Available online at www.sciencedirect.com
SCIENCE DIRECT
Theoretical Computer Science 354 (2006) 42–71
Theoretical Computer Science
www.elsevier.com/locate/tcs

CSP-CASL—A new integration of process algebra and algebraic specification

Markus Roggenbach

Department of Computer Science, University of Wales Swansea, Singleton Park, Swansea SA2 8PP, UK

Abstract

CSP–CASL integrates the process algebra CSP [T. Hoare, Communicating Sequential Processes, Prentice-Hall, Englewood cliffs, NJ, 1985; A.W. Roscoe, The Theory and Practice of Concurrency, Prentice-Hall, Englewood cliffs, NJ, 1998] with the algebraic specification language CASL [P.D. Mosses (Ed.), CASL Reference Manual, Lecture Notes in Computer Science, Vol. 2960, Springer, Berlin, 2004; E. Astesiano, M. Bidoit, B. Krieg-Brückner, H. Kirchner, P.D. Mosses, D. Sannella, A. Tarlecki, CASL—the common algebraic specification language, Theoret. Comput. Sci. 286 (2002) 153–196]. Its novel aspects include the combination of denotational semantics in the process part and, in particular, loose semantics for the data types covering both concepts of partiality and sub-sorting. Technically, this integration involves the development of a new so-called data-logic formulated as an institution. This data-logic serves as a link between the institution underlying CASL and the alphabet of communications necessary for the CSP semantics. Besides being generic in the various denotational CSP semantics, this construction leads also to an appropriate notion of refinement with clear relations to both data refinement in CASL and process refinement in CSP.

Keywords: Algebraic specification; Institution; Process algebra; CASL; CSP

1. Introduction

Among the various frameworks for the description and modelling of reactive systems, process algebra plays a prominent rôle. It has proven to be suitable at the level of requirement specification, at the level of design specifications, and also for formal refinement proofs [4]. However, process algebra does not include development techniques for data types, although data is involved in *all* of its specifications. Usually, data types are treated as given and fixed. This can be overcome by adopting techniques from algebraic specification, which is devoted to the formal description and development of abstract as well as of concretely represented data types. Algebraic specification offers initial and loose semantics as commonly used approaches [2]. The initial approach is appropriate only if the design process of a data type has already been completed. It defines a particular realisation abstractly up to isomorphism. As our main focus is the development of data types, we deal here with loose semantics of data types, which describes a class of possible models, still to be refined.

Combining process algebra and algebraic specification to form a new specification technique aims at a fruitful integration of both development paradigms. An important example of such a combination is LOTOS [10]. Here, the modelling of data relies on initial algebra semantics because of its intimate relation with term rewriting. Its semantical

E-mail address: M.Roggenbach@swan.ac.uk.

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.11.007

definition is on the operational style. CCS–CASL [22] follows a similar approach, working with initial specifications in CASL, restricting the language to conditional equational logic without sub-sorting and partiality. A quite successful development is μCRL [8]. Here, data types have a loose semantics and are specified in equational logic with total functions. Again, the underlying semantics of the process algebraic part is operational.

Specifically, we aim to enable the combination of process algebraic specification of reactive behaviour and algebraic specification of data types at any required level of detail. This allows the specifier to develop a system in a problem driven approach, where data refinement and process refinement are chosen whenever appropriate for a certain design decision.

Seen from the process algebraic side, our language combination includes the traditional monomorphic data types like strings and different kinds of numbers. Furthermore, it also deals with polymorphic data types as for instance the class of all fields. Maybe even more importantly CSP–CASL's loose specification of data types naturally corresponds to requirement documents of distributed systems in industrial contexts. Such documents often only provide an overview of the data involved, while the presentation of further details for a specific type is delayed to separate design documents. CSP–CASL is able to match such a document structure by a library of specifications, where the informal design steps are mirrored in terms of a formal refinement relation [5].

Technically, the semantics of CSP–CASL is defined in terms of institutions and their representations. We motivate and design the specialised institution $FinCommSubPFOL^{=}$ as data-logic of the process part. This institution can be represented in $SubPCFOL^{=}$, the institution underlying CASL. Therefore, this whole construction allows us to use *full* CASL in order to specify data types, which then are used to describe reactive systems in CSP.

This article concentrates on how to define the semantics of an abstract core language of CSP–CASL, i.e. the translation of the concrete syntax of formulae and processes into an abstract one; questions concerning static semantics or customising the language by syntactical encodings are only briefly sketched.

We first present short overviews on the languages involved: CSP and CASL as the languages to be integrated, and a first sketch of how CORE–CSP–CASL, the core language of CSP–CASL, shall look like. We then discuss four fundamental problems concerning the integration of data and processes. Motivated by this study, we develop in Section 4 a data-logic for the process part which we formulate as institution $CommSubPFOL^{=}$. On the one side, this data-logic can be represented in the CASL institution, on the other side it can be transformed into an alphabet of communications in such a way, that the test on equality, element-relation, renaming by predicates on the alphabet can be characterised by CASL formulae. Based on this data-logic, we define in Section 5 the semantics of CORE–CSP–CASL and show that the stated integration problems are solved within this framework. Furthermore, we introduce the notion of CSP–CASL refinement in terms of data refinement in CASL and process refinement in CSP. Section 6 presents as concrete example the specification of a simple file system in full CSP–CASL. This specification exercise demonstrates how to deal with fixed points as well as data refinement and process refinement. Finally, we relate CSP–CASL with other approaches.

2. What are CSP, CASL, and CSP–CASL?

2.1. The process algebra CSP

The process algebra CSP [9,20] is defined over an alphabet of communications A. The syntax of basic CSP processes *Proc*, cf. Fig. 1, involves elements $a \in A$ as communications, subsets $X, Y \subseteq A$ as synchronisation sets in parallel operators or for hiding certain communications, uses binary relations $R \subseteq A \times A$ in order to describe renaming, and allows non-further specified formulae φ in its conditional. As usual in process algebra, CSP introduces recursion in the form of systems of process equations. Here, (parametrised) named processes are defined in terms of basic process expressions including also process names. In this case the grammar of Fig. 1 is extended by productions $Proc ::= ProcName \mid ProcName(x)$, where x is a variable over A.

CSP is a language with many semantics, different in their style as well as in their ability to distinguish between reactive behaviours [20]. There are operational, denotational and algebraic approaches, ranging from the simple finite traces model $\mathcal{T}$ to such complex semantics as the infinite traces model with failures/divergences $\mathcal{U}$. Like the CSP syntax, all these semantics take the alphabet of communications A as a parameter.

$$
\begin{array}{lll}
Proc ::= & SKIP & \%\% \text{ terminating process} \\
| & STOP & \%\% \text{ deadlock process} \\
| & a \rightarrow Proc & \%\% \text{ action prefix} \\
| & ?x : X \rightarrow Proc & \%\% \text{ prefix choice} \\
| & Proc \fatsemi Proc & \%\% \text{ sequential composition} \\
| & Proc \,\Box\, Proc & \%\% \text{ external choice} \\
| & Proc \sqcap Proc & \%\% \text{ internal choice} \\
| & Proc \,[\![X]\!]\, Proc & \%\% \text{ generalized parallel} \\
| & Proc \,[\![X \mid Y]\!]\, Proc & \%\% \text{ alphabetized parallel} \\
| & Proc \,||\, Proc & \%\% \text{ synchronous parallel} \\
| & Proc \,|||\, Proc & \%\% \text{ interleaving} \\
| & Proc \setminus X & \%\% \text{ hiding} \\
| & Proc[[R]] & \%\% \text{ relational renaming} \\
| & \textbf{if } \varphi \textbf{ then } Proc \textbf{ else } Proc & \%\% \text{ conditional}
\end{array}
$$

Fig. 1. Syntax of basic CSP processes.

2.2. The algebraic specification language CASL

The algebraic specification language CASL [16,1] is separated into various levels, including a level of *basic specifications* and a level of *structured specifications*. Basic specifications essentially list signature items and axioms in an unstructured way, thus determining a category of first order models. Structured specifications serve to combine such basic specifications into larger specifications in a hierarchical and modular fashion.

At the level of basic specifications, one can declare sorts (keyword **sort**), operations (keyword **op**), and predicates (keyword **pred**) with given input and result sorts. Sorts may be declared to be in a *sub-sorting* relation; if s is a sub-sort of t, then terms of type s may be used wherever terms of type t are expected. Sub-sorts may also be *defined* in the form $s = \{x : t \bullet \varphi\}$, with the effect that s consists of all elements of t that satisfy φ. Operations may be declared to be partial by using a modified function arrow $\rightarrow ?$. Using the symbols thus declared, one may then write axioms in first order logic. Moreover, one can specify data types (keyword **type**), given in terms of alternatives consisting of data constructors and, optionally, selectors, which may be declared to be **generated** or **free**. Generatedness amounts to an implicit higher order induction axiom and intuitively states that all elements of the data types are reachable by constructor terms ('no junk'); freeness additionally requires that all these constructor terms are distinct ('no confusion'). Basic CASL specifications denote the class of all algebras which fulfil the declared axioms, i.e. CASL has loose semantics.

At the level of structured specifications, one has features such as parametrised named specifications, unions of specifications (keyword **and**), extensions of specifications (keyword **then**), and renaming as well as hiding of symbols. Furthermore, it is possible to choose initial semantics (keyword **free**) instead of loose semantics.

2.3. The design of CORE–CSP–CASL

CSP–CASL is a comprehensive language involving named specifications, communication channels and a wide variety of CSP operators. For the moment, we concentrate on its semantically relevant part CORE–CSP–CASL. *Syntactically*, a CORE–CSP–CASL specification consists of a data part *Sp*, which is a structured CASL specification and a process part *P* written in CSP, but wherein CASL terms are used as communications, CASL sorts denote sets of communications, relational renaming is described by a binary CASL predicate, and CASL formulae occur in the conditional:

data *Sp* **process** *P* **end**

See the next section for many concrete instances of this scheme. In choosing the loose semantics of CASL, *semantically*, such a CORE–CSP–CASL specification is a family of process denotations for a CSP process, where each model of the data part *Sp* gives rise to one process denotation.

The definition of the language CORE–CSP–CASL is generic in the choice of a specific CSP semantics. For example, all denotational CSP models [1] mentioned in [20], or even the true concurrency semantics for TCSP of [3], based on event structures, are possible parameters. Whether a CSP-semantics can be used within our construction depends on the semantics' requirements concerning what we call here the *data type of communications*. This data type takes as

[1] Indeed, the construction seems to be possible also for the operational models. We focus here on the denotational ones.

values the alphabet of communications, but additionally provides certain test functions. In this respect, our construction provides as operations

- test on equality for arbitrary CASL terms (can two communications synchronise?),
- test on membership for a CASL term concerning a CASL sort (does a communication belong to a certain subset of the alphabet of communications?),
- test whether a binary predicate holds between two CASL terms (are the terms in a renaming relation?), and
- satisfaction of a CASL first order formula (is the formula of the conditional construct true?).

As indicated in this list, we will formulate these test functions solely in CASL. To this end, we use the institution $FinCommSubPFOL^{=}$ as a link between CASL (or, more precisely, the CASL institution) and the alphabet of communications A. This alphabet A is required by the various CSP denotational semantics to describe their respective semantic domains, e.g. in case of the trace-semantics the domain $\mathcal{T}$ of all prefixed closed, non-empty subsets of $A^{\checkmark *}$. Thus, we will be able to translate the tests over the alphabet of communications, which the denotational CSP semantics need, into CASL formulae.

The above listed, seemingly small set of test operations allows for all denotational semantics described in [20], namely trace-semantics, failure-divergence-semantics and stable-failure-semantics.

3. Four issues in integrating data and processes

The data types specified by algebraic specification consist of *many-sorted algebras*. The data type of communications required by the process algebraic semantics is a *one-sorted algebra*. Thus, in order to integrate data into processes, we need to turn a many-sorted algebra into one set of values such that the above described tests are closely connected with the original data type.

3.1. Many-sorted total algebras

There are two natural ways to define the alphabet of communications in terms of the carrier sets of a CASL model: union and disjoint union of all carrier sets. To illustrate the effect of both possibilities, consider the following CORE–CSP–CASL specification:

data	**sorts** S, T
	ops $c : S;\ d : T$
process	$c \to SKIP \parallel d \to SKIP$

Its data part, written in CASL, provides two constants c and d of type S and T, respectively. The process part, written in CSP with CASL terms denoting communications, combines two processes by the synchronous parallel operator, i.e. they have to agree on all actions.

The question is, may c and d synchronise or not? In all the various CSP semantics, c and d synchronise iff they are equal. Now consider two isomorphic CASL models $\mathcal{A}$ and $\mathcal{B}$ of the data part:

$$\mathcal{A}(S) = \{*\},\ \mathcal{A}(T) = \{+\}, \mathcal{A}(c) = *,\ \mathcal{A}(d) = +\ \mathcal{B}(S) = \mathcal{B}(T) = \{\sharp\}, \mathcal{B}(c) = \mathcal{B}(d) = \sharp.$$

Choosing the union of all carrier sets as alphabet has the effect, that c and d do not synchronise for algebra $\mathcal{A}$ while they synchronise for algebra $\mathcal{B}$. Thus, isomorphic algebras give rise to different behaviour. Therefore, we define the alphabet to be the disjoint union—with the consequence that c and d do not synchronise.

3.2. Sub-sorted total algebras

This decision raises a problem if we take CASL sub-sorting into account. Modifying the data part of our example such that S is a sub-sort of T, and stating that c and d are equal in all models, we would expect these two events to

synchronise:

data	**sorts**	$S < T$
	ops	$c : S;\ d : T$
	•	$c = d$
process	$c \to SKIP \parallel d \to SKIP$	

But in our current approach, this is not the case for any model. For instance, the derived communication alphabet $\{(S, \sharp), (T, \sharp)\}$ of algebra $\mathcal{B}$ provides two different elements as semantics of c and d, respectively. The solution is to define a suitable notion of equality on the alphabet in terms of an equivalence relation. In our simple example, we can choose the smallest equivalence relation $\sim$ with

$$a \in s^{\mathcal{M}},\ b \in t^{\mathcal{M}},\ em^{\mathcal{M}}_{\langle s\rangle,t}(a) = b \Longrightarrow (s, a) \sim (t, b).$$

In the general case, this will become more complex. In this definition, $\mathcal{M}$ is a model of the data part, s and t are arbitrary sort names, a and b are elements of the respective carrier sets, and $em_{\langle s\rangle,t}$ is the implicitly defined CASL embedding function from the carrier set of s into the carrier set of t. For the algebra $\mathcal{B}$ this construction yields the one element set $\{[(S, \sharp), (T, \sharp)]\}$ ensuring that c and d synchronise—as they do in all models of the data part.

3.3. Partial algebras

Up to now, we only considered defined terms. But what should be the semantics of an undefined term? In CASL, terms arise as parts of formulae, where the enclosing formula of an undefined term is evaluated to false. In CORE–CSP–CASL, however, terms are also part of processes. Thus, for this context a new interpretation is needed. Take for example the following CSP–CASL specification:

data	**sorts**	S, T
	ops	$f : S \to ?\ T$
	•	$\forall x : S \bullet\ \neg def\ f(x)$
process	$?x : S \to f(x) \to SKIP \lvert[T]\rvert\ ?y : T \to$ **if** $def\ y$ **then** P **else** Q	

One possibility to give it a semantics would be to prescribe a certain behaviour for the case that an undefined term arises. The natural choice would be [2]

$$t \to P := CHAOS,\ \text{if } \neg defined(t).$$

With this definition the above process part would become equivalent to

$$?x : S \to CHAOS.$$

Our solution, however, is to interpret undefined terms by an *extra communication* $\perp$, i.e. we consider the definedness of a term to be observable. This is natural, as CASL includes a definedness predicate $def\ t$, which holds iff the term t is defined. With this definition, the process part of the above specification becomes equivalent to

$$?x : S \to f(x) \to Q.$$

The main motivation behind this design decision is separation of concerns. Process algebra can be seen as a mechanism which takes a data type as its parameter and uses it in order to describe a certain reactive system. In this view, the occurrence of an undefined term indicates either an open design decision concerning the data type—i.e. an issue independent of the reactive behaviour of the system—or a non-adequate use of the data type within the process algebra—i.e. an 'interface problem' between the world of data types and reactive behaviours.

