Structured CSP – A Process Algebra as an Institution*

Till Mossakowski[1] and Markus Roggenbach[2]

[1] DFKI Lab Bremen and University of Bremen, Germany
till@tzi.de
[2] University of Wales Swansea, United Kingdom
M.Roggenbach@Swan.ac.uk

Abstract. We introduce two institutions for the process algebra CSP, one for the traces model, and one for the stable failures model. The construction is generic and should be easily instantiated with further models. As a consequence, we can use structured specification constructs like renaming, hiding and parameterisation (that have been introduced over an arbitrary institution) also for CSP. With a small example we demonstrate that structuring indeed makes sense for CSP.

1 Introduction

Among the various frameworks for the description and modelling of reactive systems, process algebra plays a prominent role. Here, the process algebra CSP [13, 18] has successfully been applied in various areas, ranging from train control systems [7] over software for the international space station [6] to the verification of security protocols [19].

In this paper we extend the process algebra CSP by a 'module concept' that allows us to build complex specifications out of simpler ones. To this end, we re-use typical structuring mechanisms from algebraic specification as they are realised, e.g., in the algebraic specification language CASL [8, 4]. This approach leads to a new specification paradigm for reactive systems: our framework offers also the loose specification of CSP processes, where the structured free construct applied to a basic specification yields the usual fixed point construction by Tarski's theorem.

On the theoretical side our approach requires us to formulate the process algebra CSP as an institution [12] — the latter notion captures the essence of a logical system and allows for logic-independent structuring languages. We show that various CSP models[1] fit into this setting. The practical outcome is a flexible module concept. We demonstrate through some examples that these structuring

* This work has been supported by EPSRC under the grant EP/D037212/1 and by the German DFG under grant KR 1191/5-2.

[1] i.e. the combination of process syntax, semantic domain, semantic clauses, and a fixed-point theory in order to deal with recursion.

J.L. Fiadeiro and P.-Y. Schobbens (Eds.): WADT 2006, LNCS 4409, pp. 92–110, 2007.

mechanisms (e.g. extension, union, renaming, parametrisation) are suitable for Csp. Furthermore, formulating a process algebra as an institution links two hitherto unrelated worlds.

The paper is organised as follows: Sect. 2 discusses what a Csp signature might be. Then we describe in a generic way how to build a Csp institution. It turns out that many properties can already be proven in the generic setting. Sections 4 and 5 instantiate the generic institution with the traces model and the stable failures model, resp. Having now institutions available, we discuss how to obtain the full range of structuring mechanisms in spite of the missing pushouts of our signature category. In Sect. 7 we make structured specifications available to Csp and illustrate this with a classical example of process algebra. Sect. 8 discusses some related work and concludes the paper.

2 What Is an Appropriate Notion of a Signature Morphism?

When analysing Csp specifications, it becomes clear that there are two types of symbols that change from specification to specification: communications and process names. Pairs consisting of an alphabet A of communication symbols and of process names N (together with some type information) will eventually be the objects of our category CspSig of CSP signatures, see Sect 3.1 below. The notion of a signature morphism, however, is not as easy to determine. An institution captures how truth can be preserved under change of symbols. In this sense, we want to come up with a notion of a signature morphism that is as liberal as possible but still respects fundamental Csp properties. In this section we discuss why this requires to restrict alphabet translations to injective functions.

The process algebra Csp itself offers an operator that changes the communications of a process P, namely *functional renaming*[2] $f[P]$. Here, $f : A \rightarrow ? A$ is a (partial) function such that $dom(f)$ includes all communications occurring in P. The Csp literature, see e.g. [18], classifies functional renaming as follows: (1) Functional renaming with an injective function f preserves all process properties. (2) Functional renaming with a non-injective function f is mainly used for process abstraction. Non-injective renaming can introduce unbounded nondeterminism[3], and thus change fundamental process properties.

As a process algebra, Csp exhibits a number of fundamental algebraic laws. Among these the so-called step laws of Csp, take for example the following law $\langle \Box$**-step**$\rangle$,

[2] Note that the so-called relational renaming, which is included in our Csp dialect, subsumes functional renaming.

[3] Take for example $f[?n : \mathbf{N} \rightarrow (-n) \rightarrow \mathit{Skip}] = 0 \rightarrow \sqcap \{ (-n) \rightarrow \mathit{Skip} \mid n \in \mathbf{N}\}$, where $f(z) = 0$, if $z \geq 0$, and $f(z) = z$, if $z < 0$. As functional renaming can be expressed in terms of relational renaming, the process on the left-hand side is part of our Csp dialect. The process on the right-hand side, however, does not belong to our Csp dialect, as we restrict the internal choice operator to be binary only.

$$\begin{aligned}&(?x : A \to P) \,\Box\, (?y : B \to Q)\\ =\ &?x : A \cup B \to \mathit{if}\ x \in A \cap B\ \mathit{then}\ (P \sqcap Q)\ \mathit{else}\ (\mathit{if}\ x \in A\ \mathit{then}\ P\ \mathit{else}\ Q)\end{aligned}$$

are of a special significance: The step laws do not only hold in all the main CSP models, including the traces model $\mathcal{T}$, the failures/divergences model $\mathcal{N}$, and the stable-failures model $\mathcal{F}$. They are also essential for the definition of complete axiomatic semantics for CSP, see [18, 14]. The CSP step laws show that e.g. the behaviour of external choice $\Box$, alphabetised parallel $[\![X]\!]$ and hiding $\setminus$ crucially depends on the equality relation in the alphabet of communications. We demonstrate this here for the external choice operator $\Box$:

– Assume $a \neq b$. Then

$$\begin{aligned}&(?x : \{a\} \to P) \,\Box\, (?y : \{b\} \to Q)\\ =\ &?x : \{a, b\} \to \mathit{if}\ x \in \{a\} \cap \{b\}\ \mathit{then}\ (P \sqcap Q)\ \mathit{else}\ (\mathit{if}\ x \in \{a\}\ \mathit{then}\ P\ \mathit{else}\ Q)\\ =\ &?x : \{a, b\} \to \mathit{if}\ x \in \{a\}\ \mathit{then}\ P\ \mathit{else}\ Q\end{aligned}$$

– Mapping a and b with a non-injective function f to the same element c has the effect:

$$\begin{aligned}&f[(?x : \{a\} \to P) \,\Box\, (?y : \{b\} \to Q)]\\ =\ &((?x : \{c\} \to f[P]) \,\Box\, (?y : \{c\} \to f[Q]))\\ =\ &?x : \{c\} \to \mathit{if}\ x \in \{c\} \cap \{c\}\ \mathit{then}\ (f[P] \sqcap f[Q])\ \mathit{else}\\ &\quad (\mathit{if}\ x \in \{c\}\ \mathit{then}\ f[P]\ \mathit{else}\ f[Q])\\ =\ &?x : \{c\} \to (f[P] \sqcap f[Q])\end{aligned}$$

I.e. before the translation, the environment controls which one of the two processes P and Q is executed - after the translation this control has been lost: The process makes an internal choice between $f[P]$ and $f[Q]$. Similar examples can be extracted from the step laws for external choice $\Box$, alphabetised parallel $[\![X]\!]$ and hiding $\setminus$.

