Proof assistance for refinement in type theory

Nikos Mylonakis

March 20, 2000

Abstract

In this paper, we represent in type theory a proof system for refinement of algebraic specifications in $ASL$ [9]. The representation is not adequate but full because the use of proof obligations to represent side-conditions. Using this representation, we can develop a proof tactic to help the development of proofs of refinement.

1 Introduction

Type theories were initially used as a logical language for the foundations of mathematics. Since they also include a computational language (in particular a functional language), most of them have also been used as a framework for program development. Some expressive type theories have also been used as logical frameworks like for example the LF type theory [3].

There have been several attempts to design proof systems for the deduction of properties from algebraic specifications and for the refinement of algebraic specifications. In this paper, we concentrate on the proof systems for the refinement of $ASL$ [9] specifications presented in [4].

We use a new principle of encoding which improves the one used in LF, but not its underlying type theory. Instead we use the Uniform Theory of dependent Types ($UTT$ [5], [2]). Previous encodings of proof systems in $UTT$ by the same author [7] were adequate, in the sense that there existed a bijection between the closed derivations of a concrete judgement of the proof systems and the inhabitants of the application of the judgement to the inductive relation which encodes the proof systems. The encodings of the proof system for refinement presented in this paper is just full in the sense that there exists a total injective function $\epsilon_{ref}$ between the derivations of a concrete refinement judgement ($SP \gg SPI$) and the application of this judgement to the inductive relation which encodes the proof system for refinement. Another interesting property of $\epsilon_{ref}$ is that there exists a function $\epsilon_{ref}^{-1}$ which satisfies the following condition

$$\forall \delta \in \Delta_{H_{ASL} \gg SP} (SPI), \epsilon_{ref}^{-1} (\epsilon_{ref} \delta) = \delta$$

The encoding presented in this paper is not adequate because we use proof obligations with proof text to encode the side conditions of the proof system.
which are difficult to encode in type theory either because we can not find a syntactic characterization of the side condition or because the syntactic proofs of the side conditions are tedious or complicated.

In the paper, we first give the formal semantics of ASL and its refinement relation and after that we present the full encoding of the proof system for refinement of ASL. Before giving the encoding of the proof system, we present the adequate encodings of structured signatures and well formed specification expressions.

2 ASL

In this section, we present the formal semantics of some basic operators of ASL. The semantics of the language is inductively defined by the functions \textit{Signature} and \textit{Models}. The function \textit{Signature} must return the signature with just the visible symbols of the given specification and \textit{Models} must return the models which satisfy the specification.

We assume that the signatures are many-sorted first order signatures which form a category which is normally denoted by $\text{AlgSig}$ where morphisms are signature morphisms and inclusions are the obvious embeddings between signatures. This category has pushouts which are used for the semantics of structured specifications. The category of $\Sigma$-algebras for a given signature $\Sigma$ (which are the models of specifications) is denoted as $\text{Alg}(\Sigma)$ and see for example [1] and [8] for a semantics in an arbitrary but fixed institution [?]. In this paper, the sentences of the language are the sentences of first order logic, but we do not explicit its syntax since it is irrelevant for the setting. We will refer to the sentences of first order logic as $\text{Sen}_{FOL}(\Sigma)$ for a given signature $\Sigma$ and to the satisfaction relation between $\Sigma$-algebras and first-order sentences by $\models_{FOL,\Sigma}$.

\textbf{Definition 2.1} A reachability constraint for a given signature $\Sigma = (S, Op)$ is a pair of a set of sorts and a set of functions $(S_R, F_R)$ such that $S_R \subseteq S$ and for any $f : s_1 \times \ldots \times s_n \to s \in F_R, s \in S_R$.

\textbf{Definition 2.2} For any signature $\Sigma = (S, Op) \in \text{AlgSig}$, an algebra $A \in \text{Alg}(\Sigma)$ satisfies a reachability constraint $(S_R, F_R)$ of $\Sigma$ ($A \models (S_R, F_R)$) if the following condition holds:

$$A \models (S_R, F_R) \iff \forall s \in S_R \forall v \in A_s.$$ 

$$\exists t \in T_{(s, x)}(X_{s-s_e}). \exists \alpha : X_{s-s_e} \to A.I_\alpha(t) = v$$
Definition 2.3 The syntax of the operators of ASL is the following:

\[
SP_0 := < \Sigma, \Phi > \\
SP_1 \mid \Sigma \\
SP_1 +_\Sigma SP_2 \\
\text{rename } SP \text{ by } \sigma \\
\text{reach } SP \text{ with } (S_\mathcal{R}, F_\mathcal{R}) \\
\text{behaviour } SP \text{ wrt } \approx \\
\text{abstract } SP \text{ by } \equiv \\
SP/ \approx
\]

where the signature \( \Sigma = (S, Op) \in \text{AlgSig} \), \( \Phi \subseteq \text{Sen}_{FOL}(\Sigma) \), \( \sigma \) is a signature morphism, \( \approx \) is a partial congruence between elements of \( \Sigma \)-algebras and \( \approx \) is an equivalence relation between algebras. The semantics of the ASL operators is inductively defined as follows:

\[
\text{Signature}(< \Sigma, \Phi >) = \Sigma \\
\text{Models}(< \Sigma, \Phi >) = \{ A \mid A \models_{FOL, \Sigma} \Phi \}
\]

\[
\text{Signature}(\text{rename } SP \text{ by } \sigma) = \Sigma \\
\text{Models}(\text{rename } SP \text{ by } \sigma) = \{ A \in \text{Alg}(\Sigma) \mid A|_{\sigma} \in \text{Models}(SP) \}
\]

\[
\text{Signature}(SP|_{\Sigma}) = \Sigma \\
\text{Models}(SP|_{\Sigma}) = \{ A|_{\Sigma} \mid A \in \text{Models}(SP) \}
\]

where \( \Sigma \subseteq \text{Signature}(SP) \)
\[
\text{Signature}(SP_1 +_{\Sigma} SP_2) = \text{Signature}(SP_1) +_{\Sigma} \text{Signature}(SP_2)
\]

\[
\text{Models}(SP_1 +_{\Sigma} SP_2) = \{ A \mid A \in \text{Alg}(\text{Signature}(SP_1) +_{\Sigma} \text{Signature}(SP_2)), A|_{nt} \in \text{Models}(SP_1), A|_{nr} \in \text{Models}(SP_2) \}
\]

where \(SP_1, SP_2\) ranges over specification expressions,

\(\Sigma \subseteq \text{Signature}(SP_1), \Sigma \subseteq \text{Signature}(SP_2)\)

and \(\text{Signature}(SP_1) +_{\Sigma} \text{Signature}(SP_2)\) is the pushout of the two obvious inclusions between \(\Sigma\) and \(\text{Signature}(SP_1)\) and \(\Sigma\) and \(\text{Signature}(SP_2)\)

\[
\text{Signature}(\text{reach } SP \text{ with } (S_R, F_R)) = \text{Signature}(SP)
\]

\[
\text{Models}(\text{reach } SP \text{ with } (S_R, F_R)) = \{ A \in \text{Models}(SP) \mid A \models (S_R, F_R) \}
\]

\[
\text{Signature}(\text{behaviour } SP \text{ wrt } \approx) = \text{Signature}(SP)
\]

\[
\text{Models}(\text{behaviour } SP \text{ wrt } \approx) = \{ A/\approx \mid A \in \text{Models}(SP) \}
\]

\[
\text{Signature}(\text{abstract } SP \text{ by } \equiv) = \text{Signature}(SP)
\]

\[
\text{Models}(\text{abstract } SP \text{ by } \equiv) = \{ A \mid \exists B \in \text{Models}(SP), B \equiv A \}
\]

\[
\text{Signature}(SP/\approx) = \text{Signature}(SP)
\]

\[
\text{Models}(SP/\approx) = \{ A \mid \exists B \in \text{Models}(SP), B \approx A \}
\]

**Definition 2.4 Standard refinement:**

Assume that \(SP\) and \(SPI\) are specification expressions of ASL. \(SPI\) is a refinement of \(SP\) (denoted by \(SP \rightsquigarrow SPI\)) if the following two conditions are satisfied:

- \(\text{Signature}(SPI) = \text{Signature}(SP)\)
- \(\text{Models}(SPI) \subseteq \text{Models}(SP)\)
Notation: In the following, for any refinement $SP \rightsquigarrow SPI$ we will refer to $SP$ as the abstract specification and $SPI$ as the refined specification.

3 Encoding of signatures

The main technical problem in the encoding of structured signatures is that we have to differentiate between the new symbols introduced twice by a sum operator $SP_1 \Sigma SP_2$, which don’t belong to the common signature $\Sigma$. In order to differentiate these symbols, signatures are encoded with symbol indexes which are used to solve the name clashes in specification expressions with the sum operator.