---

[2] Note that choosing *STOP* instead of *CHAOS* would violate elementary algebraic properties of CSP. Setting $t \to P := STOP$ if $\neg defined(t)$ has e.g. as consequence $b \to SKIP = a \to SKIP \ |||\ b \to SKIP \neq a \to b \to SKIP \ \Box\ b \to a \to SKIP = b \to STOP$ if $\neg defined(a), defined(b)$.

3.4. Sub-sorted partial algebras

In the presence of both sub-sorting and partiality, communications of type '$\perp$' need further consideration. When should they synchronise? This can be studied with the following example:

data **sorts** $A, B, C < S$
ops $a : A;\ b_1, b_2 : B;\ c : C;$
$f : A \to ?\ A;\ g : C \to ?\ C$
• $a = b_1 \bullet b_2 = c$
• $\forall x : A \bullet \neg def\, f(x) \bullet \forall x : C \bullet \neg def\, g(x)$
process $f(a) \to SKIP \parallel g(c) \to SKIP$

A tempting way of dealing with $\perp$ elements would be: '*undefined elements of different sorts with common supersort synchronise iff there exist defined elements in these sorts which can synchronise*'. However, this would destroy transitivity of synchronisation. Take for example, the model $\mathcal{A}$ of the data part with

$$\mathcal{A}(A) = \{*\},\ \mathcal{A}(B) = \{*, +\},\ \mathcal{A}(C) = \{+\},\ \mathcal{A}(S) = \{*, +\},$$
$$\mathcal{A}(a) = \{*\},\ \mathcal{A}(b_1) = \{*\},\ \mathcal{A}(b_2) = \{+\},\ \mathcal{A}(c) = \{+\}.$$

Following the above suggestions, $\perp_A$ and $\perp_B$ had to synchronise as $a = b_1$, $\perp_B$ and $\perp_C$ had to synchronise as $b_2 = c$, *but* $\perp_A$ and $\perp_C$ would *not* synchronise as no element of $\mathcal{A}(A)$ will ever synchronise with any element of $\mathcal{A}(C)$.

Therefore, we will define that undefined elements of different sorts are equivalent iff the sorts belong to the same connected component in the graph of sub-sort relations. Consequently, in the above CSP–CASL specification $f(a)$ and $g(c)$, which denote the undefined communications $\perp$ of the sorts A and C, synchronise.

4. The data-logic of the process part

We formalise the above proposal in terms of a data-logic of the process part. To this end, we first introduce the institutions $PFOL^{=}$ (partial first order logic with equality) and $SubPFOL^{=}$ (sub-sorted partial first order logic with equality) closely following [14]. Based on these notions, we then define a new institution $CommSubPFOL^{=}$ (communications described in sub-sorted partial first order logic with equality) which we use as data-logic of the process part of a CORE–CSP–CASL specification. In the next step we show that there is an institution representation from $FinCommSubPFOL^{=}$ to $SubPFOL^{=}$, where $FinCommSubPFOL^{=}$ is the restriction of $CommSubPFOL^{=}$ to signatures with only finitely many sorts. Finally, we present a construction how to obtain an alphabet of communications out of a $CommSubPFOL^{=}$ model. The interesting point is that those properties of this alphabet which are relevant for the denotational CSP semantics, can be studied already in terms of $CommSubPFOL^{=}$ formulae. For this construction, it is necessary to restrict sub-sorting to relations with 'local top elements'.

The algebraic specification language CASL has $SubPCFOL^{=}$ as underlying institution, where the 'C' stands for 'with sort generations constraints'. The here described institution $SubPFOL^{=}$ is obtained from $SubPCFOL^{=}$ by omitting sort generation constraints. Thus, we can use CASL to represent the data-logic $FinCommSubPFOL^{=}$ of the process part. For the definition, discussion, and examples of both concepts of institution and (simple) institution representation, we refer to [7,14].

4.1. The institution $PFOL^{=}$

Signatures: A *many-sorted signature* $\Sigma = (S, TF, PF, P)$ 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 $(w, s) \in S^* \times S$, and
- a family $P = (P_w)_{w \in S^*}$ of *predicate symbols*.

Given a function $f : A \to B$, let $f^* : A^* \to B^*$ be its component-wise extension to finite strings. Given two signatures $\Sigma = (S, TF, PF, P)$ and $\Sigma' = (S', TF', PF', P')$, a *many-sorted signature morphism* $\sigma : \Sigma \to \Sigma'$ consists of

- a map $\sigma^S : S \to S'$,
- a map $\sigma^F_{w,s} : TF_{w,s} \cup PF_{w,s} \to TF'_{\sigma^{S*}(w),\sigma^S(s)} \cup PF'_{\sigma^{S*}(w),\sigma^S(s)}$ preserving totality, for each $w \in S^*, s \in S$, and
- a map $\sigma^P : P_w \to P_{\sigma^{S*}(w)}$.

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

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

A *many-sorted Σ-homomorphism* $h : M \to N$ is a family of functions $h = (h_s : M_s \to N_s)_{s \in S}$ 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

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

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

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

Let $\sigma : \Sigma \to \Sigma'$ be a many-sorted signature morphism, M' be a Σ'-model. Then the *reduct* $M'_{|\sigma} =: M$ of M' is the Σ-model with

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

Given a many-sorted Σ'-homomorphism $h' : M' \to N'$, its *reduct* $h'_{|\sigma} : M'_{|\sigma} \to N'_{|\sigma}$ is defined by $(h'_{|\sigma})_s := h'_{\sigma^S(s)}$ for all $s \in S$.

Sentences: Given a many-sorted signature $\Sigma = (S, TF, PF, P)$, a *variable system* over Σ is an S-sorted, pairwise disjoint family of variables $X = (X_s)_{s \in S}$. The sets $T_\Sigma(X)_s$ of *many-sorted Σ-terms* of sort s, $s \in S$, with variables in X are the least sets satisfying

- $x \in T_\Sigma(X)_s$, if $x \in X_s$, and
- $f_{w,s}(t_1, \ldots, t_n) \in T_\Sigma(X)_s$, if $t_i \in T_\Sigma(X)_{s_i}$ $(i = 1 \ldots n)$, $f \in TF_{w,s} \cup PF_{w,s}$, $w = s_1 \ldots s_n$.

Given a *total variable valuation* $v : X \to M$, the *term evaluation* $v^\sharp : T_\Sigma(X) \to ?M$ is inductively defined by

- $v^\sharp_s(x) := v(x)$ for all $x \in X_s$ and all $s \in S$.
- $v^\sharp_s(f_{w,s}(t_1, \ldots, t_n)) :=$

$$\begin{cases} (f_{w,s})_M(v^\sharp_{s_1}(t_1), \ldots, v^\sharp_{s_n}(t_n)) & \text{if } v^\sharp_s(t_i) \text{ defined } (i = 1 \ldots n) \text{ and} \\ & \quad (f_{w,s})_M(v^\sharp_{s_1}(t_1), \ldots, v^\sharp_{s_n}(t_n)) \text{ defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

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

The set $AF_\Sigma(X)$ of *many-sorted atomic Σ-formulae 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 $t_i \in T_\Sigma(X)_{s_i}$, $p_w \in P_w$, $w = s_1 \ldots s_n \in S^*$,
(2) $t_1 \stackrel{e}{=} t_2 \in AF_\Sigma(X)$, if $t_1, t_2 \in T_\Sigma(X)_s, s \in S$ (existential equations),
(3) $t_1 = t_2 \in AF_\Sigma(X)$ if $t_1, t_2 \in T_\Sigma(X)_s, s \in S$ (strong equations),
(4) $def\, t \in AF_\Sigma(X)$, if $t \in T_\Sigma(X)$ (definedness assertions).

The set $FO_\Sigma(X)$ of *many-sorted first-order Σ-formulae 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) $\varphi \wedge \psi \in FO_\Sigma(X)$, if $\varphi, \psi \in FO_\Sigma(X)$,
(4) $\varphi \Rightarrow \psi \in FO_\Sigma(X)$, if $\varphi, \psi \in FO_\Sigma(X)$,
(5) $\forall x : s \bullet \varphi \in FO_\Sigma(X)$, if $\varphi \in FO_\Sigma(X \cup \{x : s\}), s \in S$,

A *many-sorted Σ-sentence* is a closed many-sorted first order formula over Σ. Concerning the definition of the translation of many-sorted Σ sentences along a many-sorted Σ-morphism we refer to [14].

Satisfaction relation: The satisfiability of a many sorted first-order formula $\varphi \in FO_\Sigma(X)$ relative to a valuation $v : X \to M$ is defined inductively over the structure of φ:

- $v \Vdash p_w(t_1, \ldots, t_n)$ iff $v^\sharp(t_i)$ is defined $(i = 1 \ldots n)$ and $(v^\sharp(t_1), \ldots, v^\sharp(t_n)) \in (p_w)_M$.
- $v \Vdash t_1 \stackrel{e}{=} t_2$ iff $v^\sharp(t_1)$ and $v^\sharp(t_2)$ are both defined and equal.
- $v \Vdash t_1 = t_2$ iff $v^\sharp(t_1)$ and $v^\sharp(t_2)$ are either both undefined, or both are defined and equal.
- $v \Vdash def\, t$ iff $v^\sharp(t)$ is defined.
- Not $v \Vdash F$.
- $v \Vdash \varphi \wedge \psi$ iff $v \Vdash \varphi$ and $v \Vdash \psi$.
- $v \Vdash \varphi \Rightarrow \psi$ iff $v \Vdash \varphi$ implies $v \Vdash \psi$.
- $v \Vdash \forall x : s \bullet \varphi$ iff for all valuations $\zeta : X \cup \{x : s\} \to M$ with $\zeta(y) = v(y)$ for $y \neq x : s$, $y \in X$, we have $\zeta \Vdash \varphi$.

$M \vDash \varphi$ holds for a many-sorted Σ-model and a many-sorted formula φ, iff $v \Vdash \varphi$ for all variable valuations v into M. [14] proves the satisfaction condition of $PFOL^=$.

4.2. The institution $SubPFOL^=$

Signatures: A *sub-sorted signature* $\Sigma = (S, TF, PF, P, \leqslant)$ consists of a many-sorted signature (S, TF, PF, P) together with a reflexive and transitive *sub-sort relation* $\leqslant_S \subseteq S \times S$. The relation $\leqslant_S$ extends point-wise to sequences of sorts. We drop the subscript S when it is obvious from the context. For a *sub-sorted signature* $\Sigma = (S, TF, PF, P, \leqslant)$ we define *overloading relations* $\sim_F$ and $\sim_P$ for function and predicate symbols, respectively. Let $f : w_1 \to s_1$, $f : w_2 \to s_2 \in TF \cup PF$. Then $f : w_1 \to s_1 \sim_F f : w_2 \to s_2$ iff there exist $w \in S^*, s \in S$ such that $w \leqslant w_1, w \leqslant w_2, s_1 \leqslant s$, and $s_2 \leqslant s$. Let $p : w_1, p : w_2 \in P$. Then $p : w_1 \sim_P p : w_2$ iff there exists $w \in S^*$ such that $w \leqslant w_1$ and $w \leqslant w_2$.

A *sub-sorted signature morphism* $\sigma : \Sigma \to \Sigma'$ is a many-sorted signature morphism that preserves the sub-sort relation and the overloading relations,[3] i.e. for σ holds:

p1 $s_1 \leqslant s_2$ implies $\sigma^S(s_1) \leqslant \sigma^S(s_2)$ for all $s_1, s_2 \in S$
p2 $f : w_1 \to s_1 \sim_F f : w_2 \to s_2$ implies $\sigma^F_{w_1,s_1}(f) = \sigma^F_{w_2,s_2}(f)$ for all $f \in TF \cup PF$
p3 $p : w_1 \sim_P p : w_2$ implies $\sigma^P_{w_1}(p) = \sigma^P_{w_2}(p)$ for all $p \in P$.

With each sub-sorted signature $\Sigma = (S, TF, PF, P, \leqslant)$ we associate a many-sorted signature $\hat{\Sigma} = (\hat{S}, \hat{TF}, \hat{PF}, \hat{P})$, which extends the underlying many-sorted signature (S, TF, PF, P) with

- a total *injection* function symbol $\mathtt{inj} : s \to s'$ for each pair of sorts $s \leqslant_S s'$,
- a partial *projection* function symbol $\mathtt{pr} : s' \to ?s$ for each pair of sorts $s \leqslant_S s'$, and
- an unary *membership* predicate symbol $\varepsilon^s_{s'} : s'$ for each pair of sorts $s \leqslant_S s'$.

Given a sub-sorted signature morphism $\sigma : \Sigma \to \Sigma'$, we can extend it to a many-sorted signature morphism $\hat{\sigma} : \hat{\Sigma} \to \hat{\Sigma}'$ by just mapping the injections, projections and memberships in $\hat{\Sigma}$ to the corresponding injections, projections and memberships in $\hat{\Sigma}'$.

[3] Note that, thanks to preservation of subsorting, the preservation of the overloading relations can be simplified.

Models: *Sub-sorted Σ-models* are many-sorted $\hat{\Sigma}$-models satisfying in *PFOL*$^{=}$ the following set of axioms $\hat{J}(\Sigma)$ (all variables are universally quantified):

(1) $\mathtt{inj}_{s,s}(x) \stackrel{e}{=} x$ for $s \in S$.
(2) $\mathtt{inj}_{s,s'}(x) \stackrel{e}{=} \mathtt{inj}_{s,s'}(y) \Rightarrow x \stackrel{e}{=} y$ for $s \leqslant s'$.
(3) $\mathtt{inj}_{s',s''}(\mathtt{inj}_{s,s'}(x)) \stackrel{e}{=} \mathtt{inj}_{s,s''}(x)$ for $s \leqslant s' \leqslant s''$.
(4) $\mathtt{pr}_{s',s}(\mathtt{inj}_{s,s'}(x)) \stackrel{e}{=} x$ for $s \leqslant s'$.
(5) $\mathtt{pr}_{s',s}(x) \stackrel{e}{=} \mathtt{pr}_{s',s}(y) \Rightarrow x \stackrel{e}{=} y$ for $s \leqslant s'$.
(6) $\varepsilon^{s}_{s'}(x) \Leftrightarrow \mathit{def}\, \mathtt{pr}_{s',s}(x)$ for $s \leqslant s'$.
(7) $\mathtt{inj}_{s',s}(f_{w',s'}(\mathtt{inj}_{s_1,s_1'}(x_1), \ldots, \mathtt{inj}_{s_n,s_n'}(x_n))) = \mathtt{inj}_{s'',s}(f_{w'',s''}(\mathtt{inj}_{s_1,s_1''}(x_1), \ldots, \mathtt{inj}_{s_n,s_n''}(x_n)))$ for $f_{w',s'} \sim_F f_{w'',s''}$, where $w \leqslant w', w''$, $w = s_1 \ldots s_n, w' = s_1' \ldots s_n', w'' = s_1'' \ldots s_n''$, $s', s'' \leqslant s$.
(8) $p_{w'}(\mathtt{inj}_{s_1,s_1'}(x_1), \ldots, \mathtt{inj}_{s_n,s_n'}(x_n)) \Leftrightarrow p_{w''}(\mathtt{inj}_{s_1,s_1''}(x_1), \ldots, \mathtt{inj}_{s_n,s_n''}(x_n))$ for $p_{w'} \sim_P p_{w''}$, where $w \leqslant w', w''$, $w = s_1 \ldots s_n, w' = s_1' \ldots s_n', w'' = s_1'' \ldots s_n''$.

Sub-sorted Σ-morphisms are many-sorted $\hat{\Sigma}$-morphisms.

Sentences: The *Sub-sorted formulae over Σ* are the many-sorted first order formulae over $\hat{\Sigma}$. A *sub-sorted Σ-sentence* is a many-sorted first order sentences over $\hat{\Sigma}$.

Satisfaction: The satisfaction relations $v \Vdash \varphi$ and $M \vDash \varphi$ are defined as in *PFOL*$^{=}$. [14] proves the satisfaction condition of *SubPFOL*$^{=}$.

4.3. The institution CommSubPFOL$^{=}$

The definition of the institution *CommSubPFOL*$^{=}$ provides the data-logic of the process part of a CSP–CASL specification. It can be viewed as a specialisation of the institution *SubPFOL*$^{=}$.

Concerning the set of formulae, the differences are that *CommSubPFOL*$^{=}$ allows equations $t \stackrel{e}{=} t'$ and $t = t'$, where t and t' are terms of arbitrary sorts. Furthermore, a new 'element relation' is introduced. To prove the satisfaction condition, these new types of formulae make it necessary to strengthen the definition of a signature morphisms. Concerning models, in *CommSubPFOL*$^{=}$ each carrier set includes an element $\perp$ to deal explicitly with undefinedness.

