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 ϵ_{ref} between the derivations of a concrete refinement judgement ($SP \ggg SPI$) and the application of this judgement to the inductive relation which encodes the proof system for refinement. Another interesting property of ϵ_{ref} is that there exists a function ϵ_{ref}^{-1} which satisfies the following condition

$$\forall \delta \in \Delta_{\Pi^{ASL}_{AINS}\ggg}(SP \ggg SPI).\ \epsilon_{ref}^{-1}\ (\epsilon_{ref}\ \delta) \ = \ \delta$$

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 *Signature* and *Models*. The function *Signature* must return the signature with just the visible symbols of the given specification and *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 *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 Σ-algebras for a given signature Σ (which are the models of specifications) is denoted as $Alg(\Sigma)$ 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 irrellevant for the setting. We will refer to the sentences of first order logic as $Sen_{FOL}(\Sigma)$ for a given signature Σ and to the satisfaction relation between $\Sigma-$algebras and first-order sentences by $\models_{FOL,\Sigma}$.

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* $(\mathcal{S}_{\mathcal{R}}, \mathcal{F}_{\mathcal{R}})$ *such that* $\mathcal{S}_{\mathcal{R}} \subseteq S$ *and for any* $f : s_1 \times \ldots \times s_n \to s \in \mathcal{F}_{\mathcal{R}}$*,* $s \in \mathcal{S}_{\mathcal{R}}$*.*

Definition 2.2 *For any signature* $\Sigma = (S, Op) \in AlgSig$*, an algebra* $A \in Alg(\Sigma)$ *satisfies a reachability constraint* $(\mathcal{S}_{\mathcal{R}}, \mathcal{F}_{\mathcal{R}})$ *of* Σ *(*$A \models (\mathcal{S}_{\mathcal{R}}, \mathcal{F}_{\mathcal{R}})$*) if the following condition holds:*

$$A \models (\mathcal{S}_{\mathcal{R}}, \mathcal{F}_{\mathcal{R}}) \Leftrightarrow \forall s \in \mathcal{S}_{\mathcal{R}}.\forall v \in A_s.$$

$$\exists t \in T_{(S,\mathcal{F}_{\mathcal{R}})}(X_{S-\mathcal{S}_{\mathcal{R}}}).\exists \alpha : X_{S-\mathcal{S}_{\mathcal{R}}} \to A.I_\alpha(t) \;=\; v$$

Definition 2.3 *The syntax of the operators of ASL is the following:*

$$
\begin{aligned}
SP_0 ::= \;& < \Sigma, \Phi > \\
& SP_1|_\Sigma \\
& SP_1 +_\Sigma SP_2 \\
& rename \quad SP \quad \mathbf{by} \quad \sigma \\
& \mathbf{reach} \quad SP \quad \mathbf{with} \quad (\mathcal{S}_\mathcal{R}, \mathcal{F}_\mathcal{R}) \\
& \mathbf{behaviour} \quad SP \quad \mathbf{wrt} \quad \approx \\
& \mathbf{abstract} \quad SP \quad \mathbf{by} \quad \equiv \\
& SP/\approx
\end{aligned}
$$

where the signature $\Sigma = (S, Op) \in |AlgSig|$, $\Phi \subseteq Sen_{FOL}(\Sigma)$, σ *is a signature morphism,* $\approx$ *is a partial congruence between elements of* Σ*-algebras and equiv is an equivalence relation between algebras. The semantics of the ASL operators is inductively defined as follows:*

$Signature(< \Sigma, \Phi >) = \Sigma$

$Models(< \Sigma, \Phi >) = \{A \mid A \models_{FOL,\Sigma} \Phi\}$

$Signature(rename \quad SP \quad \mathbf{by} \quad \sigma) = \Sigma$

$Models(rename \quad SP \quad \mathbf{by} \quad \sigma) = \{ A \in Alg(\Sigma) \mid A|_\sigma \in Models(SP) \}$

$Signature(SP|_\Sigma) = \Sigma$

$Models(SP|_\Sigma) = \{A|_\Sigma \mid A \in Models(SP)\}$

where $\Sigma \subseteq Signature(SP)$

$$Signature(SP_1 +_\Sigma SP_2) = Signature(SP_1) +_\Sigma Signature(SP_2)$$

$$Models(SP_1 +_\Sigma SP_2) =$$

$$\{A \mid A \in Alg(Signature(SP_1) +_\Sigma Signature(SP_2)),$$

$$A|_{inl} \in Models(SP_1), A|_{inr} \in Models(SP_2)\}$$

where SP_1, SP_2 *ranges over specification expressions,*

$\Sigma \subseteq Signature(SP_1)$, $\Sigma \subseteq Signature(SP_2)$

and $Signature(SP_1) +_\Sigma Signature(SP_2)$ *is the pushout of the two obvious inclusions between* Σ *and* $Signature(SP_1)$ *and* Σ *and* $Signature(SP_2)$

$$Signature(\textbf{reach} \quad SP \quad \textbf{with} \quad (\mathcal{S}_\mathcal{R}, \mathcal{F}_\mathcal{R})) = Signature(SP)$$

$$Models(\textbf{reach} \quad SP \quad \textbf{with} \quad (\mathcal{S}_\mathcal{R}, \mathcal{F}_\mathcal{R})) = \{A \in Models(SP) \mid A \models (\mathcal{S}_\mathcal{R}, \mathcal{F}_\mathcal{R})\}$$

$$Signature(\textbf{behaviour} \quad SP \quad \textbf{wrt} \quad \approx) = Signature(SP)$$

$$Models(\textbf{behaviour} \quad SP \quad \textbf{wrt} \quad \approx) = \{ A/\approx \mid A \in Models(SP) \}$$

$$Signature(\textbf{abstract} \quad SP \quad \textbf{by} \quad \equiv) = Signature(SP)$$

$$Models(\textbf{abstract} \quad SP \quad \textbf{by} \quad \equiv) = \{A \mid \exists B \in Models(SP).B \equiv A\}$$

$$Signature(SP/\approx) = Signature(SP)$$

$$Models(SP/\approx) = \{A \mid \exists B \in Models(SP)/\approx .B \cong A\}$$

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:*

- $Signature(SPI) = Signature(SP)$
- $Models(SPI) \subseteq Models(SP)$

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 Σ. 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* $\Sigma \in |AlgSig|$*, the inductive relation Sorts is inductively defined by the following set of constructors:*

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

Definition 3.2 *For any* $\Sigma \in |AlgSig|$*, the function* $Eqbool\_Srts : Sorts \to Sorts \to Bool$ *is defined as follows:*

$$Eqbool\_Srts\ s\ s' = Primrec\ Sorts\ (s_1c\ s')\ \ldots\ (s_nc\ s')\ s$$

$$\begin{aligned} s_1c\ s' &= Primrec\ Sorts\ true\ \ldots\ false\ s' \\ &\vdots \\ s_nc\ s' &= Primrec\ Sorts\ true\ \ldots\ false\ s' \end{aligned}$$

Definition 3.3 *For any* $\Sigma \in |AlgSig|$*, the inductive relation Ops is inductively defined by the following set of constructors:*

