SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Constructing Linear Types in Isabelle/HOL

Felix Krayer

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Constructing Linear Types in Isabelle/HOL

Konstruktion linearer Typen in Isabelle/HOL

Author:	Felix Krayer
Examiner:	Florian Bruse
Supervisor:	Dmitriy Traytel, Tobias Nipkow
Submission Date:	13-11-2025

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 13-11-2025 Felix Krayer

Acknowledgments

Abstract

Contents

1 Introduction

- Datatypes in general
- Datatypes in Isabelle/HOL are built on Bounded Natural Functors (BNFs) (defined in [TPB12])
- Structure of the Thesis

2 Background

This Chapter serves to introduce BNFs and their generalization to Map-Restricted Bounded Natural Functors (MRBNFs).

2.1 Bounded Natural Functors (BNFs)

As described in Chapter 1, BNFs are essential for constructing datatypes and codatatypes in Isabelle/HOL. Especially for defining a datatype with recursion it is required that the type constructor α list is registered as a BNF, i.e., it fulfills the BNF-axioms. For example the following **datatype** command only succeeds if α list, is a BNF.

$$\textbf{datatype}\ \alpha\ \mathsf{ex} = \mathrm{A}\ \text{"}(\alpha \times (\alpha\ \mathsf{ex}))\ \mathsf{list}\text{"}$$

Since BNFs are closed under composition and fixpoints, the resulting datatype (here α ex) can be automatically registered as a BNF as well.

We write type variables as greek letters (α, β, . . .) in this thesis. However, in the Isabelle proof assistant type variables are written with a "'" in front of a name, e.g., 'a list. To copy our examples to Isabelle, one has to replace these greek letters with "'" variables. Alternatively, a "'" can be prepended to the greek letters, since for example $'\alpha$ is a valid type variable in Isabelle.

The type variables of a BNF are divided into two groups: *live* and *dead* variables or *lives* and *deads*. Live variables can be used for recursive datatype definitions, while dead ones do not allow for this. We take the function type $\alpha \Rightarrow \beta$ as an example. It's first type argument α is dead, while the second one β is live. Thus, of the following the first command succeeds while the second one fails

$$\textbf{datatype}\ \alpha\ \mathsf{success} = \mathrm{S1} \mid \mathrm{S2}\ \text{"}\alpha \Rightarrow \alpha\ \mathsf{success}\text{"}$$
$$\textbf{datatype}\ \alpha\ \mathsf{fail} = \mathrm{F1} \mid \mathrm{F2}\ \text{"}\alpha\ \mathsf{fail} \Rightarrow \alpha\text{"}$$

2.1.1 BNF constants

A BNF F with l live variables is characterized by one map and l set functions, a bound and a relator.

Map function and functors

The $l+1$-ary map function or *mapper* takes one function for each live of F as arguments as well as one F element. The domain types of these functions are the lives of F. These functions are recursively applied to the components of an element. The result is a new element of type F, where the original type variables are replaced by the range types of the mapped functions. Taking the α list type as an example, a BNF with one live α, the mapper has the type $\mathsf{map}_{\mathsf{list}} :: (\alpha \Rightarrow \alpha') \Rightarrow \alpha\ \mathsf{list} \Rightarrow \alpha'\ \mathsf{list}$.

To make F with its mapper a *functor* on the universe of all types, the mapper has to fulfill two axioms [TPB12]. First, mapping the *id* function on all lives over an element should leave it unchanged, which is formalized in Figure MAP_ID. The second property Figure MAP_COMP is concerned with mapping compositions and reads as follows: Mapping two lists of functions over an element, e.g., first $f_1 \ldots f_l$ and then $g_1 \ldots g_l$, should produce the same result as mapping the index-wise composition $(g_1 \circ f_1) \ldots (g_l \circ f_l)$ over it once. A type constructor F with a map function map_F fulfilling these two properties is considered a functor.

Set functions and naturality

A set function or *setter* is defined for each of the l live variables. Applied to an F-element, the i-th setter returns the set of all components that are part of the element and correspond to the i-th live. For example, the setter of the list type returns the set of elements in the list. We note here that when we write i as an index, we assume it to be in the range $1 \leq i \leq l$.

The set functions together with the mapper give rise to another property. We want the setters $\mathsf{set}_{F,i}$ to be natural transformations from F and map_F to the set and image function. Thus, they should fulfill the Figure SET_MAP axiom. It states that taking the i-th set of an F after mapping $f_1 \ldots f_l$ to it, results in the same set as if i-th set was taken from the original F before the image of f_i was applied to it. Figure 2.1 shows a visualization of this axiom and reads as follows: Starting from an F element first applying the setter and then mapping a function (path through the top right) results in the same as first mapping the function and then applying the setter (path through the bottom left).