Definition 3.1 For any $\Sigma \in \text{[AlgSig]}$, the inductive relation $\text{Sorts}$ is inductively defined by the following set of constructors:

$$\{ s_{\text{Sorts}} : \text{Sorts} \mid s \in \text{Sorts}(\Sigma) \}$$

Definition 3.2 For any $\Sigma \in \text{[AlgSig]}$, the function $\text{Eqbool}_{\text{Sorts}} : \text{Sorts} \to \text{Sorts} \to \text{Bool}$ is defined as follows:

$$\text{Eqbool}_{\text{Sorts}} s s' = \text{Primrec Sorts} (s_1 c s') \ldots (s_n c s') s$$

$$s_1 c s' = \text{Primrec Sorts} \text{true} \ldots \text{false} s'$$

$$\vdots$$

$$s_n c s' = \text{Primrec Sorts} \text{true} \ldots \text{false} s'$$

Definition 3.3 For any $\Sigma \in \text{[AlgSig]}$, the inductive relation $\text{Ops}$ is inductively defined by the following set of constructors:

$$\{ f_{\text{Ops}} : \text{Ops} \mid f : s_1 \times \ldots \times s_n \to s \in \Sigma \text{ and } f \text{ is not overloaded in } \Sigma \} \cup$$

$$\{ f : s_1 \times \ldots \times s_n \to s \in \Sigma \text{ and } f \text{ is overloaded in } \Sigma \}$$

Remark: We assume predefined the function $\text{Eqbool}_{\text{Ops}} : \text{Ops} \to \text{Ops} \to \text{Bool}$ defined as in the previous inductive type for the encoding of sorts.

Definition 3.4 The type $\text{Sym\_index}$ is inductively defined by the following set of constructors:

$$\text{first\_Si} : \text{Sym\_index}$$

$$\text{next\_Si} : \text{Sym\_index} \to \text{Sym\_index}$$

Remark: We assume predefined the function $\text{maxind\_Si} : \text{Sym\_index} \to \text{Sym\_index} \to \text{Sym\_index}$ which given two indexes returns the maximum of the two.
Definition 3.5 The type Ind\_sorts is defined as (Pair Sorts Sym\_index).

Definition 3.6 The type Ind\_ops is defined as (Pair Ops Sym\_index).

Definition 3.7 The type of signatures with indexes is defined as

\[ \text{Signature} = (\text{Pair} (\text{List} \text{Ind\_sorts}) (\text{List} \text{Ind\_ops})) \]

For simplicity and without loss of expressive power, we will assume a predefined total ordering between the sorts and operations of a given signature. This will avoid us to use quotient types by a permutation relation to represent signatures which are a little bit cumbersome and not really necessary for these encodings.

We will also assume predefined the following functions and inductive relations:

- the function \( \text{Ltbool\_Sorts} : \text{Ind\_sorts} \to \text{Ind\_sorts} \to \text{Bool} \) which given two indexed sorts \( s_1, s_2 \) returns true if \( s_1 \) is lower than \( s_2 \) and false otherwise.

- the function \( \text{Ltbool\_Ops} : \text{Ind\_ops} \to \text{Ind\_ops} \to \text{Bool} \) and the functions \( \text{Eqbool\_Sorts} : \text{Ind\_sorts} \to \text{Ind\_sorts} \to \text{Bool} \) and \( \text{Eqbool\_Ops} : \text{Ind\_ops} \to \text{Ind\_ops} \to \text{Bool} \).

- the functions \( \text{sort\_sl} : \text{List} \text{Ind\_sorts} \to \text{List} \text{Ind\_Sorts} \) which given a list of indexed sorts, sorts the given list eliminating repeated elements and analogously \( \text{sort\_opl} : \text{List} \text{Ind\_ops} \to \text{List} \text{Ind\_ops} \). See [6] for a verified algorithm for sorting in UTT using primitive recursion.

- the inductive relations \( \text{Sorted\_sl} : \text{List} \text{Ind\_sorts} \to \text{Prop} \) and \( \text{Sorted\_opl} : \text{List} \text{Ind\_ops} \to \text{Prop} \) which check that the lists are sorted.

The well formedness of indexed signatures is checked with the following inductive relation

Definition 3.8 The inductive relation

\[ Wf\_signature : \Pi sign : \text{Signature} \cdot \text{Prop} \]

is defined by the following constructors:

- \( Wf\_signature : \Pi sl : \text{List} \text{Ind\_sorts}, \Pi opl : \text{List} \text{Ind\_ops} \).

\[ \Pi nrsl : \text{Norep}\_\text{List} \text{Ind\_sorts} \text{Eqbool\_Sorts} \_sl. \]

\[ \Pi nropl : \text{Norep}\_\text{List} \text{Ind\_ops} \text{Eqbool\_Ops} \_opl. \]

\[ \Pi ssl : \text{Sorted} \_sl, \Pi sopl : \text{Sorted} \_opl. \]

\[ Wf\_signature (sl, opl) \]
4 Encoding of ASL specifications

In this section, we define and represent well formed specifications which can be inductively defined by the following set of rules:

**Definition 4.1** The set of well formed specifications closed by a set of free variables $X$ (denoted as $X \triangleright SP$) is inductively defined by the following rules:

\[
\frac{X \triangleright SP_1 \quad X \triangleright SP_2}{X \triangleright SP_1 +_SP SP_2} \quad \Sigma \subseteq \text{Sign}(SP_1) \land \Sigma \subseteq \text{Sign}(SP_2) \quad (\text{sum}_wfs)
\]

\[
\frac{X \triangleright SP}{X \triangleright SP} \quad \Sigma \subseteq SP \quad (\text{export}_wfs)
\]

\[
\frac{X \triangleright SP}{X \triangleright \text{rename } SP \text{ by } \sigma} \quad \text{Bij}(\text{Sign}(SP), \Sigma, \sigma) \quad (\text{rename}_wfs)
\]

\[
\frac{X \triangleright SP}{X \triangleright \text{reach } SP \text{ with } (S_R, F_R) \subseteq \text{Sign}(SP)} \quad (\text{reach}_wfs)
\]

\[
\frac{X \triangleright SP}{X \triangleright \text{behaviour } SP \text{ wrt } In, \text{Obs} \subseteq \text{Sign}(SP)} \quad (\text{behaviour}_wfs)
\]

\[
\frac{X \triangleright SP}{X \triangleright \text{abstract } SP \text{ by } \equiv} \quad \text{In, Obs} \subseteq \text{Sign}(SP) \quad (\text{abstract}_wfs)
\]

\[
\frac{X \triangleright SP}{X \triangleright \text{quotient } SP \text{ by } \approx} \quad \text{In, Obs} \subseteq \text{Sign}(SP) \quad (\text{quotient}_wfs)
\]

where $\text{Bij}(\text{Signature}(SP), \Sigma, \sigma)$ stands for the following condition:

\[
\text{Bij}(\text{Signature}(SP), \Sigma, \sigma) = (\text{Dom}(\sigma) = \text{Sign}(SP)) \land \\
\forall s, s' \in \text{Sorts}(\text{Sign}(SP)), \sigma(s) = \sigma(s') \supset s = s' \land \\
\forall s \in \text{Sorts}(\Sigma), \exists s' \in \text{Sorts}(\text{Sign}(SP)), \sigma(s') = s
\]

\[
\forall op : s_1 \times \ldots \times s_n \rightarrow s \in \text{Ops}(\text{Sign}(SP)), \forall op' : s'_1 \times \ldots \times s'_n \rightarrow s' \in \text{Ops}(\text{Sign}(SP)), \\
\sigma(op : s_1 \times \ldots \times s_n \rightarrow s) = \sigma(op' : s'_1 \times \ldots \times s'_n \rightarrow s') \supset \\
\sigma(op : s_1 \times \ldots \times s_n \rightarrow s) = op' : s'_1 \times \ldots \times s'_n \rightarrow s' \\
\forall op : s_1 \times \ldots \times s_n \rightarrow s \in \text{Ops}(\Sigma), \exists op' : s'_1 \times \ldots \times s'_n \rightarrow s' \in \text{Ops}(\text{Sign}(SP)), \\
\sigma(op' : s'_1 \times \ldots \times s'_n \rightarrow s') = op : s_1 \times \ldots \times s_n \rightarrow s
\]

7
and we assume predefined the relation $X \triangleright \phi$ which checks that the formula $\phi$ is a well-formed formula closed by $X$.

We assume predefined the inductive types $\textit{Formula}$ which defines first-order formulas and $\textit{Var_set}$ which defines variable sets. We also assume predefined the inductive relations $\textit{Wf_form} : \Pi vs : \textit{Var_set} . \Pi \textit{form} : \textit{Formula}.Prop$ and $\textit{Wf_forml} : \Pi vs : \textit{Var_set} . \Pi \textit{form} : \textit{List Formula}.Prop$ which check that the given formula and list of formulas are well-formed and closed by $vs$.