At certain points we indicate how our definitions simplify in the absence of true sub-sorting, indicating how a data-logic for partiality would look like. The here presented institution *CommSubPFOL*$^{=}$ deals with both, partiality and sub-sorting. The definition of data-logics, which cover none or only one of these two aspects, would result in a system of institutions with representations relating them with each other, with the here introduced *CommSubPFOL*$^{=}$, as well as with different CASL sub-institutions. Here, we refrain from this approach as our aim is to define an expressive combination of CSP and CASL.

Signatures: A *data-logic signature* is a sub-sorted signature $\Sigma = (S, TF, PF, P, \leqslant)$. A *data-logic signature morphism* $\sigma : \Sigma \to \Sigma'$ is a sub-sorted signature morphism that additionally reflects the sub-sort relation and does not extend the sub-sort relation, i.e. it is a many-sorted signature morphism which besides the three preservation conditions **p1**, **p2**, **p3** defined in Section 4.2 also fulfils

refl $\sigma^S(s_1) \leqslant_{S'} \sigma^S(s_2)$ implies $s_1 \leqslant_S s_2$ for all $s_1, s_2 \in S$ (reflection of the sub-sort relation) and
non-ext $\sigma^S(s_1) \leqslant_{S'} u' \wedge \sigma^S(s_2) \leqslant_{S'} u'$ implies that there exists a sort $u \in S$ with $\sigma^S(u) = u'$ for all $s_1, s_2 \in S$ and $u' \in S'$ (non-extension).

Lemma 1. *Data-logic signature morphisms compose*.

Proof. Let $\sigma_1 : \Sigma_1 \to \Sigma_2$ and $\sigma_2 : \Sigma_2 \to \Sigma_3$ be data-logic signature morphisms. As sub-sorted signatures and sub-sorted signature morphisms form a category, $\sigma := \sigma_2 \circ \sigma_1$ has the properties **p1**, **p2**, **p3**. The proof of **refl** is trivial. Concerning **non-ext**, let $s_1, s_2 \in S_1$, $u_3 \in S_3$ and $\sigma(s_1) \leqslant u_3$ as well as $\sigma(s_2) \leqslant u_3$. Thanks to non-extension of σ_2, there exists $u_2 \in S_2$ with $\sigma_2(u_2) = u_3$. Applying the reflection property of σ_2 yields $\sigma_1(s_1) \leqslant u_2$ as well as $\sigma_1(s_2) \leqslant u_2$. Thanks to non-extension of σ_1, there exists $u_1 \in S_2$ with $\sigma_1(u_1) = u_2$. Now $\sigma(u_1) = \sigma_2(\sigma_1(u_1)) = \sigma_2(u_2) = u_3$. □

Remark 2. Note that in the absence of true sub-sorting, i.e. $\leqslant_S = id_S$, $\leqslant_{S'} = id_{S'}$, the additional conditions **refl** and **non-ext** are equivalent to injectivity on sorts.

Models: A *data-logic Σ-model M* is the strict extension $M := ext(C)$ of an ordinary many-sorted model C over $\hat{\Sigma} = (\hat{S}, \hat{TF}, \hat{PF}, \hat{P})$ which satisfies in $PFOL^{=}$ the set of axioms $\hat{J}(\Sigma)$ defined in Section 4.2. Given such a $\hat{\Sigma}$-model C, its strict extension is defined by [4]

- $M_s = ext(C_s) := C_s \cup \{\bot\}$ for all $s \in \hat{S}$, where $\bot \notin C_s$ for all $s \in \hat{S}$,
- $(f_{w,s})_M(x_1, \ldots, x_n) =$

$$(f_{w,s})_{ext(C)}(x_1, \ldots, x_n) := \begin{cases} (f_{w,s})_C(x_1, \ldots, x_n) & \text{if } x_i \in C(s_i)(i = 1 \ldots n) \text{ and} \\ & (f_{w,s})_C(x_1, \ldots, x_n) \text{ is defined} \\ \bot & \text{otherwise} \end{cases}$$

for all f in $\hat{TF}_{w,s} \cup \hat{PF}_{w,s}$, and

- $(p_w)_M = (p_w)_{ext(C)} := (p_w)_C$ for all $p \in \hat{P}_w$.

This construction leads to a one-one correspondence between ordinary many-sorted models over $\hat{\Sigma}$ satisfying $\hat{J}(\Sigma)$ in $PFOL^{=}$ and Σ-models in $CommSubPFOL^{=}$: given a model C, its extension $ext(C) =: M$ is uniquely determined. Forgetting the strict extension results again in C. Concerning the properties defined in the set of axioms $\hat{J}(\Sigma)$, the extended models behave in the expected way:

Lemma 3. *In the extended model M the following hold*:

(1) $(\mathtt{inj}_{s,s})_M(x) = x$ *for* $x \in M_s, s \in S$.
(2) $(\mathtt{inj}_{s,s'})_M(x) = (\mathtt{inj}_{s,s'})_M(y) \Rightarrow x = y$ *for* $x \in M_s, y \in M_{s'}, s \leqslant s'$.
(3) $(\mathtt{inj}_{s',s''})_M((\mathtt{inj}_{s,s'})_M(x)) = (\mathtt{inj}_{s,s''})_M(x)$ *for* $x \in M_s, s \leqslant s' \leqslant s''$.
(4) $(\mathtt{pr}_{s',s})_M((\mathtt{inj}_{s,s'})_M(x)) = x$ *for* $x \in M_s, s \leqslant s'$.
(5) $(\mathtt{pr}_{s',s})_M(x) = (\mathtt{pr}_A s', s)_M(y) \neq \bot \Rightarrow x = y$ *for* $x, y \in M_{s'}, s \leqslant s'$.
(6) $(\varepsilon^s_{s'})_M(x) \Leftrightarrow (\mathtt{pr}_{s',s})_M(x) \neq \bot$ *for* $x \in M_s, s \leqslant s'$.
(7) $(\mathtt{inj}_{s',s})_M((f_{w',s'})_M((\mathtt{inj}_{s_1,s'_1})_M(x_1), \ldots, (\mathtt{inj}_{s_n,s'_n})_M(x_n))) = (\mathtt{inj}_{s'',s})_M((f_{w'',s''})_M((\mathtt{inj}_{s_1,s_1''})_M(x_1), \ldots, (\mathtt{inj}_{s_n,s_n''})_M(x_n)))$ *for* $x_i \in M_{s_i}$, $i = 1 \ldots n$, $f_{w',s'} \sim_F f_{w'',s''}$, *where* $w \leqslant w', w''$, $w = s_1 \ldots s_n, w' = s'_1 \ldots s'_n, w'' = s_1'' \ldots s_n''$, $s', s'' \leqslant s$.
(8) $(p_{w'})_M((\mathtt{inj}_{s_1,s'_1})_M(x_1), \ldots, (\mathtt{inj}_{s_n,s'_n})_M(x_n)) \Leftrightarrow (p_{w''})_M((\mathtt{inj}_{s_1,s_1''})_M(x_1), \ldots, (\mathtt{inj}_{s_n,s_n''})_M(x_n))$ *for* $p_{w'} \sim_P p_{w''}$, *where* $w \leqslant w', w''$, $w = s_1 \ldots s_n, w' = s'_1 \ldots s'_n, w'' = s_1'' \ldots s_n''$.

Proof. Simple case distinctions between $x = \bot$ and $x \neq \bot$. □

Data-logic Σ-morphisms are extended many-sorted $\hat{\Sigma}$-morphisms. Given a many-sorted morphism $\hat{h} : C \to C'$ between two many-sorted models C, C' over $\hat{\Sigma}$, which both satisfy $\hat{J}(\Sigma)$ in $PFOL^{=}$, then $ext(\hat{h}) =: h : M \to M'$ with

$$h_s(x) = ext(\hat{h}_s)(x) := \begin{cases} \hat{h}_s(x) & \text{if } x \in C(s) \\ \bot & \text{if } x = \bot \end{cases}$$

is a data-logic Σ-morphism between M and M', where $M = ext(C)$ and $M' = ext(C')$. As this extension is again uniquely determined, there is also a one-one correspondence between the many-sorted $\hat{\Sigma}$-morphisms and data-logic Σ-morphisms.

Lemma 4 (*Composition of data-logic Σ-morphisms*). *Let $h : M \to M'$ and $h' : M' \to M''$ be data-logic Σ-morphisms with underlying morphisms $\hat{h} : C \to C'$ and $\hat{h}' : C' \to C''$, respectively. Then*

$$h' \circ h = ext(\hat{h}' \circ \hat{h}).$$

Proof. Let $x = \bot \in M_s$. Then $(h' \circ h)_s(\bot) = h'_s(h_s(\bot)) = h'_s(\bot) = \bot = (ext(\hat{h}' \circ \hat{h}))(\bot)$. Let $x \neq \bot \in M_s$. Then $(h' \circ h)_s(x) = h'_s(h_s(x)) = h'_s(\hat{h}_s(x)) = \hat{h}'_s(\hat{h}_s(x)) = (\hat{h}'_s \circ \hat{h}_s)(x) = (ext(\hat{h}'_s \circ \hat{h}_s))(x)$. □

[4] In the model definition we can use the same $\bot$ symbol for all carrier sets. Later, the construction of the alphabet of communications will make some of them different from each other.

Reducts are defined as the extended $\hat{\Sigma}$-reducts. Let $\sigma : \Sigma \to \Sigma'$ be a data-logic signature morphism, let $\hat{\sigma} : \hat{\Sigma} \to \hat{\Sigma}'$ be the corresponding many-sorted signature morphism. Let M' be a Σ'-model with underlying $\hat{\Sigma}'$-model C'. Then in *SubPFOL*$^=$ the reduct $C = C'_{|\hat{\sigma}}$ of C' is given by

- $C_s = C'_{\hat{\sigma}(s)}$ for all $s \in \hat{S}$,
- $(f_{w,s})_C = (\hat{\sigma}^F_{w,s}(f))_{C'}$ for all $f \in (\hat{TF}_{w,s} \cup \hat{PF}_{w,s})$, and
- $(p_w)_C = (\hat{\sigma}^P_w(p))_{C'}$ for all $p \in \hat{P}_w$.

As *SubPFOL*$^=$ is an institution and every *CommSubPFOL*$^=$ signature morphism is also a *SubPFOL*$^=$ signature morphism, C is a $\hat{\Sigma}$-model in *SubPFOL*$^=$, i.e. it satisfies the set of axioms $\hat{J}(\hat{\Sigma})$. Thus defining the reduct as $M = M'_{|\hat{\sigma}} := ext(C)$ yields a Σ-model in *CommSubPFOL*$^=$.

Note that with this definition the models M and M' relate in the expected way. We have

- $M_s = M'_{\hat{\sigma}(s)}$ for all $s \in \hat{S}$,
- $(f_{w,s})_M = (\sigma^F_{w,s}(f))_{M'}$ for all $f \in (\hat{TF}_{w,s} \cup \hat{PF}_{w,s})$, and
- $(p_w)_M = (\sigma^P_w(p))_{M'}$ for all $p \in \hat{P}_w$.

Given a Σ'-morphism $h' : M'_1 \to M'_2$, there exists a unique underlying $\hat{\Sigma}'$-morphism $\hat{h}' : C'_1 \to C'_2$. Its reduct $\hat{h}'_{|\sigma} : C'_{1|\sigma} \to C'_{2|\sigma}$ is defined by

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

Again, as *SubPFOL*$^=$ is an institution and every *CommSubPFOL*$^=$ signature morphism is also a *SubPFOL*$^=$ signature morphism, $\hat{h}'_{|\sigma}$ is a $\hat{\Sigma}$-morphism in *SubPFOL*$^=$. Thus we know that $h'_{|\sigma} : M'_{1|\sigma} \to M'_{2|\sigma}$ with $h'_{|\sigma} := ext(\hat{h}'_{|\sigma})$ is a Σ-morphism.

Sentences: The sets $T_{\hat{\Sigma}}(X)_s$ of terms of sort $s \in S$ over Σ are the many sorted sets of *PFOL*$^=$ terms of sort $s \in S$ over $\hat{\Sigma} = (\hat{S}, \hat{TF}, \hat{PF}, \hat{P})$. Note that we use $\hat{\Sigma}$ as index of the term set over Σ.

Given a variable valuation $v : X \to M$, the *term valuation* $v^\sharp : T_{\hat{\Sigma}}(X) \to M$ is inductively defined by:

- $v^\sharp_s(x) := v(x)$ for all $x \in X_s$ and all $s \in S$.
- $v^\sharp_s(f_{w,s}(t_1, \ldots, t_n)) := (f_{w,s})_M(v^\sharp_{s_1}(t_1), \ldots, v^\sharp_{s_n}(t_n))$ for all $f \in TF_{w,s} \cup PF_{w,s}$, where $w = s_1 \ldots, s_n$, and $t_i \in T_\Sigma(X)_{s_i}$, for $i = 1, \ldots, n$.

Note that term evaluation in *CommSubPFOL*$^=$ is a *total* function thanks to the encoding of partiality in terms of the $\perp$ elements.

The set $AF_{\hat{\Sigma}}(X)$ of atomic Σ-formulae with variables in X is the least set satisfying the following rules:

(1) $p_w(t_1, \ldots, t_n) \in AF_{\hat{\Sigma}}(X)$, if $t_i \in T_{\hat{\Sigma}}(X)_{s_i}$, $p_w \in P_w$, $i = 1, \ldots, n$, $w = s_1 \ldots s_n \in S^*$,
(2) $t \stackrel{e}{=} t' \in AF_{\hat{\Sigma}}(X)$, if $t, t' \in T_{\hat{\Sigma}}(X)$ (existential equations),
(3) $t = t' \in AF_{\hat{\Sigma}}(X)$, if $t, t' \in T_{\hat{\Sigma}}(X)$ (strong equations),
(4) $def\, t \in AF_{\hat{\Sigma}}(X)$, if $t \in T_{\hat{\Sigma}}(X)$ (definedness assertions),
(5) t `in` $s' \in AF_{\hat{\Sigma}}(X)$, if $t \in T_{\hat{\Sigma}}(X)_s, s, s' \in S$ (element relation).

CommSubPFOL$^=$ extends the set of atomic formulae available in *PFOL*$^=$ in two ways: (1) equations can be formed by *any* pair of terms; (2) the formula 'element relation' allows *any* combination of sorts.

The set $FO_{\hat{\Sigma}}(X)$ of first-order Σ-formulae with variables in X is the least set satisfying the following rules:

(1) $AF_{\hat{\Sigma}}(X) \subseteq FO_{\hat{\Sigma}}(X)$,
(2) $F \in FO_{\hat{\Sigma}}(X)$ (read: false),
(3) $\varphi \wedge \psi \in FO_{\hat{\Sigma}}(X)$, if $\varphi, \psi \in FO_{\hat{\Sigma}}(X)$,
(4) $\varphi \Rightarrow \psi \in FO_{\hat{\Sigma}}(X)$, if $\varphi, \psi \in FO_{\hat{\Sigma}}(X)$,
(5) $\forall x : s \bullet \varphi \in FO_{\hat{\Sigma}}(X)$, if $\varphi \in FO_{\hat{\Sigma}}(X \cup \{x : s\}), s \in S$.

A *data-logic Σ-sentence* is a closed first order formula over Σ.

To define the translation of sentences along a data-logic signature morphism $\sigma : \Sigma \to \Sigma'$, first we deal with the translation of a variable system X along σ:

$$\sigma(X)_{s'} := \bigcup_{\sigma^S(s)=s'} X_s.$$

Then we define how to translate terms over X into terms over $\sigma(X)$ by a function $\zeta_{\sigma,X} : T_{\hat{\Sigma}}(X) \to T_{\hat{\Sigma}'}(\sigma(X))$:

- $(\zeta_{\sigma,X})_s(x : s) := x : \sigma^S(s)$ for all $x \in X_s$ and all $s \in S$.
- $(\zeta_{\sigma,X})_s(f_{w,s}(t_1, \ldots, t_n)) := \sigma^F_{w,s}(f_{w,s})((\zeta_{\sigma,X})_{s_1}(t_1), \ldots, (\zeta_{\sigma,X})_{s_n}(t_n))$ for all $f \in TF_{w,s} \cup PF_{w,s}$, where $w = s_1 \ldots s_n$, and $t_i \in T_{\hat{\Sigma}}(X)_{s_i}$, for $i = 1, \ldots, n$.