Bound and boundedness

Lastly, the BNF needs an infinite cardinal as a bound. This bound may depend on the cardinalities of the dead variables, but not on the of the live variables. In Isabelle/HOL cardinals are implemented as minimal wellorders with respect to isomorphisms [BPT14]. While details about this implementation are certainly interesting, we will not focus on

$$
\begin{tikzcd}
(\alpha_1, \ldots \alpha_l)\ F \arrow[r, "\mathsf{set}_{F,i}"] \arrow[d, "\mathsf{map}_F\ f_1\ \ldots\ f_l"'] & \alpha_i\ \mathsf{set} \arrow[d, "image\ f_i"] \\
(\beta_1, \ldots \beta_l)\ F \arrow[r, "\mathsf{set}_{F,i}"] & \beta_i\ \mathsf{set}
\end{tikzcd}
$$

for all i where α_i is a live variable of F

Figure 2.1: $\mathsf{set}_{F,i}$ as a natural transformation

these details in this thesis. For example *natLeq*, the cardinal that originates from the $\leq$ order on natural numbers, is equivalent to the smallest infinite cardinal $\aleph_0$.

Besides being a cardinal order, the bound is required to be infinite, i.e., at least $\aleph_0$ with respect to the cardinal order $\leq_o$, and regular. Regularity means that an infinite cardinal κ is stable under union, i.e., the union of any two sets of smaller cardinality than κ also has a smaller cardinality than κ.

The bound is used in the Figure SET_BD axiom to ensure that the sets obtained by the setters are bounded. This ensures that the branching of a recursively defined datatype is also bounded and thus the resulting type F as well.

Relator and shapes

The relator is used to build a relation on F by relating the components of an F element. It takes one relation for each live, that relates the corresponding type variables of the two Fs that are to be related. As an example we give the type and definition of the relator for the product type as follows:

$$
\begin{aligned}
&\mathsf{rel}_{\mathsf{prod}} ::\ (\alpha \Rightarrow \alpha' \Rightarrow \mathsf{bool}) \Rightarrow (\beta \Rightarrow \beta' \Rightarrow \mathsf{bool}) \Rightarrow (\alpha \times \beta) \Rightarrow (\alpha' \times \beta') \Rightarrow \mathsf{bool} \\
&\mathsf{rel}_{\mathsf{prod}}\ R\ Q\ p_1\ p_2 :=\ R\ (fst\ p_1)\ (fst\ p_2) \wedge Q\ (snd\ p_1)\ (snd\ p_2)
\end{aligned}
$$

Considering the list type again, we make an interesting observation: There are some α lists *xs* and *ys* that the relator cannot relate, no matter which α relation is chosen. The relator on lists is index-wise defined, i.e., the α relation must relate the elements of both lists for each index. Consequently lists of different length cannot be positively related. We think of the length of a list as its *shape*. We can generalize this idea of shape to an arbitrary type constructor F. The shape of an F element is defined by the way it is constructed and the relator can only ever relate those that have the same or equivalent shape, i.e., it will always evaluate to *false*, when two elements of different

(MAP_ID)	$\mathsf{map}_F\ \overline{id}^l\ x = x$
(MAP_COMP)	$\mathsf{map}_F\ \overline{g}^l\ (\mathsf{map}_F\ \overline{f}^l\ x) = \mathsf{map}_F\ \overline{(g \circ f)}^l\ x$
(MAP_CONG)	$(\forall i.\ \forall z \in \mathsf{set}_{F,i}\ x.\ f_i\ z = g_i\ z) \Longrightarrow \mathsf{map}_F\ \overline{f}^l\ x = \mathsf{map}_F\ \overline{g}^l\ x$
(SET_MAP)	$\forall i.\ \mathsf{set}_{F,i}(\mathsf{map}_F\ \overline{f}^l\ x) = f_i\ \grave{}\ \mathsf{set}_{F,i}\ x$
(BD)	$\mathsf{infinite}\ \mathsf{bd}_F \wedge \mathsf{regular}\ \mathsf{bd}_F \wedge \mathsf{cardinal\_order}\ \mathsf{bd}_F$
(SET_BD)	$\forall i.\ \lvert\mathsf{set}_{F,i}\ x\rvert <_o \mathsf{bd}_F$
(REL_COMPP)	$\mathsf{rel}_F\ \overline{R}^l\ \bullet\ \mathsf{rel}_F\ \overline{Q}^l\ = \mathsf{rel}_F\ \overline{(R\ \bullet\ Q)}^l$
(IN_REL)	$\mathsf{rel}_F\ \overline{R}^l\ x\ y =$
	$\exists z.\ (\forall i.\ \mathsf{set}_{F,i}\ z \subseteq \{(a,b).\ R_i\ a\ b\}) \wedge \mathsf{map}_F\ \overline{fst}^l\ z = x \wedge \mathsf{map}_F\ \overline{snd}^l\ z = y$

where $\grave{}$ is the image function on sets, $\bullet$ is the composition of relations and $<_o$ is the less than relation on cardinals

Figure 2.2: The BNF axioms

shape are given, regardless of the relations given to the relator. We can think of an element of type F as a container that has a certain *shape* with slots for *components*. These components are elements of the type constructor's type arguments.