**Definition 4.2** The type signature morphism is defined as follows:

$\textit{Signature_morphism} = \text{Pair Signature}$

$\text{Pair (List (Pair \textit{Ind_sorts} \textit{Ind_sorts})) (List (Pair \textit{Ind_ops} \textit{Ind_ops}))}$

In the appendix, one can find the following operations on signature morphisms:

- $\text{get\_dom\_sm} : \textit{Signature_morphism} \rightarrow \textit{Signature}$ which given a signature morphism, returns the domain of the signature morphism.
- $\text{get\_ran\_sm} : \textit{Signature_morphism} \rightarrow \textit{Signature}$ which given a signature morphism, returns the range of the signature morphism.
- $\text{inverse\_sm} : \textit{Signature_morphism} \rightarrow \textit{Signature_morphism}$ which given a signature morphism, returns the inverse of the signature morphism.

**Definition 4.3** The inductive type $\textit{Specification}$ is defined by the following set of constructors:

$\textit{base\_spec} : \textit{Signature} \rightarrow (\textit{List \textit{Formula}}) \rightarrow \textit{Specification}$

$\textit{sum\_spec} : \textit{Specification} \rightarrow \textit{Signature} \rightarrow \textit{Specification} \rightarrow \textit{Specification}$

$\textit{export\_spec} : \textit{Specification} \rightarrow \textit{Signature} \rightarrow \textit{Specification}$

$\textit{rename\_spec} : \textit{Specification} \rightarrow \textit{Signature_morphism} \rightarrow \textit{Specification}$

$\textit{reach\_spec} : \textit{Specification} \rightarrow \textit{Signature} \rightarrow \textit{Specification}$

$\textit{behaviour\_spec} : \textit{Specification} \rightarrow (\textit{List \textit{Ind_sorts}}) \rightarrow (\textit{List \textit{Ind_sorts}})$

$\rightarrow \textit{Specification}$
abstract_spec : Specification -> (List Ind_sorts) -> (List Ind_sorts)
               -> Specification

quotient_spec : Specification -> (List Ind_sorts) -> (List Ind_sorts)
               -> Specification

In the appendix, you can also find the following operations on signatures and specification expressions:

- new_index : Signature -> Sym_index -> Signature which given a signature and a symbol index assigns the symbol index to all the sorts and operations of the signature.

- union_Sign : Signature -> Signature -> Signature which given two signatures, returns the union of the two signatures.

- intersect_Sign : Signature -> Signature -> Signature which given two signatures, returns the intersection of the two signatures.

- diff_Sign : Signature -> Signature -> Signature which given two signatures, returns the difference of the first by the second signature.

- nameclash_Sign : Signature -> Signature -> Signature -> Signature which given three signatures returns the signature which is the intersection of the first and third and has no symbols of the second.

- Signature_sp : Specification -> Signature which given a specification expression, returns the signature of the specification.

And in the same appendix, we present the following inductive relations which are useful for the definition of the inductive relation which represents well-formed specifications:

- Same_signature : reflexivity : Signature.Prop which given two signatures checks whether they are the same.

- Subsignature : signature signature' : Signature.Prop which given two subsignatures, checks whether the first is subsignature of the second.

- Subsorts : list : List Ind_sorts, signature' : Signature.Prop which given a list of sorts and a signature checks whether the list of sorts is included in the sorts of the signature.

- Bijective : signature : Signature.relativ : Signature_morphism.Prop which given a signature and a signature morphism, checks whether the domain of the signature morphism is the same as the given signature and the signature morphism is bijective.
The following definition represents well-formed specifications:

**Definition 4.4** The inductive relation

\[ WfSpec : \Pi \text{Spec} : \Pi \text{Var} : \Pi \text{Spec} : \text{Signature} \]

is defined by the following set of constructors:

\[
\begin{align*}
\text{base}_wfspec & : \Pi \text{Spec} : \Pi \text{Var} \times \Pi \text{Spec} : \text{Signature}. \\
\text{Hfl} & : \Pi \text{list} \times \Pi \text{Form} \times \Pi \text{Spec} : \text{Signature}.
\end{align*}
\]

\[
\begin{align*}
\Pi \text{wfl} & : \Pi \text{Spec} \times \Pi \text{Var} \times \Pi \text{Spec} \times \Pi \text{Spec} : \Pi \text{Spec} \times \Pi \text{Var} \times \Pi \text{Spec} \times \Pi \text{Spec}.
\end{align*}
\]

\[
\begin{align*}
\text{sum}_wfspec & : \Pi \text{Var} : \Pi \text{Spec} : \Pi \text{Spec} \times \Pi \text{Spec} \times \Pi \text{Spec} : \Pi \text{Spec} \times \Pi \text{Var} \times \Pi \text{Spec} \times \Pi \text{Spec}.
\end{align*}
\]

\[
\begin{align*}
\text{export}_wfspec & : \Pi \text{Var} : \Pi \text{Var} \times \Pi \text{Spec} \times \Pi \text{Spec} \times \Pi \text{Var} \times \Pi \text{Spec} \times \Pi \text{Spec}.
\end{align*}
\]

\[
\begin{align*}
\text{rename}_wfspec & : \Pi \text{Var} : \Pi \text{Var} \times \Pi \text{Var} \times \Pi \text{Var} \times \Pi \text{Var} \times \Pi \text{Var} \times \Pi \text{Var}.
\end{align*}
\]

10
5 Encoding of the proof system for refinement

In this section, we present the encoding of the proof system for refinement. It uses the following preliminary definition:

**Definition 5.1** Let ASL be an ASLker specification language with an arbitrary but fixed algebraic institution AINS. Assume that $SP$ and $SP'$ are specification expressions. $SP$ is a persistent extension of $SP'$ (denoted by $PEXTOF(SP, SP')$) if the following condition holds:

- There exists an inclusion with arity $\text{Signature}(SP') \hookrightarrow \text{Signature}(SP)$
- $\text{Models}(SP) = \text{Models}(SP')|_{\text{Signature}(SP)}$. 


This proof system is inductively defined by the abstract specification expression as follows:

\[
\begin{align*}
\text{(basic)} & \quad <\Sigma, \Phi > \gg_{X} SPI \quad \text{Signature}(SPI) = \Sigma \land (SPI \models \Phi) \\
\text{(sum)} & \quad SP' \gg_{X} \text{rename} SPI|_{\text{init}(\text{Signature}(SPI))} \quad \text{by} \quad \text{inrsig}^{-1} \\
& \quad SP \gg_{X} \text{rename} SPI|_{\text{init}(\text{Signature}(SPI))} \quad \text{by} \quad \text{inlsig}^{-1} \\
\text{(sum)} & \quad SP + \Sigma SP' \gg_{X} SPI \\
\text{(export)} & \quad X \triangleright SPI' \quad \text{Signature}(SPI) = \Sigma \land \text{PEXTOF}(SPI', SPI) \\
\text{(reach)} & \quad \text{reach} SP \quad \text{with} \quad (S_{R}, F_{R}) \gg_{X} SPI \quad \text{Mod}(SPI) \models (S_{R}, F_{R}) \\
\text{(rename)} & \quad SP \gg_{X} \text{rename} SPI \quad \text{by} \quad \sigma^{-1} \\
\text{rename} & \quad SP \quad \text{by} \quad \sigma \gg_{X} SPI \\
\text{(behaviour)} & \quad SP \gg_{X} SPI / \approx \quad \text{behaviour} SP \quad \text{wrt} \quad \approx \gg_{X} SPI \\
\text{(abstract)} & \quad \text{behaviour} SP \quad \text{wrt} \quad \approx \gg_{X} SPI \quad \text{abstract} SP \quad \text{by} \quad \equiv \gg_{X} SPI \quad \text{Behc}(SPI) \\
\text{(quotient)} & \quad X \triangleright SPI' \quad \text{Cond}(SPI, SPI', SPI') \\
\text{where} & \quad \text{Cond}(SPI, SPI', SPI') = (\text{Signature}(SPI) = \text{Signature}(SPI) \land \\
& \quad \text{Mod}(SPI' / \approx) = \text{Mod}(SPI)) \\
& \quad \text{Behc}(SPI) = \text{Models}(SPI) \subseteq \text{Models(behaviour SP wrt \approx)} \\
\text{and} & \quad \text{inl : Signature}(SPI) \rightarrow \text{Signature}(SPI) + \Sigma \text{Signature}(SPI')
\end{align*}
\]
and

\[ \text{inr} : \text{Signature}(SP') \rightarrow \text{Signature}(SP) + \Sigma \text{Signature}(SP') \]

are the pushouts morphisms of \( i : \Sigma \ni \text{Signature}(SP) \) and \( ? : \Sigma \ni \text{Signature}(SP') \), \( \text{inl}(\text{Signature}(SP)) \), \( \text{inr}(\text{Signature}(SP')) \) are the obvious subsignatures of \( \text{Signature}(SP) + \Sigma \text{Signature}(SP') \) and \( \text{inl}_{\text{signature}} : \text{Signature}(SP) \rightarrow \text{inl}(\text{Signature}(SP)) \) and \( \text{inr}_{\text{signature}} : \text{Signature}(SP') \rightarrow \text{inr}(\text{Signature}(SP')) \) are the obvious signature morphisms defined with the pushouts morphisms \( \text{inl} \) and \( \text{inr} \).