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

$$\{f\_s_1\_\ldots\_s_n\_s\_Ops : Ops \mid$$

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

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

Definition 3.4 *The type* Sym_index *is inductively defined by the following set of constructors:*

$$first\_Si : Sym\_index$$

$$next\_Si : Sym\_index \to Sym\_index$$

Remark: *We assume predefined the function* $maxind\_Si : Sym\_index \to Sym\_index \to Sym\_index$ *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*

$$Signature \ = \ (Pair\ (List\ Ind\_sorts)(List\ Ind\_ops))$$

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 $Ltbool\_Srts : Ind\_sorts \to Ind\_sorts \to Bool$ which given two indexed sorts s_1, s_2 returns true if s_1 is lower than s_2 and false otherwise.
- the function $Ltbool\_Ops : Ind\_ops \to Ind\_ops \to Bool$ and the functions $Eqbool\_Isrts : Ind\_sorts \to Ind\_sorts \to Bool$ and $Eqbool\_Iops : Ind\_ops \to Ind\_ops \to Bool$.
- the functions $sort\_sl : List\ Ind\_sorts \to List\ Ind\_Sorts$ which given a list of indexed sorts, sorts the given list eliminating repeated elements and analogously $sort\_opl : List\ Ind\_ops \to List\ Ind\_ops$. See [6] for a verified algorithm for sorting in UTT using primitive recursion.
- the inductive relations $Sorted\_sl : List\ Ind\_sorts \to Prop$ and $Sorted\_opl : List\ Ind\_ops \to Prop$ 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*

$$Wfsignature : \Pi sign : Signature.Prop$$

is defined by the following constructors:

$$wfsign\_c : \Pi sl : List\ Ind\_sorts.\Pi opl : List\ Ind\_ops.$$

$$\Pi nrsl : Norep\_list\ Ind\_sorts\ Eqbool\_Isrts\ sl.$$

$$\Pi nropl : Norep\_list\ Ind\_ops\ Eqbool\_Iops\ opl.$$

$$\Pi ssl : Sorted\ sl.\Pi sopl : Sorted\ opl.$$

$$Wfsignature\ (sl, opl)$$

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 $X \blacktriangleright SP$) is inductively defined by the following rules:*

$$\frac{\{X \blacktriangleright \phi \mid \phi \in \Phi\}}{X \blacktriangleright < \Sigma, \Phi >} \quad (basic\_wfs)$$

$$\frac{X \blacktriangleright SP_1 \quad X \blacktriangleright SP_2}{X \blacktriangleright SP_1 +_\Sigma SP_2} \ \Sigma \subseteq Sign(SP_1) \wedge \Sigma \subseteq Sign(SP_2) \quad (sum\_wfs)$$

$$\frac{X \blacktriangleright SP}{X \blacktriangleright SP|_\Sigma} \ \Sigma \subseteq SP \quad (export\_wfs)$$

$$\frac{X \blacktriangleright SP}{X \blacktriangleright rename \quad SP \quad \mathbf{by} \quad \sigma} \ Bij(Sign(SP), \Sigma, \sigma) \quad (rename\_wfs)$$

$$\frac{X \blacktriangleright SP}{X \blacktriangleright \mathbf{reach} \quad SP \quad \mathbf{with} \quad (\mathcal{S}_\mathcal{R}, \mathcal{F}_\mathcal{R})} \ (S_\mathcal{R}, F_\mathcal{R}) \subseteq Sign(SP) \quad (reach\_wfs)$$

$$\frac{X \blacktriangleright SP}{X \blacktriangleright \mathbf{behaviour} \quad SP \quad \mathbf{wrt} \quad \approx} \ In, Obs \subseteq Sign(SP) \quad (behaviour\_wfs)$$

$$\frac{X \blacktriangleright SP}{X \blacktriangleright \mathbf{abstract} \quad SP \quad \mathbf{by} \quad \equiv} \ In, Obs \subseteq Sign(SP) \quad (abstract\_wfs)$$

$$\frac{X \blacktriangleright SP}{X \blacktriangleright SP/\approx} \ In, Obs \subseteq Sign(SP) \quad (quotient\_wfs)$$

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

$$Bij(Signature(SP), \Sigma, \sigma) \ = \ (Dom(\sigma) \ = \ Sign(SP)) \wedge$$

$$\forall s, s' \in Sorts(Sign(SP)).\sigma(s) = \sigma(s') \ \supset \ s = s' \ \wedge$$

$$\forall s \in Sorts(\Sigma).\exists s' \in Sorts(Sign(SP)).\sigma(s') \ = \ s$$

$$\forall op : s_1 \times \ldots \times s_n \to s \in Ops(Sign(SP)).\forall op' : s'_1 \times \ldots \times s'_n \to s' \in Ops(Sign(SP)).$$

$$\sigma(op : s_1 \times \ldots \times s_n \to s) \ = \ \sigma(op' : s'_1 \times \ldots \times s'_n \to s') \ \supset$$

$$op : s_1 \times \ldots \times s_n \to s) \ = \ op' : s'_1 \times \ldots \times s'_n \to s'$$

$$\forall op : s_1 \times \ldots \times s_n \to s \in Ops(\Sigma).\exists op' : s'_1 \times \ldots \times s'_n \to s' \in Ops(Sign(SP)).$$

$$\sigma(op' : s'_1 \times \ldots \times s'_n \to s') \ = \ op : s_1 \times \ldots \times s_n \to s$$

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

We assume predefined the inductive types $Formula$ which defines first-order formulas and Var_set which defines variable sets. We also assume predefined the inductive relations $Wfform : \Pi vs : Var\_set.\Pi form : Formula.Prop$ and $Wfforml : \Pi vs : Var\_set.\Pi form : List\ Formula.Prop$ 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:*

$$Signature\_morphism \ = \ Pair\ Signature$$

$$(Pair\ (List\ (Pair\ Ind\_sorts\ Ind\_sorts))\ (List\ (Pair\ Ind\_ops\ Ind\_ops)))$$

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

- $get\_dom\_sm : Signature\_morphism \rightarrow Signature$ which given a signature morphism, returns the domain of the signature morphism.
- $get\_ran\_sm : Signature\_morphism \rightarrow Signature$ which given a signature morphism, returns the range of the signature morphism.
- $inverse\_sm : Signature\_morphism \rightarrow Signature\_morphism$ which given a signature morphism, returns the inverse of the signature morphism.

