguated while OBJ3 terms are not.)

Mixfix declaration facilities are essentially the same for OBJ and Casl. Precedences between mixfix operations are expressed with numeric priorities in OBJ3, while Casl allows arbitrary pre-orders as in ASF+SDF. Thus, OBJ3 numeric priorities have to be translated to the pre-order induced by them. Concerning the associativity of mixfix operations, OBJ3 allows to declare rather general gathering patterns. These are not available in Casl. Casl only allows to declare binary infix operations to be left or right associative, this corresponds to special gathering patterns in OBJ3. Other gathering patterns cannot be translated to Casl. Instead, one has to parse the formulas and terms with an OBJ3 parser (using the gathering patterns) and using explicit bracketing for disambiguation wherever necessary in the translation to Casl.

Explicit sort disambiguation (written t.s in OBJ3) is also available in Casl (written t : s). Sort constraints, written as t : s in (extensions of) OBJ3, are written as membership formulas t ∈ s in Casl.

Associativity, commutativity, identity and idempotence attributes are slightly more general in OBJ3 than in Casl, since in Casl, argument and result sorts of the operations with attributes have to be the same. In the case that this condition does not hold, the attribute has to be translated to an axiom.

OBJ3's subsort declarations allow chains of subsorts to be declared. This has to be split up into several declarations in Casl. Finally, OBJ3's let definitions have to be translated to Casl's operation definitions.

Concerning functional CafeOBJ, a distinguishing feature of fun-CafeOBJ as used in the CafeOBJ report is that each sort comes equipped with an error supersort. This feature need not be treated at the level of the logic though: the right place to introduce the error supersorts is the level of language constructs, namely the function that extracts a signature in the institution from given specification should introduce the error supersorts.
11 Boolean Functions and Empty Carriers

In this section, two problems that are common to all specification language translations are discussed, namely the treatment of boolean functions and empty carriers.

11.1 Translating Boolean Functions to Predicates

Larch, ACT ONE and OBJ3 all have a built-in sort `Bool` or `Boolean` of boolean values, together with some Boolean operations.\(^{20}\) The translations of the previous sections treat `Bool` like any other sort. This means that the implicit inclusion of `Bool` into each specification must be handled at the level of structured specifications: When translating a Larch, ACT ONE or OBJ3 specification to CASL, one has to make the implicit import of `Bool` (together with possibly equality and if-then-else functions, depending of the language) explicit in the result of the translation. Further, when translating ACT ONE or OBJ3, the CASL specification of `Bool` has to be surrounded by a `free { ... }`, since the Booleans are always interpreted initially in these languages. For the declared constructor functions, ACT ONE generates special equations of sort `Bool`, which have to be included in the translation as well.

However, it may be desirable to treat `Bool` as a special sort and translate Larch’s, ACT ONE’s or OBJ3’s `Bool`-valued functions into CASL’s `predicates`. This works only for theories that do not use `Bool` as argument sort of a function.

We thus can modify the translations of Sections 6.1, 10.3 and 10.4 by mapping `Bool`-valued functions into predicates at the signature level. At the sentence level, applications of `Bool`-valued functions are translated to applications of predicates, and the standard `Bool`-functions are translated into the logical connectives.\(^{21}\) At the model level, predicates are translated to `Bool`-valued functions.

For example, concerning OBJ3, let us restrict `COSASC\(^{®}\)` (or `COSASC\(_{®}\)`) to signatures including the standard signature `BOOL` (with a fixed meaning), but without any functions having `Bool` as argument sort except those in `BOOL`, thus getting the institution `COSASC\(^{®}\)-Bool` (or the institution `COSASC\(_{®}\)-Bool`). We then have a strongly persistently liberal institution representation from

\(^{20}\) For LSL, the institution underlying Larch, Baumeister [8] has developed a formal treatment of signatures with fixed Boolean subsignature.

\(^{21}\) If-then-else is available only as a construct, not as a concept in CASL. The translation from CASL constructs to concepts can be used as translation here.

109
COSASC\textsuperscript{\textregistered}-Bool to SubFOL\textsuperscript{-}, and another one from COSASC\textsubscript{\textregistered}-Bool to SubPFOL\textsuperscript{-}. A similar remark holds for LSL: we get a representation LSL-Bool \rightarrow CFOL\textsuperscript{-}.