Summarised: Non-injective renaming can fundamentally change the behaviour of processes. One reason for this is that alphabets of communications play two roles in CSP: They are constituents of both (i) the process syntax and (ii) the semantic domain. This causes problems with non-injective functions as signature morphisms: syntax is translated covariantly while semantics is translated contravariantly.

3 The CSP Institution – General Layout

Institutions have been introduced by Goguen and Burstall [12] to capture the notion of logical system and abstract away from the details of signatures, sentences, models and satisfaction. We briefly recall the notion here.

Let $\mathcal{CAT}$ be the category of categories and functors.[4]

[4] Strictly speaking, $\mathcal{CAT}$ is not a category but only a so-called quasi-category, which is a category that lives in a higher set-theoretic universe.

Definition 1. *An* institution $I = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \models)$ *consists of*

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

such that for each $\sigma: \Sigma \longrightarrow \Sigma'$ *in* $\mathbf{Sign}$ *the following* satisfaction condition *holds:*

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

for each $M' \in |\mathbf{Mod}(\Sigma')|$ *and* $\varphi \in \mathbf{Sen}(\Sigma)$*.*

We first discuss the general layout of the Csp institution independently of a concrete Csp model.

3.1 The Category CspSig of CSP Signatures

An *object* in the category CspSig is a pair (A, N) where

- A is an alphabet of communications and
- $N = (\bar{N}, sort, param)$ collects information on process names; $\bar{N}$ is a set of process names, where each $n \in \bar{N}$ has
  - a parameter type $param(n) = \langle X_1, \ldots, X_k \rangle$, $X_i \subseteq A$ for $1 \leq i \leq k$, $k \geq 0$. A process name without parameters has the empty sequence $\langle\rangle$ as its parameter type.
  - a type $sort(n) = X \subseteq A$, which collects all communications in which the process n can possibly engage in.

By abuse of notation, we will write $n \in N$ instead of $n \in \bar{N}$ and $(a_1, \ldots, a_k) \in param(n)$ instead of $(a_1, \ldots, a_k) \in X_1 \times \ldots \times X_k$, where $param(n) = \langle X_1, \ldots, X_k \rangle$.

A *morphism* $\sigma = (\alpha, \nu) : (A, N) \to (A', N')$ in the category CspSig consists of two maps

- $\alpha : A \to A'$, an injective translation of communications, and
- $\nu : N \to N'$, a translation of process names, which has the following two properties:
  - $param'(\nu(n)) = \alpha(param(n))$: preservation of parameter types, where $\alpha(param(n))$ denotes the extension of α to sequences of sets.
  - $sort'(\nu(n)) \subseteq \alpha(sort(n))$: non-expansion of types, i.e. the translated process $\nu(n)$ is restricted to those events which are obtained by translation of its type $sort(n)$.

The non-expansion of types principle is crucial for ensuring the satisfaction condition of the CSP institution below. It ensures that the semantics of a process is frozen when translated to a larger context, i.e. even when moving to a larger alphabet, up to renaming, models for "old" names may only use "old" alphabet letters. This corresponds to a *black-box* view on processes that are imported from other specification modules.

As usual, the *composition of morphisms* $\sigma = (\alpha, \nu) : (A, N) \to (A', N')$ and $\sigma' = (\alpha', \nu') : (A', N') \to (A'', N'')$ is defined as $\sigma' \circ \sigma := (\alpha' \circ \alpha,\ \nu' \circ \nu)$.

3.2 Sentences

Given	A : alphabet of communications	
	N : set of process names	
	Z : variable system over A	
	$\mathcal{L}(A, N, Z)$: logic	
we define		
$P, Q ::=$	$n(z_1, \ldots, z_k)$	%% (possibly parametrised) process name
$\mid$	*Skip*	%% successfully terminating process
$\mid$	*Stop*	%% deadlock process
$\mid$	$a \to P$	%% action prefix with a communication
$\mid$	$y \to P$	%% action prefix with a variable
$\mid$	$?x : X \to P$	%% prefix choice
$\mid$	$P \,\Box\, Q$	%% external choice
$\mid$	$P \sqcap Q$	%% internal choice
$\mid$	*if* φ *then* P *else* Q	%% conditional
$\mid$	$P \,[\![X]\!]\, Q$	%% generalized parallel
$\mid$	$P \setminus X$	%% hiding
$\mid$	$P[[r]]$	%% relational renaming
$\mid$	$P \fatsemi Q$	%% sequential composition
where		
$n \in N$, $param(n) = \langle X_1, \ldots, X_k \rangle$ for some $k \in \mathbf{N}$, and $z_i \in (\bigcup_{Y \subseteq X_i} Z_Y) \cup X_i$ for $1 \leq i \leq k$; $a \in A$; $y \in Z$; $x \in Z_X$; $X \subseteq A$; $\varphi \in \mathcal{L}(A, N, Z)$ is a formula; and $r \subseteq A \times A$		

Fig. 1. CSP syntax

Relative to an alphabet of communications A we define a *variable system* $Z = (Z_X)_{X \in \mathcal{P}(A)}$ to be a pairwise disjoint family of variables, where subsets $X \subseteq A$ of the alphabet A are the indices.

The standard CSP literature does not reflect what kind of logic $\mathcal{L}(A, N, Z)$ is plugged into the language. A logic that is quite simple but covers those formulae usually occurring in process examples is given in Fig. 2. We record some properties of formulae:

L1 There is a substitution operator $[b/y]$ defined in an obvious way on formulae. Substitution has the following property: If $\varphi \in \mathcal{L}(A, N, Z)$, $y : Y \in Z$, and $b \in Y$, for some $Y \subseteq A$, then $\varphi[b/y] \in \mathcal{L}(A, N, Z \backslash \{y : Y\})$.

Csp *terms*, see Fig. 1 for the underlying grammar, are formed relatively to a signature (A, N), a variable system Z over A, and a logic $\mathcal{L}(A, N, Z)$. Additional Csp operators can be encoded as syntactic sugar, including the synchronous parallel operator $P \parallel Q := P \,[\![A]\!]\, Q$ and the interleaving operator $P \,|||\, Q := P \,[\![\emptyset]\!]\, Q$.