Finally, this translation is extended to formulae:

- $\sigma(t) := (\zeta_{\sigma,X})(t)$ if t is a $\hat{\Sigma}$-term in variables X,
- $\sigma(p_w(t_1, \ldots t_n)) := \sigma^p_w(p_w)(\sigma(t_1), \ldots, \sigma(t_n))$,
- $\sigma(t \stackrel{e}{=} t') := \sigma(t) \stackrel{e}{=} \sigma(t')$,
- $\sigma(t{=}t') := \sigma(t){=}\sigma(t')$,
- $\sigma(def\, t) := def\, \sigma(t)$,
- $\sigma(t \texttt{ in } s') := \sigma(t) \texttt{ in } \sigma^S(s')$,
- $\sigma(F) := F$,
- $\sigma(\varphi \wedge \psi) := \sigma(\varphi) \wedge \sigma(\psi)$,
- $\sigma(\varphi \Rightarrow \psi) := \sigma(\varphi) \Rightarrow \sigma(\psi)$,
- $\sigma(\forall x : s \bullet \varphi) := \forall x : \sigma^S(s) \bullet \sigma(\varphi)$.

Satisfaction relation: The satisfiability of a formula $\varphi \in FO_\Sigma(X)$ relative to a valuation $v : X \to M$ is inductively defined over the structure of φ:

- $v \Vdash p_w(t_1, \ldots t_n)$ iff $(v^\sharp(t_1), \ldots, v^\sharp(t_n)) \in (p_w)_M$.
- $v \Vdash t \stackrel{e}{=} t'$ iff
  - $v^\sharp_s(t) \neq \bot$, $v^\sharp_{s'}(t') \neq \bot$,
  - there exists $u \in S$ such that $s \leqslant u$ and $s' \leqslant u$, and
  - for all $u \in S$ with $s \leqslant u$ and $s' \leqslant u$ holds:
$$v^\sharp_u(\texttt{inj}_{(s,u)}(t)) = v^\sharp_u(\texttt{inj}_{(s',u)}(t')).$$
- $v \Vdash t = t'$ iff either
  - $v^\sharp_s(t) = \bot$, $v^\sharp_{s'}(t') = \bot$ and
  - there exists $u \in S$ such that $s \leqslant u$ and $s' \leqslant u$,

  or
  - $v^\sharp_s(t) \neq \bot$, $v^\sharp_{s'}(t') \neq \bot$,
  - there exists $u \in S$ such that $s \leqslant u$ and $s' \leqslant u$, and
  - for all $u \in S$ with $s \leqslant u$ and $s' \leqslant u$ holds:
$$v^\sharp_u(\texttt{inj}_{(s,u)}(t)) = v^\sharp_u(\texttt{inj}_{(s',u)}(t')).$$
- $v \Vdash def\, t$ iff $v^\sharp(t) \neq \bot$.
- $v \Vdash t \texttt{ in } s'$ iff there exists $a \in M_{s'}$ such that
  either
  - $v^\sharp_s(t) = a = \bot$, and
  - there exists $u \in S$ such that $s \leqslant u$ and $s' \leqslant u$,

  or
  - $v^\sharp_s(t) \neq \bot$, $a \neq \bot$,
  - there exists $u \in S$ such that $s \leqslant u$ and $s' \leqslant u$, and
  - for all $u \in S$ with $s \leqslant u$ and $s' \leqslant u$ holds:
$$v^\sharp_u(\texttt{inj}_{(s,u)}(t)) = (\texttt{inj}_{(s',u)})_M(a).$$

- Not $v \Vdash F$.
- $v \Vdash \varphi \wedge \psi$ iff $v \Vdash \varphi$ and $v \Vdash \psi$.
- $v \Vdash \varphi \Rightarrow \psi$ iff $v \Vdash \varphi$ implies $v \Vdash \psi$.
- $v \Vdash \forall x : s \bullet \varphi$ iff for all valuations $\zeta : X \cup \{x : s\} \to M$ with $\zeta(y) = v(y)$ for $y \neq x : s, y \in X$, and $\zeta(x : s) \neq \bot$ we have $\zeta \Vdash \varphi$.

Note that it is impossible to express t `in` s' by a combination of other formulae. The reason for this is that in a quantification $\forall x : s'$ the variable x ranges over those values of $M_{s'}$ only which are different from $\bot$.

In the absence of true sub-sorting, this definition of satisfaction for existential and strong equations directly captures the intuition we developed in Section 3.3 concerning partiality (we deliberately keep all parts of the original definitions):

- $v \Vdash^{\text{no-sub}} t \stackrel{e}{=} t'$ iff $v^\sharp_s(t) \neq \bot$, $v^\sharp_{s'}(t') \neq \bot$, $sort(t) = sort(t')$, $v^\sharp_s(t) = v^\sharp_s(t')$.
- $v \Vdash^{\text{no-sub}} t = t'$ iff either $v^\sharp_s(t) = \bot$, $v^\sharp_{s'}(t') = \bot$, $sort(t) = sort(t')$,
  or $v^\sharp_s(t) \neq \bot$, $v^\sharp_{s'}(t') \neq \bot$, $sort(t) = sort(t')$, $v^\sharp_s(t) = v^\sharp_x(t')$.

This also illustrates why in the absence of true sub-sorting it is sufficient for the desired satisfaction condition that data-logic signature morphisms are injective on the set of sorts.

Lemma 5. *Let $\sigma : \Sigma \to \Sigma'$ be signature morphism, M' be a Σ'-model, X be a variable system over Σ, and $v : \sigma(X) \to M'$ be a valuation. Define a valuation*

$$\bar{v} : \begin{cases} X & \to M'_{|\sigma}, \\ x : s & \mapsto v_{\sigma^S(s)}(x : \sigma^S(s)). \end{cases}$$

Then $\bar{v}^\sharp = v^\sharp \circ \zeta_{\sigma,X}$. Moreover, v and $\bar{v}$ are in a one-one correspondence.

Proof. Induction over the term structure:

Let $t = x,\ x \in X_s, s \in S$. Then $\bar{v}^\sharp_s(x : s) = \bar{v}(x : s) = v_{\sigma^S(s)}(x : \sigma^S(s)) = v^\sharp_{\sigma^S(s)}(x : \sigma^S(s)) = v^\sharp_{\sigma^S(s)}((\zeta_{\sigma,X})_s(x : s)) = (v^\sharp_{\sigma^S(s)} \circ \zeta_{\sigma,X})(x : s)$.

Let $t = f_{w,s}(t_1, \ldots, t_n)$. Then

$$\begin{aligned} \bar{v}^\sharp_s(f_{w,s}(t_1, \ldots, t_n)) &= (f_{w,s})_{M'_{|\sigma}}(\bar{v}^\sharp_{s_1}(t_1), \ldots, \bar{v}^\sharp_{s_n}(t_n)) \\ &= (\sigma^F_{w,s}(f_{w,s}))_{M'}((v^\sharp \circ \zeta_{\sigma,X})_{s_1}(t_1), \ldots, (v^\sharp \circ \zeta_{\sigma,X})_{s_n}(t_n)) \\ &= (v^\sharp)_{\sigma(s)}(\sigma^F_{w,s}(f_{w,s})((\zeta_{\sigma,X})_{s_1}(t_1), \ldots, (\zeta_{\sigma,X})_{s_n}(t_n))) \\ &= (v^\sharp)_{\sigma(s)}((\zeta_{\sigma,X})_s(f_{w,s}(t_1, \ldots, t_n))) \\ &= (v^\sharp \circ \zeta_{\sigma,X})_s(f_{w,s}(t_1, \ldots, t_n)). \qquad \square \end{aligned}$$

Theorem 6 (*Generalized satisfaction condition*). *Given a signature morphism $\sigma : \Sigma \to \Sigma'$, a Σ'-model M', a variable system X over Σ, and a formula $\varphi \in FO_\Sigma(X)$, we have*

$$v \Vdash \sigma(\varphi) \text{ iff } \bar{v} \Vdash \varphi$$

for all evaluations $v : \sigma(X) \to M'$, where $\bar{v}$ is defined as in Lemma 5.

Proof. Induction on the structure of φ. We demonstrate only the interesting case of existential equations. Let t_1, t_2 be terms with $sort(t_1) = s_1$ and $sort(t_2) = s_2$. We claim that the following are equivalent:

(1) for all $u' \in S'$ with $\sigma(s_1), \sigma(s_2) \leqslant u'$ holds: $v^\sharp(\mathtt{inj}_{\sigma(s_1),u'}(\sigma(t_1))) = v^\sharp(\mathtt{inj}_{\sigma(s_2),u'}(\sigma(t_2)))$

(2) for all $u \in S$ with $s_1, s_2 \leqslant u$ holds: $\bar{v}^\sharp(\mathtt{inj}_{s_1,u}(t_1)) = \bar{v}^\sharp(\mathtt{inj}_{s_2,u}(t_2))$

"⇒" Let $u \in S$ with $s_1, s_2 \leqslant u$. Thanks to **p1** this implies $\sigma(s_1), \sigma(s_2) \leqslant \sigma(u)$. Thus, the condition of (1) is true and we obtain

$$v^\sharp(\mathtt{inj}_{\sigma(s_1),\sigma(u)}(\sigma(t_1))) = v^\sharp(\mathtt{inj}_{\sigma(s_2),\sigma(u)}(\sigma(t_2))).$$

Applying Lemma 5 yields (2).

"⇐" Let $u' \in S'$ with $\sigma(s_1), \sigma(s_2) \leqslant u'$. Thanks to **non-ext** there exists $u \in S$ with $\sigma^S(u) = u'$. Applying **refl** yields $s_1, s_2 \leqslant u$. Thus, the condition of (2) is true and we obtain

$$\bar{v}^\sharp(\mathtt{inj}_{s_1,u}(t_1)) = \bar{v}^\sharp(\mathtt{inj}_{s_2,u}(t_2))$$

Applying Lemma 5 yields (1).

With this result, we can establish the equivalence:

$v \Vdash \sigma(t_1 \stackrel{e}{=} t_2)$
iff $v \Vdash \sigma(t_1) \stackrel{e}{=} \sigma(t_2)$
iff (1) $v^\sharp_{\sigma(s_1)}(\sigma(t_1)) \neq \bot$, $v^\sharp_{\sigma(s_2)}(\sigma(t_2)) \neq \bot$,
(2) there exists $u' \in S'$ such that $\sigma(s_1) \leqslant u'$ and $\sigma(s_2) \leqslant u'$, and
(3) for all $u' \in S'$ with $\sigma(s_1) \leqslant u'$ and $\sigma(s_2) \leqslant u'$ holds: $v^\sharp_{u'}(\mathtt{inj}_{(\sigma(s_1),u')}(\sigma(t_1))) = v^\sharp_{u'}(\mathtt{inj}_{(\sigma(s_2),u')}(\sigma(t_2)))$
iff (1) $\bar{v}^\sharp_{s_1}(t_1) \neq \bot$, $\bar{v}^\sharp_{s_2}(t_2) \neq \bot$,
(2) there exists $u \in S$ such that $s_1 \leqslant u'$ and $s_2 \leqslant u$, and
(3) for all $u \in S$ with $s_1 \leqslant u$ and $s_2 \leqslant u$ holds: $\bar{v}^\sharp(\mathtt{inj}_{s_1,u}(t_1)) = \bar{v}^\sharp(\mathtt{inj}_{s_2,u}(t_2))$
iff $\bar{v} \Vdash t_1 \stackrel{e}{=} t_2$. □

The satisfaction condition is a consequence of Theorem 6: *CommSubPFOL*$^=$ is an institution.

4.4. Representing *FinCommSubPFOL*$^=$ in *SubPFOL*$^=$

FinCommSubPFOL$^=$ restricts the institution *CommSubPFOL*$^=$ to signatures with only finitely many sorts. This restriction is necessary, as the translation of existential and strong equations in *SubPFOL*$^=$ yields a conjunction over all sub-sort relations within the signature. More formally, we define a simple institution representation $\mu = (\Phi, \alpha, \beta)$ (see [7,14] for definition and discussion of this concept) as follows:

The functor Φ is the embedding of data-logic signatures with finite sort sets into sub-sorted signatures.

The translation α of *FinCommSubPFOL*$^=$ formulae into *SubPFOL*$^=$ is inductively defined by

- $\alpha(p_w(t_1, \ldots, t_n)) := p_w(t_1, \ldots, t_n)$,
- If there exists no $u \in S$ with $s_1, s_2 \leqslant u$ we define $\alpha(t_1 \stackrel{e}{=} t_2) := F$; if there exists an $u \in S$ with $s_1, s_2 \leqslant u$

$$\alpha(t_1 \stackrel{e}{=} t_2) := def\, t_1 \wedge def\, t_2 \wedge \bigwedge_{u \geqslant s_1,s_2} \mathtt{inj}_{s_1,u}(t_1) = \mathtt{inj}_{s_2,u}(t_2),$$

  where $sort(t_1) = s_1$, $sort(t_2) = s_2$.
- If there exists no $u \in S$ with $s_1, s_2 \leqslant u$ we define $\alpha(t_1 = t_2) := F$; if there exists an $u \in S$ with $s_1, s_2 \leqslant u$

$$\alpha(t_1 = t_2) := (\neg def\,(t_1) \wedge \neg def\,(t_2)) \vee (def\, t_1 \wedge def\, t_2 \wedge (\bigwedge_{u \geqslant s_1,s_2} \mathtt{inj}_{s_1,u}(t_1) = \mathtt{inj}_{s_2,u}(t_2))),$$

  where $sort(t_1) = s_1$, $sort(t_2) = s_2$.
- $\alpha(def\, t) := def\, t$,
- Let t be a term of sort s. If there exists no $u \in S$ with $s, s' \leqslant u$ we define $\alpha(t\ \mathtt{in}\ s') := F$; if there exists $u \in S$ with $s, s' \leqslant u$

$$\alpha(t\ \mathtt{in}\ s') := \neg def\,(t) \vee (def\, t \wedge \exists x : s' \bullet (\bigwedge_{u \geqslant s,s'} \mathtt{inj}_{s,u}(t) = \mathtt{inj}_{s',u}(x))),$$

- $\alpha(F) := F$

- $\alpha(\varphi \wedge \psi) := \alpha(\varphi) \wedge \alpha(\psi)$,
- $\alpha(\varphi \Rightarrow \psi) := \alpha(\varphi) \Rightarrow \alpha(\psi)$,
- $\alpha(\forall x : s \bullet \varphi) := \forall x : s \bullet \alpha(\varphi)$.

In the above definition, we use the common abbreviations $\neg\varphi$ for $\varphi \Rightarrow F$, $\varphi \vee \psi$ for $\neg(\neg\varphi \wedge \neg\psi)$, T for $\neg F$, and $\exists x : s \bullet \varphi$ for $\neg \forall x : s \bullet \neg\varphi$.

The translation β of *SubPFOL*$^=$ models into *FinCommSubPFOL*$^=$ models is defined as the strict extension *ext* introduced in Section 4.3. Note that this model translation β deals with *SubPFOL*$^=$ models over those signatures only which are in the co-domain of Φ, i.e. with models over *SubPFOL*$^=$ signatures with finitely many sorts only.

To prove the representation condition, we need to introduce *partial evaluations* $v : X \to ?M$ in *SubPFOL*$^=$. This is necessary as an evaluation in *FinCommSubPFOL*$^=$ might assign $\bot$ to a variable. The evaluation $v^\sharp$ of terms and the satisfaction of formulae is defined as above (see Section 4.1), with two exceptions:

- $v^\sharp_s(x) := \begin{cases} v(x) & \text{if } v(x) \text{ is defined} \\ \text{undefined} & \text{otherwise} \end{cases}$
- $v \Vdash \forall x : s \bullet \varphi$ iff for all valuations $\zeta : X \cup \{x : s\} \to M$ with $\zeta(y) = v(y)$ for $y \neq x : s$, $y \in X$, *that are defined on* $x : s$, we have $\zeta \Vdash \varphi$.

Now, terms in *SubPFOL*$^=$ and *FinCommSubPFOL*$^=$ are related as follows:

Lemma 7. *Let $\rho : X \to ?C$ be a partial evaluation in SubPFOL$^=$. Define a total evaluation $v : X \to \beta(C) =: M$ in FinCommSubPFOL$^=$ by*

$$v_s(x) := \begin{cases} \rho_s(x) & \textit{if } \rho(x) \textit{ is defined} \\ \bot & \textit{otherwise} \end{cases}$$

Then the following holds for all $t \in T_{\hat{\Sigma}}(X)_s$:

(1) *If $\rho^\sharp_s(t)$ is defined, then $\rho^\sharp_s(t) = v^\sharp_s(t)$ and $v^\sharp_s(t) \neq \bot$.*
(2) *If $v^\sharp_s(t) \neq \bot$, then $\rho^\sharp_s(t) = v^\sharp_s(t)$ and $\rho^\sharp_s(t)$ is defined.*
(3) *$v^\sharp_s(t) = \bot$ iff $\rho^\sharp_s(t)$ is undefined.*

Proof. Induction on terms. □

Note that there is a one-one correspondence between the partial evaluations ρ and the total evaluations v.