2.1.2 BNF-axioms

We formalize the BNF-axioms in Figure 2.2 where we use the notation $\overline{f}^l = f_1 \dots f_l$ for the arguments of the mapper and the relator Additionally to the axioms we already motivated in Subsection 2.1.1 (Figure MAP_ID and Figure MAP_COMP for the functoriality of F, Figure SET_MAP to ensure that the setters are natural transformations and the boundedness of the setters Figure SET_BD), we have four additional ones.

One of those is the the congruency Figure MAP_CONG of the map function. It states that if two (lists of) functions $\overline{f}^l$ and $\overline{g}^l$ are equal when applied to the corresponding sets of all components of an F (obtained through the setters), then mapping these two lists of functions over the F each produces the same result. When this property holds, we can be sure, that the mapper only depends on how the functions $\overline{f}^l$ behave on the components of the F element.

The axiom Figure BD just ensures that the bound bd_F is a suitable cardinal, i.e., a regular and infinite one.

Distributivity of the relator is formulated in Figure IN_REL. We note here, that for showing that a type constructor is a BNF, it is only necessary to prove the inclusion $(\mathsf{rel}_F\ \overline{R}^l\ \bullet\ \mathsf{rel}_F\ \overline{Q}^l)\ x\ y \Rightarrow \mathsf{rel}_F\ \overline{(R\ \bullet\ Q)}^l\ x\ y$. The other direction follows automatically from this and the next axiom, weak pullback preservation.

Lastly, weak pullback preservation Figure IN_REL is the most abstract and complex axiom. The idea is that two elements x and y of the type $\alpha\ F$ are related through a relation R iff there exists a z that acts as a "zipped" version of x and y. The components of this z are R_i-related pairs of the components of x and y, where the first position in the pair corresponds to x and the second one to y.

2.1.3 Non-emptiness witnesses

BNF carry non-emptiness witnesses as proof that the type contains at least one element. Witnesses may depend on a subset of the BNF's live variables. For example a witness of $(\alpha_1,\ \ldots\ \alpha_l)\ F$ that depends on the first and last type variable of F, this witness has the type $\mathsf{wit}_F ::\ \alpha_1 \Rightarrow \alpha_l \Rightarrow (\alpha_1,\ \ldots\ \alpha_l)\ F$. It denotes that given witnesses for the types α_1 and α_l, a witness for F can be constructed.

Witnesses have to fulfill the following properties: For all type variables α_i the witness depends on, the witness may only contain the α_i elements w_i, that were given to the witness as arguments, i.e., $textscset_{F,i}$ applied to the witness evaluates to the singleton $\{w_i\}$. Furthermore, the witness must not contain any elements of the live type variables α_j, the witness does not depend on, i.e., $\textsc{set}_{F,j}$ must be empty. We formalize these properties in the following where $\bar{w}$ denotes the arguments that the witness depends on.

(WITS) $\qquad \forall i.\ \mathsf{set}_{F,i}\ (\mathsf{wit}_F\ \bar{w}) = (\mathtt{if}\ \mathsf{wit}_F \text{ depends on } \alpha_i\ \mathtt{then}\ \{w_i\}\ \mathtt{else}\ \emptyset)$

If multiple types of witnesses exist for a given F, then the ones with the fewest arguments are most useful for showing non-emptiness. Concretely, we say a witness $\mathsf{wit}_{F,1}$ *subsumes* $\mathsf{wit}_{F,2}$, when $\mathsf{wit}_{F,1}$ depends on a true subset of the arguments of $\mathsf{wit}_{F,2}$. In this case we ignore the subsumed witness, as the other one is more useful. However, when two witnesses have overlapping dependencies but neither depends on a subset of the other we are interested in both, even if one has a smaller number of arguments than the other.

2.1.4 BNF examples

Further examples of BNFs are is the product type $(\alpha,\ \beta)$ prod, a binary type constructor with infix notation $\alpha \times \beta$, and the type of finite sets α fset. The latter is interesting for the reason that it is a subtype of the set type, which is not a BNF. By enforcing finiteness for the elements of the type it is possible to give a bound for the set function, fulfilling the Figure SET_BD axiom, which is not possible for the unrestricted set type. Since unboundedness is the only reason that the set type is not a BNF, α fset can be shown to be a BNF.

To show, how BNFs can be combined to create new ones, we consider the type constructor (α, β) plist = $(\alpha \times \beta)$ list. We define for it a map function ($\mathsf{map}_{\mathsf{plist}}$) and two set functions ($\mathsf{set1}_{\mathsf{plist}}$ and $\mathsf{set2}_{\mathsf{plist}}$) as well as a relator $\mathsf{rel}_{\mathsf{plist}}\ R\ Q$. The exact definitions are given as such:

$$\begin{aligned}
\mathsf{map}_{\mathsf{plist}}\ f\ g\ &= \mathsf{map}_{\mathsf{list}}\ (\mathsf{map}_{\mathsf{prod}}\ f\ g) \\
\mathsf{set1}_{\mathsf{plist}}\ xs &= \mathsf{set}_{\mathsf{list}}\ (\mathsf{map}_{\mathsf{list}}\ fst\ xs) \\
\mathsf{set2}_{\mathsf{plist}}\ xs &= \mathsf{set}_{\mathsf{list}}\ (\mathsf{map}_{\mathsf{list}}\ snd\ xs) \\
\mathsf{rel}_{\mathsf{plist}}\ R\ Q &= \mathsf{rel}_{\mathsf{list}}(\mathsf{rel}_{\mathsf{prod}}\ R\ Q)
\end{aligned}$$

where we use the standard map, set and relator functions of the list and product type.