Definition 4.3 *The inductive type Specification is defined by the following set of constructors:*

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

$$sum\_spec : Specification \ \rightarrow \ Signature \ \rightarrow \ Specification \ \rightarrow \ Specification$$

$$export\_spec : Specification \ \rightarrow \ Signature \ \rightarrow \ Specification$$

$$rename\_spec : Specification \ \rightarrow \ Signature\_morphism \ \rightarrow \ Specification$$

$$reach\_spec : Specification \ \rightarrow \ Signature \ \rightarrow \ Specification$$

$$behaviour\_spec : Specification \ \rightarrow \ (List\ Ind\_sorts) \ \rightarrow \ (List\ Ind\_sorts)$$

$$\rightarrow \ Specification$$

$$abstract\_spec : Specification \rightarrow (List\ Ind\_sorts) \rightarrow (List\ Ind\_sorts)$$

$$\rightarrow Specification$$

$$quotient\_spec : Specification \rightarrow (List\ Ind\_sorts) \rightarrow (List\ Ind\_sorts)$$

$$\rightarrow Specification$$

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

- $new\_index : Signature \rightarrow Sym\_index \rightarrow Signature$ which given a signature and a symbol index assigns the symbol index to all the sorts and operations of the signature.
- $union\_Sign : Signature \rightarrow Signature \rightarrow Signature$ which given two signatures, returns the union of the two signatures.
- $intersect\_Sign : Signature \rightarrow Signature \rightarrow Signature$ which given two signatures, returns the intersection of the two signatures.
- $diff\_Sign : Signature \rightarrow Signature \rightarrow Signature$ which given two signatures, returns the difference of the first by the second signature.
- $nameclash\_sign : Signature \rightarrow Signature \rightarrow Signature \rightarrow Signature$ which given three signatures returns the signature which is the intersection of the first and third and has no symbols of the second.
- $Signature\_sp : Specification \rightarrow Signature$ 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:

- $Same\_signature : \Pi sign, sign' : Signature.Prop$ which given two signatures checks whether they are the same.
- $Subsignature : \Pi sign, sign' : Signature.Prop$ which given two subsignatures, checks whether the first is subsignature of the second.
- $Subsorts : \Pi sl : List\ Ind\_sorts.sign' : Signature.Prop$ which given a list of sorts and a signature checks whether the list of sorts is included in the sorts of the signature.
- $Bijective : \Pi sign : Signature.\Pi signm : Signature\_morphism.Prop$ which given a signature and a signature morphism, checks whether the domain of the signature morphism is tha same as the given signature and the signature morphism is bijective.

The following definition represents well-formed specifications:

Definition 4.4 *The inductive relation*

$$Wfspec : \Pi vs : Var\_set.\Pi sp : Specification.Prop$$

is defined by the following set of constructors:

$base\_wfsp : \Pi vs : Var\_set.\Pi sign : Signature.$

$\Pi fl : list\ Formula.\Pi wfs : Wfsignature\ sign.$

$\Pi wffl : Wfforml\ vs\ fl.Wfspec\ (base\_spec\ sign\ fl)$

$sum\_wfsp : \Pi vs : Var\_set.\Pi sp : Specification.\Pi sign : Signature.$

$\Pi sp' : Specification.\Pi wfsign : Wfsignature\ sign.$

$\Pi subsp : Subsignature\ sign\ (Signature\ sp).$

$\Pi subsp' : Subsignature\ sign\ (Signature\ sp').$

$\Pi wfsp : Wfspec\ vs\ sp.\Pi wfsp' : Wfspec\ vs\ sp'.$

$Wfspec\ vs\ (sum\_spec\ sp\ sign\ sp')$

$export\_wfsp : \Pi vs : Var\_set.\Pi sp : Specification.\Pi sign : Signature.$

$\Pi wfsign : Wfsignature\ sign.\Pi wfsp : Wfspecification\ vs\ sp.$

$\Pi subs : Subsignature\ sign\ (Signature\ sp).$

$Wfspecification\ vs\ (export\_spec\ sp\ sign)$

$rename\_wfsp : \Pi vs : Var\_set.\Pi sp : Specification.$

$\Pi signm : Signature\_morphism.\Pi bij : Bijective\ (Signature\ sp)\ signm.$

$\Pi wfsp : Wfspecification\ vs\ sp.Wfspecification\ vs\ (rename\_spec\ sp\ signm)$

$reach\_wfsp : \Pi vs : Var\_set.\Pi sp : Specification.\Pi sign : Signature.$

$\Pi wfsp : Wfspecification\ vs\ sp.\Pi subs : Subsignature\ sign\ (Signature\ sp).$

$Wfspecification\ vs\ (reach\_spec\ sp\ sign)$

$behaviour\_wfsp : \Pi vs : Var\_set.\Pi sp : Specification.\Pi Obs, In : (List\ Ind\_sorts).$

$\Pi wfsp : Wfspecification\ vs\ sp.$

$\Pi subs : Subsort\ In\ (Signature\ sp).\Pi subs : Subsort\ Obs\ (Signature\ sp).$

$Wfspecification\ vs\ (behaviour\_spec\ sp\ sign)$

$abstract\_wfsp : \Pi vs : Var\_set.\Pi sp : Specification.\Pi Obs, In : (List\ Ind\_sorts).$

$\Pi wfsp : Wfspecification\ vs\ sp.$

$\Pi subs : Subsort\ In\ (Signature\ sp).\Pi subs : Subsort\ Obs\ (Signature\ sp).$

$Wfspecification\ vs\ (abstract\_spec\ sp\ sign)$

$quotient\_wfsp : \Pi vs : Var\_set.\Pi sp : Specification.\Pi Obs, In : (List\ Ind\_sorts).$

$\Pi wfsp : Wfspecification\ vs\ sp.$

$\Pi subs : Subsort\ In\ (Signature\ sp).\Pi subs : Subsort\ Obs\ (Signature\ sp).$

$Wfspecification\ vs\ (quotient\_spec\ sp\ sign)$

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 $Signature(SP') \hookrightarrow Signature(SP)$*
- $Models(SP) = Models(SP')|_{Signature(SP)}$.

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

$$(basic_{\ggg}) \qquad \frac{}{< \Sigma, \Phi > \ggg_X \ SPI} \ Signature(SPI) \ = \ \Sigma \ \wedge \ (SPI \ \models \ \Phi)$$

$$(sum_{\ggg}) \qquad \frac{\begin{array}{c} SP' \ \ggg_X \ rename \ \ SPI|_{inr(Signature(SP'))} \ \ \mathbf{by} \ \ inrsig^{-1} \\ SP \ \ggg_X \ rename \ \ SPI|_{inl(Signature(SP))} \ \ \mathbf{by} \ \ inlsig^{-1} \end{array}}{SP +_{\Sigma} SP' \ \ggg_X \ SPI}$$

$$(export_{\ggg}) \qquad \frac{\begin{array}{c} X \ \blacktriangleright \ SPI' \\ SP \ \ggg_X \ SPI' \end{array}}{SP|_{\Sigma} \ \ggg_X \ SPI} \ Signature(SPI) \ = \ \Sigma \ \wedge \ PEXTOF(SPI', SPI)$$

$$(reach_{\ggg}) \qquad \frac{SP \ \ggg_X \ SPI}{\mathbf{reach} \ \ SP \ \ \mathbf{with} \ \ (\mathcal{S}_{\mathcal{R}}, \mathcal{F}_{\mathcal{R}}) \ \ggg_X \ SPI} \ Mod(SPI) \ \models \ (\mathcal{S}_{\mathcal{R}}, \mathcal{F}_{\mathcal{R}})$$

$$(rename_{\ggg}) \qquad \frac{SP \ \ggg_X \ rename \ \ SPI \ \ \mathbf{by} \ \ \sigma^{-1}}{rename \ \ SP \ \ \mathbf{by} \ \ \sigma \ \ggg_X \ SPI}$$

$$(behaviour_{\ggg}) \qquad \frac{SP \ \ggg_X \ SPI/\approx}{\mathbf{behaviour} \ \ SP \ \ \mathbf{wrt} \ \ \approx \ggg_X \ SPI}$$