For the purpose of turning Csp into an institution, the use of variables needs to be made more precise. Given a system of *global variables* G and a system of *local variables* L, which are disjoint, we define the system of *all variables* $Z := G \cup L$. We define the set of process terms $T_{(A,N)}(G, L)$ over a signature (A, N) to be the least set satisfying the following rules:

- $n(z_1, \ldots, z_k) \in T_{(A,N)}(G, L)$ if $n \in N$, $param(n) = \langle X_1, \ldots, X_k \rangle$ for some $k \in \mathbf{N}$, and $z_i \in (\bigcup_{Y \subseteq X_i} Z_Y) \cup X_i$ for $1 \leq i \leq k$;
- $Skip,\ Stop \in T_{(A,N)}(G, L)$.
- $a \to P \in T_{(A,N)}(G, L)$ if $a \in A$ and $P \in T_{(A,N)}(G, L)$
- $x \to P \in T_{(A,N)}(G, L)$ if $x \in G \cup L$ and $P \in T_{(A,N)}(G, L)$
- $?x : X \to P \in T_{(A,N)}(G, L)$ if $P \in T_{(A,N)}(G, L \cup \{x : X\})$.
- $P \,\Box\, Q,\ P \sqcap Q \in T_{(A,N)}(G, L)$ if $P, Q \in T_{(A,N)}(G, L)$.
- *if* φ *then* P *else* $Q \in T_{(A,N)}(G, L)$ if $P, Q \in T_{(A,N)}(G, L)$ and $\varphi \in \mathcal{L}(A, N, Z)$.
- $P \,[\![X]\!]\, Q \in T_{(A,N)}(G, L)$ if $P, Q \in T_{(A,N)}(G, L)$ and $X \subseteq A$.
- $P \setminus X \in T_{(A,N)}(G, L)$, if $P \in T_{(A,N)}(G, L)$ and $X \subseteq A$.
- $P[[r]] \in T_{(A,N)}(G, L)$ if $P \in T_{(A,N)}(G, L)$ and $r \subseteq A \times A$.
- $P \fatsemi Q \in T_{(A,N)}(G, L)$ if $P \in T_{(A,N)}(G, L)$ and $Q \in T_{(A,N)}(G, \emptyset)$.

The set of global variables remains constant in all rules; local variables are effected in the rules for prefix choice and sequential composition: prefix choice adds a new local variable; sequential composition deletes all local variables.

The Csp semantics deals with variables using substitution on the syntax level. Here, $P[b/y]$ denotes the process P in which every free occurrence of the variable

Formulae in $\mathcal{L}(A, N, Z)$:		
$t_1 = t_2$	t_1, t_2 terms over (N, A) and Z	
$t \in X$	t a term over (N, A) and Z; $X \subseteq A$	
Terms over (N, A) *and* Z:		
a	$a \in A$	(alphabet symbol)
x	$x \in Z$	(variable)

Fig. 2. A simple logic for formulae occurring in CSP processes

$$
\begin{array}{ll}
n(z_1,\ldots,z_k)[a/y] & = n(y_1,\ldots,y_k) \text{ with } y_i = \begin{cases} a & \text{if } z_i = y \\ z_i & \text{otherwise.} \end{cases} \\
\mathit{Skip}[b/y] & = \mathit{Skip} \\
\mathit{Stop}[b/y] & = \mathit{Stop} \\
(a \to P)[b/y] & = a \to P[b/y] \\
(x \to P)[b/y] & = \begin{cases} b \to P[b/y]\ ; & x = y \\ x \to P[b/y]\ ; & x \neq y \end{cases} \\
(?x : X \to P)[b/y] & = \begin{cases} ?x : X \to P & ;\ x = y \\ ?x : X \to P[b/y]\ ; & x \neq y \end{cases} \\
(P \,\Box\, Q)[b/y] & = P[b/y] \,\Box\, Q[b/y] \\
(P \sqcap Q)[b/y] & = P[b/y] \sqcap Q[b/y] \\
(\mathit{if}\ \varphi\ \mathit{then}\ P\ \mathit{else}\ Q)[b/y] & = \mathit{if}\ \varphi[b/y]\ \mathit{then}\ P[b/y]\ \mathit{else}\ Q[b/y] \\
(P \,[\![X]\!]\, Q)[b/y] & = P[b/y] \,[\![X]\!]\, Q[b/y] \\
(P \setminus X)[b/y] & = P[b/y] \setminus X \\
(P[[r]])[b/y] & = (P[b/y])[[r]] \\
(P \fatsemi Q)[b/y] & = P[b/y] \fatsemi Q[b/y]
\end{array}
$$

Fig. 3. Substitution

$y : Y$ is replaced by a communication $b \in Y$. Fig. 3 gives the formal definition[5]. We write $P[a_1/x_1, a_2/x_2, \ldots, a_n/x_n]$ for $(\ldots((P[a_1/x_1])[a_2/x_2])\ldots)[a_n/x_n]$.

A *process definition* over a signature (A, N) is an equation

$$p(x_1, \ldots, x_k) = P$$

where $p \in N$, the x_i are variables with $x_i : X_i$, where X_i is the *i-th* component of $\mathit{param}(p)$, and P is a term. A process definition is a *sentence* if $P \in T_{(sort(p),N)}(\{x_1 : X_1, \ldots, x_k : X_k\}, \emptyset)$.

3.3 Translation Along a Signature Morphism

Let $\sigma = (\alpha, \nu) : (A, N) \to (A', N')$ be a signature morphism. Given a variable system $Z = (Z_X)_{X \in \mathcal{P}(A)}$ over (A, N) we obtain a variable system $\sigma(Z)$ over (A', N') by $\sigma(Z)_{X'} := \bigcup_{\alpha(X)=X'} Z_X$. For an individual variable $x : X$ this translation yields $\sigma(x : X) = \alpha(x : X) = x : \alpha(X)$. For the *translation of formulae* we require:

L2 $\mathcal{L}$ has a formula translation of the type $\sigma : \mathcal{L}(A, N, Z) \to \mathcal{L}(A', N', \sigma(Z))$ with the following property: given a formula $\varphi \in \mathcal{L}(A, N, Z)$, then $\sigma(\varphi) \in \mathcal{L}(\alpha(A), N', \sigma(Z))$.

L3 Formula translation composes, i.e., for all signature morphisms $\sigma = (\alpha, \nu) : (A, N) \to (A', N')$, $\sigma' = (\alpha', \nu') : (A', N') \to (A'', N'')$, and $\varphi \in \mathcal{L}(A, N, Z)$ holds: $(\sigma' \circ \sigma)(\varphi) = \sigma'(\sigma(\varphi))$.

[5] The rule for prefix choice deals with free and bound variables. In the case of sequential composition only a substitution with a global variable can have an effect on the process Q.

Properties **L2** and **L3** are indeed satisfied by our simple logic given in Fig. 2.

Fig. 4 gives the rules for *term translation*. *Translation of process definitions* is defined as

$$\sigma(p(x_1, \ldots, x_k) = P) := \sigma(p(x_1, \ldots, x_k)) = \sigma(P).$$

The translation of process definitions composes.

$$\begin{array}{lll}
\sigma(n(z_1, \ldots, z_k)) := \nu(n)(\alpha(z_1), \ldots, \alpha(z_k)) & \sigma(P \sqcap Q) & := \sigma(P) \sqcap \sigma(Q) \\
\sigma(\mathit{Stop}) := \mathit{Stop} & \sigma(\mathit{if}\ \varphi\ \mathit{then}\ P\ \mathit{else}\ Q) & := \mathit{if}\ \sigma(\varphi)\ \mathit{then}\ \sigma(P) \\
\sigma(\mathit{Skip}) := \mathit{Skip} & & \mathit{else}\ \sigma(Q) \\
\sigma(a \to P) := \alpha(a) \to \sigma(P) & \sigma(P \,[\![X]\!]\, Q) & := \sigma(P) \,[\![\alpha(X)]\!]\, \sigma(Q) \\
\sigma(x \to P) := x \to \sigma(P) & \sigma(P \setminus X) & := \sigma(P) \setminus \alpha(X) \\
\sigma(?x : X \to P) := ?x : \alpha(X) \to \sigma(P) & \sigma(P[[r]]) & := \sigma(P)[[\alpha(r)]] \\
\sigma(P \,\Box\, Q) := \sigma(P) \,\Box\, \sigma(Q) & \sigma(P \fatsemi Q) & := \sigma(P) \fatsemi \sigma(Q) \\
\multicolumn{3}{c}{\text{where } \alpha(r) := \{(\alpha(x), \alpha(y)) \mid (x, y) \in r\}}
\end{array}$$