Concerning ACT ONE, we can use GCond\textsuperscript{-}-Bool \rightarrow FOL\textsuperscript{-} constructed in an analogous way. (This representation can also be used for ASF, although ASF does not have an implicitly imported Bool). Now in this setting (of translating Bool-valued functions to CASL predicates), the special axioms of sort Bool generated for constructors in ACT ONE (stating injectivity of constructors and disjointness of their images) can be generated with a free type \ldots construct in CASL, which generates exactly the same axioms as first-order formulas.

11.2 The Empty Carrier Problem

All institutions of order-sorted and partial algebras which we have defined in this paper do not admit empty carriers. The reader may be surprised by this, since many-sorted and subsorted algebras usually are defined in a way allowing empty carriers [28,36].

The main reason to disallow empty carriers in CASL was that when allowing empty carriers, the satisfaction of a sentence like

$$\forall x : s \cdot a = b$$

would depend not only on the equality of the interpretations of a and b, but also on the question whether the carrier for sort s is empty or not. This may lead to confusion, especially when the variable x : s is not introduced locally, but through a global variable declaration.

Surprisingly, another reason to forbid empty carriers lies in the framework of order-sorted algebra that has been developed by advocates of empty carriers: Lemma 10.3, which is crucial for proving the satisfaction condition of COSASC\textsubscript{\textregistered}, only holds if we forbid models that mix empty and non-empty carriers.\textsuperscript{22} We here additionally forbid models consisting only of empty carriers, but this additional restriction is not essential in any respect.

Forbidding empty carriers is not a great loss, since it is unlikely that a datatype contains no data. Moreover, it is even an advantage: since the standard definitions of first-order and higher-order logic forbid empty carriers as well, there is a straightforward bridge from CASL to FOL\textsuperscript{-} or HOL\textsuperscript{-}-based theo-

\textsuperscript{22}It also holds if we restrict ourselves to confluent sets of axioms [36], but this does not help for defining an institution.
rem proving tools (which usually assume non-empty carriers). Empty carriers would overly complicate the use of such tools for CASL.

It has been argued [35] that allowing empty carriers is necessary to get initial models and free extensions. However, Corollary 4.16 shows that initial models and free extensions exist if we restrict ourselves to strict signatures. Now in [35] it is argued that there are useful theories that are not strict (e.g. the theory of pre-ordered sets). However, such theories can still be used as parameter theory for free extensions, since Corollary 4.16 only requires the target theory to be strict. It will hardly be the case that one wants to have initial or free semantics for non-strict theories.

For the rare cases where an example specification should really use empty carriers, and for the re-use of tools supporting empty carriers, we now examine the relation between institutions with and without empty carriers and set up representations between these institutions. Then, by composing with one of these representations, we can shift the representations of Fig. 3, 4 and 5 to representations between institutions that differ in their behaviour w.r.t. empty carriers.

For any institution $I$ introduced in the previous sections, let $I^{mI}$ denote $I$ with a slightly modified model functor that allows the possibility of empty carriers in the model.

Let us now consider the representation of $I$ in $I^{mI}$ and vice versa at the first-order level.\footnote{Note that most formulations of first-order logic exclude empty carriers anyway, so these representations will be needed very rarely only.} There are easy representations between $FOL=^=$ and $(FOL=^=)^{mI}$:

- For the representation of $(FOL=^=)^{mI}$ in $FOL=^=$, we can simply use representation (4) from the first-order level (Section 4.1.4) and delete the axiom $\exists x : s \bullet D_{s}(x)$ stating non-emptiness of the set of "defined" elements. This gives a representation of $(PFOL^{-})^{mI}$ in $FOL^{-}$, which then can be composed with the inclusion of $(FOL=^=)^{mI}$ into $(PFOL^{-})^{mI}$. As in (4), we thus get a strongly persistently liberal representation with the weak (injective)-amalgamation property.