$$(abstract_{\ggg}) \qquad \frac{\mathbf{behaviour} \ \ SP \ \ \mathbf{wrt} \ \ \approx \ggg_X \ SPI}{\mathbf{abstract} \ \ SP \ \ \mathbf{by} \ \ \equiv \ggg_X \ SPI} \ Behc(SP)$$

$$(quotient_{\ggg}) \qquad \frac{\begin{array}{c} X \ \blacktriangleright \ SPI' \\ SP \ \ggg_X \ SPI' \end{array}}{SP/\approx \ggg_X \ SPI} \ Cond(SP, SPI, SPI')$$

where

$$Cond(SP, SPI, SPI') \ = \ (Signature(SPI) \ = \ Signature(SP) \ \wedge$$

$$Mod(SPI'/\approx) \ = \ Mod(SPI))$$

$$Behc(SP) \ = \ Models(SP) \ \subseteq \ Models(\mathbf{behaviour} \ \ SP \ \ \mathbf{wrt} \ \ \approx)$$

and

$$inl : Signature(SP) \ \rightarrow \ Signature(SP) +_{\Sigma} Signature(SP')$$

and

$$inr : Signature(SP') \rightarrow Signature(SP) +_{\Sigma} Signature(SP')$$

are the pushouts morphisms of $i : \Sigma \hookrightarrow Signature(SP)$ and $i' : \Sigma \hookrightarrow Signature(SP')$, $inl(Signature(SP)), inr(Signature(SP'))$ are the obvious subsignatures of $Signature(SP) +_{\Sigma} Signature(SP')$ and $inlsign : Signature(SP) \rightarrow inl(Signature(SP))$ and $inrsign : Signature(SP') \rightarrow inr(Signature(SP'))$ are the obvious signature morphisms defined with the pushouts morphisms *inl* and *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 of if there exists a derivation of the sequent $SP \ggg SPI$ in $\Delta_{\Pi^{ASL}_{AINS}\ggg}(SP \ggg SPI)$*

For the definition of the proof system for refinement we need the resulting signatures after applying a pushout morphism (inl,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:

- $inl\_sums : Specification \rightarrow Signature \rightarrow Specification \rightarrow Signature$
- $inr\_sums : Specification \rightarrow Signature \rightarrow Specification \rightarrow Signature$
- $inlsm\_sums : Specification \rightarrow Signature \rightarrow Specification \rightarrow Signature\_morphism$
- $inrsm\_sums : Specification \rightarrow Signature \rightarrow Specification \rightarrow Signature\_morphism$

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

Definition 5.2 *The type $Proof\_symbol$ is inductively defined by this incomplete set of constructors:.*

$$\begin{array}{l} a, \ldots, z : Var\_symbol \\ A, \ldots, Z : Var\_symbol \\ \_,', \$, \ldots : Var\_symbol \end{array}$$

Definition 5.3 *The type $Proof\_text$ is defined as $Ne\_list\ Proof\_text$.*

Definition 5.4 *The inductive relation*

$$Basic\_po : \Pi sp : Specification.\Pi fl : List\ Formula.\Pi pt : Proof\_text.Prop$$

is defined by the following constructors:

$$basicpo\_c : \Pi sp : Specification.\Pi fl : List\ Formula.\Pi pt : Proof\_text.$$

$$Basic\_po\ sp\ fl\ pt$$

Definition 5.5 *The inductive relation*

$$Pext\_po : \Pi sp, sp' : Specification.\Pi pt : Proof\_text.Prop$$

is defined by the following constructors:

$$pextpo\_c : \Pi sp, sp' : Specification.\Pi pt : Proof\_text.$$

$$Pext\_po\ sp\ sp'\ pt$$

Definition 5.6 *The inductive relation*

$$Reach\_po : \Pi sp : Specification.\Pi rsign : Signature.\Pi pt : Proof\_text.Prop$$

is defined by the following constructors:

$$reachpo\_c : \Pi sp : Specification.\Pi rsign : Signature.\Pi pt : Proof\_text.$$

$$Reach\_po\ sp\ rsign\ pt$$

Definition 5.7 *The inductive relation*

$$Behcomp\_po : \Pi sp : Specification.\Pi pt : Proof\_text.Prop$$

is defined by the following constructors:

$$behcomp\_c : \Pi sp : Specification.\Pi pt : Proof\_text.$$

$$Behcomp\_po\ sp\ pt$$

Definition 5.8 *The inductive relation*

$$Qmodeq\_po : \Pi sp, sp' : Specification.\Pi pt : Proof\_text.Prop$$

is defined by the following constructors:

$$qmodeqpo\_c : \Pi sp, sp' : Specification.\Pi pt : Proof\_text.$$

$$qmodeq\_po\ sp\ sp'\ pt$$

Definition 5.9 *The inductiive relation*

$$Proof\_obligation : Prop$$

is defined by the following constructors:

$basicpo\_cc : \Pi sp : Specification.\Pi fl : List\ Formula.\Pi pt : Proof\_text.$

$\Pi bpr : Basic\_po\ sp\ fl\ pt.Proof\_obligation$

$pextpo\_cc : \Pi sp, sp' : Specification.\Pi pt : Proof\_text.$

$\Pi epr : Pext\_po\ sp\ sp'\ pt.Proof\_obligation$

$reachpo\_cc \Pi sp : Specification.\Pi rsign : Signature.\Pi pt : Proof\_text.$

$\Pi rpr : Reach\_po\ sp\ rsign\ pt.Proof\_obligation$

$behcomp\_cc : \Pi sp : Specification.\Pi pt : Proof\_text.$

$\Pi bpr : Behcomp\_po\ sp\ pt.Proof\_obligation$

$qmodeqpo\_c : \Pi sp, sp' : Specification.\Pi pt : Proof\_text.$

$\Pi qpr : qmodeq\_po\ sp\ sp'\ pt.Proof\_obligation$

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*

$RefineASLFOL : \Pi sp : Specification.\Pi vs : Var\_set.\Pi sp' : Specification.Prop$

is defined by the following set of constructors:

$basic\_ref : \Pi vs : Var\_set.\Pi sign : Signature.\Pi fl : List\ Formula.$

$\Pi wfl : Wfforml\ vs\ fl.\Pi sp : Specification.\Pi pt : Proof\_text.$

$\Pi sames : Same\_signature\ sign\ (Signature\_sp\ sp).\Pi bpo : Basic\_po\ sp\ fl\ pt.$

$RefineASLFOL\ (base\_spec\ sign\ fl)\ vs\ sp$

$sum\_ref : \Pi sp, sp', spi : Specification.\Pi sign : Signature.\Pi vs : Var\_set.$

$\Pi refsp : RefineASLFOL\ sp\ vs$

$(rename\_spec\ (export\_spec\ spi\ (inl\_sums\ sp\ sign\ sp'))$

$(inverse(inlsm\_sums\ sp\ sign\ sp')).$

$\Pi refsp' : RefineASLFOL\ sp'\ vs$

$(rename\_spec\ (export\_spec\ spi\ (inr\_sums\ sp\ sign\ sp'))$

$(inverse(inrsm\_sums\ sp\ sign\ sp')).$

$RefineASLFOL\ (sum\_spec\ sp\ sign\ sp')\ vs\ spi$

$ren\_ref : \Pi vs : Var\_set.\Pi sp, spi : Specification.\Pi sm : Signature\_morphism.$