The proof of the following theorem can be found in [1] and in [4].

**Theorem 5.1** For any specification expressions \( SP \) and \( SPI \), \( SP \rightsquigarrow SPI \) if and only if there exists a derivation of the sequent \( SP \Rightarrow SPI \) in \( \Delta_{\Pi_{\text{ASL}}}^{\text{ASL}}(SP \Rightarrow SPI) \)

For the definition of the proof system for refinement we need the resulting signatures after applying a pushout morphism \( (\text{inl}, \text{inr}) \) to the signatures of the left and right specification expressions of a sum operator respectively. Apart from these two definitions, we need also the definitions of the pushout morphisms associated to the three signatures of a sum operator. These definitions are also in the appendix and they have the following names and arities:

- \( \text{inl}_{\text{sums}} : \text{Specification} \rightarrow \text{Signature} \rightarrow \text{Specification} \rightarrow \text{Signature} \)
- \( \text{inr}_{\text{sums}} : \text{Specification} \rightarrow \text{Signature} \rightarrow \text{Specification} \rightarrow \text{Signature} \)
- \( \text{inlsm}_{\text{sums}} : \text{Specification} \rightarrow \text{Signature} \rightarrow \text{Specification} \rightarrow \text{Signature}_{\text{morphism}} \)
- \( \text{inrsm}_{\text{sums}} : \text{Specification} \rightarrow \text{Signature} \rightarrow \text{Specification} \rightarrow \text{Signature}_{\text{morphism}} \)

Now, we define the inductive relations which represent the proof obligations of the proof system.

**Definition 5.2** The type \( \text{Proof}_{\text{symbol}} \) is inductively defined by this incomplete set of constructors:

\[
a, \ldots, z : \text{Var}_{\text{symbol}} \\
A, \ldots, Z : \text{Var}_{\text{symbol}} \\
\_\_\_\_\_ : \text{Var}_{\text{symbol}} \\
\]

**Definition 5.3** The type \( \text{Proof}_{\text{text}} \) is defined as \( \text{Nil} \text{List Proof}_{\text{text}} \).

**Definition 5.4** The inductive relation

\[
\text{Basic}_{\text{po}} : \Pi_{\text{sp}} \text{Specification} \Pi_{\text{fl}} \text{List Formula} \Pi_{\text{pt}} \text{Proof}_{\text{text}} \text{Prop}
\]
is defined by the following constructors:

\[
\text{basic}_{\Pi} \cdot \Pi_{sp} : \text{Specification}, \Pi_{fl} : \text{List Formula}, \Pi_{pt} : \text{Proof text}.
\]

\[
\text{Basic}_{\Pi} \cdot \Pi_{sp} \Pi_{fl} \Pi_{pt}
\]

**Definition 5.5** The inductive relation

\[
\text{Pext}_{\Pi} \cdot \Pi_{sp}, \Pi_{sp'} : \text{Specification}, \Pi_{pt} : \text{Proof text}, \Pi_{prop}
\]

is defined by the following constructors:

\[
\text{pext}_{\Pi} \cdot \Pi_{sp}, \Pi_{sp'} : \text{Specification}, \Pi_{pt} : \text{Proof text}.
\]

\[
\text{Pext}_{\Pi} \cdot \Pi_{sp} \Pi_{sp'} \Pi_{pt}
\]

**Definition 5.6** The inductive relation

\[
\text{Reach}_{\Pi} \cdot \Pi_{sp} : \text{Specification}, \Pi_{rsign} : \text{Signature}, \Pi_{pt} : \text{Proof text}, \Pi_{prop}
\]

is defined by the following constructors:

\[
\text{reach}_{\Pi} \cdot \Pi_{sp} : \text{Specification}, \Pi_{rsign} : \text{Signature}, \Pi_{pt} : \text{Proof text}.
\]

\[
\text{Reach}_{\Pi} \cdot \Pi_{sp} \Pi_{rsign} \Pi_{pt}
\]

**Definition 5.7** The inductive relation

\[
\text{Behcomp}_{\Pi} \cdot \Pi_{sp} : \text{Specification}, \Pi_{pt} : \text{Proof text}, \Pi_{prop}
\]

is defined by the following constructors:

\[
\text{behcomp}_{\Pi} \cdot \Pi_{sp} : \text{Specification}, \Pi_{pt} : \text{Proof text}.
\]

\[
\text{Behcomp}_{\Pi} \cdot \Pi_{sp} \Pi_{pt}
\]

**Definition 5.8** The inductive relation

\[
\text{Qmodeq}_{\Pi} \cdot \Pi_{sp}, \Pi_{sp'} : \text{Specification}, \Pi_{pt} : \text{Proof text}, \Pi_{prop}
\]

is defined by the following constructors:

\[
\text{qmodeq}_{\Pi} \cdot \Pi_{sp}, \Pi_{sp'} : \text{Specification}, \Pi_{pt} : \text{Proof text}.
\]

\[
\text{qmodeq}_{\Pi} \cdot \Pi_{sp} \Pi_{sp'} \Pi_{pt}
\]
**Definition 5.9** The inductive relation

\[ \text{Proof obligation} : \text{Prop} \]

is defined by the following constructors:

- \( \text{basicpo} \circ \text{cc} : \Pi \text{sp} : \text{Specification} \Pi \text{fl} : \text{List Formula} \Pi \text{pt} : \text{Proof text} \). 
- \( \Pi \text{pr} : \text{Basic} \circ \text{po} \circ \text{sp} \circ \text{fl} \circ \text{pt}. \text{Proof obligation} \)
- \( \text{pxtppo} \circ \text{cc} : \Pi \text{sp} , \text{sp} ' : \text{Specification} \Pi \text{pt} : \text{Proof text} \).
- \( \Pi \text{pr} : \text{Pext} \circ \text{po} \circ \text{sp} \circ \text{sp} ' \circ \text{pt}. \text{Proof obligation} \)
- \( \text{reachpo} \circ \text{cc} : \Pi \text{sp} : \text{Specification} \Pi \text{rsign} : \text{Signature} \Pi \text{pt} : \text{Proof text} \).
- \( \Pi \text{pr} : \text{Reach} \circ \text{po} \circ \text{sp} \circ \text{rsign} \circ \text{pt}. \text{Proof obligation} \)
- \( \text{behecomp} \circ \text{cc} : \Pi \text{sp} : \text{Specification} \Pi \text{pt} : \text{Proof text} \).
- \( \Pi \text{pr} : \text{Behecomp} \circ \text{po} \circ \text{sp} \circ \text{pt}. \text{Proof obligation} \)
- \( \text{qmodeqpo} \circ \text{cc} : \Pi \text{sp} , \text{sp} ' : \text{Specification} \Pi \text{pt} : \text{Proof text} \).
- \( \Pi \text{pr} : \text{Qmodeq} \circ \text{po} \circ \text{sp} \circ \text{sp} ' \circ \text{pt}. \text{Proof obligation} \)

And finally, we define the inductive relation which represents the proof system for refinement and we present the theorem which establishes the adequacy of the representation.

**Definition 5.10** The inductive relation

\[ \text{RefineASLFOL} : \Pi \text{sp} : \text{Specification} \Pi \text{vs} : \text{Var set} \Pi \text{sp} ' : \text{Specification} \Pi \text{prop} \]

is defined by the following set of constructors:

- \( \text{basic_ref} : \Pi \text{vs} : \text{Var set} \Pi \text{sign} : \text{Signature} \Pi \text{fl} : \text{List Formula} \).
- \( \Pi \text{wfl} : \text{Wform} \circ \text{vs} \circ \text{fl} \circ \text{sp} : \text{Specification} \Pi \text{pt} : \text{Proof text} \).
- \( \Pi \text{same} : \text{Same} \circ \text{signature} \circ \text{sign} (\text{Signature} \circ \text{sp} \circ \text{sp}) \Pi \text{bpo} : \text{Basic} \circ \text{po} \circ \text{sp} \circ \text{fl} \circ \text{pt} \).

\[ \text{RefineASLFOL} (\text{base} \circ \text{spec} \circ \text{sign} \circ \text{fl}) \circ \text{vs} \circ \text{sp} \]
sum_ref : \Pi_{sp, sp', spi : \text{Specification}}. \Pi_{\text{sign} : \text{Signature}}. \Pi_{vs : \text{Var_set}}.

\Pi_{ref sp : \text{Refine_ASLFOL} \ sp \ vs}

\ (rename_{spec} (export_{spec} \ spi \ (inl\_sums \ sp \ \text{sign} \ sp'))

\ (inverse(inlsm\_sums \ sp \ \text{sign} \ sp')).

\Pi_{ref sp' : \text{Refine_ASLFOL} \ sp' \ vs}

\ (rename_{spec} (export_{spec} \ spi \ (inv\_sums \ sp \ \text{sign} \ sp'))

\ (inverse(inrms\_sums \ sp \ \text{sign} \ sp')).