- $FOL^{-}$ can be represented in $(FOL=^=)^{mI}$ by adding to a signature an axiom

  $$\exists x : s \bullet true$$

  for each sort $s$. Axioms and models are translated identically. This gives a subinstitution representation. $\square$

Instead of $FOL=^=$ we could have used any other institution at the first-order level and get the same results.
At the positive conditional level, we do not have such strong relations.\textsuperscript{24} We have the following representations between $(SubP)\text{Horn}^=\!\!\!\!$ and $(SubP)\text{Horn}^=\!\!\!\!mt$:

- $(\text{Horn}^=\!\!\!\!mt)$ can be represented in $\text{Horn}^=\!\!\!\!$ in much the same way as $(\text{FOL}^=\!\!\!\!mt)$ can be represented in $\text{FOL}^=\!\!\!\!$ above. We have to delete not only the axioms of form $\exists x : s \bullet D_s(x)$, but also the operations $\bot$ and the axioms involving them in order to get theories in Horn form. This gives a strongly persistently liberal representation (however, due to the deletion of $\bot$, weak (injective)-amalgamation is lost).

The same thing also works for the superinstitutions of $\text{Horn}^=\!\!\!\!$ defined at the positive conditional level.

Concerning representing $(\text{Cond}^=\!\!\!\!mt)$ in $\text{Cond}^=\!\!\!\!$, we can use representation (8) from the positive conditional level and delete the newly introduced constants that guarantee the existence of “defined” elements.

- $\text{Horn}^=\!\!\!\!$ can be represented in $\text{Horn}^=\!\!\!\!mt$ by adding to a signature an overloaded constant $c : s$ for each sort $s$. Axioms and models are translated identically. This gives a model-bijective institution representation. The same thing also works for the sub- and superinstitutions of $\text{Horn}^=\!\!\!\!$ defined at the positive conditional level. $\square$

12 Conclusion

We have considered the relation of C\text{ASL} to other specification languages at the level of specification in-the-small. To this end, we have defined a number of subinstitutions of the institution $\text{SubPCFOL}^=\!\!\!\!$ underlying the specification language C\text{ASL}.

Among these subinstitutions of $\text{SubPCFOL}^=\!\!\!\!$, we have defined a number of institution representations. Altogether, three graphs of institutions and representations arise: two at the first-order level (one with, one without sort generation constraints), and one at the positive conditional level. We also have classified the institution representations according to different properties, and have shown how these properties lead to a good interaction with theorem proving. More specifically, the properties allow (1) the lifting of theorem provers for flat specifications, (2) for structured specifications without \textbf{free}, (3) for structured specifications including \textbf{free} and/or (4) the lifting of loose semantics. As a consequence, we get

(1) All the institutions of the positive conditional level are liberal w.r.t. so-called strict theory morphisms (this is proved by lifting \textbf{free} constructions

\textsuperscript{24}Note that at the positive conditional level, there are much more frameworks allowing empty carriers than in the first-order case.
from well-known institutions to other subinstitutions of Casl).

(2) First-order theorem provers with induction (or, at the level of positive
conditional specifications, conditional term rewriters and paramodulation)
are lifted for use for theorem proving within basic specifications in Casl. At the first-order level, this result also holds for structured
specifications. Actually, we have used the composition of translations
(7) \( \circ (6') \circ (5\alpha) \) (cf. Section 4.1) from the first-order level, going from
the Casl logic \( SubPCFOL^= \) to second-order logic \( SOL^= \) as the basis
for the logical encoding provided within the Casl tool set (CATS) [51].
CATS also provides the intermediate steps of this composite encoding,
having targets \( CFOL^= \) and \( SubCFOL^= \) (the former roughly being first-
order logic with induction). Thus it is possible to use CATS to connect
Casl to standard first-order or higher-order theorem provers.

As a first practical example, we have used this to encode Casl in Is-
abelle/HOL in the HOL-Casl system, see [58] for details. The experience
was that the institution representation can serve as a semantical basis for
the re-use of theorem provers, but to make it work in practice requires
more work. One important point is the question of a partial inverse of the
representation, which makes it possible to display intermediate formulas
in a proof in the syntax of the represented institution.

(3) Moreover, we have described the translation of a number of well-known
specification languages (Larch, OBJ3, functional CafeOBJ, ACT, ASF,
and HEP-theories) into Casl. We have set up formally precise
translations at the level of the underlying logics (formalized as institutions),
and informally discussed how this lifts to the level of language constructs.
Thus, specification libraries and case studies developed in these languages
can be translated to Casl.