To show that (α, β) plist is a BNF, we have to prove the BNF-axioms for it. Besides the definitions above, we give $\aleph_0$ as the bound $\mathsf{bd}_{\mathsf{plist}}$.

2.2 Syntaxes with bindings

2.3 Map-Restricted Bounded Natural Functors (MRBNFs)

Type constructors that involve names or bindings often violate the requirements of BNFs. Considering for example the type of distinct lists α dlist, a subtype of α list that describes only lists containing pairwise distinct α atoms. The issue with this type is that the standard map function on lists cannot guarantee that the resulting list is still distinct, i.e., that it is still part of the type. Thus in BNF terms the type variable of α dlist is dead. However, by restricting the mapper to only use bijections, the distinctness of the resulting list can be ensured.

MRBNFs are a generalization of BNFs. Restricting the map function of a functor to *small-support* functions or *small-support bijections* for certain type variables allows us to reason about type constructors in terms of BNF properties, even in cases where this would not be possible otherwise. We call type variables that that are restricted to small-support functions *free* variables or *frees* and those restricted to small-support bijections *bound* variables or *bounds*. This allows us to define MRBNFs with four types of variables (lives, frees, bounds and deads) as opposed to BNFs which only distinguish between lives and deads. Our example from the beginning of this section, the distinct list α dlist is a MRBNF with α as a bound variable.

A small-support function leaves most arguments unchanged, meaning it acts like the identity function on them. Concretely defined, the cardinality of the set of arguments the function changes must be smaller than the cardinality of the argument type itself:

$$\mathsf{small\_supp}\ f = |\{x :: \alpha.\ f\ x \neq x\}| <_o |\Omega_\alpha|$$

where Ω_α is the universe of type α.

Considering a polymorphic type that is meant to represent simple λ-terms, where α is the space of variable names. If we want to substitute a free variable x in a term T by a term N, we may run into the following problem: If $T = \lambda y.T'$, we need to ensure that there are no name clashes with y in the new term N before we substitute x by N in T'. This is done by choosing a fresh y' and renaming y to y' in T'.

The mapper and setters are expanded to work for the frees and bounds just as they do for lives. Thus, a MRBNF F with l lives, fr frees and b bounds has $l + fr + b$ setters and a mapper with arity $l + fr + b + 1$. Since the mapper takes small-support functions and bijections for the free and bound variables, which have the same type for their domain and range, this transfers to F as well. This means that the type variables for frees and bounds are the same for the F argument of the mapper and the result, while the lives can change type.

We keep the original relator that only relates live variables with given relations and relates the free and bound variables to be with equality. Thinking in our model of F elements being shapes with atoms in slots, the regular relator rel_F requires the frees and bounds in each slot to be the same for both elements that are compared.

To relate F elements that are not equal in all frees and bounds, we introduce a new map-restricted relator $\mathsf{mr\_rel}_F$. It takes a function for each free and bound - with the appropriate restrictions to small-support and bijectivity - in addition to the relations for the lives. The new arguments are placed in front of the relations for the live variables. It is then defined in terms of the relator as follows:

$$\mathsf{mr\_rel}_F\ \langle \overline{u}^b\ \overline{v}^{fr} \rangle\ \overline{R}^l\ x\ y = \mathsf{rel}_F\ \overline{R}^l\ x\ (\mathsf{map}_F\ \langle \overline{id}^l\ \overline{v}^{fr}\ \overline{u}^b \rangle\ y)$$

where we write f^l for the functions or relations of the live variables $f_1 \ldots f_l$ and analogously v^{fr} and u^b for frees and bounds. Furthermore, we write the arguments of the map function as $[\overline{f}^l\ \overline{v}^{fr}\ \overline{u}^b]$. The argument order of the mapper might be different, as the lives, bounds and frees do not have to be separated, but can be interlaced. For example for the type $(\alpha,\ \beta,\ \gamma)\ F$ where α and γ are free, while β is bound, the mapper has the following type:

$$\mathsf{map}_F :: (\alpha \Rightarrow \alpha) \Rightarrow (\beta \Rightarrow \beta) \Rightarrow (\gamma \Rightarrow \gamma) \Rightarrow (\alpha,\ \beta,\ \gamma)\ F \Rightarrow (\alpha,\ \beta,\ \gamma)\ F$$

Thus, in our notation $\mathsf{map}_F\ \langle \overline{f}^l\ \overline{v}^{fr}\ \overline{u}^b \rangle$ we assume the $\langle \dots \rangle$ to interlace the arguments correctly to $\mathsf{map}_F\ v_1\ u_1\ v_2$. Analogously for the $\mathsf{mr\_rel}_F$, $\langle \overline{u}^b\ \overline{v}^{fr} \rangle$ interlaces the functions according to the order of bounds and frees.