Theorem 8 (*Generalized representation condition*). *With the notation of Lemma* 7 *the following holds*: $\rho \Vdash \alpha(\varphi) \Leftrightarrow v \Vdash \varphi$.

Proof. By induction on the structure of φ. We prove here the interesting cases of existential equations and of quantification.

$\underline{t_1 \stackrel{e}{=} t_2}$:

Let t_1 and t_2 have no common super-sort. Then $\rho \Vdash \alpha(t_1 \stackrel{e}{=} t_2)$ iff $\rho \Vdash F$ iff $v \Vdash t_1 \stackrel{e}{=} t_2$.

Let t_1 and t_2 have common super-sorts $u_1, \ldots, u_k$, $sort(t_1) = s_1$ and $sort(t_2) = s_2$. Then

$\rho \Vdash \alpha(t_1 \stackrel{e}{=} t_2)$

iff $\rho \Vdash def\, t_1 \wedge def\, t_2 \wedge \bigwedge_{u \in \{u_1,\ldots,u_k\}} \mathtt{inj}_{s_1,u}(t_1) = \mathtt{inj}_{s_2,u}(t_2)$

iff (1) $\rho^\sharp_{s_1}(t_1)$ defined,

(2) $\rho^\sharp_{s_2}(t_2)$ defined, and

(3) for all $u \in \{u_1, \ldots, u_k\}$: $\rho^\sharp_u(\mathtt{inj}_{s_1,u}(t_1)) = \rho^\sharp_u(\mathtt{inj}_{s_2,u}(t_2))$

iff (1) $v^\sharp_{s_1}(t_1) \neq \bot$,

(2) $v^\sharp_{s_2}(t_2) \neq \bot$, and

(3) for all $u \in \{u_1, \ldots, u_k\}$: $v^\sharp_u(\mathtt{inj}_{s_1,u}(t_1)) = v^\sharp_u(\mathtt{inj}_{s_2,u}(t_2))$

iff $v \Vdash t_1 \stackrel{e}{=} t_2$.

$\underline{\forall x : s \bullet \varphi :}$

$\rho \Vdash \alpha(\forall x : s \bullet \varphi)$

iff $\rho \Vdash \forall x : s \bullet \alpha(\varphi)$

iff for all valuations $\zeta : X \cup \{x : s\} \to M$ with $\zeta(y) = \rho(y)$ for $y \neq x : s$, $y \in X$, that are defined on $x : s$, we have $\zeta \Vdash \alpha(\varphi)$

iff for all valuations $\eta : X \cup \{x : s\} \to M$ with $\eta(y) = \begin{cases} \zeta(x) & \text{if } \zeta(x) \text{ is defined} \\ \bot & \text{otherwise} \end{cases}$

for $y \neq x : s$, $y \in X$, such that $\eta(x : s) \neq \bot$, we have $\eta \Vdash \varphi$.

iff $v \Vdash \forall x : s \bullet \varphi$. $\square$

Remark 9 (*Representing the subset of 'CASL-formulae'*). Restricting the set of *CommSubPFOL*$^=$ formulae to those which follow the syntax production rules of *SubPFOL*$^=$ allows us to choose the translation α of *FinCommSubPFOL*$^=$ formulae into *SubPFOL*$^=$ formulae as identity:

- For predicates, α is already the identity.
- For existential equations, the restriction to the syntax production rules of *SubPFOL*$^=$ means that both terms t_1, t_2 have to be of the same sort s. Thus, we only have to deal with the second case of the definition of α. Here, $\bigwedge_{u \geqslant s} \mathtt{inj}_{s,u}(t_1) = \mathtt{inj}_{s,u}(t_2)$ is equivalent to $t_1 = t_2$, thanks to (1) of $\hat{J}(\hat{\Sigma})$. With this result we may replace the remaining $def\, t_1 \wedge def\, t_2 \wedge t_1 = t_2$ by $t_1 \stackrel{e}{=} t_2$.
- For strong equations, the restriction to these syntax production rules means that both terms t_1, t_2 have to be of the same sort s. With the same arguments as for existential equations we can define α as identity.
- For definedness, α is already the identity.
- If α is the identity for the atomic formulae, then it is so for *FALSE*, conjunction, implication, and quantification.

Thus, we can study the satisfaction of closed formulae within this subset directly within *SubPFOL*$^=$.

4.5. An alphabet of communications

Given a model M over a data-logic signature $\Sigma = (S, TF, PF, P, \leqslant)$, what is the corresponding alphabet A of communications? Our examples of Section 3 indicate what to do: take the disjoint sum of all carrier sets and model the additional equalities between terms as an equivalence relation $\sim$. Unfortunately, the notion of strong equality defined within *CommSubPFOL*$^=$ fails to be transitive:

Example 10. Let $S := \{s, s', s'', u, u'\}$ be a set of sorts, and $\leqslant$ the reflexive and transitive closure of $s, s' \leqslant u, s', s'' \leqslant u'$. Let t, t', t'' be terms of sorts s, s', s'', respectively, let M be a model. Then even with $M \models t = t'$ and $M \models t' = t''$ we have $M \not\models t = t''$ as s and s'' have no common super-sort.

Thus, we need a further restriction to *CommSubPFOL*$^=$. A *signature with local top elements* is a data-logic signature $\Sigma = (S, TF, PF, P, \leqslant)$, where for all $u, u', s \in S$ the following holds: if $u, u' \geqslant s$ then there exists $t \in S$ with $t \geqslant u, u'$.

Relatively to a model M for a signature with top elements, we define an alphabet of communications

$$A(M) := \left(\biguplus_{s \in S} M_s \right)_{/\sim}$$

where $(s, x) \sim (s', x')$ iff either

- $x = x' = \bot$ and
- there exists $u \in S$ such that $s \leqslant u$ and $s' \leqslant u$,

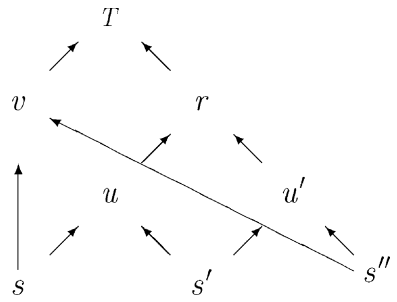

Fig. 2. Sort relations for the equivalence proof.

or

- $x \neq \bot, x' \neq \bot$,
- there exists $u \in S$ such that $s \leqslant u$ and $s' \leqslant u$, and
- for all $u \in S$ with $s \leqslant u$ and $s' \leqslant u$ the following holds:

$$(\mathtt{inj}_{(s,u)})_M(x) = \mathtt{inj}_{(s',u)_M}(x')$$

for $s, s' \in S$, $x \in M_s, x' \in M_{s'}$.

Lemma 11. *In CommSubPFOL$^=$ restricted to signatures with local top elements the following holds*:

(1) *Existential and strong equality are transitive*.
(2) *The relation $\sim$ is an equivalence relation for any model M*.

Proof. (1) Let t, t', t'' be terms of sorts s, s', s'', respectively. Let $v : X \to M$ be an evaluation in data-logic. Let $v \Vdash t \stackrel{e}{=} t'$, $v \Vdash t' \stackrel{e}{=} t''$.

Then $v_s^\sharp(t) \neq \bot$, $v_{s'}^\sharp(t') \neq \bot$, and $v_{s''}^\sharp(t'') \neq \bot$. Furthermore, there exist sorts u, u' such that $s, s' \leqslant u$ and $s', s'' \leqslant u'$. As the signature has local top elements and $s' \leqslant u$ as well as $s' \leqslant u'$, there exists a sort r with $u, u' \leqslant r$ and thus $s, s'' \leqslant r$.

Now let v be a sort with $s, s'' \leqslant v$. As also $s, s'' \leqslant r$, there exists a sort T with $v, t \leqslant T$. Fig. 2 summarises these sort relations. Using the equations in $\hat{J}(\Sigma)$ and the consequences of $v \Vdash t \stackrel{e}{=} t'$, $v \Vdash t' \stackrel{e}{=} t''$, we can prove the equality of t and t'' embedded in the super-sort v:

$$\begin{aligned}
& v_v^\sharp(\mathtt{inj}_{s,v}(t)) \\
= \; & v_v^\sharp(\mathtt{pr}_{T,v}(\mathtt{inj}_{v,T}(\mathtt{inj}_{s,v}(t)))) \\
= \; & v_v^\sharp(\mathtt{pr}_{T,v}(\mathtt{inj}_{r,T}(\mathtt{inj}_{u,r}(\mathtt{inj}_{s,u}(t))))) \\
= \; & v_v^\sharp(\mathtt{pr}_{T,v}(\mathtt{inj}_{r,T}(\mathtt{inj}_{u,r}(\mathtt{inj}_{s',u}(t'))))) \\
= \; & v_v^\sharp(\mathtt{pr}_{T,v}(\mathtt{inj}_{r,T}(\mathtt{inj}_{u',r}(\mathtt{inj}_{s',u'}(t'))))) \\
= \; & v_v^\sharp(\mathtt{pr}_{T,v}(\mathtt{inj}_{r,T}(\mathtt{inj}_{u',r}(\mathtt{inj}_{s'',u'}(t'')))))) \\
= \; & v_v^\sharp(\mathtt{pr}_{T,v}(\mathtt{inj}_{s'',T}(t''))) \\
= \; & v_v^\sharp(\mathtt{pr}_{T,v}(\mathtt{inj}_{v,T}(\mathtt{inj}_{s'',v}(t')))) \\
= \; & v_v^\sharp(\mathtt{inj}_{s'',v}(t'')).
\end{aligned}$$

This shows the transitivity of existential equations.

To prove the transitivity of strong equations, it remains to consider the situation $v \Vdash t{=}t'$, $v \Vdash t'{=}t''$ where $v_s^\sharp(t) = v_{s'}^\sharp(t') = v_{s''}^\sharp(t'') = \bot$. Here, we know that there exist sorts u, u' with $s, s' \leqslant u$ and $s', s'' \leqslant u'$ and therefore a sort $r \geqslant u, u'$. As $s, s'' \leqslant r$, we obtain $v \Vdash t{=}t''$.

(2) Reflexivity and symmetry are trivial. Transitivity is analogue to (1), as $\sim$ essentially uses the same definition as strong equality. □

Let M be a model in *CommSubPFOL*$^{=}$ restricted to signatures with local top elements, let A be the corresponding alphabet of communications. Then we define

(1) A family of mappings $(\mathtt{emb}_s)_{s\in S}$ by

$$\mathtt{emb}_s : \begin{cases} M_s & \to A \\ a & \mapsto (s,a)_{/\sim} \end{cases}$$

(2) A predicate $\bar{p}_{s,s'}$ on $A \times A$ for any binary predicate symbol $p_{s,s'} \in P, s, s' \in S$, by

$$\bar{p}_{s,s'} := \{(\mathtt{emb}_s(a), \mathtt{emb}_{s'}(b)) \mid (a,b) \in (p_{s,s'})_M\}$$

These notions yield the following relations between M and tests on A:

Theorem 12 (*Relation between logic and alphabet*). *Let M be a model in CommSubPFOL$^{=}$ restricted to signatures with local top elements, let $v : X \to M$ be a variable valuation, t, t' terms of sorts s, s', respectively, and $p_{s,s'}$ be a binary predicate symbol. Then*

(1) $v \Vdash t = t' \Leftrightarrow \mathtt{emb}_s(v^\sharp_s(t)) = \mathtt{emb}_{s'}(v^\sharp_{s'}(t'))$
(2) $v \Vdash t \ \mathtt{in}\ s' \Leftrightarrow \mathtt{emb}_s(v^\sharp_s(t)) \in \mathtt{emb}_{s'}(M_{s'})$
(3) $v \Vdash p_{s,s'}(t,t') \Leftrightarrow (\mathtt{emb}_s(v^\sharp_s(t)), \mathtt{emb}_{s'}(v^\sharp_{s'}(t'))) \in \bar{p}_{s,s'}$

Proof.

(1) $v \Vdash t = t'$ iff $(s, v^\sharp_s(t)) \sim (s', v^\sharp_{s'}(t'))$ iff $[v^\sharp_s(t)] = [v^\sharp_{s'}(t')]$ iff $\mathtt{emb}_s(v^\sharp_s(t)) = \mathtt{emb}_{s'}(v^\sharp_{s'}(t'))$.
(2) $v \Vdash t \ \mathtt{in}\ s'$
iff there exists $a \in M_{s'}$ with either
- $a = \bot = v^\sharp_s(t)$ and there exists $u \in S$ with $s, s' \leqslant u$ or
- $a \neq \bot$, $v^\sharp_s(t) \neq \bot$, there exists $u \in S$ with $s, s' \leqslant u$ and for all $v \in S$ with $s, s' \leqslant v$: $(\mathtt{inj}_{s',v})_M(a) = v^\sharp_v(\mathtt{inj}_{s,v}(t))$.

iff $(s, v^\sharp_s(t)) \sim (s', a)$
iff $\mathtt{emb}_s(v^\sharp_s(t)) = \mathtt{emb}_{s'}(a) \in \mathtt{emb}_{s'}(M_{s'})$.
(3) Let $(\mathtt{emb}_s(v^\sharp_s(t)), \mathtt{emb}_{s'}(v^\sharp_{s'}(t'))) \in \bar{p}_{s,s'}$. Then there exist $(a,b) \in (p_{s,s'})_M$ such that $(s,a) \sim \mathtt{emb}_s(v^\sharp_s(t))$, $(s',b) \sim \mathtt{emb}_{s'}(v^\sharp_{s'}(t'))$. As predicates never hold for $\bot$, this has as a consequence: $v^\sharp_s(t) \neq \bot$ and $v^\sharp_{s'}(t') \neq \bot$. Choosing s as a common super-sort of s as sort of a and s as sort of t, we obtain $a = v^\sharp_s(t)$. In the same way, we may conclude $b = v^\sharp_{s'}(t')$ and have finally $v \Vdash p_{s,s'}(t,t')$.
Let $v \Vdash p_{s,s'}(t,t')$. Then $(v^\sharp_s(t), v^\sharp_{s'}(t')) \in (p_{s,s'})_M$ and thus $(\mathtt{emb}_s(v^\sharp_s(t)), \mathtt{emb}_{s'}(v^\sharp_{s'}(t'))) \in \bar{p}_{s',s''}$. □

5. Core–CSP–CASL semantics

We now use the above described construction of a data type of communications from a *SubPFOL*$^{=}$ model over a signature Σ to define the semantics of a CORE–CSP–CASL specification

data *Sp* **process** *P* **end**

Our construction of Section 4 involves two conditions:

(1) The signature Σ needs to be finite (necessary in the representation of *CommSubPFOL*$^{=}$) – this holds for any specification written in CASL.
(2) Sub-sorting is restricted to sub-sort relations which have local top elements (necessary for the transitivity of strong equality).

Thus, *Sp* can be any structured CASL specification, provided its sub-sort relation has local top elements. This condition holds e.g. for nearly all of the specifications in the CASL library of Basic Datatypes [19].

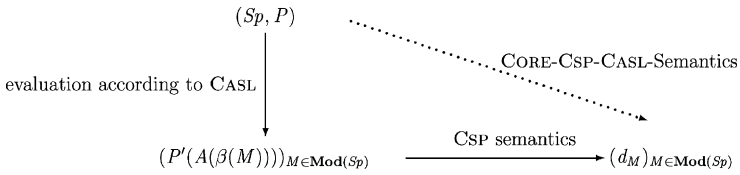

Fig. 3. Overview of the CORE–CSP–CASL semantics construction.

First, we present an overview of the two-step semantics of CORE–CSP–CASL. Then we define how to evaluate in the first step the CASL elements within processes and show how—in the second step—the various denotational CSP semantics can be applied within our approach. In this formal setting of CORE–CSP–CASL without recursion, we study again the integration issues raised in Section 3 and demonstrate that CORE–CSP–CASL solves them in the desired way. Then we complete our semantics of CORE–CSP–CASL by adding recursion to the process part. Finally, we define a notion of refinement and show how to decompose it into the refinement notions of CASL and CSP, respectively.