Refine_ASLFOL (sum_{spec} \ sp \ \text{sign} \ sp') \ vs \ spi

rename_ref : \Pi_{vs : \text{Var_set}}. \Pi_{sp, spi : \text{Specification}}. \Pi_{sm : \text{Signature_morphism}}.

\Pi_{ref sp : \text{Refine_ASLFOL} \ sp \ vs \ (rename_{spec} \ spi \ (inverse_{sm} \ sm))}.

Refine_ASLFOL (rename_{spec} \ spi \ sm) \ vs \ sp

exp_ref : \Pi_{vs : \text{Var_set}}. \Pi_{sp, spi, spi' : \text{Specification}}.

\Pi_{\text{sign} : \text{Signature}}. \Pi_{\text{pt} : \text{Proof_text}}

\Pi_{wfsp' : \text{Wf specification} \ vs \ spi'}.

\Pi_{same_s : \text{Same signature sign} \ (\text{Signature} \ sp \ spi)}. \Pi_{bpo : \text{Bextof} \ po \ spi' \ spi \ \text{pt}}

\Pi_{ref sp : \text{Refine_ASLFOL} \ sp \ vs \ spi'}.

Refine_ASLFOL (export_{spec} \ sp \ \text{sign}) \ vs \ spi
ref\_reach : \Pi vs : \text{Var\_set:\Pi sp, spi : Specification.}

\Pi \text{sign : Signature.\Pi pt : Proof\_text.}

\Pi \text{reachpo : Reach\_po sp sign pt.}

\Pi \text{ref \_fr : RefineASLFO\_L sp vs spi}

\text{RefineASLFO\_L (reach\_spec sp sign) vs spi}

\text{ref\_behaviour : \Pi vs : \text{Var\_set:\Pi sp, spi : Specification.\Pi sl, sl' : List\_Ind\_sorts.}}

\Pi \text{ref \_fr : RefineASLFO\_L sp vs (quotient\_spec spi sl sl')}

\text{RefineASLFO\_L (behaviour\_spec sp sl sl') vs spi}

\text{ref\_abstract : \Pi vs : \text{Var\_set:\Pi sp, spi : Specification.}}

\Pi \text{sl, sl' : List\_Ind\_sorts.\Pi pt : Proof\_text}

\Pi \text{ref \_fr : RefineASLFO\_L (behaviour\_spec sp sl sl') vs spi}

\text{beh\_po : Behcomp\_po sp pt.}

\text{RefineASLFO\_L (abstract\_spec sp sl sl') vs spi}

\text{ref\_quotient : \Pi vs : \text{Var\_set:\Pi sp, spi, spi' : Specification.}}

\Pi \text{sl, sl' : List\_Ind\_sorts.\Pi pt : Proof\_text}

\Pi \text{wfs\_spi' : Wf\_specification vs spi'.}

\Pi \text{ref \_fr : RefineASLFO\_L sp vs spi'.}

\Pi \text{sams : Same\_signature \text{(Signature\_sp sp) (Signature\_sp spi).}}

\Pi \text{beh\_po : Qmodeq\_po spi spi' pt.}

\text{RefineASLFO\_L (quotient\_spec sp sl sl') vs spi}

17
Assuming predefined the following encoding and decoding functions on well formed specification:

\[ \epsilon_{sp} : \text{Var}_\text{set} \rightarrow \text{SP EX}(\text{ASL}) \rightarrow \text{Specification} \]

\[ \epsilon_{sp}^{-1} : \text{Var}_\text{set} \rightarrow \text{Specification} \rightarrow \text{SP EX}(\text{ASL}) \]

where \( \text{SP EX}(\text{ASL}) \) denotes the set of specification expressions of \( \text{ASL} \), we can prove the following theorem:

**Theorem 5.2** For any sequence of variables \( X \), for any specification expression \( sp, sp' \in \text{SP EX}(\text{ASL}) \) such that \( X \triangleright sp \) and \( X \triangleright sp' \), there exists a total injective function \( \epsilon_{ref} \) between closed derivations of the judgement \( sp \gg sp' \) and the inhabitants of the inductive relation

\[ \text{Ref ineASLFOL} \left( \epsilon_{sp} \ (\epsilon_{vs} \ X) \ sp \right) \ (\epsilon_{vs} \ X) \ (\epsilon_{sp} \ (\epsilon_{vs} \ X) \ sp') \]

, and there exists an injective function \( \epsilon_{ref}^{-1} \) such that for all derivations \( \delta \) of the judgement \( sp \gg_X sp' \), \( \epsilon_{ref}^{-1} \ (\epsilon_{ref} \ \delta) = \delta \)

**Proof 5.1** The proof is similar to the ones presented for the encoding of the proof systems presented in [8] but obviously a little bit simpler and the definition of the function \( \epsilon_{ref}^{-1} \) is performed in the same way as in the same proof systems.

### 6 A tactic for proofs of refinement

In this section we present a tactic to assist the developments of proofs of refinement. We define a functional program which given two specification expressions and a variable set, builds interactively a proof which shows that the second specification expression is a refinement of the first listing the proofs obligations which must be externally proved in order to guarantee the correctness of the proof. If it is not possible to give the refinement proof, the tactic fails and it is denoted by the predefined exception \( \text{Fail}_{\text{ref}} \).

The functional program is inductively defined by the first specification expression because of the way the proof system is defined, and it requires to raise proof obligations for the basic, export, reach, abstract and quotient operator. The interactivity is needed to determine the specification expression in the export and quotient operator which has to be a refinement of the subspecification of the export and quotient operator respectively. To achieve this interaction, we assume predefined a function \( \text{get}_\text{wspec} \) which gets from the input a specification expression together with a proof which is well formed.

The function which will be denoted as \( \text{Ref}_{\text{tactic}} \) is inductively defined as
follows:
\[ \text{Ref\_tactic (base\_spec \_sign \_fl) vs \_sp} = \]
\[
\begin{align*}
&\text{if } (\text{fst (same\_signature} f \text{ sign (Signature sp}})) \text{ then} \\
&\quad (\text{basic\_ref vs sign fl sp "BASIC\_PO"}) \\
&\quad (\text{snd (same\_signature} f \text{ sign (Signature sp}})) \\
&\quad (\text{basicpo\_sp fl "BASIC\_PO"}), \\
&\quad [\text{basicpo\_sp fl "BASIC\_PO" (basicpo\_sp fl "BASIC\_PO")}] \\
&\text{else Fail\_ref}
\end{align*}
\]
\[ \text{Ref\_tactic (sum\_spec \_sp \_sign \_sp')} vs \_sp'' = \]
\[
\begin{align*}
&\text{(sum\_ref sp sp' sp'' sign vs (fst ref\_tactsp1) (fst ref\_tactsp2)}, \\
&\quad \text{concat (snd ref\_tactsp1) (snd ref\_tactsp2))} \\
&\text{where} \\
&\quad \text{ref\_tactsp1} = (\text{Ref\_tactic sp vs} \\
&\quad (\text{rename\_spec (export\_spec sp' (inl\_sums sp sign sp'))}) \\
&\quad (\text{inverse(inlsm\_sums sp sign sp')}), \\
&\quad \text{ref\_tactsp2} = (\text{Ref\_tactic sp vs} \\
&\quad (\text{rename\_spec (export\_spec sp'' (inr\_sums sp sign sp'))}) \\
&\quad (\text{inverse(inrsm\_sums sp sign sp')}).
\end{align*}
\]
\[ \text{Ref\_tactic (rename\_spec \_sp \_sign\_nm) vs \_sp' =} \]
\[
\begin{align*}
&\text{(ren\_ref vs sp sp' sm(fst ref\_tactsp), (snd ref\_tactsp))} \\
&\text{where} \\
&\quad \text{ref\_tactsp} = \text{Ref\_tactic sp vs (rename\_spec \_sp' (inverse\_sm sm)}
\end{align*}
\]
$$Ref\_jactic\ (export\_spec\ sp\ sign)\ vs\ sp' =$$

$$if\ (fst\ (same\_signature\ f\ sign\ (Signature\_sp\ sp')))\ then$$

$$(exp\_ref\ vs\ sp\ sp'\ (fst\ getsp)\ sign\ "EXPORT\_PO")$$

$$(snd\ getsp)\ (snd\ (same\_signature\ f\ sign\ (Signature\_sp\ sp')))$$

$$(\text{extend}\ (fst\ getsp)\ sp'\ "EXPORT\_PO")\ (fst\ ref\_tactsp),$$

$$(\text{extend}\ (fst\ getsp)\ sp'\ "EXPORT\_PO")\ (snd\ ref\_tactsp))$$

$$else\ F ail\_ref$$

$$where$$

$$gets p = get\_w\ f\ spec$$

$$ref\_tactsp = Ref\_jactic\ sp\ (fst\ getsp)$$

$$Ref\_jactic\ (reach\_spec\ sp\ sign)\ vs\ sp' =$$

$$(ref\_reach\ vs\ sp\ sp'\ sign\ "REACH\_PO")$$

$$(reach\_pc\ sp\ sign\ "REACH\_PO")$$

$$(fst\ ref\_tactsp),\ \text{cons}\ Proof\_obligation\ (reach\_pc\ sp\ sign\ "REACH\_PO")$$