$\Pi refsp : RefineASLFOL\ sp\ vs\ (rename\_spec\ spi\ (inverse\_sm\ sm)).$

$RefineASLFOL\ (rename\_spec\ spi\ sm)\ vs\ sp$

$exp\_ref : \Pi vs : Var\_set.\Pi sp, spi, spi' : Specification.$

$\Pi sign : Signature.\Pi pt : Proof\_text$

$\Pi wfsp' : Wfspecification\ vs\ spi'.$

$\Pi sames : Samesignature\ sign\ (Signature\_sp\ spi).\Pi bpo : Pextof\_po\ spi'\ spi\ pt$

$\Pi refsp : RefineASLFOL\ sp\ vs\ spi'.$

$RefineASLFOL\ (export\_spec\ sp\ sign)\ vs\ spi$

$ref\_reach : \Pi vs : Var\_set.\Pi sp, spi : Specification.$

$\Pi sign : Signature.\Pi pt : Proof\_text.$

$\Pi reachpo : Reach\_po\ sp\ sign\ pt.$

$\Pi refr : RefineASLFOL\ sp\ vs\ spi$

$RefineASLFOL\ (reach\_spec\ sp\ sign)\ vs\ spi$

$ref\_behaviour : \Pi vs : Var\_set.\Pi sp, spi : Specification.\Pi sl, sl' : List\ Ind\_sorts.$

$\Pi refr : RefineASLFOL\ sp\ vs\ (quotient\_spec\ spi\ sl\ sl')$

$RefineASLFOL\ (behaviour\_spec\ sp\ sl\ sl')\ vs\ spi$

$ref\_abstract : \Pi vs : Var\_set.\Pi sp, spi : Specification.$

$\Pi sl, sl' : List\ Ind\_sorts.\Pi pt : Proof\_text$

$\Pi refr : RefineASLFOL\ (behaviour\_spec\ sp\ sl\ sl')\ vs\ spi$

$\Pi behpo : Behcomp\_po\ sp\ pt.$

$RefineASLFOL\ (abstract\_spec\ sp\ sl\ sl')\ vs\ spi$

$ref\_quotient : \Pi vs : Var\_set.\Pi sp, spi, spi' : Specification.$

$\Pi sl, sl' : List\ Ind\_sorts.\Pi pt : Proof\_text$

$\Pi wfspi' : Wfspecification\ vs\ spi'.$

$\Pi refr : RefineASLFOL\ sp\ vs\ spi'.$

$\Pi sams : Same\_signature\ (Signature\_sp\ sp)\ (Signature\_sp\ spi).$

$\Pi behpo : Qmodeq\_po\ spi\ spi'\ pt.$

$RefineASLFOL\ (quotient\_spec\ sp\ sl\ sl')\ vs\ spi$

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

$$\epsilon_{sp} : Var\_set \rightarrow SPEX(ASL) \rightarrow Specification$$

$$\epsilon_{sp}^{-1} : Var\_set \rightarrow Specification \rightarrow SPEX(ASL)$$

where $SPEX(ASL)$ denotes the set of specification expressions of ASL, we can prove the following theorem:

Theorem 5.2 *For any sequence of variables X, for any specification expression $sp, sp' \in SPEX(ASL)$ such that $X \blacktriangleright sp$ and $X \blacktriangleright sp'$, there exists a total injective function ϵ_{ref} between closed derivations of the judgement $sp \ggg sp'$ and the inhabitants of the inductive relation*

$$RefineASLFOL\ (\epsilon_{sp}\ (\epsilon_{vs}\ X)\ sp)\ (\epsilon_{vs}\ X)\ (\epsilon_{sp}\ (\epsilon_{vs}\ X)\ sp')$$

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

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 ϵ_{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 $Fail\_ref$.

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 get_wfspec which gets from the input a specification expression together with a proof which is well formed.

The function which will be denoted as Ref_tactic is inductively defined as follows:

$Ref\_tactic\ (base\_spec\ sign\ fl)\ vs\ sp\ =$

$if\ (fst\ (same\_signaturef\ sign\ (Signature\ sp)))\ then$

$(basic\_ref\ vs\ sign\ fl\ sp\ \text{“}BASIC\_PO''$

$(snd\ (same\_signaturef\ sign\ (Signature\ sp)))$

$(basicpo\_c\ sp\ fl\ \text{“}BASIC\_PO''),$

$[basicpo\_cc\ sp\ fl\ \text{“}BASIC\_PO''\ (basicpo\_c\ sp\ fl\ \text{“}BASIC\_PO'')])$

$else\ Fail\_ref$

$Ref\_tactic\ (sum\_spec\ sp\ sign\ sp')\ vs\ sp''\ =$

$(sum\_ref\ sp\ sp'\ sp''\ sign\ vs\ (fst\ reftactsp1)\ (fst\ reftactsp2),$

$concat\ (snd\ reftactsp1)\ (snd\ reftactsp2))$

$where$

$reftactsp1\ =\ (Ref\_tactic\ sp\ vs$

$(rename\_spec\ (export\_spec\ sp''\ (inl\_sums\ sp\ sign\ sp'))$

$(inverse(inlsm\_sums\ sp\ sign\ sp')).$

$reftactsp2\ =\ (Ref\_tactic\ sp\ vs$

$(rename\_spec\ (export\_spec\ sp''\ (inr\_sums\ sp\ sign\ sp'))$

$(inverse(inrsm\_sums\ sp\ sign\ sp')).$

$Ref\_tactic\ (rename\_spec\ sp\ signm)\ vs\ sp'\ =$

$(ren\_ref\ vs\ sp\ sp'\ sm(fst\ reftactsp), (snd\ reftactsp))$

$where$

$reftactsp\ =\ Ref\_tactic\ sp\ vs\ (rename\_spec\ sp'\ (inverse\_sm\ sm)$

$Ref\_tactic\ (export\_spec\ sp\ sign)\ vs\ sp'\ =$

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

$(exp\_ref\ vs\ sp\ sp'\ (fst\ getsp)\ sign\ \text{``}EXPORT\_PO''$

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

$(pextpo\_c\ (fst\ getsp)\ sp'\ \text{``}EXPORT\_PO'')\ (fst\ reftactsp),$

$(cons\ Proof\_obligation\ (pextpo\_cc\ (fst\ getsp)\ sp'\ \text{``}EXPORT\_PO''$

$(pextpo\_c\ (fst\ getsp)\ sp'\ \text{``}EXPORT\_PO''))\ (snd\ reftactsp))$

$else\ Fail\_ref$

$where$

$getsp\ =\ get\_wfspec$

$reftactsp\ =\ Ref\_tactic\ sp\ (fst\ getsp)$

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

$(ref\_reach\ vs\ sp\ sp'\ sign\ \text{``}REACH\_PO''$

$(reachpo\_c\ sp\ sign\ \text{``}REACH\_PO'')$

$(fst\ reftactsp),\ cons\ Proof\_obligation\ (reachpo\_cc\ sp\ sign\ \text{``}REACH\_PO''$

