Chapter6:Adequate encodings of proof systems for ASL

1 Introduction

In this chapter, we present the encodings of two more interesting proof systems presented in the previous chapter and in the appendix you can find the complete encoding of the proof system for higher-order logic. In this chapter, first we explain how to encode the finitary versions of the non-compositional proof systems for ASL inductively defined by specification expressions and finally, we present a full encoding for the proof system for refinement of ASL specifications.

All the previous encodings which we have presented so far are adequate (including the ones of the appendix), in the sense that there exists 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, but the second presented in this chapter is just full in the sense that there exists a total injection 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

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

This last encoding 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.

2 Adequate encoding of Π_{HOL}^{RBASL}

The encoding of the non-compositional proof system Π_{HOL}^{RBASL} is based on the proof system for higher-order logic presented in the appendix. Here, we just present the general lines to develop the adequate encoding.

The main difficulty is the calculation of the symbols of the signature associated to a proof system of a given specification expression. In order to give an effective procedure to calculate the different pushouts which appear in the definitions, the encoded signature is extended with symbol indexes which are used to solve the name clashes in specification expressions with, for example, the sum operator or the rename and export operator. 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 and we have to differentiate between the hidden symbols of a specification expression of the form $SP|_\Sigma$ with the visible symbols with the same name after applying a renaming to $SP|_\Sigma$. Additionally, for the behavioural operators we have to proceed in a similar way as in the rename operator to control the name clashes generated by the disjoint copy of the signature of the specification which is required.

The encoded signature is calculated with the function ϵ_{sym} which given a specification expression and a symbol index returns the signature (with symbol indexes associated to sorts and operations) associated to the proof system of the given specification expression.

First, we define signatures with indexes and then we define the function ϵ_{sym}. 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.

Definition 2.1 *For any $\Sigma \in |AlgSig|$, the inductive relation Sorts is inductively defined by the following set of constructors:*

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

Remark: *We assume predefined the equality function $Eqbool\_Srts : Sorts \rightarrow Sorts \rightarrow Bool$*

Definition 2.2 *For any $\Sigma \in |AlgSig|$, the inductive relation Ops is inductively defined by the following set of constructors:*

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

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

$f : s_1 \times \ldots \times s_n \rightarrow s \in \Sigma$ *and f is overloaded in* $\Sigma\}$

Remark: *We assume predefined the equality function $Eqbool\_Ops : Ops \rightarrow Ops \rightarrow Bool$ section.*

Definition 2.3 *The type Sym_index is defined as Var_index.*

Remark: *We assume predefined the function $maxind\_Si : Sym\_index \rightarrow Sym\_index \rightarrow Sym\_index$ which given two indexes returns the maximum of the two.*

Definition 2.4 *The type Ind_sorts is defined as $(Pair\ Sorts\ Sort\_index)$.*

Definition 2.5 *The type Ind_ops is defined as $(Pair\ Ops\ Op\_index)$.*

Definition 2.6 *The type of signatures with indexes is defined as*

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

We assume predefined the following functions and inductive relations:

- the function $Ltbool\_Srts : Ind\_sorts \rightarrow Ind\_sorts \rightarrow Bool$ 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 [McK92] 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 2.7 *The inductive relation*

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

is defined by the following constructors:

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

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

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

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

$$Wfsignature\ (sl, opl)$$

Definition 2.8 *The encoding function ϵ_{sym} which given a symbol index and a specification expression returns a pair the signature with indexes associated to the proof system of the specification expression together with the highest index used is inductively defined as follows:*

$$\epsilon_{sym}\ (ind, <\Sigma, \Phi>)\ =\ mkpair\ (\epsilon_{symisl}\ (ind, Sorts(\Sigma)), \epsilon_{symopl}\ (ind, Ops(\Sigma))\ ind$$

$where$

$$\epsilon_{symisl}\ (ind, [])\ =\ (nil\ Ind\_sorts)$$

$$\epsilon_{symisl}(ind, cons\ s\ sl)\ =\ (cons\ Ind\_sorts\ (mkpair\ s\_Srts\ ind)\ (\epsilon_{symisl}\ (ind, sl)))$$

$$\epsilon_{symopl}\ (ind, [])\ =\ nil\ ops$$

$$\epsilon_{symopl}(ind, cons\ op : s_1 \times \ldots \times s_n \rightarrow s\ opl)\ =$$

$$(cons\ (mkpair\ (op\_Ops\ ind)$$

$$(\epsilon_{symopl}\ (ind, opl)))\quad ,\ if\ op\ not\ overloaded\ in\ \Sigma$$

$$(cons\ (mkpair\ (op\_s_1\_\ldots\_s_n\_s\_Ops\ ind)$$

$$(\epsilon_{symopl}\ (ind, opl)))\quad ,\ if\ op\ overloaded\ in\ \Sigma$$

$$\epsilon_{sym}\ (ind, SP_1\ +_\Sigma\ SP_2)\ =$$

$$mkpair((r_{inl''}\ (SP_1, \Sigma, SP_2) \cup r_{inr''}(SP_1, \Sigma, SP_2)))\ (next\_Vi\ (maxim\_Si\ ind_1\ ind_2))$$