Most of the specification languages that we have examined are based on a
rather straightforwardly defined subinstitution of the Casl institution. The
exception is OBJ3. When attempting to translate OBJ3 to Casl, we first
had to clarify what the institution underlying OBJ3 is. Actually, we have
found two institutions for OBJ3, serving different methodological purposes.
One institution (\( COSASC^\oplus \)) treats error values generated by retracts as first-
class citizens, allowing error recovery, while the other one (\( COSASC_\oplus \)) treats
error values as virtual values, allowing a clean separation of ordinary and error
values.

We also have addressed the level of language constructs and have (informally)
shown how the constructs for writing specifications in-the-small from other
languages can be translated into Casl. A formal treatment would need a
formal definition of the other languages, which is not available in all cases and
which also would be too detailed.

Finally, we have shown how the problem that most of the existing algebraic
specification languages differ from CASL in that they have a semantics allowing models with empty carriers can be solved.

An important future extension of the translations will be to consider also the translation among different concepts for specifications in the large. Here, the main difficulty will be the different philosophies behind the module systems: While OBJ3, CafeOBJ and ACT TWO use a colimit-based approach where the same name can have different meanings (depending on its origin module), CASL follows a same-name-same-thing philosophy.

Acknowledgments I would like to thank Maura Cerioli, Răzvan Diaconescu, Joseph A. Goguen, Anne Haxthausen, Kolyang, Hans-Jörg Kreowski, Bernd Krieg-Brückner, José Meseguer, Peter D. Mosses, Markus Roggenbach, Grigore Rosu, Don Sannella, Lutz Schröder and Andrzej Tarlecki for intensive and sometimes controversial, but always productive discussions, and further, all the (other) participants of CoFI. Special thanks to Lutz Schröder for help with the appendix.

A Preservation of Freeness

The following propositions are important for proving properties about persistently liberal institution representations. We will use the following terminology: Given a functor $F$ left adjoint to $G$, $\eta^G$ and $\varepsilon^F$ (or just $\eta$ and $\varepsilon$, if no confusion can arise) will denote the unit and counit of some corresponding adjoint situation.

**Proposition A.1** Given $R: B \to A$ with left adjoint $L$ such that $R \circ L \cong id$, then $\eta, \varepsilon_L$, and $R\varepsilon$ are isomorphisms. Moreover, $L$ is full.

**Proof.** Let $\delta: R \circ L \to id$ be a natural isomorphism, and let $A$ be an object in $A$.

Since $R\varepsilon \circ \eta_R = id$ for any adjunction, $R\varepsilon_L \circ \delta^{-1}_{RLA} \circ \delta_{RLA} \circ \eta_{RLA} = id$. Hence, the upper triangle commutes. By naturality of $\delta \circ \eta$, also the square commutes. Thus, also the lower triangle commutes. But this shows $(\delta \circ \eta)_{RLA}$ to be an isomorphism. Since $\delta$ is an isomorphism, $\eta_{RLA}$ is an isomorphism as well. By
naturality of $\eta$,

$$
\begin{array}{ccc}
RLA & \xrightarrow{\eta_{RLA}} & RLRLA \\
\downarrow{\delta_A} & & \downarrow{R\delta_A} \\
A & \xrightarrow{\eta_A} & RLA
\end{array}
$$

commutes. Hence, $\eta_A$ is an isomorphism. Since $\varepsilon_L \circ L\eta = id$ and $R\varepsilon \circ \eta_R = id$ for any adjunction, $\varepsilon_L$ and $R\varepsilon$ are isomorphisms as well.

Fullness of $L$ follows with the dual of [1, 19.14]. □

**Proposition A.2** Let the following diagram of categories and functors be given.

$$
\begin{array}{ccc}
A & \xrightarrow{L} & B \\
\downarrow{U} & & \downarrow{V} \\
X & \xleftarrow{L'} & Y
\end{array}
$$

Assume further that

- $U \circ R = R' \circ V$,
- $R \circ L \cong id$ and $R' \circ L' \cong id$,
- $L$ is left adjoint to $R$, and
- $L'$ is left adjoint to $R'$.

Then

1. If $B \in B$ is $V$-free over $L'X$ for some $X \in X$, then $RB$ is $U$-free over $X$.
2. $A \in A$ is $U$-free over $X \in X$ iff $LA$ is $V$-free over $L'X$.
3. If $B \in B$ is strongly persistently $V$-free and $VB$ is in the image of $L'$, then $RB$ is strongly persistently $U$-free.
4. Further assume that $L' \circ U = V \circ L$. Then $A \in A$ is strongly persistently $U$-free iff $LA$ is strongly persistently $V$-free.