$$(reach\_pc\ sp\ sign\ "REACH\_PO")\ (snd\ ref\_tactsp))$$

$$where$$

$$ref\_tactsp = Ref\_jactic\ sp\ vs\ sp'$$

20
\[ \text{Ref\_tactic}(\text{behaviour\_spec } sp \text{ obsl } inl) \text{ vs } sp' = \]

\[ (\text{ref\_behaviour vs } sp \text{ sp'} \text{ obsl } inl (\text{fst } \text{ref\_tactsp}) , (\text{snd } \text{ref\_tactsp})) \]

where
\[ \text{ref\_tactsp} = \text{Ref\_tactic } sp \text{ vs (quotient\_spec } sp' \text{ obsl } inl) \]

\[ \text{Ref\_tactic}(\text{abstract\_spec } sp \text{ obsl } inl) \text{ vs } sp' = \]

\[ (\text{ref\_abstract vs } sp \text{ sp'} \text{ obsl } inl (\text{fst } \text{ref\_tactsp}) (\text{behcomp\_spec } sp \text{ “ABSTRACT\_PO"}), \]

\[ \text{cons } \text{Proof\_Obligation } (\text{behcomp\_spec } sp \text{ “ABSTRACT\_PO"}) \]

\[ (\text{behcomp\_spec } sp \text{ “ABSTRACT\_PO"}) (\text{snd } \text{ref\_tactsp})) \]

where
\[ \text{ref\_tactsp} = \text{Ref\_tactic } (\text{behaviour\_spec } sp \text{ obsl } inl) \text{ vs } sp \]

\[ \text{Ref\_tactic}(\text{quotient\_spec } sp \text{ obsl } inl) \text{ vs } sp' = \]

\[ \text{if } (\text{fst } (\text{same\_signature } f (\text{Signature\_sp } sp) (\text{Signature\_sp } sp'))) \text{ then } \]

\[ (\text{ref\_quotient vs } sp \text{ sp'} (\text{fst } \text{getsp}) \text{ obsl } inl \text{ “QUOTIENT\_PO"}) \]

\[ (\text{snd } \text{getsp}) (\text{fst } \text{ref\_tactsp}) (\text{snd } (\text{same\_signature } f) \]

\[ (\text{Signature\_sp } sp) (\text{Signature\_sp sp'}))] \]

\[ (\text{qmodeq\_spec } sp \text{ sp'} \text{ “QUOTIENT\_PO"}), \text{ cons } \text{Proof\_Obligation} \]

\[ (\text{qmodeq\_spec sp sp'} (\text{qmodeq\_spec } sp \text{ sp'} \text{ “QUOTIENT\_PO"}))) \]

\[ \text{else } \text{Fail } \text{ref} \]

where
\[ \text{getsp} = \text{get\_wfspec} \]
\[ \text{ref\_tactsp} = \text{Ref\_tactic } sp \text{ vs (fst } \text{get\_wfspec)} \]

where we assume predefined the function same\_signature\_f which given two signatures returns a boolean which states whether the two signatures are equal or not, and a proof that the two signatures are equal which is an inhabitant.
of the inductive relation Same signature applied to the two given signatures. In case the first boolean is false, the proof returned is the proof that the two empty signatures are the same. We also assume predefined the functions with the same name defined using primitive recursion in UTT in the paper but using the functional programming language for the development of tactics.

7 Conclusions

In this paper, we have represented in type theory a proof system for refinement of algebraic specifications in ASL. First, we have presented the encoding of signatures with indexes. Indexes were needed to solve the name clashes between subspecifications of structured specifications. Then, we have presented well formed specification which can easily be defined by an inductive relation and finally we give a representation of the proof system for refinement. The representation is not adequate but full because the use of proof obligations to represent side-conditions. Using this representation, we can develop a proof tactic to help the development of proofs of refinement.

References


A Predefined functions of this paper

A.1 Functions on signature morphisms

Definition A.1 The function $\text{get\_dom\_sm} : \text{Signature\_morphism} \rightarrow \text{Signature}$ is defined as follows:

$$
\text{get\_dom\_sm} \, \text{signm} = (\text{sort\_sl} \, (\text{get\_dom\_spl} \, (\text{fst} \, (\text{snd} \, \text{signm}))))),

\text{sort\_opl} \, (\text{get\_dom\_oppl} \, (\text{snd} \, (\text{snd} \, \text{signm}))))
$$

$$
\text{get\_dom\_spl} \, \text{spl} = \text{map} \, \text{fst} \, \text{spl}
$$

$$
\text{get\_dom\_oppl} \, \text{oppl} = \text{map} \, \text{fst} \, \text{oppl}
$$

Definition A.2 The function $\text{get\_ran\_sm} : \text{Signature\_morphism} \rightarrow \text{Signature}$ is defined as follows:

$$
\text{get\_ran\_sm} \, \text{signm} = (\text{sort\_sl} \, (\text{get\_ran\_spl} \, (\text{fst} \, (\text{snd} \, \text{signm}))))),

\text{sort\_opl} \, (\text{get\_ran\_oppl} \, (\text{snd} \, (\text{snd} \, \text{signm}))))
$$

$$
\text{get\_ran\_spl} \, \text{spl} = \text{map} \, \text{snd} \, \text{spl}
$$

$$
\text{get\_ran\_oppl} \, \text{oppl} = \text{map} \, \text{snd} \, \text{oppl}
$$

Definition A.3 The function $\text{inverse\_sm} : \text{Signature\_morphism} \rightarrow \text{Signature\_morphism}$
is defined as follows:

\[
\text{inverse} \, \text{sm} \, \text{sm} = \text{mkpair} \, (\text{getran} \, \text{sm} \, \text{sm}) \, (\text{invert} \, \text{pairs} \, \text{sm})
\]

where

\[
\text{invert} \, \text{pairs} \, \text{sm} = \text{mkpair} \, (\text{invp} \, \text{sl} \, (\text{fst} \, (\text{snd} \, \text{sm}))) \, (\text{invp} \, \text{op} \, (\text{snd} \, (\text{snd} \, \text{sm})))
\]

\[
\text{invp} \, \text{sl} \, \text{sl} = \text{map} \, \text{invp} \, \text{sl}
\]

\[
\text{invp} \, \text{op} \, \text{sl} = \text{map} \, \text{invp} \, \text{op} \, \text{sl}
\]

\[
\text{invp} \, \text{p} = (\text{snd} \, \text{p}, \text{fst} \, \text{p})
\]

A.2 Operations on signatures and specification expressions

**Definition A.4** The function new\_index : Signature \rightarrow Sym\_index \rightarrow Signature is defined as follows:

\[
\text{new} \, \text{index} \, \text{sign} \, \text{ind} = \text{mkpair} \, (\text{map} \, (\text{updinds} \, \text{ind}) \, (\text{fst} \, \text{sign}))
\]

\[
(\text{map} \, (\text{updindop} \, \text{ind}) \, (\text{snd} \, \text{sign}))
\]

where

\[
\text{updinds} \, \text{s} \, \text{ind} = (\text{fst} \, \text{s}, \text{ind})
\]

\[
\text{updindop} \, \text{op} \, \text{ind} = (\text{fst} \, \text{op}, \text{ind})
\]

**Definition A.5** The function union\_Sign : Signature \rightarrow Signature \rightarrow Signature is defined as follows:

\[
\text{union} \, \text{Sign} \, \text{sign} \, \text{sign} \, \text{sign} = \text{mkpair} \, (\text{union} \, \text{Srt} \, (\text{first} \, \text{sign}) \, (\text{first} \, \text{sign}))
\]

\[
(\text{union} \, \text{Ops} \, (\text{snd} \, \text{sign}) \, (\text{snd} \, \text{sign}'))
\]

**Definition A.6** The function union\_Srt : (List \_\text{Ind\_sorts}) \rightarrow (List \_\text{Ind\_sorts}) \rightarrow
(List Ind_sorted) is defined as follows:

\[
\text{union Srts } l l' = \text{Primrec (List Ind_sorted)) } l' \text{ gene_uSrts } l
\]

where

\[
\text{gene_uSrts } s s l sl f = \text{add_iff not in sl s sl f}
\]

\[
\text{add_iff not in sl s sl } = \text{Primrec Bool (cons s sl} | \text{ sl (not in sl s sl)}
\]

\[
\text{not in sl s sl } = \text{Primrec(List Ind_sorted) true (gene_ninsl s} | \text{ sl)
\]

\[
\text{gene_ninSl } s s' sl b =
\]

\[
\text{Primrec bool(not bool EqboolSrts s s') b b}
\]