$where$

$$mkpair\ indsymsp_1\ ind_1\ =\ \epsilon_{sym}\ (ind, SP_1)$$

$$mkpair\ indsymsp_2\ ind_2\ =\ \epsilon_{sym}\ (ind, SP_2)$$

$$nameclashset\ =\ \{symb \mid symb\ \in\ Symbols(SP_1),$$

$$symb,\ \in\ Symbols(SP_2), symb\ \notin\ \Sigma\}$$

$$r_{inl''}(SP_1, \Sigma, SP_2)\ =\ \{(symb, next\_Vi\ (maxind\_Si\ ind_1\ ind_2))\ |$$

$$symb \in nameclashset\ \} \cup$$

$$\{(symb, ind)\ |(symb, ind) \in indsymsp_1, symb \notin nameclashset\}$$

$$r_{inr''}(SP_1, \Sigma, SP_2) \ = \ \{(symb, ind) \mid (symb, ind) \in indsymsp_2 \ \}$$

$$\epsilon_{sym}\ (ind, SP_1|_\Sigma) \ = \ \epsilon_{sym}\ (ind, SP_1)$$

$$\epsilon_{sym}\ (ind, rename \quad SP \quad \textbf{by} \quad \sigma) \ =$$

$$mkpair\ ((r_{\sigma''}(SP, \sigma)) \ \cup \ \{(symb, ind) \mid symb \in \Sigma\})(next\_Vi\ ind_1)$$

$$where$$

$$mkpair\ indsymsp_1\ ind_1 \ = \ \epsilon_{sym}\ (ind, SP)$$

$$nameclashset \ = \ \{symb \mid symb \ \in \ Symbols(SP), symb \ \notin \ Signature(SP)$$

$$symb \ \in \ \Sigma \ \}$$

$$r_{\sigma''}(SP, \Sigma) \ = \ \{(symb, next\_Vi\ ind_1) \mid symb \ \in \ nameclashset\} \cup$$

$$\{(symb, ind') \mid (symb, ind') \ \in \ indsymsp_1,$$

$$symb \notin nameclashset, symb \notin Signature(SP)\}$$

$$where\ \sigma : Signature(SP) \rightarrow \Sigma$$

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

$$\epsilon_{sym}\ (ind, \textbf{behaviour} \quad SP \quad \textbf{wrt} \quad \approx) \ = \ mkpair\ ((r_{Copy''}(SP, Copy)) \cup$$

$$\{(symb, ind) \mid symb \in Copy(Signature(SP))\})\ (next\_Vi\ ind_1)$$

$$where$$

$$mkpair\ indsymsp_1\ ind_1 \ = \ \epsilon_{sym}\ (ind, SP)$$

$$nameclashset \;=\; \{symb \mid symb \in Symbols(SP), symb \notin Signature(SP),$$

$$symb \in Copy(Signature(SP)) \}$$

$$r_{Copy''}(SP,\Sigma) \;=\; \{(symb, next\_Vi\ ind_1) \mid symb \in nameclashset\} \cup$$

$$\{(symb, ind') \mid (symb, ind') \in indsymsp_1, symb \notin nameclashset,$$

$$\{(\pi_s\_Ops, (next\_Vi\ ind_1)) \mid s \in Sorts(Signature(SP))\} \cup$$

$$\{(\sim_s\_Ops, next\_Vi\ ind_1) \mid s \in Sorts(Signature(SP))\}$$

$$\epsilon_{sym}\ (ind, \textbf{abstract}\quad SP\quad \textbf{by}\quad \equiv) \;=\; \epsilon_{sym}\ (ind, \textbf{behaviour}\quad SP/\approx\quad \textbf{wrt}\quad \approx)$$

$$\epsilon_{sym}\ (ind, SP/\approx) \;=\; mkpair\ ((r_{Copy''-1}(ind_1, SP, Copy)) \cup$$

$$\{(symb, ind) \mid symb \in Copy(Signature(SP))\})\ (next\_Vi\ ind_1)$$

$$where$$

$$mkpair\ indsymsp_1\ ind_1 \;=\; \epsilon_{sym}\ (ind, SP)$$

$$nameclashset \;=\; \{symb \mid symb \in Symbols(SP), symb \notin Signature(SP)$$