$(reachpo\_c\ sp\ sign\ \text{``}REACH\_PO''))\ (snd\ reftactsp))$

$where$

$reftactsp\ =\ Ref\_tactic\ sp\ vs\ sp'$

$Ref\_tactic\ (behaviour\_spec\ sp\ obsl\ inl)\ vs\ sp' =$

$(ref\_behaviour\ vs\ sp\ sp'\ obsl\ inl\ (fst\ reftactsp),\ (snd\ reftactsp))$

$where$

$reftactsp = Ref\_tactic\ sp\ vs\ (quotient\_spec\ sp'\ obsl\ inl)$

$Ref\_tactic\ (abstract\_spec\ sp\ obsl\ inl)\ vs\ sp' =$

$(ref\_abstract\ vs\ sp\ sp'\ obsl\ inl\ (fstreftactsp)\ (behcomp\_c\ sp\ \text{``}ABSTRACT\_PO''),$

$cons\ Proof\_Obligation\ (behcomp\_cc\ sp\ \text{``}ABSTRACT\_PO''$

$(behcomp\_c\ sp\ \text{``}ABSTRACT\_PO''))\ (snd\ reftactsp))$

$where$

$reftactsp = Ref\_tactic\ (behaviour\_spec\ sp'\ obsl\ inl)\ vs\ sp$

$Ref\_tactic\ (quotient\_spec\ sp\ obsl\ inl)\ vs\ sp' =$

$if\ (fst\ (same\_signaturef\ (Signature\_sp\ sp)\ (Signature\_sp\ sp')))\ then$

$(ref\_quotient\ vs\ sp\ sp'\ (fst\ getsp)\ obsl\ inl\ \text{``}QUOTIENT\_PO''$

$(snd\ getsp)\ (fst\ reftactsp)\ (snd\ (same\_signaturef$

$(Signature\_sp\ sp)\ (Signature\_sp\ sp')))$

$(qmodeq\_c\ sp\ sp'\ \text{``}QUOTIENT\_PO''),\ cons\ Proof\_obligation$

$(qmodeq\_cc\ sp\ sp'\ (qmodeq\_c\ sp\ sp'\ \text{``}QUOTIENT\_PO'')))$

$elseFail\_ref$

$where$

$getsp = get\_wfspec$
$reftactsp = Ref\_tactic\ sp\ vs\ (fst\ get\_wfspec)$

where we assume predefined the function same_signaturef 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

[1] Michel Bidoit, María Victoria Cengarle, and Rolf Hennicker. Proof systems for structured specifications and their refinements. Chapter 11 of the book Algebraic Foundations of Systems Specification.

[2] Healfdene Goguen. *A Typed Operational Semantics for Type Theory*. PhD thesis, University of Edinburgh, September 1994. Also published as Technical Report CST-110-94, Department of Computer Science.

[3] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[4] Rolf Hennicker. *Structured Specifications with Behavioural Operators: Semantics, Proof Methods and Applications*. Habilitationsschrift, Institut für Informatik, Ludwig-Maximilians-Universität München, June 1997.

[5] Zhaohui Luo. *Computation and Reasoning: A Type Theory for Computer Science*. Clarendon Press Oxford, 1994.

[6] James Hugh McKinna. *Deliverables: A Categorical Approach to Program Development in Type Theory*. PhD thesis, University of Edinburgh, November 1992.

[7] Nikos Mylonakis. Adequate encodings of proof systems for algebraic specifications in UTT. *Submitted to CADE-13 Workshop on Proof Search in Type-Theoretic Languages*, 1996.

[8] Nikos Mylonakis. *A type-theoretic approach to proof support for algebraic design frameworks*. PhD thesis, Universitat Politècnica de Catalunya, 2000. To appear.

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

A Predefined functions of this paper

A.1 Functions on signature morphisms

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

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

$$sort\_opl\ (get\_dom\_oppl\ (snd\ (snd\ signm))))$$

$$get\_dom\_spl\ spl\ = map\ fst\ spl$$

$$get\_dom\_oppl\ oppl\ = map\ fst\ oppl$$

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

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

$$sort\_opl\ (get\_ran\_oppl\ (snd\ (snd\ signm))))$$

$$get\_ran\_spl\ spl\ = map\ snd\ spl$$

$$get\_ran\_oppl\ oppl\ = map\ snd\ oppl$$

Definition A.3 *The function* $inverse\_sm : Signature\_morphism \rightarrow Signature\_morphism$

is defined as follows:

$$inverse\_sm\ sm\ =\ mkpair\ (get\_ran\_sm\ sm)\ (invert\_pairs\ sm)$$

$where$

$$invert\_pairs\ sm\ = mkpair(invp\_sl(fst\ (snd\ sm)))(invp\_opl(snd\ (snd\ sm)))$$

$$invp\_sl\ sl\ =\ map\ invp\ sl$$

$$invp\_opl\ sl\ =\ map\ invp\ opl$$

$$invp\ p\ =\ (snd\ p, fst\ p)$$

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:*

$$new\_index\ sign\ ind\ =\ mkpair\ (map\ (updinds\ ind)\ (fst\ sign)\ )$$

$$(map\ (updindop\ ind)\ (snd\ sign))$$

$where$

$$updinds\ s\ ind\ =\ (fst\ s, ind)$$

$$updindop\ op\ ind\ =\ (fst\ op, ind)$$

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

$$union\_Sign\ sign\ sign'\ =\ mkpair\ (union\_Srt\ (first\ sign)\ (first\ sign'))$$

$$(union\_Ops\ (snd\ sign)\ (snd\ sign'))$$

Definition A.6 *The function* $union\_Srt : (List\ Ind\_sorts) \rightarrow (List\ Ind\_sorts) \rightarrow$

$(List\ Ind\_sorts)$ *is defined as follows:*

$$union\_Srts\ l\ l' \ = \ Primrec\ (List\ Ind\_sorts)\ l'\ genc\_uSrts\ l$$

where

$$genc\_uSrts\ s\ sl\ slf = add\_if\_not\_in\_sl\ s\ slf$$

$$add\_if\_not\_in\_sl\ s\ sl \ = \ Primrec\ Bool\ (cons\ s\ sl)\ sl\ (not\_in\_sl\ s\ sl)$$

$$not\_in\_sl\ s\ sl \ = \ Primrec(List\ Ind\_sorts)\ true\ (genc\_ninsl\ s)\ sl$$

$$genc\_ninsl\ s\ s'\ sl\ b \ =$$

$$Primrec\ bool(not\_bool\ Eqbool\_Isrts\ s\ s')\ b\ b$$

Definition A.7 *The function* $union\_Ops : (List\ Ind\_ops) \rightarrow (List\ Ind\_ops) \rightarrow (List\ Ind\_ops)$ *is defined as follows:*

$$union\_Ops\ l\ l' \ = \ Primrec\ (List\ Ind\_ops)\ l'\ genc\_uOps\ l$$

where

$$genc\_uOps\ op\ opl\ oplf = add\_if\_not\_in\_opl\ op\ oplf$$

$$add\_if\_not\_in\_opl\ op\ opl \ = \ Primrec\ Bool\ (cons\ op\ opl)\ opl\ (not\_in\_opl\ op\ opl)$$

$$not\_in\_opl\ op\ opl \ = \ Primrec\ (list\ Ind\_ops)\ true\ (genc\_ninopl\ op)\ opl$$

$$genc\_ninopl\ op\ op'\ opl\ b = \ Primrec\ bool\ (not\_bool\ (Eqbool\_Iops\ op\ op'))\ b\ b$$