**Proof.** (1)

By Proposition A.1, $\eta^R$ and $\varepsilon^L_U$ are isomorphisms, and $L$ is full. Assume that $B \in B$ is $V$-free over $L'X$, i.e. there is a $V$-universal arrow $\eta^V_{U,Y}: L'X \rightarrow VB$. By composition with the $R$-universal arrow $\eta^R_{X,Y}: X \rightarrow R'L'X$ we get an $R'V$-universal arrow $\eta^{RV}_{X,Y} = R'\eta^V_{U,Y} \circ \eta^R_{X,Y}: X \rightarrow R'VB$. By its universality, there is a morphism $g: B \rightarrow LRB$ with $R'Vg \circ \eta^{RV}_{X,Y} = U \eta^R_{RB} \circ \eta^{RV}_{X,Y}$. 

115
Now $R'\varepsilon_B \circ R'g \circ \eta_X^{R'v} = U R \varepsilon_B \circ U \eta_{RB} \circ \eta_X^{R'v} = \eta_X^{R'v}$. By universality of $\eta_X^{R'v}$, we get $\varepsilon_B^{R'} \circ g = \text{id}$, and thus also $\varepsilon_B^{R'} \circ g \circ \varepsilon_B^{R'} = \varepsilon_B^{R'}$. Since $L$ is full, $g \circ \varepsilon_B^{R'}$ is the $L$-image of an $A$-morphism. By co-universality of $\varepsilon_B^{R'}$, we get $g \circ \varepsilon_B^{R'} = \text{id}$. Thus, $g: B \to LRB$ is an isomorphism. But then, $\eta_X^U := \eta_X^{R'v}: X \to URB$ can be shown to be a $U$-universal arrow as follows: If $f : X \to UA$ is an $X$-morphism, by universality of $\eta_X^{R'v}$, there is a unique morphism $f^\#: B \to LA$ satisfying $R'f^\# \circ \eta_X^{R'v} = U \eta_A^L \circ f$. Now $(\eta_A^L)^{-1} \circ R(f^\# \circ g^{-1}) \circ \eta_{RB}$ is a morphism from $RB$ to $A$ with $U((\eta_A^L)^{-1} \circ R(f^\# \circ g^{-1}) \circ \eta_{RB}) \circ \eta_X^{R'v} = U(\eta_A^L)^{-1} \circ UR(f^\#) \circ UR(g^{-1}) \circ U \eta_{RB} \circ U \varepsilon_B \circ URg \circ \eta_X^{R'v} = U(\eta_A^L)^{-1} \circ UR(f^\#) \circ UR(g^{-1}) \circ U \varepsilon_B \circ URg \circ \eta_X^{R'v} = U(\eta_A^L)^{-1} \circ UR(f^\#) \circ \eta_X^{R'v} = U(\eta_A^L)^{-1} \circ U \eta_A^L \circ f = f$. Since $g$ is an isomorphism, uniqueness follows from that of $f^\#$. \( \square \)

Proof of (2), “$\Rightarrow$”:

Let $\eta_X^U : X \to UA$ be $U$-universal, and $\eta_A^R : A \to RLA$ be $R$-universal. Then

$$\eta_{L',X} := L'X \xrightarrow{L'\eta_X^U} L'UA \xrightarrow{L'U\eta_A^R} L'URLA = L'R'VLA \xrightarrow{\varepsilon_{VLA}^L} VLA$$

is a $V$-universal arrow; Let $f : L'X \to VB$ be an $X$-morphism. By co-universality of $\varepsilon_{VB}^L$, there is some unique $f^\#: L'X \to R'VB = URB$ with $\varepsilon_{VB}^L \circ Uf = f$. By $UR$-universality of $U\eta_A^R \circ \eta_X^U$, there is some unique $f^\#: LA \to B$ with $URf^\# \circ U\eta_A^R \circ \eta_X^U = f$. By also considering the commutativity of the square (due to naturality of $\varepsilon^L$), $f^\#$ is the unique morphism from $LA$ to $B$ with $Vf^\# \circ \eta_{L',X} = f$.  