$$, symb \in Copy(Signature(SP)) \}$$

$$r_{Copy''-1}(SP,\Sigma) \;=\; \{(symb, next\_Vi\ ind_1) \mid symb \in nameclashset\} \cup$$

$$\{(symb, ind') \mid (symb, ind') \in indsymsp_1,$$

$$symb \notin nameclashset\} \cup$$

$$\{(\pi_s\_Ops, (next\_Vi\ ind_1)) \mid s \in Sorts(Signature(SP))\} \cup$$

$$\{(\sim_s\_Ops, next\_Vi\ ind_1) \mid s \in Sorts(Signature(SP))\}$$

The rest of the encoding of the proof system is very similar to the encoding of higher-order logic, with the additional task to encode for each specification expression the specific assumptions which are defined by the function Γenv.

For example, the encoding of the proof system for a specification expression of the form $SP_1 +_\Sigma SP_2$ would consist of the encoding of the rules of the proof system of higher-order logic for formulas with the symbols of SP_1 and for formulas with the symbols of SP_2 (both appropriately indexed via the previous function) and the encoding of the assumptions would be the encoding of the assumptions associated to SP_1 together with the ones associated to SP_2 appropiately renamed to the symbols with indexes of the proof system.

3 Encoding of the proof system for refinement

In this section, we present the full encoding of the following proof system for refinement of ASL presented in previous chapter. We explicit the set of free variables and well-formedness conditions in order to be able to give an adequate encoding of the proof system:

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

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

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

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

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

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

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

$$(quotient_{\ggg}) \qquad \frac{X \blacktriangleright SPI' \quad SP \ggg_X SPI'}{SP/\approx \ggg_X SPI} \; Cond(SP, SPI, SPI')$$

First, we give adequate encodings of well-formed specification expressions and after that we give the encoding of the proof system for refinement. The sentences of well-formed specifications will be the higher-order terms of type *Prop* presented in the appendix but this is quite irrellevant for the presentation.

Definition 3.1 *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:*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Definition 3.2 *The type signature morphism is defined as follows:*

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

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

Remark: *The first component of type Signature is the domain signature of the signature morphism.*

In appendix G, 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 3.3 *The inductive type Specification is defined by the following set of constructors:*

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

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

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

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

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

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

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

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

In appendix G, you can 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 3.4 *The inductive relation*

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

is defined by the following set of constructors:

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

$\Pi htl : Holterm\_list.\Pi wfs : Wfsignature\ sign.$

$\Pi wfhtl : Wfhtermlist\ vs\ htl.Wfspec\ (base\_spec\ sign\ htl)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 start the encoding of the proof system for refinement presented in last chapter. First, we define the inductive relations which represent the proof obligations of the proof system.

Definition 3.5 *The type* $Proof\_symbol$ *is defined as* Var_smbol.

Definition 3.6 *The type* $Proof\_text$ *is defined as* $Ne\_list\ Proof\_text$.

Definition 3.7 *The inductive relation*

$$Basic\_po : \Pi sp : Specification.\Pi ht : Holterm.\Pi pt : Proof\_text.Prop$$

is defined by the following constructors:

$$basicpo\_c : \Pi sp : Specification.\Pi ht : List\ Holterm.\Pi pt : Proof\_text.$$

$$Basic\_po\ sp\ ht\ pt$$

Definition 3.8 *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 3.9 *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 3.10 *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 3.11 *The inductive relation*

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

is defined by the following constructors:

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

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

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 3.12 *The inductive relation*

$RefineRBASLHOL : \Pi sp : Specification.\Pi vs : Holvar\_set.\Pi sp' : Specification.Prop$

is defined by the following set of constructors:

$basic\_ref : \Pi vs : Holvar\_set.\Pi sign : Signature.\Pi htl : List\ Holterm.$

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

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

$RefineRBASLHOL\ (base\_spec\ sign\ htl)\ vs\ sp$

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

$\Pi refsp : RefineRBASLHOL\ sp\ vs$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\epsilon_{sp} : Holvar\_set \rightarrow SPEX(RBASL) \rightarrow Specification$$

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

we can prove the following theorem:

Theorem 3.13 *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*

$$RefineRBASLHOL\ (\epsilon_{sp}\ (\epsilon_{vs}\ X)\ sp)\ (\epsilon_{vs}\ X)\ (\epsilon_{sp}\ (\epsilon_{vs}\ X)\ sp').$$

There exists also 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:
The proof is similar to the ones presented for the proof systems for higher-order logic but obviously a little bit simpler and the definition of the function ϵ_{ref}^{-1} is performed in the same way as in the proof systems for higher-order logic.

References

[McK92] James Hugh McKinna. *Deliverables: A Categorical Approach to Program Development in Type Theory.* PhD thesis, University of Edinburgh, November 1992.