Definition A.8 *The function* $intersect\_Sign\ :\ Signature\ \rightarrow\ Signature\ \rightarrow$

Signature is defined as follows:

$$inrtersect\_Sign\ sign\ sign' \ = \ mkpair\ (fst\ (inter\_Srt\ (first\ sign)\ (first\ sign')))$$

$$(fst\ (inter\_Ops\ (snd\ sign)\ (snd\ sign')))$$

$$where$$

$$inter\_Srt\ sl\ sl' \ = \ Primrec\ (List\ Ind\_sorts)\ (nil, sl)\ addifinsecsl\ sl'$$

$$addifinsecsl\ s\ sl\ psl \ = \ Primrec\ bool\ (cons\ s\ (fst\ psl), snd\ psl)$$

$$psl\ (is\_in\_bool\ Eqbool\_Isrts\ (snd\ psl))$$

$$inter\_Ops\ opl\ opl' \ = \ Primrec\ (List\ Ind\_ops)\ (nil, opl)\ addifinsecopl\ sl'$$

$$addifinsecopl\ op\ opl\ popl \ = \ Primrec\ bool\ (cons\ s\ (fst\ popl), snd\ popl)$$

$$popl\ (is\_in\_bool\ Eqbool\_Iops\ (snd\ psl))$$

Definition A.9 *The function $diff\_Sign : Signature \to Signature \to Signature$ is defined as follows:*

$$diff\_Sign\ sign\ sign' \ = \ mkpair\ (diff\_Srt\ (first\ sign)\ (first\ sign'))$$

$$(diff\_Ops\ (snd\ sign)\ (snd\ sign'))$$

$$where$$

$$diff\_Srt\ sl\ sl' \ = Primrec\ (List\ Ind\_sorts)\ sl\ gencsl\_diff\ sl'$$

$$gencsl\_diff\ s\ sl\ sl' \ = \ remove\ Eqbool\_Isrts\ s\ sl'$$

$$diff\_Ops\ opl\ opl' \ = Primrec\ (List\ Ind\_ops)\ opl\ gencopl\_diff\ opl'$$

$$gencopl\_diff\ op\ opl\ opl' \ = \ remove\ Eqbool\_Iops\ op\ opl'$$

Definition A.10 *The function $nameclash\_sign : Signature \ \to \ Signature \ \to \ Signature \ \to \ Signature$ is defined as follows:*

$$nameclash\_sign\ signsp\ sign\ signsp' \ =$$

$$diff\_sign\ (intersect\_sign\ signsp\ signsp')\ sign$$

Definition A.11 *The function $Signature\_ind\_sp : Specification \ \to \ Sym\_index \ \to$*

$Signature$ is defined as follows:

$$Signature\_ind\_sp\ sp\ ind = Primrec\ Specification\ (basec\_sign\ ind)\ (sumc\_sign\ ind)\ (expc\_sign\ ind)$$

$$(renc\_sign\ ind)\ (reachc\_sign\ ind)\ (behc\_sign\ ind)\ (quotc\_sign\ ind)\ (abstrc\_sign\ ind)\ sp$$

$where$

$$basec\_sign\ ind\ sign\ fl\ =\ (new\_index\ sign\ ind, ind)$$

$$sumc\_sign\ ind\ sp\ sign\ sp'\ signsp\ signsp'\ =$$

$$mkpair\ (union\_sign\ (new\_index\ (nameclash\_sign\ (fst\ signsp)$$

$$sign\ (fst\ signsp'))$$

$$(next\_Si\ (maxind\_Si\ (snd\ signsp)\ (snd\ signsp'))))$$

$$(union\_sign\ (diff\_sign\ (diff\_sign\ (fst\ signsp)\ sign)$$

$$(nameclash\_sign\ (fst\ signsp)\ sign\ (fst\ signsp')))\ (fst\ signsp'))$$

$$(next\_Si\ (maxind\_Si\ (snd\ signsp)\ (snd\ signsp')))$$

$$renc\_sign\ ind\ sp\ signm\ signsp\ =\ (get\_ran\_sm\ signm, ind)$$

$$expc\_sign\ ind\ sp\ sign\ signsp\ =\ (sign, ind)$$

$$reachc\_sign\ ind\ sp\ reachsgn\ signsp\ =\ signsp$$

$$behc\_sign\ ind\ sp\ obssl\ inssl\ signsp\ =\ signsp$$

$$absc\_sign\ ind\ sp\ obssl\ inssl\ signsp\ =\ signsp$$

$$quoc\_sign\ ind\ sp\ obssl\ inssl\ signsp\ =\ signsp$$

Definition A.12 *The function* $Signature\_sp : Specification\ \rightarrow\rightarrow\ Signature$ *is defined as follows:*

$$Signature\_sp\ sp\ =\ fst\ (Signature\_ind\_sp\ sp\ first\_Vi)$$

A.3 Some inductive relations

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

$$basec\_Sams : \Pi sign : Same\_signature\ (mkpair\ (nil\ Ind\_sorts)\ (nil\ Ind\_ops))$$

$$(mkpair\ (nil\ Ind\_sorts)\ (nil\ Ind\_ops))$$

$$gencs\_Sams : \Pi s : Ind\_sorts.\Pi sign, sign' : Signature.\Pi sams : Same\_signature\ sign\ sign'.$$

$$Same\_signature\ (sort\_sl\ (cons\ s\ (fst\ sign), (snd\ sign)))$$

$$(sort\_sl\ (cons\ s\ (fst\ sign'), (snd\ sign')))$$

$$gencop\_Sams : \Pi op : Ops.\Pi sign, sign' : Signature.\Pi sams : Same\_signature\ sign\ sign'.$$

$$Same\_signature\ (fst\ sign, (sort\_opl\ (consop\ (snd\ sign)))$$

$$(fst\ sign, (sort\_opl\ (cons\ op\ (snd\ sign)))$$

Definition A.14 *The inductive relation* $Subsignature : \Pi sign, sign' : Signature.Prop$ *is defined by the following set of constructors:*

$$basec\_Subsign : \Pi sign : Signature.Subsignature\ (mkpair\ (nil\ Ind\_sorts)\ (nil\ Ind\_ops))\ sign$$

$$gencs\_Subsign : \Pi s : Ind\_sorts.\Pi sign, sign' : Signature.$$

$$\Pi isins : Is\_in\_List\ s\ (fst\ sign').$$

$$Subsignature\ (sort\_sl\ (cons\ s\ (fst\ sign), (snd\ sign)))\ sign'$$

$$gencop\_Subs : \Pi op : Ops.\Pi sign, sign' : Signature.\Pi isins : Is\_in\_List\ op\ (snd\ sign').$$

$$Subsignature\ (fst\ sign, (sort\_opl\ (cons\ op\ (snd\ sign)))\ sign'$$

Definition A.15 *The inductive relation* $Subsorts : \Pi sl : List\ Ind\_sorts.sign' : Signature.Prop$ *is defined by the following set of constructors:*

$$basec\_Subs : \Pi sign : Signature.Subsorts\ (nil\ Ind\_sorts)\ sign$$

$$gencs\_Subs : \Pi s : Ind\_sorts.\Pi sl : List\ Ind\_sorts.\Pi sign : Signature.$$

$$\Pi isins : Is\_in\_List\ s\ (fst\ sign').$$

$$Subsorts\ (sort\_sl\ (cons\ s\ sl))\ sign'$$

Definition A.16 *The inductive relation*

$$Bijective : \Pi sign : Signature.\Pi signm : Signature\_morphism.Prop$$

is defined by the following constructors:

