

Fresh identifiers

Andrei Popescu Thomas Bauereiss

May 14, 2024

Abstract

This entry defines a type class with an operator returning a fresh identifier, given a set of already used identifiers and a preferred identifier. The entry provides a default instantiation for any infinite type, as well as executable instantiations for natural numbers and strings.

Contents

1	The type class <i>fresh</i>	1
2	Fresh identifier generation for natural numbers	2
3	Fresh identifier generation for strings	3
3.1	A partial order on strings	3
3.2	Incrementing a string	4
3.3	The fresh-identifier operator	4
3.4	Lifting to string literals	5
4	Fresh identifier generation for infinite types	6

1 The type class *fresh*

```
theory Fresh
  imports Main
begin
```

A type in this class comes with a mechanism to generate fresh items. The fresh operator takes a list of items to be avoided, xs , and a preferred element to be generated, x .

It is required that implementations of fresh for specific types produce x if possible (i.e., if not in xs).

While not required, it is also expected that, if x is not possible, then implementation produces an element that is as close to x as possible, given a notion of distance.

```

class fresh =
  fixes fresh :: 'a set  $\Rightarrow$  'a  $\Rightarrow$  'a
  assumes fresh-notIn:  $\bigwedge$  xs x. finite xs  $\Longrightarrow$  fresh xs x  $\notin$  xs
  and fresh-eq:  $\bigwedge$  xs x. x  $\notin$  xs  $\Longrightarrow$  fresh xs x = x

```

The type class *fresh* is essentially the same as the type class *infinite* but with an emphasis on fresh item generation.

```

class infinite =
  assumes infinite-UNIV:  $\neg$  finite (UNIV :: 'a set)

```

We can subclass *fresh* to *infinite* since the latter has no associated operators (in particular, no additional operators w.r.t. the former).

```

subclass (in fresh) infinite
   $\langle$ proof $\rangle$ 

```

end

2 Fresh identifier generation for natural numbers

```

theory Fresh-Nat
  imports Fresh
begin

```

Assuming $x \leq y$, *fresh2 xs x y* returns an element outside the interval (x, y) that is fresh for *xs* and closest to this interval, favoring smaller elements:

```

function fresh2 :: nat set  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  nat where
fresh2 xs x y =
  (if x  $\notin$  xs  $\vee$  infinite xs then x else
   if y  $\notin$  xs then y else
   fresh2 xs (x-1) (y+1))
 $\langle$ proof $\rangle$ 
termination
 $\langle$ proof $\rangle$ 

```

```

lemma fresh2-notIn: finite xs  $\Longrightarrow$  fresh2 xs x y  $\notin$  xs
 $\langle$ proof $\rangle$ 

```

```

lemma fresh2-eq: x  $\notin$  xs  $\Longrightarrow$  fresh2 xs x y = x
 $\langle$ proof $\rangle$ 

```

```

declare fresh2.simps[simp del]

```

```

instantiation nat :: fresh
begin

```

fresh xs x y returns an element that is fresh for *xs* and closest to *x*, favoring smaller elements:

```

definition fresh-nat :: nat set  $\Rightarrow$  nat  $\Rightarrow$  nat where

```

fresh-nat $xs\ x \equiv fresh2\ xs\ x\ x$

instance $\langle proof \rangle$

end

Code generation

lemma *fresh2-list*[code]:
 fresh2 (set *xs*) $x\ y =$
 (if $x \notin set\ xs$ then x else
 if $y \notin set\ xs$ then y else
 fresh2 (set *xs*) ($x-1$) ($y+1$))
 $\langle proof \rangle$

Some tests:

value [*fresh* {}] (1::nat),
 fresh {3,5,2,4} 3]

end

3 Fresh identifier generation for strings

theory *Fresh-String*
 imports *Fresh*
begin

3.1 A partial order on strings

The first criterion is the length, and the second the encoding of last character.

definition *ordst* :: *string* \Rightarrow *string* \Rightarrow *bool* **where**
ordst $X\ Y \equiv$
 ($length\ X \leq length\ Y \wedge X \neq [] \wedge Y \neq [] \wedge of-char\ (last\ X) < (of-char\ (last\ Y))$
 :: *nat*)
 $\vee (length\ X < length\ Y)$

definition *ordstNS* :: *string* \Rightarrow *string* \Rightarrow *bool* **where**
ordstNS $X\ Y \equiv X = Y \vee ordst\ X\ Y$

lemma *ordst-antirefl*: $\neg ordst\ X\ X$
 $\langle proof \rangle$

lemma *ordst-trans*:
assumes *As1*: *ordst* $X\ Y$ **and** *As2*: *ordst* $Y\ Z$
shows *ordst* $X\ Z$
 $\langle proof \rangle$

lemma *ordstNS-refl*: *ordstNS* $X\ X$

<proof>

lemma *ordstNS-trans:*

ordstNS X Y \implies ordstNS Y Z \implies ordstNS X Z

<proof>

lemma *ordst-ordstNS-trans:*

ordst X Y \implies ordstNS Y Z \implies ordst X Z

<proof>

lemma *ordstNS-ordst-trans:*

ordstNS X Y \implies ordst Y Z \implies ordst X Z

<proof>

3.2 Incrementing a string

If the last character is \geq 'a' and $<$ 'z', then *upChar* increments this last character; otherwise *upChar* appends an 'a'.

