Fresh identifiers

Andrei Popescu   Thomas Bauereiss

March 17, 2025

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 *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*
⟨*proof*⟩

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 $(x-1)$ $(y+1)$)
⟨*proof*⟩
termination
⟨*proof*⟩

lemma *fresh2-notIn*: $finite\ xs \Longrightarrow fresh2\ xs\ x\ y \notin xs$
⟨*proof*⟩

lemma *fresh2-eq*: $x \notin xs \Longrightarrow fresh2\ xs\ x\ y = x$
⟨*proof*⟩

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 ≡ *fresh2 xs x x*

instance ⟨*proof*⟩

end

Code generation

lemma *fresh2-list*[*code*]:
  fresh2 (*set xs*) *x y* =
    (*if x* ∉ *set xs then x else*
     if y ∉ *set xs then y else*
     fresh2 (*set xs*) (*x*−*1*) (*y*+*1*))
  ⟨*proof*⟩

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* ⇒ *string* ⇒ *bool* **where**
ordst X Y ≡
 (*length X* ≤ *length Y* ∧ *X* ≠ [] ∧ *Y* ≠ [] ∧ *of-char* (*last X*) < (*of-char*(*last Y*) :: *nat*))
 ∨ (*length X* < *length Y*)

definition *ordstNS* :: *string* ⇒ *string* ⇒ *bool* **where**
ordstNS X Y ≡ *X* = *Y* ∨ *ordst X Y*

lemma *ordst-antirefl*: ¬ *ordst X X*
⟨*proof*⟩

lemma *ordst-trans*:
assumes *As1*: *ordst X Y* **and** *As2*: *ordst Y Z*
shows *ordst X Z*
⟨*proof*⟩

lemma *ordstNS-refl*: *ordstNS X X*

⟨*proof*⟩

lemma *ordstNS-trans*:
ordstNS X Y $\Longrightarrow$ *ordstNS Y Z* $\Longrightarrow$ *ordstNS X Z*
⟨*proof*⟩

lemma *ordst-ordstNS-trans*:
ordst X Y $\Longrightarrow$ *ordstNS Y Z* $\Longrightarrow$ *ordst X Z*
⟨*proof*⟩

lemma *ordstNS-ordst-trans*:
ordstNS X Y $\Longrightarrow$ *ordst Y Z* $\Longrightarrow$ *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* ⇒ *string* **where**
upChar Y =
 (*if* (*Y* ≠ [] ∧ *of-char*(*last Y*) ≥ (*97* :: *nat*) ∧
 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* ⇒ *string* ⇒ *string*
where
Up: *Y* ∈ *Xs* $\Longrightarrow$ *finite Xs* $\Longrightarrow$ *fresh-string Xs Y* = *fresh-string* (*Xs* − {*Y*}) (*upChar Y*)
|
Fresh: *Y* ∉ *Xs* ∨ *infinite Xs* $\Longrightarrow$ *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* $\Longrightarrow$ *fresh-string Xs Y* ∉ *Xs*

⟨*proof*⟩

Code generation:

lemma *fresh-string-if*:
  fresh-string Xs Y = (
    if Y ∈ *Xs* ∧ *finite Xs then fresh-string* (*Xs* − {*Y*}) (*upChar Y*)
    else Y)
  ⟨*proof*⟩

lemmas *fresh-string-list*[*code*] = *fresh-string-if*[**where** *Xs* = *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* ≡ (∀ *c* ∈ *set str*. ¬*digit7 c*)

lemma *map-ascii-of-idem*:
  is-ascii str ⟹ *map String.ascii-of str* = *str*
  ⟨*proof*⟩

lemma *is-ascii-butlast*:
  is-ascii str ⟹ *is-ascii* (*butlast str*)
  ⟨*proof*⟩

lemma *ascii-char-of*:
  fixes *c* :: *nat*
  assumes *c* < *128*
  shows ¬*digit7* (*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 ⟹ *is-ascii* (*upChar str*)
  ⟨*proof*⟩

lemma *is-ascii-fresh-string*:
is-ascii Y $\Longrightarrow$ *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* ⇒ *String.literal* ⇒ *String.literal*
is *fresh-string*
⟨*proof*⟩

instance ⟨*proof*⟩

end

end

Code generation:

context
includes *literal.lifting*
begin

lift-definition *upChar-literal* :: *String.literal* ⇒ *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* ∈ *xs* ∧ *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