**Definition A.7** The function union Ops : (List Ind_ops) → (List Ind_ops) → (List Ind_ops) is defined as follows:

\[
\text{union Ops } l l' = \text{Primrec (List Ind_ops) l' gene_uOps } l
\]

where

\[
\text{gene_uOps op opl opl f} = \text{add iff not in op op f}
\]

\[
\text{add_iff not in op op op} = \text{Primrec Bool (cons op opl} | \text{ op op (not in op op op)}
\]

\[
\text{not in op op op } = \text{Primrec (list Ind_ops) true (gene_ninop op.opl}
\]

\[
\text{gene_ninop op op op } b = \text{Primrec bool(not_bool Eqbool_tops op op') b b}
\]

**Definition A.8** The function intersect_Sign : Signature → Signature →
Signature is defined as follows:

\[ \text{intersectSign} \, \text{sign} \, \text{sign'} = \text{mkpair} \left( \text{fst} \left( \text{interSrt} \left( \text{first sign} \right) \left( \text{first sign'} \right) \right) \right) \]
\[ \left( \text{fst} \left( \text{interOps} \left( \text{snd sign} \right) \left( \text{snd sign'} \right) \right) \right) \]

where

\[ \text{interSrt} \, \text{sl} \, \text{sl'} = \text{Primrec} \left( \text{List Indorts} \right) \left( \text{nil, sl} \right) \text{addfinsecl sl'} \]
\[ \text{addfinsecl} \, \text{sl} \, \text{psl} = \text{Primrec bool} \left( \text{cons s} \left( \text{fst psl}, \text{snd psl} \right) \right) \]
\[ \text{psl} \left( \text{is_in_bool} \right) \text{Eqbool Isrts} \left( \text{snd psl} \right) \]

\[ \text{interOps} \, \text{opl} \, \text{opl'} = \text{Primrec} \left( \text{List Indops} \right) \left( \text{nil, opl} \right) \text{addfinsecopl sl'} \]
\[ \text{addfinsecopl} \, \text{opl} \, \text{popl} = \text{Primrec bool} \left( \text{cons s} \left( \text{fst popl}, \text{snd popl} \right) \right) \]
\[ \text{popl} \left( \text{is_in_bool} \right) \text{Eqbool Iops} \left( \text{snd psl} \right) \]

**Definition A.9** The function \( \text{diff \_Sign} \) : Signature \( \rightarrow \) Signature \( \rightarrow \) Signature is defined as follows:

\[ \text{diff \_Sign} \, \text{sign} \, \text{sign'} = \text{mkpair} \left( \text{diff \_Srt} \left( \text{first sign} \right) \left( \text{first sign'} \right) \right) \]
\[ \left( \text{diff \_Ops} \left( \text{snd sign} \right) \left( \text{snd sign'} \right) \right) \]

where

\[ \text{diff \_Srt} \, \text{sl} \, \text{sl'} = \text{Primrec} \left( \text{List Indorts} \right) \left( \text{sl gensecl diff sl'} \right) \]
\[ \text{gensecl} \, \text{diff s} \, \text{sl} \, \text{sl'} = \text{remove Eqbool Isrts} \, \text{s} \, \text{sl'} \]
\[ \text{diff \_Ops} \, \text{opl} \, \text{opl'} = \text{Primrec} \left( \text{List Indops} \right) \left( \text{opl gencopl diff opl'} \right) \]
\[ \text{gencopl} \, \text{diff opl} \, \text{opl} \, \text{opl'} = \text{remove Eqbool Iops} \, \text{opl} \, \text{opl'} \]

**Definition A.10** The function \( \text{nameclash \_Sign} \) : Signature \( \rightarrow \) Signature \( \rightarrow \) Signature \( \rightarrow \) Signature is defined as follows:

\[ \text{nameclash \_Sign} \, \text{signsp} \, \text{sign} \, \text{signsp'} = \]
\[ \text{diff \_Sign} \left( \text{intersect \_Sign} \, \text{signsp} \, \text{signsp'} \right) \, \text{sign} \]

**Definition A.11** The function \( \text{Signature \_Indsp} \) : Specification \( \rightarrow \) Sym \( \rightarrow \)
Signature is defined as follows:

\[ \text{Signature} \text{sp} \text{sp} \text{ind} = \text{Primrec Specification} \ (\text{basec} \text{sign ind}) \ (\text{suma} \text{sign ind}) \ (\text{expc} \text{sign ind}) \\
(\text{renc} \text{sign ind}) \ (\text{reache} \text{sign ind}) \ (\text{behe} \text{sign ind}) \ (\text{quote} \text{sign ind}) \ (\text{abstre} \text{sign ind}) \text{sp} \]

where

\[ \text{basec} \text{sign ind} \text{ sign fl} = (\text{new index} \text{ sign ind, ind)} \]

\[ \text{suma} \text{sign ind} \text{ sp sign spf} \text{ signsp spsignsp'} = \]

\[ \text{mkpair} \ (\text{union} \text{sign} \ (\text{new index} \text{ (nameclash} \text{sign} \ (\text{fst signspsp} \text{)})) \text{sign} \ (\text{fst signsp'})) \]

\[ (\text{next} \text{Si} \ (\text{maxind} \text{Si} \ (\text{snd signspsp} \text{)} \ (\text{snd signsp'})))) \]

\[ (\text{union} \text{sign} \ (\text{diff} \text{sign} \ (\text{diff} \text{sign} \ (\text{fst signspsp} \text{)} \text{sign})) \]

\[ (\text{nameclash} \text{sign} \ (\text{fst signspsp} \text{)} \ (\text{fst signsp'}))) \ (\text{fst signsp'})] \]

\[ (\text{next} \text{Si} \ (\text{maxind} \text{Si} \ (\text{snd signspsp} \text{)} \ (\text{snd signsp'})))) \]

\[ \text{renc} \text{sign ind} \text{ sp signm signspp} = (\text{get ran sm signspp, ind)} \]

\[ \text{expc} \text{sign ind} \text{ sp sign signspsp} = (\text{sign, ind)} \]

\[ \text{reache} \text{sign ind} \text{ sp reachspsp} = \text{signsp} \]

\[ \text{behe} \text{sign ind} \text{ sp obssl insl signspsp} = \text{signsp} \]

\[ \text{abse} \text{sign ind} \text{ sp obssl insl signspsp} = \text{signsp} \]

\[ \text{quoc} \text{sign ind} \text{ sp obssl insl signspsp} = \text{signsp} \]

**Definition A.12** The function \( \text{Signature sp: Specification} \rightarrow \rightarrow \text{Signature} \) is defined as follows:

\[ \text{Signature sp sp} = \text{fst} \ (\text{Signature ind sp sp first Vi}) \]
A.3 Some inductive relations

**Definition A.13** The inductive relation $\text{Same}_{\text{signature}} : \Pi \text{sign, sign'} : \text{Signature}.\text{Prop}$ is defined by the following set of constructors:

$$\text{basec}_\text{sams} : \Pi \text{sign} : \text{Same}_{\text{signature}} \ (\text{mkpair} \ (\text{nil Ind}_{\text{sorts}}) \ (\text{nil Ind}_{\text{ops}}))$$

$$\ (\text{mkpair} \ (\text{nil Ind}_{\text{sorts}}) \ (\text{nil Ind}_{\text{ops}}))$$

$$\text{gencs}_\text{sams} : \Pi \text{s} : \text{Ind}_{\text{sorts}}.\Pi \text{sign, sign'} : \text{Signature}.\Pi \text{sams} : \text{Same}_{\text{signature}} \text{ sign sign'}.$$  

$$\text{Same}_{\text{signature}} \ (\text{sort sl} \ (\text{cons s} \ (\text{fst sign}), (\text{snd sign})))$$

$$\ (\text{sort sl} \ (\text{cons s} \ (\text{fst sign'}), (\text{snd sign'})))$$

$$\text{gencop}_\text{sams} : \Pi \text{op} : \text{Ops}.\Pi \text{sign, sign'} : \text{Signature}.\Pi \text{sams} : \text{Same}_{\text{signature}} \text{ sign sign'}. $$

$$\text{Same}_{\text{signature}} \ (\text{fst sign}, (\text{sort op} \ (\text{consop} \ (\text{snd sign}))))$$

$$\ (\text{fst sign}, (\text{sort op} \ (\text{consop} \ (\text{snd sign}))))$$

**Definition A.14** The inductive relation $\text{Sub}_{\text{signature}} : \Pi \text{sign, sign'} : \text{Signature}.\text{Prop}$ is defined by the following set of constructors:

$$\text{basec}_\text{subsign} : \Pi \text{sign} : \text{Signature}.\text{Sub}_{\text{signature}} \ (\text{mkpair} \ (\text{nil Ind}_{\text{sorts}}) \ (\text{nil Ind}_{\text{ops}})) \ \text{sign}$$

$$\ \text{gencs}_\text{subsign} : \Pi \text{s} : \text{Ind}_{\text{sorts}}.\Pi \text{sign, sign'} : \text{Signature}.\Pi \text{isins} : \text{Is in List s (fst sign')}. $$

$$\text{Sub}_{\text{signature}} \ (\text{sort sl} \ (\text{cons s} \ (\text{fst sign}), (\text{snd sign}))) \ \text{sign'}$$

$$\text{gencop}_\text{subsign} : \Pi \text{op} : \text{Ops}.\Pi \text{sign, sign'} : \text{Signature}.\Pi \text{isins} : \text{Is in List op (snd sign')}. $$

$$\text{Sub}_{\text{signature}} \ (\text{fst sign}, (\text{sort op} \ (\text{consop} \ (\text{snd sign}))) \ \text{sign'})$$