In the following, we assume all CASL specifications to have local top elements.

5.1. The two-step semantics of CORE–CSP–CASL

The semantics of CORE–CSP–CASL is defined in a two-step approach, see Fig. 3. Let (Sp, P) be a CORE–CSP–CASL specification, i.e. Sp is a CASL specification and P is a CSP process, where CASL terms are used as communications, CASL sorts denote sets of communications, relational renaming is described by a binary CASL predicate, and CASL formulae occur in the conditional (cf. Section 2.3).

In the first step, the evaluation according to CASL, we translate (Sp, P) into a $\mathbf{Mod}(Sp)$-indexed family of CSP processes $(P'(A(\beta(M))))_{M\in\mathbf{Mod}(Sp)}$. Here, we define for each model M of Sp a CSP process $P'(A(\beta(M)))$ over the alphabet of communications $A(\beta(M))$ induced by M. This alphabet is obtained by first applying the model translation β from $SubPFOL^{=}$ models into $FinCommSubPFOL^{=}$ models, cf. Section 4.4. Then, we use the alphabet construction A of Section 4.5 to transform the $FinCommSubPFOL^{=}$ model $\beta(M)$ into an alphabet of communications. Besides the derivation of a suitable alphabet, it is also necessary to evaluate the CASL terms, sorts, formulae, and relations occurring in P. To this end, we define an evaluation function $[\![\_]\!]\_$, which takes a CSP–CASL process specification and an evaluation $v : X \to \beta(M)$ in data-logic as parameters and yields a CSP process over $A(\beta(M))$. Here, the evaluations v deal with CSP binding.

In the second step, the evaluation according to CSP, we apply point-wise a denotational CSP semantics. This translates a process $P'(A(\beta(M)))$ into its denotation d_M in the semantic domain of the chosen CSP semantics.

5.2. Evaluation according to CASL

In the following, S always stands for a *finite* set of sorts.

Let M be a model over a sub-sorted signature $\Sigma = (S, TF, PF, P, \leqslant)$, i.e. let M be a CASL model. Let $\beta(M)$ be its translation into a $CommSubPFOL^{=}$ model. Let $v : X \to \beta(M)$ be a variable valuation. Then the semantics of the CASL elements of the process part is defined by

- $[\![s]\!]_v := \mathtt{emb}_s(\beta(M)_s)$ for $s \in S$.
- $[\![p_{s_1 s_2}]\!]_v := \{(\mathtt{emb}_{s_1}(x), \mathtt{emb}_{s_2}(y)) \mid (x, y) \in (p_{s_1 s_2})_{\beta(M)}\}$ for $p \in P_{s_1 s_2}, s_1, s_2 \in S$.
- $[\![t]\!]_v := \mathtt{emb}_s(v^{\sharp}_s(t))$ for $t \in T_{\hat{\Sigma}}(X)_s$.
- $[\![\varphi]\!]_v := \begin{cases} \mathit{true} & \text{if } v \Vdash \varphi \\ \mathit{false} & \text{if not } v \Vdash \varphi \end{cases}$ where φ is a sub-sorted formula over Σ.

Theorem 12 summarises how the evaluated sorts, terms and predicates relate with their origins in the data-logic $FinCommSubPFOL^{=}$, and therefore—after applying the translation α of $FinCommSubPFOL^{=}$ formulae into $SubPFOL^{=}$ formulae—also with their origins in $SubPFOL^{=}$. For the sub-sorted formulae over Σ, according to Remark 9,

$$
\begin{array}{ll}
[\![SKIP]\!]_\nu & := SKIP \\
[\![STOP]\!]_\nu & := STOP \\
[\![t \to P]\!]_\nu & := [\![t]\!]_\nu \to [\![P]\!]_\nu \\
[\![?x : s \to P]\!]_\nu & := ?x : [\![s]\!]_\nu \to [\![P]\!]_{(\lambda z.\nu)} \\
[\![P \fatsemi Q]\!]_\nu & := [\![P]\!]_\nu \fatsemi [\![Q]\!]_\emptyset \\
[\![P \,\Box\, Q]\!]_\nu & := [\![P]\!]_\nu \,\Box\, [\![Q]\!]_\nu \\
[\![P \sqcap Q]\!]_\nu & := [\![P]\!]_\nu \sqcap [\![Q]\!]_\nu \\
[\![P \,[\![s]\!]\, Q]\!]_\nu & := [\![P]\!]_\nu \,[\![[\![s]\!]_\nu]\!]\, [\![Q]\!]_\nu \\
[\![P \,[\![s_1 \mid s_2]\!]\, Q]\!] & := [\![P]\!]_\nu \,[\![[\![s_1]\!]_\nu \mid [\![s_2]\!]_\nu]\!]\, [\![Q]\!]_\nu \\
[\![P \,||\, Q]\!]_\nu & := [\![P]\!]_\nu \,||\, [\![Q]\!]_\nu \\
[\![P \,|||\, Q]\!]_\nu & := [\![P]\!]_\nu \,|||\, [\![Q]\!]_\nu \\
[\![P \setminus s]\!]_\nu & := [\![P]\!]_\nu \setminus [\![s]\!]_\nu \\
[\![P[[p]]]\!]_\nu & := [\![P]\!]_\nu [[[\![p]\!]_\nu]] \\
[\![\textbf{if } \varphi \textbf{ then } P \textbf{ else } Q]\!]_\nu & := \textbf{if } [\![\varphi]\!]_\nu \textbf{ then } [\![P]\!]_\nu \textbf{ else } [\![Q]\!]_\nu
\end{array}
$$

Fig. 4. Evaluation according to CASL.

the translation α is not necessary and we have thanks to Theorem 8

$$\rho \Vdash \varphi \Leftrightarrow v \Vdash \varphi,$$

where $\rho : X \to M$ is the partial evaluation corresponding to v.

The variable valuations v are necessary to model the CSP binding concept, see Fig. 4 for its definition. At the level of basic CORE–CSP–CASL processes, we need only to bind elements of the alphabet of communications to variable names. Valuations are also allowed to bind the 'undefined' values $\perp$.

The CSP prefix choice operator $?x : S \to P$ binds x in P. Thus, the clause for prefix choice turns the current environment v into a function $(\lambda z.v)$ which takes a substitution as its argument:

$$[\![\_\_]\!]_{\lambda z.v}[a/x] := [\![\_\_]\!]_{v[a/x]}.$$

Here, $v[a/x](y) := v(y)$ for $y \neq x$ and $v[a/x](x) := a$. Substitutions are the way how the various CSP semantics model the binding concept of the prefix choice operator.

Example 13 (*The semantics of the prefix operator in* $\mathcal{T}$). In the CSP traces model $\mathcal{T}$, the semantics of the prefix operator is defined as

$$traces(?x : X \to P) := \{\langle\rangle\} \cup \{\langle a\rangle \frown t \mid t \in traces(P[a/x]), a \in X\}.$$

Here, $\langle\rangle$ denotes the empty trace and $\frown$ is the concatenation of traces. Combining this semantic clause with the evaluation according to CASL in an environment v, we obtain

$$
\begin{array}{l}
traces([\![?x : s \to P]\!]_v) = \\
traces(?x : [\![s]\!]_v \to [\![P]\!]_{(\lambda z.v)}) = \\
\{\langle\rangle\} \cup \{\langle a\rangle \frown t \mid t \in traces([\![P]\!]_{v[a/x]}), a \in [\![s]\!]_v\}.
\end{array}
$$

All other CSP operators just preserve the environment, with the exception of sequential composition. Here, the definition of $[\![\_\_]\!]\_\_$ lifts the CSP declarative view on variables to CORE–CSP–CASL: if the process P terminates, none of its bindings survives, i.e. the following process Q starts within the empty environment $\emptyset$.

Given a CORE–CSP–CASL specification (Sp, P), we now define for $M \in \mathbf{Mod}(Sp)$:

$$P'(A(\beta(M))) := [\![P]\!]_\emptyset.$$

5.3. Evaluation according to CSP

The process $P'(A(\beta(M)))$, see Fig. 3, is an ordinary CSP process over the alphabet of communications $A(\beta(M))$, i.e. we can apply any CSP semantics to it which covers the set of CSP operators involved. This is the case for all denotational

$$
\begin{array}{lcl}
traces(SKIP) & = & \{\langle\rangle, \langle\checkmark\rangle\} \\
traces(STOP) & = & \{\langle\rangle\} \\
traces(a \rightarrow P) & = & \{\langle\rangle\} \cup \{\langle a\rangle \frown s \mid s \in traces(P)\} \\
traces(?x : X \rightarrow P) & = & \{\langle\rangle\} \cup \{\langle a\rangle \frown s \mid s \in traces(P[a/x]), a \in X\} \\
traces(P \fatsemi Q) & = & (traces(P) \cap A^*) \\
 & & \quad \cup \{s \frown t \mid s \frown \langle\checkmark\rangle \in traces(P), t \in traces(Q)\} \\
traces(P \,\Box\, Q) & = & traces(P) \cup traces(Q). \\
traces(P \sqcap Q) & = & traces(P) \cup traces(Q). \\
traces(P \,[\![\, X \,]\!]\, Q) & = & \bigcup \{s \,[\![\, X \,]\!]\, t \mid s \in traces(P) \wedge t \in traces(Q)\} \\
traces(P \,[\![\, X \mid Y \,]\!]\, Q) & = & \{s \in (X \cup Y)^{*\checkmark} \mid s \restriction X \cup \{\checkmark\} \in traces(P) \wedge \\
 & & \qquad\qquad s \restriction Y \cup \{\checkmark\} \in traces(Q) \quad \} \\
traces(P \mid\mid Q) & = & traces(P) \cap traces(Q) \\
traces(P \,|||\, Q) & = & \bigcup \{s \,|||\, t \mid s \in traces(P) \wedge t \in traces(Q)\} \\
traces(P \setminus X) & = & \{s \setminus X \mid s \in traces(P)\} \\
traces(P[[R]]) & = & \{t \mid \exists\, s \in traces(P) \,.\, sR^*t\} \\
traces(\textbf{if } \varphi \textbf{ then } P \textbf{ else } Q) & = & \begin{cases} traces(P); & \varphi \text{ evaluates to } true \\ traces(Q); & \varphi \text{ evaluates to } false \end{cases}
\end{array}
$$

Fig. 5. Semantic clauses for the traces model $\mathcal{T}$.

CSP semantics described in [20], namely the traces model $\mathcal{T}$, the failure divergence model $\mathcal{N}$, and the stable failures model $\mathcal{F}$. For simplicity, we look here at the traces model only.

Given an alphabet of communications A, the *traces model* $\mathcal{T}$ takes the set of all non-empty, prefix-closed subsets of

$$A^{*\checkmark} := A^* \cup \{s \frown \langle\checkmark\rangle \mid s \in A^*\}.$$

as semantic domain. The symbol ✓ denotes termination and is not an element of A.

The domain $\mathcal{T}$ can be seen as a complete partial order (with bottom element), where

$$S \sqsubseteq T :\Leftrightarrow S \subseteq T$$

for $S, T \in \mathcal{T}$. It can also be turned into a complete metric space, where the distance function is defined by

$$d(S,T) := inf\{2^{-n} \mid S \downarrow n = T \downarrow n,\ n \in \mathbf{N}\}$$

for $S, T \in \mathcal{T}$. Here, $s \downarrow n := s$ for $length(s) \leqslant n$, $s \frown t \downarrow n := s$ for $length(s) = n$ for traces $s, t \in A^{*\checkmark}$, and $S \downarrow n := \{s \downarrow n \mid s \in S\}$ for $S \in \mathcal{T}$. Both variants of $\mathcal{T}$, the cpo $(\mathcal{T}, \sqsubseteq)$ and the cms $(\mathcal{T}, d)$, are used (1) to define a semantics of recursive processes in terms of fixed points, and also (2) to prove refinement between fixed points by fixed-point induction.

Fig. 5 summarises the semantic clauses for the traces model. Here, P and Q are CSP processes over an alphabet of communications A, $X, Y \subseteq A$ are sets of communications, and $R \subseteq A \times A$ is a binary relation over A. The necessary notations on traces are defined in Fig. 6.

5.4. The integration issues revisited

With the above defined semantics of CORE–CSP–CASL without recursion, we are now able to study our motivating examples of Section 3 in a formal setting. This also verifies that our design actually results in the desired semantics.

5.4.1. The semantics of the CORE–CSP–CASL specification of Section 3.1

Here, we study how a many-sorted total algebra behaves in our semantics. The data part of the CSP–CASL specification of Section 3.1 defines the sub-sorted signature $\Sigma = (\{S, T\}, \{c : S, d : T\}, \emptyset, \emptyset, \{S \leqslant S, T \leqslant T\})$. There are no axioms present in the data part. Thus, let M be an arbitrary sub-sorted model of Σ. Then, the meaning of the process part for this

(1) $\forall s, t \in A^*,\ a, b \in A$:

$$
\begin{array}{rcl}
\langle\rangle \ |||\ s & = & \{s\} \\
s \ |||\ \langle\rangle & = & \{s\} \\
\langle a\rangle \frown s \ |||\ \langle b\rangle \frown t & = & \{\langle a\rangle \frown u \mid u \in s \ ||| \ \langle b\rangle \frown t\} \\
 & & \cup\ \{\langle b\rangle \frown u \mid u \in \langle a\rangle \frown s \ ||| \ t\} \\
s \ |||\ t \frown \langle\checkmark\rangle & = & \{\} \\
s \frown \langle\checkmark\rangle \ |||\ t & = & \{\} \\
s \frown \langle\checkmark\rangle \ |||\ t \frown \langle\checkmark\rangle & = & \{u \frown \langle\checkmark\rangle \mid u \in s \ ||| \ t\}
\end{array}
$$

(2) $\forall s, t \in A^*,\ x, x' \in X,\ y, y' \in A \setminus X$:

$$
\begin{array}{rcl}
s \ [\![X]\!]\ t \frown \langle\checkmark\rangle & = & \{\} \\
s \frown \langle\checkmark\rangle \ [\![X]\!]\ t & = & \{\} \\
s \frown \langle\checkmark\rangle \ [\![X]\!]\ t \frown \langle\checkmark\rangle & = & \{u \frown \langle\checkmark\rangle \mid u \in s \ [\![X]\!] \ t\} \\
s \ [\![X]\!]\ t & = & t \ [\![X]\!] \ s \\
\langle\rangle \ [\![X]\!]\ \langle\rangle & = & \{\langle\rangle\} \\
\langle\rangle \ [\![X]\!]\ \langle x\rangle & = & \{\} \\
\langle\rangle \ [\![X]\!]\ \langle y\rangle & = & \{\langle y\rangle\} \\
\langle x\rangle \frown s \ [\![X]\!]\ \langle y\rangle \frown t & = & \{\langle y\rangle \frown u \mid u \in \langle x\rangle \frown s \ [\![X]\!] \ t\} \\
\langle x\rangle \frown s \ [\![X]\!]\ \langle x\rangle \frown t & = & \{\langle x\rangle \frown u \mid u \in s \ [\![X]\!] \ t\} \\
\langle x\rangle \frown s \ [\![X]\!]\ \langle x'\rangle \frown t & = & \{\} \qquad \text{if } x \neq x' \\
\langle y\rangle \frown s \ [\![X]\!]\ \langle y'\rangle \frown t & = & \{\langle y\rangle \frown u \mid u \in s \ [\![X]\!] \ \langle y'\rangle \frown t\} \\
 & & \cup\ \{\langle y'\rangle \frown u \mid u \in \langle y\rangle \frown s \ [\![X]\!] \ t\}
\end{array}
$$

(3) $s \setminus X$ is defined to be $s \restriction (A \setminus X)$ for any trace s

(4) If $s \in A^*$ and $X \subseteq A$ then $s \restriction A$ means the sequence s *restricted* to X: the sequence whose members are those of s which are in X.
$\langle\rangle \restriction X = \langle\rangle$ and
$(s \frown \langle a\rangle) \restriction X = (s \restriction X) \frown \langle a\rangle$ if $a \in X$, $s \restriction X$ otherwise.