2.3.1 MRBNF axioms

MRBNFs require the same axioms as BNFs with slight modifications. We take the formalized axioms from Figure 2.2 as a base and explain the differences.

For the Figure MAP_COMP, Figure MAP_CONG and Figure SET_MAP axioms, we add the assumptions that the functions that correspond to frees and bounds are small-support functions and that the ones corresponding to bounds are additionally bijections.

Furthermore, while Figure REL_COMPP stays unchanged, using the original relator, Figure IN_REL is changed to be defined in terms of the map-restricted relator.

2.3.2 binder datatypes

MRBNFs can be used in a **binder_datatype** command to produce a datatype with bindings.

In the resulting MRBNF the free and bound type variables are required to be *large* and *regular*. Largeness is necessary to ensure that there are always fresh names available. It is defined as the cardinality of the type being at least $\aleph_0$ or $\aleph_1$ depending on whether it is a datatype of codatatype. In Isabelle the requirements of largeness and regularity are combined in dedicated type classes, `var` and `covar` respectively.

TODO: cite [Bla+19]

3 Linearizing MRBNFs

3.1 Linearization of MRBNFs (In theory)

In this section we define the linearization of a MRBNF F on a subset of it's *live* variables. Linearization means, that the resulting type only contains elements for which all atoms of the linearized variables are distinct. We say F is *non-repetitive* on these variables. This type is also a MRBNF with the same variable types (*live*, *free*, *bound*, *dead*), except for the linearized variables that change their type from *live* to *bound*. This means that the new map function is restricted to only allow bijective and small-support functions on these variables.

We formalize the idea of distinctness of components as *non-repetitiveness* on the linearized variables. In our notation we use L as the set of indices of live variables we linearize on, i.e., $L \subseteq \{1 \ldots l\}$. Furthermore we define a combinator $\overline{R}^l \bowtie^L \overline{Q}^l = [(\texttt{if } i \in L \texttt{ then } Q_i \texttt{ else } R_i) \texttt{ for } 1 \leq i \leq l]$ that selects the second argument if i is a variable to be linearized on and the first one, if not.

3.1.1 Non-repetitiveness

At the core of linearization lies the notion of *non-repetitiveness*. An element x of a type is considered to be non-repetitive with respect to a type variable α if it does not contain repeating α-atoms. For example, a α list is non-repetitive, if all of its α-elements it contains are pairwise distinct. To define non-repetitiveness for an arbitrary MRBNF, we have to express this property in terms of its map, set and relator functions. Considering α lists again, we can show a list xs to be distinct, iff for each other list ys of the same length, we can find a function f such that $ys = \mathsf{map}_{\mathsf{list}}\ f\ xs$. If xs were not distinct, there must exist two indices with the same α element in xs. Furthermore, there exists a ys that has different elements at these two indices and thus a function mapping xs to ys cannot exist, since it would have to map two same elements in xs to two differing ones in ys.

In Subsection 2.1.1 we proposed the idea to think about elements of a BNF (or MRBNF) F as containers with a certain shape with atoms in slots specified by the shape. Using this model, we can generalize the notion of lists having the same length to F elements having the same shape. We can express this through the relator by using the *top* relation that relates everything with each other as the argument. Thus, we give the definition of equivalent shape and non-repetitiveness for list:

$$\mathsf{eq\_shape}_{\mathsf{list}}\ x\ y = \mathsf{rel}_{\mathsf{list}}\ top\ x\ y$$
$$\mathsf{nonrep}_{\mathsf{list}}\ x = \forall y.\ \mathsf{eq\_shape}_{\mathsf{list}}\ x\ y \Longrightarrow (\exists f.\ y = \mathsf{map}_{\mathsf{list}}\ f\ x)$$

Note that we use the regular relator that only relates live variables with given relations while it requires equality for all frees and bounds.

More interesting is the case of $(\alpha,\ \beta)$ alist which we only want to be non-repetitive on α. For our purpose of defining non-repetitiveness on a subset of the live variables, we fix the other live variables to be equal when defining equivalent shape.

Based on this, x is a non-repetitive element, if for all other elements y with equal shape, a function exists through which x can be mapped to y. In our example of list, this holds for all lists with distinct elements (given a second list, one can easily define a function mapping the distinct elements of x to that list). It does not hold for lists with repeating elements, because no f exists that could map two equal elements at different positions in this list to distinct elements in an arbitrary second list.