116
Proof of (2), \(\Longleftrightarrow\):
Follows from (1) with \(B = LA\), since \(RLA \cong A\). \(\Box\)

Proof of (3): Follows from (1) with \(X\) such that \(L'X = VB\), where Proposition A.1 ensures that the unit constructed in the proof is an isomorphism, which leads to strongly persistent freeness by Proposition 2.10. \(\Box\)

Proof of (4):
Follows from (2) with \(X = UA\), since by assumption, \(LUA = VLA\), where \(\epsilon_{L'}^{V} = \epsilon_{L}^{V}\) and proposition A.1 ensure that the unit constructed in the proof is an isomorphism, which leads to strongly persistent freeness by Proposition 2.10. \(\Box\)

B Locally finitely presentable categories

**Definition B.1** \([2]\) An object \(K\) of a category \(\mathcal{K}\) is called **finitely presentable** provided that its hom-functor

\[ \text{hom}(K, -) : \mathcal{K} \to \text{Set} \]

preserves directed colimits.

For example, a set is finitely presentable in \(\text{Set}\) iff it is finite. A many-sorted algebra is finitely presentable in \(\text{Mod}(S, F)\) iff it can be presented by finitely many generators and finitely many equations in the usual algebraic sense.

**Definition B.2** \([2]\) A category \(\mathcal{K}\) is called **locally finitely presentable** provided that is cocomplete and has a set \(\mathcal{A}\) of finitely presentable objects such that every object is a directed colimit of objects from \(\mathcal{A}\).

**Proposition B.3** \([2]\) Each locally finitely presentable category is complete. \(\Box\)
Locally finitely presentable categories are categories which satisfy some completeness properties (completeness and cocompleteness) and some smallness properties (roughly speaking, they are categories of structures with operations of finite arities).

C Effective equivalence relations

Definition C.1 (cf. [2, 3.4(8)])

A relation on an object $K$ is a subobject of $K \times K$ (usually represented by a pair $e_1, e_2: E \to K$ of morphisms such that the morphism $\langle e_1, e_2 \rangle: E \to K \times K$ is a monomorphism). We call $e_1, e_2: E \to K$ an equivalence relation provided that it is

1. reflexive, i.e., the diagonal of $K \times K$ is contained in the subobject represented by $\langle e_1, e_2 \rangle$
2. symmetric, i.e., the monomorphisms $\langle e_1, e_2 \rangle$ and $\langle e_2, e_1 \rangle$ represent the same subobject
3. transitive, i.e., when we form the pullback of $e_2$ and $e_1$:

\[
\begin{array}{c}
\begin{array}{c}
E' \\
\downarrow e' \\
E \\
\downarrow e_1 \\
K
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
E \\
\downarrow e_2 \\
E \\
\downarrow e_1 \\
K
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
K \\
f
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
K' \\
f
\end{array}
\end{array}
\]

then the subobject represented by $\langle e_1 \circ e'_1, e_2 \circ e'_2 \rangle$ is contained in that represented by $\langle e_1, e_2 \rangle$.

Definition C.2 A kernel pair of a morphism $f: K \to K'$ in a category is a pair $e_1, e_2: E \to K$ such that

\[
\begin{array}{c}
\begin{array}{c}
E \\
\downarrow e_2 \\
K
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
E \\
\downarrow e_1 \\
K
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
K \\
f
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
K' \\
f
\end{array}
\end{array}
\]

is a pullback.
A category has effective equivalence relations, if every equivalence relation is
the kernel pair of some morphism.

References

    In C. Choppy, D. Bert, and P. Mosses, editors, Recent Trends in Algebraic
    Development Techniques, 14th International Workshop, WADT'99, Bonas,
    France, volume 1827 of Lecture Notes in Computer Science. Springer-Verlag,
    2000.
[4] E. Astesiano, M. Bidot, H. Kirchner, B. Krieg-Brückner, P. D. Mosses,
    D. Sannella, and A. Tarlecki. CASL: The common algebraic specification
    language. Theoretical Computer Science, this volume.
[8] H. Baumeister. Relations between Abstract Datatypes modeled as Abstract
    publication, 2000.
    1990.
    Addison-Wesley, 1989.
    Computer Science, this volume.
    two-valued model theory for partial algebras. Algebra Universalis, 15:306–358,
    1982.


120


123