Fig. 4. Term translation

3.4 Models and Reducts

Let $\mathcal{D}(A)$ be a CSP domain constructed relatively to a set of communications A. Examples of $\mathcal{D}(A)$ are the domain $\mathcal{T}(A)$ of the CSP traces model, see Section 4, and the domain $\mathcal{F}(A)$ of the CSP stable failures model, see Section 5. A *model* M over a signature (A, N) assigns to each n and for all $a_1, \ldots, a_k \in \mathit{param}(n)$ a type correct element of the semantic domain $\mathcal{D}(A)$, i.e.

$$M(n(a_1, \ldots, a_k)) \in \mathcal{D}(\mathit{sort}(n)) \subseteq \mathcal{D}(A).$$

We define model categories to be partial orders, that is, there is a morphism between models M_1 and M_2, iff $M_1 \sqsubseteq M_2$. Here $\sqsubseteq$ is the pointwise extension of the partial order used in the denotational CSP semantics for the chosen domain $\mathcal{D}$; see the individual domains for the concrete choice of the partial order.

Given an injective (total) alphabet translation $\alpha : A \to A'$ we define its partial inverse as

$$\hat{\alpha} : \begin{array}{c} A' \to ?\ A \\ a' \mapsto \end{array} \begin{cases} \hat{\alpha}(a) & ;\ \text{if } a \in A \text{ is such that } \alpha(a) = a' \\ \mathit{undefined} & ;\ \text{otherwise} \end{cases}$$

Let $\hat{\alpha}_{\mathcal{D}} : \mathcal{D}(A') \to ? \mathcal{D}(A)$ be the extension of $\hat{\alpha}$ to semantic domains – to be defined for any domain individually.

The *reduct* of a model M' along σ is defined as

$$M'|_\sigma\ (n(a_1, \ldots, a_k)) = \hat{\alpha}_{\mathcal{D}}(M'(\nu(n)(\alpha(a_1), \ldots, \alpha(a_k)))).$$

As for reducts it is clear that we work with domains, we usually omit the index and write just $\hat{\alpha}$. On the level of domains, we define the following **reduct condition** on α and $\hat{\alpha} : \forall X \subseteq A : \ \hat{\alpha}(\mathcal{D}(\alpha(X))) \subseteq \mathcal{D}(X)$.

Theorem 2 (Reducts are type correct). *Let α and $\hat{\alpha}$ fulfil the reduct condition. Then reducts are type correct, i.e. $M' \mid_\sigma (n(a_1, \ldots, a_k)) \in \mathcal{D}(sort(n))$.*

3.5 Satisfaction

Given a map $denotation : M \times P \to \mathcal{D}(A)$, which – given a model M – maps a closed process term $P \in T_{(A,N)}(\emptyset, \emptyset)$ to its denotation in $\mathcal{D}$, we define the satisfaction relation of our institution[6]:

$$\begin{gathered} M \models p(x_1, \ldots, x_k) = P \\ :\Leftrightarrow \\ \forall (a_1, \ldots, a_k) \in param(p). \\ denotation_M(p(a_1, \ldots a_k)) = denotation_M(P[a_1/x_1, \ldots, a_k/x_k]) \end{gathered}$$

Remark 3. We can replace the logic $\mathcal{L}(A, N, Z)$ by any other logic that comes with a satisfaction relation

$$\models \subseteq \textsc{CspMod}_{\mathcal{D}}(A, N) \times \mathcal{L}(A, N, \emptyset)$$

and satisfies laws **L1** to **L3** above, plus

L4 The logic fulfils a satisfaction condition, i.e., forall $\varphi \in \mathcal{L}(A, N, \emptyset)$ holds:

$$M' \mid_\sigma \models \varphi[a_1/x_1, \ldots, a_n/x_n] \Leftrightarrow M' \models \sigma(\varphi)[\alpha(a_1)/x_1, \ldots, \alpha(a_n)/x_n]$$

To be concise with the Csp semantics, which deals with variables using substitution on the syntax level, it is necessary to include here a (possibly empty) substitution, see the reduct property stated in Theorem 4 below.

The Csp models give interpretations to the process names. The formulae used in practical Csp examples usually only reason about data, not on processes. Thus, in the satisfaction condition above the notion of a model and its reduct will vanish in most logic instances.

If the chosen Csp model has the reduct property and the extension of α and $\hat{\alpha}$ are inverse functions on $\mathcal{D}(A)$ and $\mathcal{D}(\alpha(A))$, the satisfaction condition holds:

Theorem 4 (Satisfaction condition). *Let $\sigma = (\alpha, \nu) : (A, N) \to (A', N')$ be a signature morphism. Let M' be a (A', N')-model over the domain $\mathcal{D}(A')$. Let the following* **reduct property** *hold:*

$$\begin{aligned} &denotation_{M' \mid \sigma}(P[a_1/x_1, \ldots, a_n/x_n]) \\ &= \hat{\alpha}(denotation_{M'}(\sigma(P)[\alpha(a_1)/x_1, \ldots, \alpha(a_n)/x_n])) \end{aligned}$$

for all $P \in T_{(A,N)}(\{x_1 : X_1, \ldots x_n : X_n\}, \emptyset)$, $a_i \in X_i \subseteq A$ for $1 \leq i \leq n$, $n \geq 0$. Let α and $\hat{\alpha}$ be inverses on $\mathcal{D}(A)$ and $\mathcal{D}(\alpha(A))$. Under these conditions, we have for all process definitions $p(x_1, \ldots, x_k) = P$ over (A, N) :

$$M' \mid_\sigma \models p(x_1, \ldots, x_k) = P \Leftrightarrow M' \models \sigma(p(x_1, \ldots, x_k) = P).$$

[6] Here and in the following we use '$:\Leftrightarrow$' as an abbreviation for 'iff, by definition'.

4 The CSP Traces Model as an Institution

Given an alphabet A and an element $\checkmark \notin A$ (denoting successful termination) we define sets $A^{\checkmark} := A \cup \{\checkmark\}$ and $A^{*\checkmark} := A^* \cup \{t \frown \langle\checkmark\rangle \mid t \in A^*\}$. The domain $\mathcal{T}(A)$ of the traces model is the set of all subsets T of $A^{*\checkmark}$ for which the following healthiness condition holds:

T1 T is non-empty and prefix closed.