$bij\_ctr : \Pi sign : Signature.\Pi signm : Signature\_morphism.$

$\Pi norepssd : Norep\_list\ Ind\_sorts\ Eqbool\_Isrts\ (fst\ (get\_dom\_sm\ signm)).$

$\Pi norepsst : Norep\_list\ Ind\_sorts\ Eqbool\_Isrts\ (fst\ (get\_ran\_sm\ signm)).$

$\Pi norepsopd : Norep\_list\ Ind\_ops\ Eqbool\_Iops\ (snd\ (get\_dom\_sm\ signm)).$

$\Pi norepsopt : Norep\_list\ Ind\_ops\ Eqbool\_Iops\ (snd\ (get\_ran\_sm\ signm)).$

$\Pi samesignd : Same\_signature\ sign\ (get\_dom\_sm\ signm).$

$\Pi samesignt : Same\_signature\ (first\ signm)\ (get\_ran\_sm\ signm).$

$Bijective\ sign\ signm$

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

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

$$inl\_sums\ sp\ sign\ sp' = Signature\_sp\ sp$$

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

$inr\_sums\ sp\ sign\ sp' =$

$union\_sign\ (new\_index\ (nameclash\_sign\ (Signature\_sp\ sp)\ sign\ (Signature\_sp\ sp'))$

$(next\_Vi\ (maxind\_Si\ (snd\ (Signature\_ind\_sp\ sp\ first\_Vi))$

$(snd\ (Signature\_ind\_sp\ sp'\ first\_Vi)))))$

$(diff\_sign\ (Signature\_sp\ sp')\ (nameclash\_sign\ (Signature\_sp\ sp)\ sign(Signature\_sp\ sp')))$

Definition A.19 *The function* $inlsm\_sums : Specification \rightarrow Signature \rightarrow$

$Specification \rightarrow Signature\_morphism$ *is defined as follows:*

$$inlsm\_sums\ sp\ sign\ sp' \ =$$

$$(join\ Ind\_sorts\ Ind\_sorts\ (fst\ (Signature\_sp\ sp))\ \ (fst\ (Signature\_sp\ sp)),$$

$$join\ Ind\_ops\ Ind\_ops\ (snd\ (Signature\_sp\ sp))(snd\ (Signature\_sp\ sp)))$$

Definition A.20 *The function* $inrsm\_sums : Specification \rightarrow Signature \rightarrow Specification \rightarrow Signature\_morphism$ *is defined as follows:*

$$inrsm\_sums\ sp\ sign\ sp' \ =$$

$$(concat\ (prod\ Ind\_sorts\ Ind\_sorts)\ (join\ Ind\_sorts\ Ind\_sorts$$

$$(Fst\ (nameclash\_sign\ (Signature\_sp\ sp)\ sign\ (Signature\_sp\ sp')))$$

$$(Fst\ (new\_index\ (nameclash\_sign\ (Signature\_sp\ sp)\ sign\ (Signature\_sp\ sp'))$$

$$(next\_Vi\ (maxind\_Si\ (snd\ (Signature\_ind\_sp\ sp\ first\_Vi))$$

$$(snd\ (Signature\_ind\_sp\ sp'\ first\_Vi)))))))$$

$$(join\ Ind\_sorts\ Ind\_sorts\ (Fst\ (diff\_sign\ (Signature\_sp\ sp'\ first\_Si)$$

$$(nameclash\_sign\ (Signature\_sp\ sp\ first\_Si)\ sign\ (Signature\_sp\ sp'\ first\_Si))))$$

$$(Fst\ (diff\_sign\ (Signature\_sp\ sp'\ first\_Si)\ (nameclash\_sign$$

$$(Fst\ (Signature\_sp\ sp\ first\_Si))\ sign\ (Signature\_sp\ sp'\ first\_Si)))))),$$

$(concat\ (prod\ Ind\_ops\ Ind\_ops)\ (join\ Ind\_ops\ Ind\_ops$

$(snd\ (nameclash\_sign\ (Signature\_sp\ sp\ first\_Si)\ sign\ (Signature\_sp\ sp'\ first\_Si)))$

$(snd\ (new\_index\ (nameclash\_sign\ (Signature\_sp\ sp\ first\_Si)$

$sign\ (Signature\_sp\ sp'\ first\_Si))$

$(next\_Vi\ (maxind\_Si\ (snd\ (Signature\_ind\_sp\ sp\ first\_Vi))$

$(snd\ (Signature\_ind\_sp\ sp'\ first\_Vi)))))))$

$(join\ Ind\_ops\ Ind\_ops\ (snd\ (diff\_sign\ (Signature\_sp\ sp'\ first\_Si)$

$(nameclash\_sign\ (Signature\_sp\ sp\ first\_Si)\ sign\ (Signature\_sp\ sp'\ first\_Si))))$

$(snd\ (diff\_sign\ (Signature\_sp\ sp'\ first\_Si)\ (nameclash\_sign$

$(Signature\_sp\ sp\ first\_Si)\ sign\ (Signature\_sp\ sp'\ first\_Si))))))$

References

[1] Michel Bidoit, María Victoria Cengarle, and Rolf Hennicker. Proof systems for structured specifications and their refinements. Chapter 11 of the book Algebraic Foundations of Systems Specification.

[2] Healfdene Goguen. *A Typed Operational Semantics for Type Theory.* PhD thesis, University of Edinburgh, September 1994. Also published as Technical Report CST-110-94, Department of Computer Science.

[3] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[4] Rolf Hennicker. *Structured Specifications with Behavioural Operators: Semantics, Proof Methods and Applications.* Habilitationsschrift, Institut für Informatik, Ludwig-Maximilians-Universität München, June 1997.

[5] Zhaohui Luo. *Computation and Reasoning: A Type Theory for Computer Science.* Clarendon Press Oxford, 1994.

[6] James Hugh McKinna. *Deliverables: A Categorical Approach to Program Development in Type Theory.* PhD thesis, University of Edinburgh, November 1992.

[7] Nikos Mylonakis. Adequate encodings of proof systems for algebraic specifications in UTT. *Submitted to CADE-13 Workshop on Proof Search in Type-Theoretic Languages*, 1996.

[8] Nikos Mylonakis. *A type-theoretic approach to proof support for algebraic design frameworks*. PhD thesis, Universitat Politècnica de Catalunya, 2000. To appear.

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