For MRBNFs with more than one live variable, we can give a definitions of *non-repetitiveness* and having *equal shape* on a subset L of the live variables. In that case, we consider x and y of type F to have equal shape with respect to the variables $\{\alpha_i.\ i \in L\}$, iff they are *equal* in the atoms corresponding to the non linearized lives and are related with *top* in on the linearized variables. Consequently for the map in the nonrep definition, the *id* function is applied to the non linearized lives, since they are already required to be equal.

$$\mathsf{eq\_shape}^L_F\ x\ y = \mathsf{rel}_F\ (\overline{(=)}^l \bowtie^L \overline{top}^l)\ x\ y$$
$$\mathsf{nonrep}^L_F\ x = \forall y.\ \mathsf{eq\_shape}^L_F\ x\ y \Longrightarrow (\exists \overline{f}^l.\ y = \mathsf{map}_f\ \langle(\overline{id}^l \bowtie^L \overline{f}^l)\ \overline{id}^{fr}\ \overline{id}^b\rangle\ x)$$

3.1.2 Conditions for linearization

A MRBNFs has to fulfill two properties to be linearized. First, to ensure that the resulting type constructor is non-empty, it is required, that there exists a non-repetitive element (with respect to the linearized variables): $\exists x.\ \mathsf{nonrep}\ x$

Furthermore, even though MRBNFs are already required to preserve weak pullbacks defined as Figure IN_REL in Figure 2.2, for the linearization it is required that they preserve *all* pullbacks. Formalized this means that the existence of z in the equation has to be fulfilled uniquely, i.e., for each R-related x and y there exists *exactly one* z fulfilling the property Figure IN_REL. For example the strong pullback preservation is fulfilled by the α list and $\alpha\ \beta$ prod functor but not by α fset, the type constructor for finite sets of αs.

We note here that the requirement of strong pullback preservation can be omitted, when the MRBNF is linearized on all its live variables, i.e., when the linearized MRBNF has no live variables. This is because in this case the *relation exchange* lemma explained in Subsection 3.1.3 becomes trivial. In all other cases, that lemma is the sole reason, strong pullback preservation is required.

3.1.3 Intermediate lemmas

We want to prove the MRBNF axioms for the linearized MRBNF. For this we utilize some intermediate lemmas which we present in this section.

F is strong From the pullback preservation with uniqueness we can prove the following lemma. In fact this notion of strength is equivalent to pullback preservation:

(F_STRONG)
$$\mathsf{rel}_F\ \overline{R}^l\ x\ y\ \wedge\ \mathsf{rel}_F\ \overline{Q}^l\ x\ y \Longrightarrow \mathsf{rel}_F\ \overline{(inf\ R\ Q)}^l\ x\ y$$

where the infimum inf of two relations R and Q relates exactly those elements that are related by both R and Q.

Relation exchange The *exchange of relations* is a consequence of the previous property, paragraph F_STRONG: If two elements x and y are related through the relator with two different lists $\overline{R}^l = R_1 \dots R_l$ and $\overline{Q}^l = Q_1 \dots Q_l$ of atom-level relations, then x and y are also related with any index-wise combination of $\overline{R}^l$ or $\overline{Q}^l$. For each index i either the relation R_i or Q_i is selected.

For our purpose of linearization, we are specifically interested in the case, where for all live variables that we linearize on the relation from $\overline{R}^l$ is chosen and for all others the relation from $\overline{Q}^l$ relation, i.e., the result of the combinator $\bowtie^L$. This results in the following lemma for a MRBNF F:

(REL_EXCHANGE)
$$\mathsf{rel}_F\ \overline{R}^l\ x\ y\ \wedge\ \mathsf{rel}_F\ \overline{Q}^l\ x\ y \Longrightarrow \mathsf{rel}_F\ (\overline{Q}^l \bowtie^L \overline{R}^l)\ x\ y$$

In the specific case, that the MRBNF is linearized on *all* of it's live variables, the combination that is chosen is exactly $\overline{R}$. Then the lemma becomes trivial, since its goal is equal to it's first assumption in this case.

As a consequence of this, the previous lemma paragraph F_STRONG is not needed to prove this lemma. Furthermore, this lemma is the sole reason why paragraph F_STRONG and strong pullback preservation are needed for the linearization. Thus the requirement of pullback preservation can be lifted, in the case that the linearization is applied to all live variables at the same time.

map peresrving non-repetitiveness

$$\mathsf{small\_supp}\ \overline{v}^{fr}\ \wedge\ \mathsf{small\_supp}\ \overline{u}^{b}\ \wedge\ \mathsf{bijective}\ \overline{u}^{b}\ \wedge$$
$$\mathsf{nonrep}_F^L\ x\ \wedge\ \mathsf{bijective}\ \overline{f}^{l}\ \Longrightarrow \mathsf{nonrep}_F^L\ (\mathsf{map}_F\ \langle(\overline{g}^{l} \bowtie_L \overline{f}^{l})\ \overline{v}^{fr}\ \overline{u}^{b}\rangle\ x)$$

3.1.4 Proving the MRBNF axioms

3.1.5 Lifting Witnesses

Existing witnesses of the original MRBNF that do not depend on any of the linearized variables can be lifted to be witnesses of the linearized MRBNF.