The domain $\mathcal{T}(A)$ gives rise to the notion of trace refinement $S \sqsubseteq_{\mathcal{T}} T :\Leftrightarrow T \subseteq S$. $(\mathcal{T}(A), \sqsubseteq_{\mathcal{T}})$ forms a complete lattice, with $A^{*\checkmark}$ as its bottom and $\{\langle\rangle\}$ as its top. *Morphisms* in the category $Mod_{\mathcal{T}}(A, N)$ are defined as:

$$M_1 \to M_2 :\Leftrightarrow$$
$$\forall\, n \in N : \forall\, a_1, \ldots, a_k \in param(n) : M_2(n(a_1, \ldots, a_k)) \sqsubseteq_{\mathcal{T}} M_1(n(a_1, \ldots, a_k))$$

$I(n(a_1, \ldots, a_k)) = \{\langle\rangle\}$, i.e. the model which maps all instantiated process names to the denotation of *Stop* is initial in $Mod_{\mathcal{T}}(A, N)$; $F(n(a_1, \ldots, a_k)) = A^{*\checkmark}$ is final in $Mod_{\mathcal{T}}(A, N)$.

Let $\sigma = (\alpha, \nu) : (A, N) \to (A', N')$ be a signature morphism. We extend the map α canonically to three maps $\alpha^{\checkmark}$, $\alpha^{*\checkmark}$ and $\alpha^{*\checkmark}_{\mathcal{T}}$ to include the termination symbol, to extend it to strings, and to let it apply to elements of the semantic domain, respectively. In the same way we can extend $\hat{\alpha}$, the partial inverse of α, to three maps $\hat{\alpha}^{\checkmark}$, $\hat{\alpha}^{*\checkmark}$ and $\hat{\alpha}^{*\checkmark}_{\mathcal{T}}$. With these notions, it holds that:

Theorem 5 (Reducts in the traces model are well-behaved)

1. *Let* $T' \in \mathcal{T}(A')$. *Then* $\hat{\alpha}(T') \in \mathcal{T}(A)$.
2. $\forall\, X \subseteq A :\ \hat{\alpha}(\mathcal{T}(\alpha(X))) \subseteq \mathcal{T}(X)$.

$$
\begin{aligned}
traces_M(n(a_1, \ldots, a_k)) &= M(n(a_1, \ldots, a_k)) \\
traces_M(Skip) &= \{\langle\rangle, \langle\checkmark\rangle\} \\
traces_M(Stop) &= \{\langle\rangle\} \\
traces_M(a \to P) &= \{\langle\rangle\} \cup \{\langle a\rangle \frown s \mid s \in traces_M(P)\} \\
traces_M(?\ x : X \to P) &= \{\langle\rangle\} \cup \{\langle a\rangle \frown s \mid s \in traces_M(P[a/x]), a \in X\} \\
traces_M(P \,\Box\, Q) &= traces_M(P) \cup traces_M(Q) \\
traces_M(P \sqcap Q) &= traces_M(P) \cup traces_M(Q) \\
traces_M(\mathit{if}\ \varphi\ \mathit{then}\ P\ \mathit{else}\ Q) &= \text{if } M \models \varphi \text{ then } traces_M(P) \text{ else } traces_M(Q) \\
traces_M(P \,[\![X]\!]\, Q) &= \bigcup\{t_1 \,[\![X]\!]\, t_2 \mid t_1 \in traces_M(P),\ t_2 \in traces_M(Q)\} \\
traces_M(P \setminus X) &= \{t \setminus X \mid t \in traces_M(P)\} \\
traces_M(P[[r]]) &= \{t \mid \exists\, t' \in traces_M(P).\ (t', t) \in r^*\} \\
traces_M(P \fatsemi Q) &= (traces_M(P) \cap A^*) \cup \\
&\quad \{t_1 \frown t_2 \mid t_1 \frown \langle\checkmark\rangle \in traces_M(P),\ t_2 \in traces_M(Q)\}
\end{aligned}
$$

Fig. 5. Semantic clauses of the basic processes in the traces model $\mathcal{T}$

Fig. 5 gives the semantic clauses of the traces model, see [18] for the definition of the various operators on traces. Note that thanks to the rules imposed on the use of variables, there is no need to provide a denotation for a process term of the form $x \rightarrow P$: In the clause for prefix choice $?x : X \rightarrow P$, which is the only way to introduce a variable x, every free occurrence of x in the process P is syntactically substituted by a communication.

Lemma 6 (Terms, Substitutions and Reducts). *With $traces_M$ as denotation function, the traces model has the reduct property stated in Theorem 4.*

As reducts are healthy and the reduct property holds, reducts are well formed. Thanks to Lemma 6 and Theorem 4, the Csp traces model forms an institution.

5 The CSP Stable Failures Model as an Institution

Given an alphabet A the domain $\mathcal{F}(A)$ of the stable failures model consists of those pairs

$$(T, F), \quad \text{where } T \subseteq A^{*\checkmark} \text{ and } F \subseteq A^{*\checkmark} \times \mathcal{P}(A^{\checkmark}),$$

satisfying the following healthiness conditions:

T1 T is non-empty and prefix closed.
T2 $(s, X) \in F \Rightarrow s \in T$.
T3 $s \frown \checkmark \in T \Rightarrow (s \frown \checkmark, X) \in F$ for all $X \subseteq A^{\checkmark}$.
F2 $(s, X) \in F \wedge Y \subseteq X \Rightarrow (s, Y) \in F$.
F3 $(s, X) \in F \wedge \forall\, a \in Y : s \frown \langle a \rangle \notin T \Rightarrow (s, X \cup Y) \in F$.
F4 $s \frown \langle \checkmark \rangle \in T \Rightarrow (s, A) \in F$.

The domain $\mathcal{F}(A)$ gives rise to the notion of stable failures refinement

$$(T, F) \sqsubseteq_{\mathcal{F}} (T', F') :\Leftrightarrow T' \subseteq T \wedge F' \subseteq F$$

$(\mathcal{F}(A), \sqsubseteq_{\mathcal{F}})$ forms a complete lattice with $(A^{*\checkmark}, A^{*\checkmark} \times \mathcal{P}(A^{\checkmark}))$ as its bottom and $(\{\langle\rangle\}, \emptyset)$ as its top. See [18] for a complete definition of the stable failures model. *Morphisms* in the category $Mod_{\mathcal{F}}(A, N)$ are defined as:

$$M_1 \rightarrow M_2 :\Leftrightarrow$$
$$\forall\, n \in N : \forall\, a_1, \ldots, a_k \in param(n) : M_2(n(a_1, \ldots, a_k)) \sqsubseteq_{\mathcal{F}} M_1(n(a_1, \ldots, a_k))$$

$I(n(a_1, \ldots, a_k)) = (\{\langle\rangle\}, \emptyset)$, i.e. the model which maps all instantiated process names to the denotation of the immediately diverging process, is initial in $Mod_{\mathcal{T}}(A, N)$; $F(n(a_1, \ldots, a_k)) = (A^{*\checkmark}, A^{*\checkmark} \times \mathcal{P}(A^{\checkmark}))$ is final in $Mod_{\mathcal{T}}(A, N)$.

The semantic clauses of the stable failures model are given by a pair of functions: $fd_M(P) = (traces_M(P), failures_M(P))$ – see [18] for the definition.