fun *upChar* :: *string* \Rightarrow *string* **where**

upChar Y =

*(if (Y \neq [] \wedge of-char(last Y) \geq (97 :: nat) \wedge
of-char(last Y) $<$ (122 :: nat))*

then (butlast Y) @

[char-of(of-char(last Y) + (1 :: nat))]

else Y @ "a"

)

lemma *upChar-ordst: ordst Y (upChar Y)*

<proof>

3.3 The fresh-identifier operator

fresh Xs Y changes *Y* as little as possible so that it becomes disjoint from all strings in *Xs*.

function *fresh-string* :: *string set* \Rightarrow *string* \Rightarrow *string*

where

Up: Y \in Xs \implies finite Xs \implies fresh-string Xs Y = fresh-string (Xs - {Y}) (upChar Y)

|

Fresh: Y \notin Xs \vee infinite Xs \implies fresh-string Xs Y = Y

<proof>

termination

<proof>

lemma *fresh-string-ordstNS: ordstNS Y (fresh-string Xs Y)*

<proof>

lemma *fresh-string-set: finite Xs \implies fresh-string Xs Y \notin Xs*

<proof>

Code generation:

lemma *fresh-string-if*:

fresh-string $Xs\ Y =$ (
 if $Y \in Xs \wedge \text{finite } Xs$ then *fresh-string* ($Xs - \{Y\}$) (*upChar* Y)
 else Y)
<proof>

lemmas *fresh-string-list*[*code*] = *fresh-string-if*[**where** $Xs = \text{set } Xs$ **for** Xs , *simplified*]

Some tests:

value [*fresh-string* {} "Abc",
 fresh-string {"X", "Abc"} "Abd",
 fresh-string {"X", "Y"} "Y",
 fresh-string {"X", "Yaa", "Ya", "Yaa"} "Ya",
 fresh-string {"X", "Yaa", "Yz", "Yza"} "Yz",
 fresh-string {"X", "Y", "Yab", "Y"} "Y"]

Here we do locale interpretation rather than class instantiation, since *string* is a type synonym for *char list*.

interpretation *fresh-string*: *fresh* **where** *fresh* = *fresh-string*
<proof>

3.4 Lifting to string literals

abbreviation *is-ascii* $str \equiv (\forall c \in \text{set } str. \neg \text{digit7 } c)$

lemma *map-ascii-of-idem*:

is-ascii $str \implies \text{map } \text{String.ascii-of } str = str$
<proof>

lemma *is-ascii-butlast*:

is-ascii $str \implies \text{is-ascii } (\text{butlast } str)$
<proof>

lemma *ascii-char-of*:

fixes $c :: \text{nat}$
assumes $c < 128$
shows $\neg \text{digit7 } (\text{char-of } c)$
<proof>

lemmas *ascii-of-char-of-idem* = *ascii-char-of*[*THEN* *String.ascii-of-idem*]

lemma *is-ascii-upChar*:

is-ascii $str \implies \text{is-ascii } (\text{upChar } str)$
<proof>

```

lemma is-ascii-fresh-string:
  is-ascii Y  $\implies$  is-ascii (fresh-string Xs Y)
  <proof>

```

For string literals we can properly instantiate the class.

```

instantiation String.literal :: fresh
begin

```

```

context
  includes literal.lifting
begin

```

```

lift-definition fresh-literal :: String.literal set  $\Rightarrow$  String.literal  $\Rightarrow$  String.literal
  is fresh-string
  <proof>

```

```

instance <proof>

```

```

end

```

```

end

```

Code generation:

```

context
  includes literal.lifting
begin

```

```

lift-definition upChar-literal :: String.literal  $\Rightarrow$  String.literal is upChar
  <proof>

```

```

lemma upChar-literal-upChar[code]:
  upChar-literal s = String.implode (upChar (String.explode s))
  <proof>

```

```

lemma fresh-literal-if:
  fresh xs y = (if y  $\in$  xs  $\wedge$  finite xs then fresh (xs - {y}) (upChar-literal y) else y)
  <proof>

```

```

lemmas fresh-literal-list[code] = fresh-literal-if[where xs = set xs for xs, simplified]

```

```

end

```

Some tests:

```

value [fresh {} (STR "Abc"),
  fresh {STR "X", STR "Abc"} (STR "Abd"),
  fresh {STR "X", STR "Y"} (STR "Y"),
  fresh {STR "X", STR "Yaa", STR "Ya", STR "Yaa"} (STR "Ya"),
  fresh {STR "X", STR "Yaa", STR "Yz", STR "Yza"} (STR "Yz"),

```

fresh {*STR* "X", *STR* "Y", *STR* "Yab", *STR* "Y"} (*STR* "Y")]

end

4 Fresh identifier generation for infinite types

theory *Fresh-Infinite*

imports *Fresh*

begin

This is a default fresh operator for infinite types for which more specific (smarter) alternatives are not (yet) available.

definition (**in** *infinite*) *fresh* :: 'a set \Rightarrow 'a \Rightarrow 'a **where**
fresh *xs* *x* \equiv if *x* \notin *xs* \vee *infinite* *xs* then *x* else (SOME *y*. *y* \notin *xs*)

sublocale *infinite* < *fresh* **where** *fresh* = *fresh*
{*proof*}

end