**Definition A.15** The inductive relation $\text{Sub}_{\text{sorts}} : \Pi \text{sl} : \text{List Ind}_{\text{sorts}}.\Pi \text{sign'} : \text{Signature}.\text{Prop}$ is defined by the following set of constructors:

$$\text{basec}_\text{subs} : \Pi \text{sign} : \text{Signature}.\text{Sub}_{\text{sorts}} \ (\text{nil Ind}_{\text{sorts}}) \ \text{sign}$$

$$\ \text{gencs}_\text{subs} : \Pi \text{s} : \text{Ind}_{\text{sorts}}.\Pi \text{sl} : \text{List Ind}_{\text{sorts}}.\Pi \text{sign} : \text{Signature}.\Pi \text{isins} : \text{Is in List s (fst sign')}. $$

$$\text{Sub}_{\text{sorts}} \ (\text{sort sl} \ (\text{cons s sl}) \ \text{sign'})$$
Definition A.16 The inductive relation

\[ \text{Bijective} : \text{IIsign} : \text{Signature} \Rightarrow \text{IIsignm} : \text{Signature\_morphism} \Rightarrow \text{Prop} \]

is defined by the following constructors:

\[ \text{bijctr} : \text{IIsign} : \text{Signature} \Rightarrow \text{IIsignm} : \text{Signature\_morphism}. \]

\[ \text{Piorepsd} : \text{Norep\_list\_Ind\_sorts} \Rightarrow \text{Eqbool\_Isrts} (\text{fst} (\text{get\_dom\_sm} \text{signm})). \]

\[ \text{Piorepsst} : \text{Norep\_list\_Ind\_sorts} \Rightarrow \text{Eqbool\_Isrts} (\text{fst} (\text{get\_ran\_sm} \text{signm})). \]

\[ \text{Piorepsopd} : \text{Norep\_list\_Ind\_ops} \Rightarrow \text{Eqbool\_Jops} (\text{snd} (\text{get\_dom\_sm} \text{signm})). \]

\[ \text{Piorepsopt} : \text{Norep\_list\_Ind\_ops} \Rightarrow \text{Eqbool\_Jops} (\text{snd} (\text{get\_ran\_sm} \text{signm})). \]

\[ \text{Piamesignd} : \text{Same\_signature \_sign} (\text{get\_dom\_sm} \text{signm}). \]

\[ \text{Piamesignt} : \text{Same\_signature} (\text{first} \text{signm}) (\text{get\_ran\_sm} \text{signm}). \]

\[ \text{Bijective} \text{ sign signm} \]

A.4 Operations associated to the pushouts morphisms of structured specifications

Definition A.17 The function ind\_sums : Specification \rightarrow Signature \rightarrow Specification \rightarrow Specification is defined as follows:

\[ \text{ind\_sums sp sign sp}' = \text{Signature}\_\text{sp sp} \]

Definition A.18 The function inr\_sums : Specification \rightarrow Signature \rightarrow Specification \rightarrow Specification is defined as follows:

\[ \text{inr\_sums sp sign sp}' = \]

\[ (\text{union\_sign} (\text{new\_index} (\text{nameclash\_sign} (\text{Signature}\_\text{sp sp}) \text{sign} (\text{Signature}\_\text{sp sp}'))) \]

\[ (\text{next\_Vi} (\text{maxind\_Si} (\text{snd} (\text{Signature\_ind\_sp sp} \text{first\_Vi})))) \]

\[ (\text{snd} (\text{Signature\_ind\_sp sp}' \text{first\_Vi}))) \]

\[ (\text{diff\_sign} (\text{Signature}\_\text{sp sp}') (\text{nameclash\_sign} (\text{Signature}\_\text{sp sp} \text{sign}(\text{Signature}\_\text{sp sp}')))) \]

Definition A.19 The function inl\_sm\_sums : Specification \rightarrow Signature \rightarrow
Specification → Signature_morphism is defined as follows:

\[\text{inrsm\_sums}\ sp\ \text{sign}\ sp' =\]

\[\text{join}\ Ind\_sorts\ Ind\_sorts\ (\text{fst}\ (\text{Signature}\_sp\ sp)) (\text{fst}\ (\text{Signature}\_sp\ sp')),\]

\[\text{join}\ Ind\_ops\ Ind\_ops\ (\text{snd}\ (\text{Signature}\_sp\ sp)(\text{snd}\ (\text{Signature}\_sp\ sp)))\]

**Definition A.20** The function \(\text{inrsm\_sums} : \text{Specification} → \text{Signature} → \text{Specification} → \text{Signature}_\text{morphism} \) is defined as follows:

\[\text{inrsm\_sums}\ sp\ \text{sign}\ sp' =\]

\[\text{concat}\ (\text{prod}\ Ind\_sorts\ Ind\_sorts)\ (\text{join}\ Ind\_sorts\ Ind\_sorts\)

\[\ (\text{Fst}\ (\text{nameclash}\_sign\ (\text{Signature}\_sp\ sp)\ \text{sign}\ (\text{Signature}\_sp\ sp'))))\]

\[\ (\text{Fst}\ (\text{new}\_index\ \text{nameclash}\_sign\ (\text{Signature}\_sp\ sp)\ \text{sign}\ (\text{Signature}\_sp\ sp'))))\]

\[\ (\text{next}_Vi\ \text{maxind}_Si\ (\text{snd}\ (\text{Signature}\_indsp\ sp\ \text{first}_Vi))\]

\[\ (\text{snd}\ (\text{Signature}\_indsp\ sp'\ \text{first}_Vi))))))))\]

\[\ (\text{join}\ Ind\_sorts\ Ind\_sorts\ (\text{Fst}\ (\text{diff}\_sign\ (\text{Signature}\_sp\ sp'\ \text{first}_Si))\)

\[\ (\text{nameclash}\_sign\ (\text{Signature}\_sp\ sp\ \text{first}_Si)\ \text{sign}\ (\text{Signature}\_sp\ sp'\ \text{first}_Si))))))\]

\[\ (\text{Fst}\ (\text{diff}\_sign\ (\text{Signature}\_sp\ sp'\ \text{first}_Si)\ \text{nameclash}\_sign\)

\[\ (\text{Fst}\ (\text{Signature}\_sp\ sp\ \text{first}_Si)\ \text{sign}\ (\text{Signature}\_sp\ sp'\ \text{first}_Si))))))),\]
\[
\text{(concat } \text{prod Ind}\_\text{ops} \text{ Ind}\_\text{ops}) \text{ (join Ind}\_\text{ops} \text{ Ind}\_\text{ops} \\
\text{(snd (nameclash}\_\text{sign} \text{ (Signature}_\text{sp sp first}_Si \text{ sign} \text{ (Signature}_\text{sp sp'} first}_Si))] \\
\text{(snd (new}\_\text{index (nameclash}\_\text{sign} \text{ (Signature}_\text{sp sp first}_Si)} \\
\text{ sign (Signature}_\text{sp sp'} first}_Si])) \\
\text{(next}_Vi \text{ (maxind}_Si \text{ (snd (Signature}_\text{ind}\_\text{sp sp first}_Vi))] \\
\text{(snd (Signature}_\text{ind}\_\text{sp sp'} first}_Vi))]]) \\
\text{(join Ind}\_\text{ops} \text{ Ind}\_\text{ops} \text{ (snd (diff}\_\text{sign (Signature}_\text{sp sp first}_Si)} \\
\text{nameclash}\_\text{sign (Signature}_\text{sp sp first}_Si \text{ sign} \text{ (Signature}_\text{sp sp'} first}_Si)]) \\
\text{(snd (diff}\_\text{sign (Signature}_\text{sp sp'} first}_Si \text{ (nameclash}\_\text{sign}} \\
\text{(Signature}_\text{sp sp first}_Si \text{ sign (Signature}_\text{sp sp'} first}_Si)])])]
\]

References


[8] Nikos Mylonakis. *A type-theoretic approach to proof support for algebraic
To appear.

[9] Don Sannella and Martin Wirsing. A kernel language for algebraic speci-
fication and implementation. In *Proc. Intl. Conf. on Foundations of Com-
putation Theory*, Borgholm, Sweden, number 158 in Springer LNCS, pages
413–27, 1983.