Following the same extension pattern for $\alpha : A \to A'$ as demonstrated for the traces model, we obtain:

Theorem 7 (Reducts in the stable failures model are well-behaved)

1. *Let* $(T', F') \in \mathcal{F}(A')$. *Then* $\hat{\alpha}(T', F') \in \mathcal{F}(A)$.
2. $\forall X \subseteq A : \ \hat{\alpha}(\mathcal{F}(\alpha(X))) \subseteq \mathcal{F}(X)$.

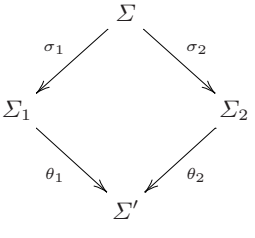
Lemma 8 (Terms, Substitutions and Reducts). *With* fd_M *as denotation function, the stable failures model has the reduct property stated in Theorem 4.*

As reducts are healthy and the reduct property holds, reducts are well formed in the stable failures model. Thanks to Lemma 8 and Theorem 4, the Csp stable failures model forms an institution.

6 Pushouts and Amalgamation

The existence of pushouts and amalgamation properties shows that an institution has good modularity properties. The amalgamation property (called 'exactness' in [9]) is a major technical assumption in the study of specification semantics [20] and is important in many respects. To give a few examples: it allows the computation of normal forms for specifications [3, 5], and it is a prerequisite for good behaviour w.r.t. parametrisation [10] and conservative extensions [9, 17]. The proof system for development graphs with hiding [15], which allow a management of change for structured specifications, is sound only for institutions with amalgamation. A Z-like state based language has been developed over an arbitrary institution with amalgamation [2].

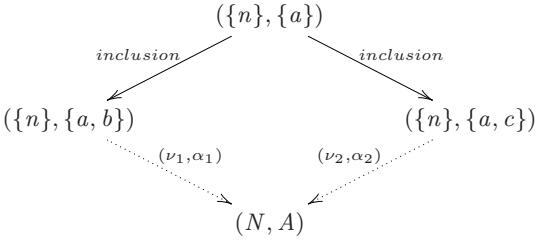
The mildest amalgamation property is that for pushouts. It is also called semi-exactness. An institution is said to be *semi-exact*, if for any pushout of signatures



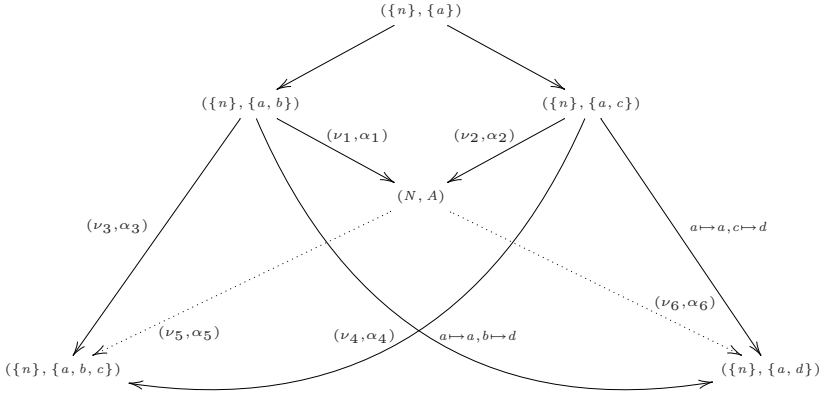
any pair $(M_1, M_2) \in \mathbf{Mod}(\Sigma_1) \times \mathbf{Mod}(\Sigma_2)$ that is *compatible* in the sense that M_1 and M_2 reduce to the same Σ-model can be *amalgamated* to a unique Σ'-model M (i.e., there exists a unique $M \in \mathbf{Mod}(\Sigma')$ that reduces to M_1 and M_2, respectively), and similarly for model morphisms.

Proposition 9. *CspSig does not have pushouts.*

Proof. Suppose that there is a pushout



By the pushout property, we have the following mediating morphisms:



Since α_1 and α_6 are injective, A must have cardinality 2, which implies that α_1 and α_2 are bijective. But then, $\{a,b\} = Im(\alpha_3) = Im(\alpha_5) = Im(\alpha_4) = \{a,c\}$, a contradiction. □

However, this result is not as severe as it might look. Let $CspSig^{noninj}$ be *CspSig* with the restriction dropped that α must be injective. Then we have:

Proposition 10. *$CspSig^{noninj}$ has pushouts, and any such pushout of a span in CspSig actually is a square in CspSig (although not a pushout in CspSig).*

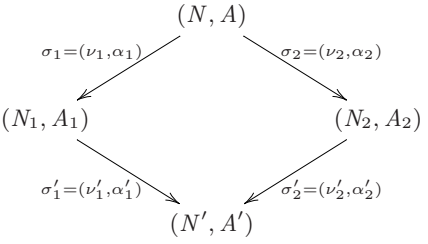
Proof. **Set** has pushouts, and monomorphisms in **Set** are stable under pushouts ([1, Exercise 11P]). This lifts to the indexed level in $CspSig^{noninj}$ and *CspSig*. □

Note that the phenomenon that pushouts of *CspSig*-spans in $CspSig^{noninj}$ are squares but not pushouts in *CspSig* is due to the fact that mediating morphisms are generally not in *CspSig*.

Pushouts in $CspSig^{noninj}$ give us an amalgamation property:

Theorem 11. *$CspSig^{noninj}$-pushouts of CspSig-morphisms have the semi-exactness property for the traces model and the stable failures model.*

Proof. Let



be a $CspSig^{noninj}$-pushout of $CspSig$-morphisms, and let M_i be an (N_i, A_i)-model w.r.t. the trace or the stable failure semantics $(i = 1, 2)$ such that $M_1 \mid_{\sigma_1} = M_2 \mid_{\sigma_2}$. We construct an (N', A')-model M' as follows:

$$M'(n) = \begin{cases} \alpha_1(M_1(n_1)), \text{if } n_1 \text{ is such that } \nu_1(n_1) = n \\ \alpha_2(M_2(n_2)), \text{if } n_2 \text{ is such that } \nu_2(n_2) = n \end{cases}$$

This is well-defined because $M_1 \mid_{\sigma_1} = M_2 \mid_{\sigma_2}$. It is clear that $M' \mid_{\theta_i} = M_i$ $(i = 1, 2)$. Due to the non-expansion of types principle for signature morphisms, M' is unique. □

In fact, this result generalizes easily to multiple pushouts. Moreover, the initial (=empty) signature has the terminal model category. Since all colimits can be formed by the initial object and multiple pushouts, this shows that we even have exactness (when colimits are taken in $CspSig^{noninj}$).