(5) Definition of the Relation sR^*t. :
$\langle a_1, \ldots a_n\rangle R^* \langle b_1, \ldots b_m\rangle \quad \Leftrightarrow \quad n = m \wedge \forall\, i \leq n \,.\, a_i R b_i$

Fig. 6. Used notations of the traces model $\mathcal{T}$.

model M is

$$
\begin{aligned}
& traces([\![c \to SKIP \parallel d \to SKIP]\!]_\emptyset) \\
= \ & traces(\mathtt{emb}_S(\emptyset^\sharp_S(c)) \to SKIP \parallel \mathtt{emb}_T(\emptyset^\sharp_T(d)) \to SKIP) \\
= \ & traces(\mathtt{emb}_S(\emptyset^\sharp_S(c)) \to SKIP) \cap traces(\mathtt{emb}_T(\emptyset^\sharp_T(d)) \to SKIP) \\
= \ & \{\langle\rangle, \langle \mathtt{emb}_S(\emptyset^\sharp_S(c))\rangle, \langle \mathtt{emb}_S(\emptyset^\sharp_S(c)), \checkmark\rangle\} \cap \{\langle\rangle, \langle \mathtt{emb}_T(\emptyset^\sharp_T(d))\rangle, \langle \mathtt{emb}_T(\emptyset^\sharp_T(d)), \checkmark\rangle\}
\end{aligned}
$$

In order to decide, whether this intersection only has the empty trace, we need to know whether

$$\mathtt{emb}_S(\emptyset^\sharp_S(c)) = \mathtt{emb}_T(\emptyset^\sharp_T(d)).$$

According to Theorem 12, this is equivalent to

$$\emptyset \Vdash c = d \text{ in } CommSubPFOL^=.$$

Thanks to Theorem 8, this is equivalent to

$$\emptyset \Vdash \alpha(c = d) \text{ in } SubPFOL^=.$$

which evaluates to

$$\emptyset \Vdash F \text{ in } SubPFOL^=$$

Thus, $\mathtt{emb}_S(\emptyset^\sharp_S(c)) \neq \mathtt{emb}_T(\emptyset^\sharp_T(d))$ and the intersection evaluates to $\{\langle\rangle\}$.

5.4.2. The semantics of the CORE–CSP–CASL *specification of Section 3.2*

Section 3.2 deals with sub-sorted total algebras. In its example, the data part defines the sub-sorted signature $\Sigma = (\{S,T\},\{c:S, d:T\},\emptyset,\emptyset,\{S \leqslant S, T \leqslant T, S \leqslant T\})$. As axiom we have $\mathtt{inj}_{S,T}(c) = d$.[5] Let M be a Σ model in which this axiom holds. Concerning the process part, as above we obtain for M

$$\{\langle\rangle, \langle \mathtt{emb}_S(\emptyset^\sharp_S(c))\rangle, \langle \mathtt{emb}_S(\emptyset^\sharp_S(c)), \checkmark\rangle\} \cap \{\langle\rangle, \langle \mathtt{emb}_T(\emptyset^\sharp_T(d))\rangle, \langle \mathtt{emb}_T(\emptyset^\sharp_T(d)), \checkmark\rangle\}.$$

In order to decide, whether this intersection only has the empty trace, we need to know whether

$$\begin{array}{l} \quad \mathtt{emb}_S(\emptyset^\sharp_S(c)) = \mathtt{emb}_T(\emptyset^\sharp_S(d)) \\ \text{iff } \emptyset \Vdash c = d \text{ in } \mathit{CommSubPFOL}^= \\ \text{iff } \emptyset \Vdash \alpha(c = d) \text{ in } \mathit{SubPFOL}^= \\ \text{iff } \emptyset \Vdash \mathtt{inj}_{S,T}(c) = \mathtt{inj}_{T,T}(d) \text{ in } \mathit{SubPFOL}^= \end{array}$$

which holds as $\mathtt{inj}_{T,T}(d) = d$ is in the set of axioms $\hat{J}$ and $\mathtt{inj}_{S,T}(c) = d$ is true in M. Thus, the trace semantics of the process part for M is

$$\{\langle\rangle, \langle \mathtt{emb}_S(\emptyset^\sharp_S(c))\rangle, \langle \mathtt{emb}_S(\emptyset^\sharp_S(c)), \checkmark\rangle\}.$$

5.4.3. The semantics of the CORE–CSP–CASL *specification of Section 3.3*

Here, we show the effects of partiality on synchronisation. The data part of the CSP–CASL specification of Section 3.3 defines the sub-sorted signature $\Sigma = (\{S,T\},\emptyset,\{f : S \to ?\ T\},\emptyset,\{S \leqslant S, T \leqslant T\})$. As axiom we have $\forall x : S \bullet \neg \mathit{def}\, f(x)$. Let M be a Σ-model where f_M is undefined for all values in M_S.

Concerning the semantics of the process part, first we study if the multiple prefix $?x : S$ synchronises over T, i.e. whether $\Vdash x : S \ \mathtt{in}\ T$. This is not the case, as $\alpha(x : S \ \mathtt{in}\ T) = F$. Thus, the left process can perform the multiple prefix independent of the right process. Then we need to check whether $f(x)$ synchronises over T, i.e. whether $\Vdash f(x) \ \mathtt{in}\ T$. This is the case, as $\alpha(f(x) \ \mathtt{in}\ T) = \neg \mathit{def}\ \ (f(x)) \vee \ldots$, and $\neg \mathit{def}\ f(x)$ holds in M. The multiple prefix $?y : T$ synchronises over T, as

$$\begin{array}{l} \quad \Vdash y : T \ \mathtt{in}\ T \text{ in } \mathit{CommSubPFOL}^= \\ \text{iff } \Vdash \alpha(y : T \ \mathtt{in}\ T) \text{ in } \mathit{SubPFOL}^= \\ \text{iff } \Vdash \neg \mathit{def}\,(y : T) \vee (\mathit{def}\, y : T \wedge \exists x : T \bullet y = x) \end{array}$$

holds in M : if a partial evaluation $\rho(y)$ is undefined, $\neg \mathit{def}\,(y : T)$ is true, if $\rho(y)$ is defined, its value can be used as a witness for the existential quantifier. Therefore, the left process synchronises with the multiple prefix $?y : T$ and y is bound to $\bot$. Finally, we need to know whether $v \Vdash \mathit{def}\ y$ holds for $v(y) = \bot$. This is not the case, as $\rho \Vdash \mathit{def}\ y$ does not hold for $\rho(y)$ undefined. Thus, the process Q is executed. This results in the trace set

$$\{\langle\rangle,\ \langle \mathtt{emb}_S(x)\rangle,\ \langle \mathtt{emb}_S(x), \mathtt{emb}_T(\bot)\rangle, \langle \mathtt{emb}_S(x), \mathtt{emb}_T(\bot)\rangle \frown t \mid x \in \beta(M)_S,\ t \in \mathit{traces}(Q)\}\,.$$

5.4.4. The semantics of the CORE–CSP–CASL *specification of Section 3.4*

The data part of the CSP–CASL specification of Section 3.4 defines the sub-sorted signature $\Sigma = (\{A,B,C,S\},\{a : A, b_1, b_2 : B, c : C\},\{f : A \to ?\ A, g : C \to ?\ C\},\emptyset,\{A \leqslant S, B \leqslant S, C \leqslant S, A \leqslant A, B \leqslant B, C \leqslant C, S \leqslant S\})$. The here interesting axioms are $\forall x : S \bullet \neg \mathit{def}\, f(x)$ and $\forall x : T \bullet \neg \mathit{def}\ g(x)$. Let M be a Σ-model where f_M is undefined for all values in M_A and g_M is undefined for all values in M_C. Then

$$\begin{array}{l} \quad \mathtt{emb}_S(\emptyset^\sharp_S(f(a))) = \mathtt{emb}_T(\emptyset^\sharp_T(g(c))) \\ \text{iff } \emptyset \Vdash f(a) = g(c) \text{ in } \mathit{CommSubPFOL}^= \\ \text{iff } \emptyset \Vdash \alpha(f(a) = g(c)) \text{ in } \mathit{SubPFOL}^= \\ \text{iff } \emptyset \Vdash \neg \mathit{def}\, f(a) \wedge \neg \mathit{def}\ g(c) \text{ in } \mathit{SubPFOL}^= \end{array}$$

which is true thanks to the above stated axioms. Thus, the trace set for M is

$$\{\langle\rangle, \langle \mathtt{emb}_A(\bot)\rangle, \langle \mathtt{emb}_A(\bot), \checkmark\rangle\}.$$

[5] The CASL static analysis translates the axiom $c = d$ into this formula in $\mathit{SubPFOL}^=$.

$$\begin{array}{ll} \textbf{data} & \textbf{sorts}\ \ Nat \\ & \textbf{ops}\ \ \ 0 : Nat;\ suc : Nat \to Nat \\ \textbf{process} & \textbf{let}\ P(n : Nat) = n \to P(suc(n)) \\ & \textbf{in}\ \ 0 \to P(suc(n)) \end{array}$$

Fig. 7. A CORE–CSP–CASL specification with recursion in the process part.

5.5. Adding recursion to the process part

Up to now we only studied the semantics of basic CORE–CSP–CASL processes. In order to add recursion, we extend the syntax of the process part by the construct

let *ProcessDefinition*$^+$ **in** *Proc*.

The **let** part consists of a nonempty list of process definitions of the form

$$\begin{array}{rl} \mathit{ProcessDefinition} ::= & \mathit{PN} = \mathit{Proc} \\ & \mid \mathit{PN}(x : S) = \mathit{Proc}. \end{array}$$

Here, the left-hand side of an equation is either a process name or a process name with one variable x of a sort S as a parameter. For the right-hand side of a *ProcessDefinition* as well as for the **in** part, we extend the grammar of *Proc* presented in Fig. 1 by two new clauses:

$$\mathit{Proc} ::= \mathit{PN} | \mathit{PN}(t) | \ldots,$$

where *PN* is a process name and $PN(t)$ is process name with a CASL term t as parameter. The **in** part of a recursive process definition provides the process we would like to specify.

Fig. 7 shows an example of a CORE–CSP–CASL specification including a recursive process definition. It consists of a loose specification of the natural numbers in the data part, and specifies a process which, in any model M of the data part, communicates the values of the terms 0, $suc(0)$, $suc(suc(0)), \ldots$

In recursive process definitions we assume that all process names occurring on the right-hand side of a process definition or in the resulting process are defined, that there is exactly one process definition for each process name, that in a process definition with a variable x declared on the left-hand side this is the only free variable on the right-hand side, etc.

Let M be a model over a sub-sorted signature $\Sigma = (S, TF, PF, P, \leqslant)$, i.e. let M be a CASL model. Let $\beta(M)$ be its translation into a *CommSubPFOL*$^=$ model. Then the **let** part of a recursive process definition induces the following set of variables $V_{\beta(M)}$:

(1) Any process name *PN* on the left-hand side of a *ProcessDefinition* yields a process variable $PN \in V_{\beta(M)}$.
(2) Any process name with a variable declaration $PN(x : S)$ on the left-hand side of a *ProcessDefinition* yields a set of variables $\{PN_a \mid a \in \beta(M)_S\} \subseteq V_{\beta(M)}$.
(3) $V_{\beta(M)}$ does not include any other variables.

Let $\mathcal{D}$ be the semantic domain of a denotational CSP model over the alphabet of communications $A(\beta(M))$. For example, in the traces model $\mathcal{D}$ is the set $\mathcal{T}$ of all non-empty, prefix-closed subsets of $A(\beta(M))^{*\surd}$. Then a process environment $\mathcal{E}_{\beta(M)}$ over $V_{\beta(M)}$ is a total map

$$\mathcal{E}_{\beta(M)} : V_{\beta(M)} \to \mathcal{D}.$$

In order to deal with recursive processes, we now extend our evaluation function $[\![\_\!]\!]\_\_$ by a process environment as a second parameter. For process names and process names with parameters we define

- $[\![PN]\!]_{v,\mathcal{E}_{\beta(M)}} := \mathcal{E}_{\beta(M)}(PN)$
- $[\![PN(t)]\!]_{v,\mathcal{E}_{\beta(M)}} := \mathcal{E}_{\beta(M)}(PN_{v(t)})$.

The clauses of Fig. 4 only pass the process environment $\mathcal{E}_{\beta(M)}$ without changing it, for example

$$[\![t \to P]\!]_{v,\mathcal{E}_{\beta(M)}} := [\![t]\!]_{v,\mathcal{E}_{\beta(M)}} \to [\![P]\!]_{v,\mathcal{E}_{\beta(M)}}.$$

The evaluations of CASL elements defined in Section 5.2 just ignore the new parameter, as e.g.

$$[\![s]\!]_{v,\mathcal{E}_{\beta(M)}} := \mathtt{emb}_s(\beta(M)_s).$$

Now, the **let** part of a CORE–CSP–CASL specification with recursion in the process part is turned to an (in general: infinite) system of process equations:

(1) Every *ProcessDefinition* of type $PN = Proc$ yields an equation

$$[\![PN]\!]_{\emptyset,\mathcal{E}_{\beta(M)}} = [\![Proc]\!]_{\emptyset,\mathcal{E}_{\beta(M)}}.$$

(2) Every *ProcessDefinition* of type $PN(x : S) = Proc$ yields a set of equations

$$[\![PN_{v(x)}]\!]_{v,\mathcal{E}_{\beta(M)}} = [\![Proc]\!]_{v,\mathcal{E}_{\beta(M)}},$$

where $v \in \{v' : \{x : S\} \to \beta(M)\}$ is an evaluation; i.e. each possible value of the variable x in $\beta(M)_S$ yields an equation.

The semantics of a CORE–CSP–CASL specification with recursion in the process part is defined iff *for each model M* of the data part this system of equations has a unique solution $\mathcal{E}_{\beta(M)}$ in the chosen CSP model. In this case, the semantics is the $\mathbf{Mod}(Sp)$-indexed family

$$([\![P]\!]_{\emptyset,\mathcal{E}_{\beta(M)}})_{M \in \mathrm{Mod}(Sp)},$$

where P is the process of the **in** part. The theory of CSP semantics offers a large variety of techniques to treat such systems of equations [20].

5.6. Refinement

For a denotational CSP model with domain $\mathcal{D}$, the semantic domain of CORE–CSP–CASL consists of the I-indexed families of process denotations $d_M \in \mathcal{D}$, i.e.

$$(d_M)_{M \in I},$$

where I is a class of $SubPFOL^{=}$ models. As refinement $\rightsquigarrow_{\mathcal{D}}$ we define on these elements

$$\begin{array}{l}(d_M)_{M\in I} \rightsquigarrow_{\mathcal{D}} (d'_{M'})_{M'\in I'} \\ \quad\text{iff} \\ I' \subseteq I \wedge \forall M' \in I' : d_{M'} \sqsubseteq_{\mathcal{D}} d'_{M'},\end{array}$$

where $I' \subseteq I$ denotes inclusion of model classes over the same signature, and $\sqsubseteq_{\mathcal{D}}$ is the refinement notion in the chosen CSP model $\mathcal{D}$. In the traces model $\mathcal{T}$ we have for instance $T \sqsubseteq_{\mathcal{T}} T' :\Leftrightarrow T' \subseteq T$, where T and T' are prefixed closed sets of traces. [6] The definitions of CSP refinements for $\mathcal{D} \in \{\mathcal{T}, \mathcal{N}, \mathcal{F}, \mathcal{I}, \mathcal{U}\}$, cf. [20], which are all based on set inclusion, yield that CSP–CASL refinement is a preorder.

Concerning *data refinement*, we directly obtain the following characterisation:

$$\left.\begin{array}{c}\textbf{data } Sp \textbf{ process } P \textbf{ end} \\ \rightsquigarrow_{\mathcal{D}} \\ \textbf{data } Sp' \textbf{ process } P \textbf{ end}\end{array}\right\} \quad \text{if} \quad \left\{\begin{array}{l}1.\ \Sigma(Sp) = \Sigma(Sp'), \\ 2.\ \mathbf{Mod}(Sp') \subseteq \mathbf{Mod}(Sp)\end{array}\right.$$

[6] We follow here the CSP convention, where T' *refines* T is written as $T \sqsubseteq_{\mathcal{D}} T'$, i.e. the more specific process is on the right-hand side of the symbol.

$$(Sp, Ch, P)$$
$$\text{syntactic encoding}\downarrow$$
$$(Sp \textbf{ then } Sp_{Ch}, P')$$

Fig. 8. Syntactic encoding.

