

Lifting Definition Option*

René Thiemann

December 14, 2021

Abstract

We implemented a command, **lift-definition-option**, which can be used to easily generate elements of a restricted type $\{x :: 'a. P x\}$, provided the definition is of the form $\lambda y_1 \dots y_n. \text{if } \text{check } y_1 \dots y_n \text{ then } \text{Some } (\text{generate } y_1 \dots y_n :: 'a) \text{ else } \text{None}$ and $\text{check } y_1 \dots y_n \implies P (\text{generate } y_1 \dots y_n)$ can be proven.

In principle, such a definition is also directly possible using one invocation of **lift-definition**. However, then this definition will not be suitable for code-generation. To this end, we automated a more complex construction of Joachim Breitner which is amenable for code-generation, and where the test $\text{check } y_1 \dots y_n$ will only be performed once. In the automation, one auxiliary type is created, and Isabelle's lifting- and transfer-package is invoked several times.

This entry is outdated as in the meantime the lifting- and transfer-package has the desired functionality in an even more general way. Therefore, only the examples are kept.

Contents

1	Examples	2
1.1	A simple restricted type without type-parameters	2
1.2	Examples with type-parameters in the restricted type.	2
1.3	Example from IsaFoR/CeTA	3
1.4	Code generation tests and derived theorems	3

```
theory Lifting-Definition-Option-Examples
imports
  Main
begin
```

*This research is supported by FWF (Austrian Science Fund) project Y 757.

1 Examples

1.1 A simple restricted type without type-parameters

```
typedef restricted = { i :: int. i mod 2 = 0 } morphisms base restricted
by (intro exI[of - 4]) auto
setup-lifting type-definition-restricted
```

Let us start with just using a sufficient criterion for testing for even numbers, without actually generating them, i.e., where the generator is just the identity function.

```
lift-definition(code-dt) restricted-of-simple :: int ⇒ restricted option is
λ x :: int. if x ∈ {0, 2, 4, 6} then Some x else None by auto
```

We can also take several input arguments for the test, and generate a more complex value.

```
lift-definition(code-dt) restricted-of-many-args :: nat ⇒ int ⇒ bool ⇒ restricted
option is
λ x y (b :: bool). if int x + y = 5 then Some ((int x + 1) * (y + 1)) else None
by clarsimp presburger
```

No problem to use type parameters.

```
lift-definition(code-dt) restricted-of-poly :: 'b list ⇒ restricted option is
λ xs :: 'b list. if length xs = 2 then Some (int (length (xs))) else None by auto
```

1.2 Examples with type-parameters in the restricted type.

```
typedef 'f restrictedf = { xs :: 'f list. length xs < 3 } morphisms basef restrictedf
by (intro exI[of - Nil]) auto
setup-lifting type-definition-restrictedf
```

It does not matter, if we take the same or different type-parameters in the lift-definition.

```
lift-definition(code-dt) test1 :: 'g ⇒ nat ⇒ 'g restrictedf option is
λ (e :: 'g) x. if x < 2 then Some (replicate x e) else None by auto
```

```
lift-definition(code-dt) test2 :: 'f ⇒ nat ⇒ 'f restrictedf option is
λ (e :: 'f) x. if x < 2 then Some (replicate x e) else None by auto
```

Tests with multiple type-parameters.

```
typedef ('a,'f) restr = { (xs :: 'a list,ys :: 'f list) . length xs = length ys }
morphisms base' restr
by (rule exI[of - ([], [])], auto)
setup-lifting type-definition-restr
```

```
lift-definition(code-dt) restr-of-pair :: 'g ⇒ 'e list ⇒ nat ⇒ nat ⇒ ('e,nat) restr
option is
λ (z :: 'g) (xs :: 'e list) (y :: nat) n. if length xs = n then Some (xs,replicate n y)
else None
```

by *auto*

1.3 Example from IsaFoR/CeTA

An argument filter is a mapping π from n-ary function symbols into lists of positions, i.e., where each position is between 0 and n-1. In IsaFoR, (Isabelle's Formalization of Rewriting) and CeTA [1], the corresponding certifier for term rewriting related properties, this is modelled as follows, where a partial argument filter in a map is extended to a full one by means of an default filter.

```
typedef 'f af = { ( $\pi :: 'f \times nat \Rightarrow nat\ list$ ). ( $\forall f\ n.$  set ( $\pi (f,n)$ )  $\subseteq \{0 ..< n\}$ )}  
morphisms af Abs-af by (rule exI[of -  $\lambda$  -. []], auto)
```

```
setup-lifting type-definition-af
```

```
type-synonym 'f af-impl = (('f  $\times$  nat)  $\times$  nat list)list
```

```
fun fun-of-map-fun :: ('a  $\Rightarrow$  'b option)  $\Rightarrow$  ('a  $\Rightarrow$  'b)  $\Rightarrow$  ('a  $\Rightarrow$  'b) where  
  fun-of-map-fun m f a = (case m a of Some b  $\Rightarrow$  b | None  $\Rightarrow$  f a)
```

```
lift-definition(code-dt) af-of :: 'f af-impl  $\Rightarrow$  'f af option is  
   $\lambda$  s :: 'f af-impl. if ( $\forall$  fidx  $\in$  set s. ( $\forall$  i  $\in$  set (snd fidx). i < snd (fst fidx)))  
    then Some (fun-of-map-fun (map-of s) ( $\lambda$  (f,n). [0 ..< n])) else None  
using map-of-SomeD by (fastforce split: option.splits)
```

1.4 Code generation tests and derived theorems

```
export-code
```

```
  restricted-of-many-args  
  restricted-of-simple  
  restricted-of-poly  
  test1  
  test2  
  restr-of-pair  
  af-of
```

```
in Haskell
```

```
lemma restricted-of-simple-Some:
```

```
  restricted-of-simple x = Some r  $\Longrightarrow$  base r = x
```

```
using restricted-of-simple.rep-eq[of x]
```

```
apply (split if-splits)
```

```
apply (simp-all only: option.map option.inject option.simps(3))
```

```
done
```

```
end
```

Acknowledgements

We thank Andreas Lochbihler for pointing us to Joachim’s solution, and we thank Makarius Wenzel for explaining us, how we can go back from states to local theories within Isabelle/ML.

References

- [1] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468. Springer, 2009.