7 Structuring and Parametrization for CSP

Mostly following [20], in this section we recall a popular set of institution-independent structuring operations, which seems to be quite universal and which can also be seen as a kernel language for the CASL structuring constructs [8].

basic specifications For any signature $\Sigma \in |\ \mathbf{Sign}\ |$ and finite set $\Gamma \subseteq \mathbf{Sen}(\Sigma)$ of Σ-sentences, the *basic specification* $\langle \Sigma, \Gamma \rangle$ is a specification with:

$$Sig(\langle \Sigma, \Gamma \rangle) := \Sigma$$
$$\mathbf{Mod}(\langle \Sigma, \Gamma \rangle) := \{M \in \mathbf{Mod}(\Sigma) \mid M \models \Gamma\}$$

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

$$Sig(SP_1 \cup SP_2) := \Sigma$$
$$\mathbf{Mod}(SP_1 \cup SP_2) := \mathbf{Mod}(SP_1) \cap \mathbf{Mod}(SP_2)$$

***translation*:** For any signature morphism $\sigma\colon \Sigma \to \Sigma'$ and Σ-specification SP, SP **with** σ is a specification with:

$$Sig(SP \textbf{ with } \sigma) := \Sigma'$$
$$\mathbf{Mod}(SP \textbf{ with } \sigma) := \{M' \in \mathbf{Mod}(\Sigma') \mid M' \mid_\sigma \in \mathbf{Mod}(SP)\}$$

***hiding*:** For any set SYs of symbols in a signature Σ' generating a subsignature Σ of Σ', and Σ'-specification SP', SP' **reveal** SYs is a specification with:

$Sig(SP' \textbf{ reveal } SYs) \quad := \Sigma$

$\textbf{Mod}(SP' \textbf{ reveal } SYs) := \{M' |_\sigma \mid M' \in \textbf{Mod}(SP')\}$

where $\sigma: \Sigma \to \Sigma'$ is the inclusion signature morphism.

***free specification*:** For any signature morphism $\sigma: \Sigma \to \Sigma'$ and Σ'-specification SP', **free** SP' **along** σ is a specification with:

$Sig(\textbf{free } SP' \textbf{ along } \sigma) = \Sigma'$

$\textbf{Mod}(\textbf{free } SP' \textbf{ along } \sigma) = \{M' \in \textbf{Mod}(SP') \mid$

M' is strongly persistently $(\textbf{Mod}(\sigma): \textbf{Mod}(SP') \longrightarrow \textbf{Mod}(\Sigma))$-free $\}$

Given categories $\mathbf{A}$ and $\mathbf{B}$ and a functor $G: \mathbf{B} \longrightarrow \mathbf{A}$, an object $B \in \mathbf{B}$ is called *G-free (with unit $\eta_A: A \longrightarrow G(B)$) over $A \in \mathbf{A}$*, if for any object $B' \in \mathbf{B}$ and any morphism $h: A \longrightarrow G(B')$, there is a unique morphism $h^\# : B \longrightarrow B'$ such that $G(h^\#) \circ \eta_A = h$. An object $B \in \mathbf{B}$ is called *strongly persistently G-free* if it is G-free with unit id over $G(B)$ (id denotes the identity).

***parametrisation*:** For any (formal parameter) specification SP, (body) specification SP' with signature inclusion $\sigma: Sig(SP) \longrightarrow Sig(SP')$ and specification name SN, the declaration

$$SN[SP] = SP'$$

names the specification SP' with the name SN, using formal parameter SP. The formal parameter SP can also be omitted; in this case, we just have named a specification for future reference.

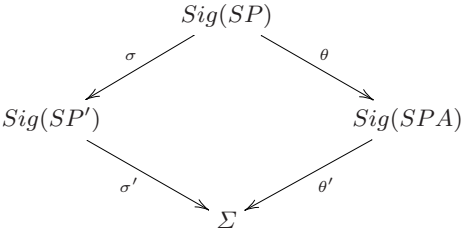
***instantiation*:** Given a named specification $SN[SP] = SP'$ with signature inclusion $\sigma: Sig(SP) \longrightarrow Sig(SP')$ and an (actual parameter) specification SPA and a fitting morphism $\theta: Sig(SP) \longrightarrow Sig(SPA)$, $SN[SPA \textbf{ fit } \theta]$ is a specification with

$Sig(SN[SPA \textbf{ fit } \theta]) \quad := \Sigma$

$\textbf{Mod}(SN[SPA \textbf{ fit } \theta]) :=$

$\{M \in \textbf{Mod}(\Sigma) \mid M |_{\sigma'} \in \textbf{Mod}(SP'), M |_{\theta'} \in \textbf{Mod}(SPA)\}$

where



is a pushout (note that for CSP, we take the pushout in $CspSig^{noninj}$, as discussed in Sect. 6).

In CASL, we can also *extend* specifications with new declarations and axioms. This is written $SP \textbf{ then } SP'$, where SP' is a specification fragment. Since we do not want to deal with specification fragments formally here, we just note that

spec NoLoss =
- $OneCoin = coin \rightarrow Skip$
- $AtLeastOneCoin = OneCoin \sqcap coin \rightarrow AtLeastOneCoin$
- $NoLoss = AtLeastOneCoin;\ item \rightarrow NoLoss$

end

spec MachineForTeaAndCoffee
[**{**NoLoss **then** $NoLoss \sqsubseteq VM \setminus \{button\}$ **}** **reveal** VM]
=
$TeaAndCoffee = \quad VM[[\{(item, coffee), (button, c\text{-}button)\}]]$
$\qquad\qquad \Box\ VM[[\{(item, tea), (button, t\text{-}button)\}]]$
end

spec UnfairMachine =
$UnfairMachine = button \rightarrow coin \rightarrow coin \rightarrow item \rightarrow UnfairMachine$
end

spec UnfairMachineForTeaAndCoffee =
{ MachineForTeaAndCoffee [UnfairMachine **fit** $UnfairMachine \mapsto VM$]
} **reveal** TeaAndCoffee
end

Fig. 6. Process Instantiation

the semantics of extension is similar to that of union, and refer to [8] for full formal details.

In standard Csp, the cpo approach defines the meaning of a system of recursive process equations to be its smallest fixed-point, if such a smallest fixed-point exists. To determine this fixed point, Tarski's fixed-point theorem is applied to the function underlying the system of equations. Take for example, the system $P = P,\ Q = a \rightarrow Q$. Over the alphabet $A = \{a\}$ it has $traces(P) = \{\langle\rangle\}, traces(Q) = a^*$ as its smallest solution. However, there are other fixed-points, as the equation $P = P$ holds for every process, i.e. $traces(P) = \{\langle\rangle, \langle a\rangle\},\ traces(P) = \{\langle\rangle, \langle a\rangle, \langle aa\rangle\}$, etc. also yield fixed-points. As structured Csp works with loose semantics,

$$\textbf{spec}\ \textsc{Loose} = \ \bullet\ P = P\ \ \bullet\ Q = a \rightarrow Q\ \textbf{end}$$

has the set of *all* fixed-points as its semantics. Choosing initial semantics by adding the keyword **free**, however, i.e.

$$\textbf{spec}\ \textsc{Initial} = \textbf{free}\ \{\ \bullet\ P = P\ \ \bullet\ Q = a \rightarrow Q\ \}\ \textbf{end}$$

has the *smallest* fixed point as its semantics thanks to our choice of morphisms in the model categories.

