Lazy List Examples

Jasmin Christian Blanchette Andrei Popescu
Dmitriy Traytel

January 30, 2015

Contents

1 lfilter

Here, we define a monomorphic filter function on lazy lists over natural numbers. Unlike, the theory presented in the paper in general the rudimentary package currently does not support polymorphism.

inductive *llist-in* **where**
  llist-in (*LCons x xs*) *x*
| *llist-in xs y* $\Longrightarrow$ *llist-in* (*LCons x xs*) *y*

abbreviation *lset xs* $\equiv$ $\{x.$ *llist-in xs x*$\}$

function *lfilter*$_{rec}$ :: (*nat* $\Rightarrow$ *bool*) $\Rightarrow$ *llist* $\Rightarrow$ (*llist* + ((*nat* $\Rightarrow$ *bool*) $\times$ *llist*)) $\Sigma\Sigma 0$ *F* $\Sigma\Sigma 0$ **where**
  lfilter$_{rec}$ *P xs* =
    (*if* $\forall x \in$ *lset xs*. $\neg$ *P x then GUARD0* (*Inl* ())
    else if P (*head xs*) *then GUARD0* (*Inr* (*head xs*, *CONT0* (*P*, *tail xs*)))
    else lfilter$_{rec}$ *P* (*tail xs*))
by *pat-completeness auto*
termination
proof (*relation measure* (λ(*P*, *xs*). *LEAST n*. *P* (*head* ((*tail* ^^ *n*) *xs*))), *rule wf-measure*, *clarsimp*)
  fix *P xs x*
  assume *llist-in xs x P x* $\neg$ *P* (*head xs*)
  from *this*(*1*,*2*) **obtain** *a* **where** *P* (*head* ((*tail* ^^ *a*) *xs*))
    by (*atomize-elim*, *induct xs x rule*: *llist-in.induct*) (*auto simp*: *J.dtor-ctor funpow-Suc-right*
    simp del: *funpow.simps*(*2*) *intro*: *exI*[*of* - *0*] *exI*[*of* - *Suc i* **for** *i*])
  moreover
  with ‹$\neg$ *P* (*head xs*)›
   have (*LEAST n*. *P* (*head* ((*tail* ^^ *n*) *xs*))) = *Suc* (*LEAST n*. *P* (*head* ((*tail* ^^ *Suc n*) *xs*)))

by (*intro Least-Suc*) *auto*
then show (*LEAST n*. *P* (*head* ((*tail* ^^ *n*) (*tail xs*)))) < (*LEAST n*. *P* (*head* ((*tail* ^^ *n*) *xs*)))
by (*simp add*: *funpow-swap1*[*of tail*])
qed

definition *lfilter* :: (*nat* ⇒ *bool*) ⇒ *llist* ⇒ *llist* **where**
lfilter P xs = *corecUU0* (*split* $lfilter_{rec}$) (*P*, *xs*)

lemma *lfilter-code*:
lfilter P xs =
(*if* ∀ *x* ∈ *lset xs*. ¬ *P x then LNil*
else if P (*head xs*) *then LCons* (*head xs*) (*lfilter P* (*tail xs*))
else lfilter P (*tail xs*))
unfolding *lfilter-def*
by (*subst corecUU0*, *unfold split*, *subst* $lfilter_{rec}$*.simps*)
(*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def*
*eval0-*𝔬𝔭*0 eval0-leaf0*′ *o-eq-dest*[*OF Abs-*Σ*0-natural*] *corecUU0*
imp-conjL[*symmetric*] *imp-conjR conj-commute del*: *not-all*)

Here is an alternative definition using partial_function.

partial-function (*tailrec*) $lfilter2_{rec}$ ::
(*nat* ⇒ *bool*) ⇒ *llist* ⇒ (*llist* + ((*nat* ⇒ *bool*) × *llist*)) ΣΣ*0 F* ΣΣ*0* **where**
$lfilter2_{rec}$ *P xs* =
(*if* ∀ *x* ∈ *lset xs*. ¬ *P x then GUARD0* (*Inl* ())
else if P (*head xs*) *then GUARD0* (*Inr* (*head xs*, *CONT0* (*P*, *tail xs*)))
else $lfilter2_{rec}$ *P* (*tail xs*))

definition *lfilter2* :: (*nat* ⇒ *bool*) ⇒ *llist* ⇒ *llist* **where**
lfilter2 P xs = *corecUU0* (*split* $lfilter2_{rec}$) (*P*, *xs*)

lemma *lfilter2-code*:
lfilter2 P xs =
(*if* ∀ *x* ∈ *lset xs*. ¬ *P x then LNil*
else if P (*head xs*) *then LCons* (*head xs*) (*lfilter2 P* (*tail xs*))
else lfilter2 P (*tail xs*))
unfolding *lfilter2-def*
by (*subst corecUU0*, *unfold split*, *subst* $lfilter2_{rec}$*.simps*)
(*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def*
*eval0-*𝔬𝔭*0 eval0-leaf0*′ *o-eq-dest*[*OF Abs-*Σ*0-natural*] *corecUU0*
imp-conjL[*symmetric*] *imp-conjR conj-commute del*: *not-all*)

end