For this it is necessary to show that they are non-repetitive on the linearized elements, i.e., that they are part of the new type. From WITS (Subsection 2.1.3) we know that any witness not depending on the linearized lives does not contain atoms from these lives. Thus, we can show that these witnesses are non-repetitive, since an element with no α atoms is trivially non-repetitive on α.

Other witnesses that depend on the linearized variables cannot be lifted and have to be discarded. Even if they are non-repetitive, witnesses of a MRBNF may only depend on lives and not on bounds, which the linearized lives turn into.

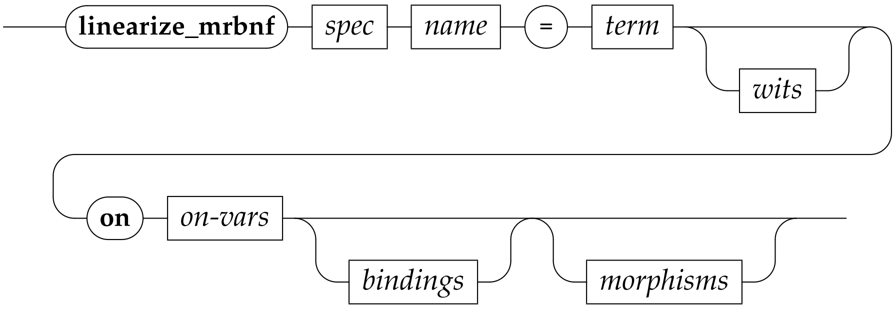
Additionally, new witnesses may be specified for the resulting MRBNF. For these the property WITS defined in Subsection 2.1.3 has to be proven, i.e., that they only consist of the atoms given to them as arguments.

When an liftable witness of the original MRBNF exists or a new witness fulfilling WITS is specified, the existence of a non-repetitive element we motivated in Subsection 3.1.2 is trivially proven.

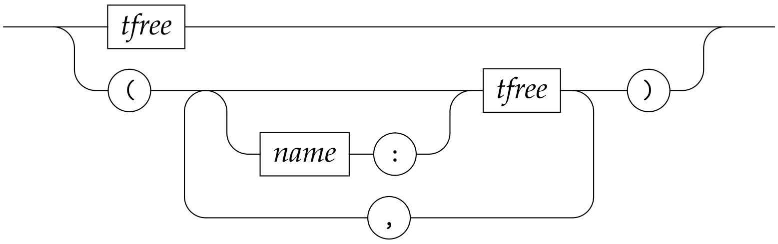
3.1.6 Preservation of strength

3.2 Linearization of MRBNFs (In Isabelle)

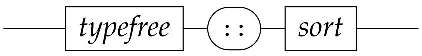
We implement a command that allows the user to linearize an existing MRBNF or BNF on one or multiple of it's live variables. The syntax of the command is given in the following:



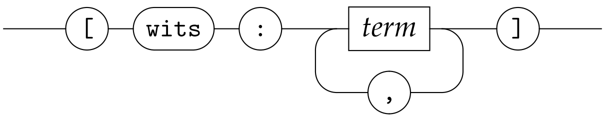
spec



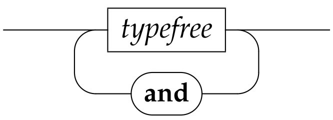
tfree



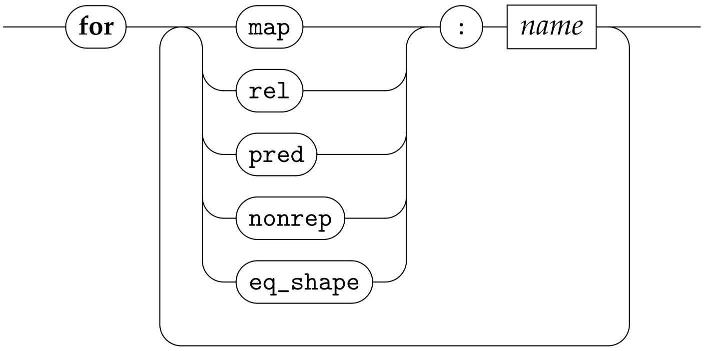
wits



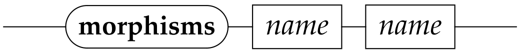
on-vars



bindings



morphisms



With this command, we can linearize our example by writing the following line in Isabelle:

$$\textbf{linearize\_mrbnf}\ (\text{keys: } \alpha :: \text{var},\ \text{vals: } \beta)\ \text{alist} = (\alpha :: \text{var} \times \beta)\ \text{list}\ \textbf{on}\ \alpha$$

Since for $(\alpha \times \beta)$ list both type variables are live and we only linearize on α, it is necessary to prove strong pullback preservation for this MRBNF.

After the user has written the command, the conditions for linearization we presented in Subsection 3.1.2 have to be proven, i.e., non-emptiness of the linear type and strong pullback preservation.