In order to illustrate the practical use of structured Csp specifications, we consider the classical example of process algebra: the development of a vending machine for tea and coffee, following [13], see Fig. 6. For simplicity, we omit explicit signature declarations and derive the alphabet and the process names

from the symbols used. The owner of a vending machine will insist the machine never to make a loss. The process *NoLoss* with $sort(NoLoss) = \{coin, item\}$ in the specification NoLoss has the property that at any time the number of *coin*s inserted to the machine is bigger than the number of *item*s delivered. The specification MachineForTeaAndCoffee describes how to turn the specification of a non-dedicated vending machine *VM* into the specification of a machine for selling tea and coffee. Here, we assume $sort(VM) = \{coin, item, button\}$. *VM* is loosely specified by the condition $NoLoss \sqsubseteq VM \setminus \{button\}$, i.e. $VM \setminus \{button\}$ does not make any loss. The specification MachineForTeaAndCoffee takes the machine *VM* as its parameter and defines the machine *TeaAndCoffee* by renaming the *item* to be delivered into *tea* and *coffee*, resp., and the *button* into *c-button* and *t-button*, resp. However, only those vending machines *VM* are accepted as an actual parameter that fulfil the condition specified by *NoLoss*: This is expressed via the refinement condition $NoLoss \sqsubseteq VM \setminus \{button\}$ in the parameter[7]. The *UnfairMachine*, which lets the customer pay twice for one item, fulfils this requirement in the traces model as well as in the stable failures model. Therefore, it is a legal parameter. Instantiating MachineForTeaAndCoffee with the process *UnfairMachine* yields a process *CoffeeAndTea*, where the customer has to pay twice for tea and coffee.

The semantics of the specifications above behaves as expected. For example, for the basic specification NoLoss, we get:

- $Sig(\text{NoLoss}) = (A, (\bar{N}, sort, param))$ with
- $A = \{coin, item\}$
- $\bar{N} = \{OneCoin, AtLeastOneCoin, NoLoss\}$
- $sort(OneCoin) = \{coin\}$, $sort(AtLeastOneCoin) = \{coin\}$, $sort(NoLoss) = A$.
- $param(n) = \langle\rangle$.
- **Mod**(NoLoss) consists of one model M with

$$M(NoLoss) = \{s \mid s \text{ is prefix of } t \in (coin^+ \ item)^*\}$$

It is quite typical that Csp specifications have exactly one model; indeed, in this respect, Csp resembles more a programming language than a specification language. However, using refinement, we can also write useful loose specifications. Consider $SP = \{\text{NoLoss } \textbf{then } NoLoss \sqsubseteq VM \setminus \{button\}\} \textbf{ reveal } VM$. This has the following semantics:

- $Sig(SP) = (A, (\bar{N}, sort, param))$ with
- $A = \{coin, item, button\}$
- $\bar{N} = \{VM\}$
- $sort(VM) = \{coin, item, button\}$.
- $param(n) = \langle\rangle$.
- **Mod**(SP) consists of models M that provide a trace set $M(VM)$ with the following property: if the action *button* is removed from $M(VM)$, the resulting trace set is contained in $M(NoLoss)$ above.

[7] Note that Csp refinement $P \sqsubseteq Q$ is equivalent to $P = P \sqcap Q$.

That is, SP can be see as a requirement specification on a vending machine, allowing several actual vending machine implementations. SP is the formal parameter of a parametrised specification that can be instantiated with different vending machines. Moreover, due to the amalgamation property of Theorem 11, we can ensure that each vending machine model can be extended to a model of the appropriately instantiated specification MachineForTeaAndCoffee.

8 Conclusion and Future Work

Our institutions for Csp use injective signature morphisms, due to the fact that the alphabet plays a double role, in the process syntax and the semantic domains, and both aspects are mapped covariantly — a contravariant mapping would destroy important laws for of Csp processes.

Languages like Unity and CommUnity [11] split the alphabet of communications into 'data' – to be translated covariantly – and 'actions' – to be translated contravariantly. The advantage of this approach is that the contravariant translation makes it possible to 'split' actions. We avoid such a partition of the alphabets of communications as Csp with its relational renaming already offers a means of 'splitting' an action on the term level. A rich set of algebraic laws allows to relates the new process with the old one.

We have demonstrated that with our Csp institutions, structured specifications have a semantics that fits with what one would expect in the Csp world. In particular, we can use loose semantics and parameterisation in combination with Csp refinement in a very useful way, going beyond what has been developed in the Csp community so far. Future work will extend the institutions presented here with an algebraic data type part, aiming at an institution for the language Csp-Casl [16]. For this, it is probably useful to distinguish between a syntactic and a semantic alphabet, at the price of complicating algebraic laws like the $\langle \Box$**-step**$\rangle$ law by using equality on the semantic alphabet in a subtle way, but with the advantage of allowing for non-injective alphabet translations.

Acknowledgment. The authors would like to thank José Fiadeiro, Yoshinao Isobe, Grigore Rosu, Erwin R. Catesbeiana, Andrzej Tarlecki, and Lutz Schröder for helpful discussions on what the right Csp institution might be.

References

1. J. Adámek, H. Herrlich, and G. Strecker. *Abstract and Concrete Categories.* Wiley, New York, 1990.
2. H. Baumeister. *Relations between Abstract Datatypes modeled as Abstract Datatypes.* PhD thesis, Universität des Saarlandes, 1998.
3. J. Bergstra, J. Heering, and P. Klint. Module algebra. *Journal of the ACM*, 37:335–372, 1990.
4. M. Bidoit and P. D. Mosses. Casl *User Manual.* LNCS 2900. Springer, 2004.
5. T. Borzyszkowski. Logical systems for structured specifications. *Theoretical Computer Science*, 286:197–245, 2002.

6. B. Buth, J. Peleska, and H. Shi. Combining methods for the livelock analysis of a fault-tolerant system. In *AMAST'98*, LNCS 1548. Springer, 1998.
7. B. Buth and M. Schrönen. Model-checking the architectural design of a fail-safe communication system for railway interlocking systems. In *FM'99*, LNCS 1709. Springer, 1999.
8. CoFI (The Common Framework Initiative). CASL *Reference Manual*. LNCS 2960. Springer Verlag, 2004.
9. R. Diaconescu, J. Goguen, and P. Stefaneas. Logical support for modularisation. In *Logical Environments*, pages 83–130. Cambridge, 1993.
10. H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 2*. Springer, 1990.
11. J. Fiadeiro. *Categories for Software Engineering*. Springer, 2004.
12. J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39:95–146, 1992.
13. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
14. Y. Isobe and M. Roggenbach. A complete axiomatic semantics for the CSP stable-failures model. In *CONCUR'06*, LNCS 4137. Springer, 2006.
15. T. Mossakowski, S. Autexier, and D. Hutter. Development graphs – proof management for structured specifications. *Journal of Logic and Algebraic Programming*, 67(1-2):114–145, 2006.
16. M. Roggenbach. CSP-CASL - a new integration of process algebra and algebraic specification. *Theoretical Computer Science*, 354(1):42–71, 2006.
17. M. Roggenbach and L. Schröder. Towards trustworthy specifications I: Consistency checks. In *WADT'01*, LNCS 2267. Springer, 2001.
18. A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998.
19. P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *The Modelling and Analysis of Security Protocols: the CSP Approach*. Addison-Wesley, 2001.
20. D. Sannella and A. Tarlecki. Specifications in an arbitrary institution. *Information and Computation*, 76:165–210, 1988.