The crucial point is that we fix both the signature of the data part and the process P. For *process refinement*, a similar characterisation is obvious

$$\left.\begin{array}{c}\textbf{data } Sp \textbf{ process } P \textbf{ end}\\ \rightsquigarrow_{\mathcal{D}}\\ \textbf{data } Sp \textbf{ process } P' \textbf{ end}\end{array}\right\} \quad \text{if} \quad \left\{\begin{array}{l}\text{for all } M \in \mathbf{Mod}(Sp):\\ [\![[\![P]\!]_{\emptyset:\emptyset\to\beta(M)}]\!]_{\text{CSP}} \sqsubseteq_{\mathcal{D}} [\![[\![P']\!]_{\emptyset:\emptyset\to\beta(M)}]\!]_{\text{CSP}}.\end{array}\right.$$

Here, $[\![\_]\!]_{\text{CSP}}$ is the evaluation according to the CSP denotational semantics, and $\emptyset : \emptyset \to \beta(M)$ is the empty evaluation into the *CommSubPFOL*$^{=}$ model $\beta(M)$. For this result, we only fix the specification Sp.

6. An example in full CSP–CASL: specifying a file system

With studying CORE–CSP–CASL up to now, we have concentrated on the semantically relevant part of our combination of CASL and CSP. The full language CSP–CASL offers more features, namely it integrates CSP–CASL specifications into CASL libraries and it uses communication channels in the process part. We give a brief overview of these additional features and study then how to model a file system in CSP–CASL.

6.1. *Full* CSP–CASL

In full CSP–CASL, a specification with name N consists of a data part Sp, which is a structured CASL specification, an (optional) channel part Ch to declare channels, which are typed according to the data specification, and a process part P:

ccspec $N =$ **data** Sp **channel** Ch **process** P **end**.

See Fig. 9 for a concrete instance of this scheme.

In the channel part Ch, the statement $c : s$ declares a channel c of sort s. Here, s needs to be a sort defined in Sp. In the process part P, sending a value v of sort s over the channel c is encoded as a communication $c!v$. Receiving a value x from a channel c is written $c?x : s \to P$, which semantically is treated as the CSP prefix choice operator.

Such a specification in full CSP–CASL can be transformed by several syntactic encodings into a specification in CORE–CSP–CASL. In this translation, the treatment of channels is the most prominent one. CSP handles channels as special subsets of the communication alphabet. Consequently, the channel part Ch of a CSP–CASL specification is modelled within CASL. The channel part Ch gives rise to a specification fragment Sp_{Ch}, which monomorphically extends the data part Sp to a CASL specification Sp **then** Sp_{Ch}. As all models of Sp **then** Sp_{Ch}, which extend the same model of Sp, are identical up to isomorphism, and all models of Sp can be extended to at least one model of Sp **then** Sp_{Ch}, this construction neither adds new diversity nor does it remove any interpretation of the data part. The extended specification Sp **then** Sp_{Ch} provides new CASL sorts and operations, with which—in accordance to the original treatment of channels in CSP—the process part P is rewritten to a form P' without channels. Fig. 8 illustrates this step.

Besides dealing with channels, the syntactic encoding also eliminates certain CSP operators, as for instance the 'time-out' $P \rhd Q$, which is replaced by its semantic equivalent $(P \sqcap STOP) \Box Q$. Also, convenient abbreviations for CSP processes like $RUN(s)$, where s is a sort, or $CHAOS$ are resolved.

6.2. *Specifying a file system*

A file-system, cf. Fig. 9, deals with different kind of DATA, namely with *File*s and *Attribute*s associated with them. Here, we organise the *File*s and *Attribute*s as PAIRS—a specification from the CASL standard libraries, which is imported

library FILESYSTEM **version** 1.0
from Basic/StructuredDatatypes **version** 1.0 **get** PAIR

spec DATA =
  sorts $Attribute, File$
then
  PAIR$[File][Attribute]$ **with sort** $Pair[File][Attribute] \mapsto FileAndAttribute$
end

spec STATE =
  DATA
then
  sort $State$
  op $setAttr : State \times FileAndAttribute \to State;$
    $getAttr : State \times File \to ?\ Attribute;$
    $initial : State$
end

ccspec FILESYSTEM =
  data STATE
  channels $set : FileAndAttribute;$
    $get : File;$
    $reply : Attribute$
  process
    let $P(s : State) = set?fa \to P(setAttr(s, fa))$
      $\Box\ get?f \to reply!getAttr(s, f) \to P(s)$
    in $P(initial)$
end

Fig. 9. Specification of a simple file system.

spec STATE1 = STATE **then**
  $\forall s : State;\ f, f' : File;\ a : Attribute$
  • $\neg def\ getAttr(initial, f)$
  • $getAttr(setAttr(s, pair(f, a)), f') = a\ when\ f = f'\ else\ getAttr(s, f')$
end
ccspec FILESYSTEM1 =
  data STATE1
  channels $set : FileAndAttribute;$
    $get : File;$
    $reply : Attribute$
  process
    let $P(s : State) = set?fa \to P(setAttr(s, fa))$
      $\Box\ get?f \to reply!getAttr(s, f) \to P(s)$
    in $P(initial)$
end

Fig. 10. A refinement in the data part.

at the begin of the library FILESYSTEM. A file-system has also a STATE. A *State* is observed by an operation *getAttr*, which returns the *Attribute* associated to a specific *File*. A *State* might be changed, by associating an *Attribute* to a *File*. It is convenient to have a distinguished *initial* state. Note that as there are no sub-sort relations declared, the underlying signature of STATE has local top elements.

Both specifications DATA and STATE are loose, i.e. it is left open, what a *File* or an *Attribute* might be. There are no axioms specifying properties of the operations *setAttr* and *getAttr*. Also, there is no prescribed structure of a *State*. The CSP–CASL specification FILESYSTEM uses these two CASL specifications to define a process, which offers to its environment the choice between setting an *Attribute* to a *File* and asking for the *Attribute* of a *File*.

Although the specification FILESYSTEM includes a recursive process definition built on a loosely specified sort, we can easily prove that the underlying system of equations has a unique solution in the CSP traces model $\mathcal{T}$: the

```
ccspec FILESYSTEM2 =
    data STATE1
    channels set   : FileAndAttribute;
             get   : File;
             reply : Attribute
    process
        let P(s : State) =   set?fa → P(setAttr(s, fa))
                           □ get?f → reply!getAttr(s, f) → P(s)
        in set?fa → P(setAttr(initial, fa))
end
```

Fig. 11. A refinement in the process part.

external choice operator is non-destructive and consists of two processes starting with action prefix—a constructive CSP operator.

While FILESYSTEM only provides the signatures how to access and manipulate states, FILESYSTEM1, see Fig. 10, ensures that there is no information available in the state *initial* and that *getAttr* replies the information added by *setAttr*. FILESYSTEM1 is obtained from FILESYSTEM by a simple data refinement: adding axioms to STATE yields a smaller model class. Thus, according to our first result in Section 5.6, FILESYSTEM1 is a CSP–CASL refinement of FILESYSTEM.

Fig. 11 provides an example of a refinement in the process part. Here, we show with the technique of CSP fixed-point induction that FILESYSTEM2 refines FILESYSTEM1. The significant condition is to prove

$$\begin{array}{rl} & set?fa \to P(setAttr(initial, fa)) \\ & \Box\ get?f \to reply!getAttr(initial, f) \to P(initial) \\ \sqsubseteq_{\mathcal{T}} & set?fa \to P(setAttr(initial, fa)) \end{array}$$

This holds as removing a nondeterministic options within the CSP traces model $\mathcal{T}$ leads to a refinement. As the data is fixed and there is a refinement in the process part, FILESYSTEM2 is a CSP–CASL refinement of FILESYSTEM1 according the second characterisation of refinement in Section 5.6.

7. Relation with other approaches

There are various proposals of reactive CASL extensions—see Fig. 12 for a small selection. Our definition of CSP–CASL, like CCS–CASL [21,22] or CASL-CHART [18], combines CASL with reactive systems of a particular kind. All these approaches result in specification frameworks able to model actual reactive systems. CASL–LTL [17] and COCASL [15] take a more fundamental approach: they extend CASL internally. In the case of CASL–LTL, the logic is extended by temporal operators, while COCASL dualizes the CASL sort generation constraints as well as the structured **free** by co-algebraic constructions. In both cases the result is more like a meta-framework, which allows one e.g. to model the semantics of a process algebra.

According to [22] CCS–CASL restricts CASL to many-sorted conditional equations without partiality and sub-sorting with initial semantics, i.e. the language available to describe data types is a true subset of CASL. As CCS–CASL is based on Milner's value passing CCS, there is no need to turn a many-sorted algebra into one set of communications: synchronisation is only possible between names (parametrised with variables) and co-names (parametrised with terms). Thus, algebraic specification is used to give semantics to the passed values. The available names and co-names are not treated as "data". For a comparison between CSP and CCS see e.g. [6].

The language LOTOS [10] integrates the algebraic specification language ACT ONE with a process algebraic language based on a combination of concepts of CCS and CSP. Concerning the relation of ACT ONE and CASL, we refer to [14], which defines a representation of the institution underlying ACT ONE in *FOL* =, a sub-language of CASL. Furthermore, LOTOS uses initial semantics, while CASL provides both, initial and loose semantics. ACT ONE does neither include sub-sorting nor partiality. Thus, it has only to deal with the first of our four integration issues. This is actually present in the language, as LOTOS takes the CSP approach of synchronisation. In defining synchronisation in terms of so-called gates, which are considered to be different if they have different names, it provides the same solution to this issue as we use in CSP–CASL.

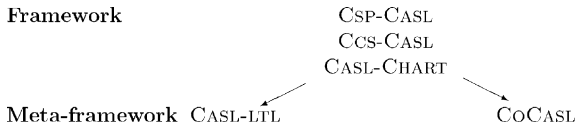

Fig. 12. Relationship between CSP–CASL and other reactive CASL extensions.

In its data part μCRL [8] uses equational logic with a predefined type of Booleans with a fixed interpretation. The logic is restricted to total functions, sub-sorting is not available. There is no formulation of this logic as an institution available, but following the ideas of [14] it should be possible to represent it within the institution underlying CASL. μCRL uses loose semantics, thus defining a model-indexed family as semantics of a specification. The μCRL solution to our first integration issue is to use the values of a data type as parameters to actions, where different action names make the data different.

The model checker FDR extends CSP by a functional 'programming' language for data types. The alphabet of communications is described in terms of channels, which are considered to be different. Thus, FDR and CSP–CASL take the same approach to solve the first integration problem, see Section 3.1: in FDR channel names stand for types, which in CASL are denoted as sorts. FDR does not provide sub-sorting. Concerning partiality, functions like the division of two natural numbers m/n are included—but the situation of undefined results is not properly treated.

Thus, none of the above described combinations of data types with process algebras addresses the problems of partiality or sub-sorting. Concerning the different solutions offered to our first integration issue, they all follow the paradigm 'disjoint union', realised by different technical means. Here, we think that our treatment is on the right level: it does not introduce a new data type construction *outside* the algebraic specification language and therefore allows to translate the question of synchronisation into the question if a certain formula is valid. As our short discussion of full CSP–CASL showed, channels can be treated as a special data construct *inside* CASL and are as such a 'derived' concept.

8. Conclusion and future work

In this paper, we introduce the language CSP–CASL as a new kind of integration of process algebra and algebraic specification. Against the trend set by E-LOTOS [12] in replacing the algebraic specification language for the data part by a functional one, we claim that data refinement is a powerful specification paradigm, and it is interesting to study a language covering the specification of functional as well as of reactive system properties at an appropriate level of abstraction. A case study in an industrial context has shown that CSP–CASL is capable to deal with complex systems at different levels of detail [5].

On the CASL side, CSP–CASL includes many-sorted first order logic with sort generation constraints, sub-sorting and partiality as well as all structuring constructs. Concerning CSP, CSP–CASL is generic in the choice of the denotational CSP semantics. The two characterisations of Section 5.6 and the discussion of our example in Section 6.2 demonstrate that our notion of refinement is intuitive and also of practical use.

Concerning the language CSP–CASL, it will be useful to have also parametrised specifications available. This rises the question whether CSP–CASL can be formalised within the framework of institutions. On the tools side, future work will include establishing a clear relation of CSP–CASL with the model checker FDR, and the development of tool support for theorem proving in CSP–CASL. For the latter, we intend to integrate the theorem provers HOL–CASL [13] and CSP-Prover [11].

Acknowledgements

It is a pleasure to thank the anonymous referees for their valuable feedback; P. Scollo for being a helpful editor; Erwin R. Catesbeiana (jr) for his steady support; and Y. Isobe, T. Mossakowski and J.V. Tucker for long and fruitful discussions on the foundations of CSP–CASL.

References

[1] E. Astesiano, M. Bidoit, B. Krieg-Brückner, H. Kirchner, P.D. Mosses, D. Sannella, A. Tarlecki, CASL—the common algebraic specification language, Theoret. Comput. Sci. 286 (2002) 153–196.

[2] E. Astesiano, H.-J. Kreowski, B. Krieg-Brückner (Eds.), Algebraic Foundations of System Specification, Springer, Berlin, 2000.

[3] C. Baier, M.E. Majster-Cederbaum, The connection between an event structure semantics and an operational semantics for TCSP, Acta Informatica 31 (1994) 81–104.

[4] J.A. Bergstra, A. Ponse, S.A. Smolka (Eds.), Handbook of Process Algebra, Elsevier, Amsterdam, 2001.

[5] A. Gimblett, M. Roggenbach, H. Schlingloff, Towards a Formal Specification of an Electronic Payment System in CSP-CASL, in: J.L. Fiadeiro, P.D. Mosses, F. Orejas (Eds.), Recent Trends in Algebraic Development Techniques, Revised Selected Papers of WADT, Lecture Notes in Computer Science, Vol. 3423, Springer, Berlin, 2005, pp. 61–78.

[6] R.J. van Glabbeek, Notes on the methodology of CCS and CSP, Theoret. Comput. Sci. 177 (1997) 329–349.

[7] J.A. Goguen, R.M. Burstall, Institutions: Abstract model theory for specification and programming, J. ACM 39 (1992) 95–146.

[8] J.F. Groote, A. Ponse, The syntax and semantics of μCRL, in: A. Ponse, C. Verhoef, S.F.M. van Vlijmen (Eds.), in: Algebra of Communicating Processes '94, Workshops in Computing Series, Springer, Berlin, 1995, pp. 26–62.

[9] T. Hoare, Communicating Sequential Processes, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[10] ISO 8807, Lotos—a formal description technique based on the temporal ordering of observational behaviour, Geneva, 1989.

[11] Y.Isobe, M.Roggenbach, A Generic Theorem Prover of CSP Refinement, in: N. Halbwachs, L.D. Zuck (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Proceedings of TACAS 2005, Lecture Notes in Computer Science, Vol. 3440, Springer, Berlin, 2005, pp. 108 – 123.

[12] JTCI/SC7/WG14, The E-LOTOS final draft international standard, 2001, Available at ⟨ftp://ftp.inrialpes.fr/pub/vasy/publications/elotos/elotos-fdis⟩.

[13] T. Mossakowski, CASL: from semantics to tools, in: S. Graf, M. Schwartzbach (Eds.), TACAS 2000, Lecture Notes in Computer Science, Vol. 1785, Springer, Berlin, 2000, pp. 93–108.

[14] T. Mossakowski, Relating CASL with other specification languages: the institution level, Theoret. Comput. Sci. 286 (2002) 367–475.

[15] T. Mossakowski, L. Schröder, M. Roggenbach, H. Reichel, Algebraic-coalgebraic specification in COCASL, JALP, to appear.

[16] P.D. Mosses (Ed.), CASL Reference Manual, Lecture Notes in Computer Science, Vol. 2960, Springer, Berlin, 2004.

[17] G. Reggio, E. Astesiano, C. Choppy, CASL-LTL, Technical Report DISI-TR-99-34, Università di Genova, 2000.

[18] G. Reggio, L. Repetto, CASL-CHART: a combination of statecharts and of the algebraic specification language CASL, AMAST'00, Lecture Notes in Computer Science, Vol. 1816, Springer, Berlin, 2000, pp. 243–257.

[19] M. Roggenbach, T. Mossakowski, L. Schröder, CASL libraries, in: P. Mosses (Ed.), CASL Reference Manual, Lecture Notes in Computer Science, Vol. 2960, Springer, Berlin, 2004.

[20] A.W. Roscoe, The Theory and Practice of Concurrency, Prentice-Hall, Englewood Cliffs, NJ, 1998.

[21] G. Salaün, M. Allemand, C. Attiogbé, A formalism combining CCS and CASL, Technical Report 00.14, University of Nantes, 2001.

[22] G. Salaün, M. Allemand, C. Attiogbé, Specification of an access control system with a formalism combining CCS and CASL, Parallel and Distributed Processing, IEEE, 2002, Silver Spring, MD, pp. 211–219.