These conditions are given dynamically to the user. For example, it is only necessary to show strong pullback preservation, when the resulting MRBNF has live variables remaining. Furthermore, as mentioned in Subsection 3.1.5, the non-emptiness of the non-repetitive type is easily proven when the user specified a non-emptiness witness, or a liftable witness of the original type exists. Thus, the user is not asked to show the existence of a non-repetitive element in these cases.

Furthermore, since the original MRBNF already fulfills weak pullback preservation, we extract the uniqueness property of strong pullback preservation and require the user to prove only this. Strong pullback preservation can be proven from weak pullback preservation together with the uniqueness we specify as follows:

$$\begin{aligned}\forall x\ y.\ (&\mathsf{map}_F\ [\overline{fst}^{l}\ \overline{id}^{fr}\ \overline{id}^{b}]\ x = \mathsf{map}_F\ [\overline{fst}^{l}\ \overline{id}^{fr}\ \overline{id}^{b}]\ y \wedge \\ &\mathsf{map}_F\ [\overline{snd}^{l}\ \overline{id}^{fr}\ \overline{id}^{b}]\ x = \mathsf{map}_F\ [\overline{snd}^{l}\ \overline{id}^{fr}\ \overline{id}^{b}]\ y) \Longrightarrow \\ &x = y\end{aligned}$$

4 Examples

4.1 POPLmark challenge: Pattern

The POPLmark challenge [Ayd+05] presents a selection of problems to benchmark the progress in formalizing programming language metatheory. The challenges are built around formalizing aspects of *System* $F_{<:}$ calculus, a polymorphic typed lambda calculus with subtyping. We are interested in part 2B of this challenge, which has the goal to formalize and proof *type soundness* for terms with pattern matching over records. Type soundness is considered in terms of *preservation* (evaluating a term preserves its type) and *progress* (a term is either a value or can be evaluated).

We focus on the `record` terms `pattern-let`. A record is a term defined as a set of pairs, where the first element is a label and the second element a term: $\{(\mathtt{l}_j, \mathtt{t}_j)\}$. The labels l within a record must be pairwise distinct. A pattern is defined as either a typed variable or a set of (label, patten) pairs with pairwise distinct labels: $\mathtt{p} ::= \mathtt{x} : \mathtt{T} \mid \{(\mathtt{l}_j, \mathtt{p}_j)\}$

A formalization of part 2B of the POPLmark challenge in Isabelle/HOL is presented by Blanchette et al. [Bla+19]. They use *binder_datatypes* to abstract types, variables and terms. A central notion in this formalization is the *labeled finite set* (α, β) lfset that is used in the representation of records and patterns. This type constructor is a subtype of $(\alpha \times \beta)$ fset that only includes elements that are non-repetitive on α. This restriction is necessary, because for both records and patterns the label α must be mutually distinct, i.e., the set representing them has to be non-repetitive.

While by construction $(\alpha \times \beta)$ fset is a BNF (and an MRBNF since all BNFs are also MRBNFs) with both variables being live, (α, β) lfset is a MRBNFs with α as a bound variable, since it is non-repetitive on α. While this is a linearization, the finite set on pairs does not fulfill strong pullback preservation. Thus the approach and command we presented in Chapter 3 cannot be used here. Because of an alternate, equivalent description on non-repetitiveness specific to this type, it is still possible to manually linearize this MRBNF.

For the pattern a different type is used. It is constructed by linearizing an intermediate type prepat that is defined using the **datatype** command:

datatype $(\alpha,\ \beta)$ prepat = PPVar "α" "β typ" | PPRec "(string, $(\alpha,\ \beta)$ prepat) lfset"

Abbreviations

BNF Bounded Natural Functor

MRBNF Map-Restricted Bounded Natural Functor

List of Figures

List of Tables

Bibliography

[Ayd+05] B. E. Aydemir, A. Bohannon, M. Fairbairn, J. N. Foster, B. C. Pierce, P. Sewell, D. Vytiniotis, G. Washburn, S. Weirich, and S. Zdancewic. "Mechanized metatheory for the masses: The POPLmark challenge." In: *International Conference on Theorem Proving in Higher Order Logics*. Springer. 2005, pp. 50–65.

[Bla+19] J. C. Blanchette, L. Gheri, A. Popescu, and D. Traytel. "Bindings as bounded natural functors." In: *Proceedings of the ACM on Programming Languages* 3.POPL (2019), pp. 1–34.

[BPT14] J. C. Blanchette, A. Popescu, and D. Traytel. "Cardinals in Isabelle/HOL." In: *International Conference on Interactive Theorem Proving*. Springer. 2014, pp. 111–127.

[TPB12] D. Traytel, A. Popescu, and J. C. Blanchette. "Foundational, compositional (co) datatypes for higher-order logic: Category theory applied to theorem proving." In: *2012 27th Annual IEEE Symposium on Logic in Computer Science*. IEEE. 2012, pp. 596–605.