

# Formalization of All Examples from [1]

Jasmin Christian Blanchette

Andrei Popescu

Dmitriy Traytel

January 30, 2015

## Contents

<b>1</b>	<b>Stream Examples</b>	<b>2</b>
1.1	one . . . . .	2
1.2	plus . . . . .	2
1.3	nat . . . . .	2
1.4	id . . . . .	2
1.5	fib . . . . .	3
1.6	sum . . . . .	3
1.7	alternate . . . . .	3
1.8	interleave . . . . .	4
1.9	merge . . . . .	4
1.10	dup . . . . .	5
1.11	inv . . . . .	5
1.12	thue . . . . .	5
1.13	times . . . . .	6
1.14	ham . . . . .	6
1.15	even . . . . .	6
1.16	odd . . . . .	7
1.17	drop . . . . .	7
1.18	diff . . . . .	7
1.19	Some Proofs . . . . .	7
<b>2</b>	<b>Binary Tree Examples</b>	<b>8</b>
2.1	one . . . . .	8
2.2	plus . . . . .	8
2.3	divide . . . . .	9
2.4	bird . . . . .	9
2.5	mirror . . . . .	10
2.6	Some Proofs . . . . .	10

## References

- [1] R. Hinze and D. W. H. James. Proving the unique fixed-point principle correct. Technical Report CS-RR-11-03, Department of Computer Science, University of Oxford, 2011.

# 1 Stream Examples

## 1.1 one

**definition** *one* :: *stream* **where**  
  *one* = *dtor\_corec\_J* ( $\lambda\_. (1, \text{Inr } ())$ ) ()

**lemma** *head\_one*[*simp*]: *head one* = 1  
  **unfolding** *one\_def J.dtor\_corec*  
  **by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *tail\_one*[*simp*]: *tail one* = *one*  
  **unfolding** *one\_def J.dtor\_corec*  
  **by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *one\_code*[*code*]: *one* = *SCons* 1 *one*  
  **by** (*metis J.ctor\_dtor prod.collapse head\_one tail\_one*)

## 1.2 plus

**definition** *plus* :: *stream*  $\Rightarrow$  *stream*  $\Rightarrow$  *stream* **where**  
  *plus s t* = *dtor\_corec\_J* ( $\lambda(s, t). (\text{head } s + \text{head } t, \text{Inr } (\text{tail } s, \text{tail } t))$ ) (s, t)

**lemma** *head\_plus*[*simp*]: *head (plus s t)* = *head s* + *head t*  
  **unfolding** *plus\_def J.dtor\_corec map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def* **by** *simp*

**lemma** *tail\_plus*[*simp*]: *tail (plus s t)* = *plus (tail s) (tail t)*  
  **unfolding** *plus\_def J.dtor\_corec map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def* **by** *simp*

**lemma** *plus\_code*[*code*]: *plus (SCons m s) (SCons n t)* = *SCons (m + n) (plus s t)*  
  **by** (*smt2 J.ctor\_dtor J.dtor\_ctor fst\_conv snd\_conv prod.collapse head\_plus tail\_plus*)

**lemma** *plus\_uniform*: *plus xs ys* = *alg01 (xs, ys)*  
  **unfolding** *plus\_def*  
  **apply** (*rule fun\_cong[OF sym[OF J.dtor\_corec\_unique]]*)  
  **unfolding** *alg01*  
  **by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def fun\_eq\_iff convol\_def 01\_def alg01\_def*)

## 1.3 nat

**definition** *nat* :: *stream* **where**  
  *nat* = *corecUU1* ( $\lambda\_. \text{GUARD1 } (0, \text{PLS1 } (\text{CONT1 } ()), \text{END1 one}))$ ) ()

**lemma** *head\_nat*[*simp*]: *head nat* = 0  
  **unfolding** *nat\_def corecUU1*  
  **by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def J.dtor\_ctor eval1\_leaf1'*)

**lemma** *tail\_nat*[*simp*]: *tail nat* = *plus nat one*  
  **apply** (*subst nat\_def*) **unfolding** *corecUU1*  
  **by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def J.dtor\_ctor eval1\_leaf1'*  
    *eval1\_op1 alg01\_Inr o\_eq\_dest[OF Abs\_01\_natural] plus\_uniform nat\_def*)

**lemma** *nat\_code*[*code*]: *nat* = *SCons* 0 (*plus nat one*)  
  **by** (*metis J.ctor\_dtor prod.collapse head\_nat tail\_nat*)

## 1.4 id

**definition** *id* :: *stream*  $\Rightarrow$  *stream* **where**  
  *id s* = *dtor\_corec\_J* ( $\lambda s. (\text{head } s, \text{Inl } (\text{tail } s))$ ) s

**lemma** *head\_id*[simp]: *head* (*id* *s*) = *head* *s*  
**unfolding** *id\_def* *J.dtor\_corec*  
**by** (*simp* *add*: *map\_pre\_J\_def* *BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *tail\_id*[simp]: *tail* (*id* *s*) = *tail* *s*  
**unfolding** *id\_def* *J.dtor\_corec*  
**by** (*simp* *add*: *map\_pre\_J\_def* *BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *id\_code*[code]: *id* (*SCons* *m* *s*) = *SCons* *m* *s*  
**by** (*metis* *J.ctor\_dtor\_prod.collapse* *head\_id* *tail\_id*)

## 1.5 fib

**definition** *fib* :: *stream* **where**  
*fib* = *corecUU1* ( $\lambda x s. \text{GUARD1 } (0, \text{PLS1 } (\text{SCONS1 } (1, \text{CONT1 } xs), \text{CONT1 } xs))) ()$ )

**lemma** *head\_fib*[simp]: *head* *fib* = 0  
**unfolding** *fib\_def* *corecUU1*  
**by** (*simp* *add*: *map\_pre\_J\_def* *BNF\_Comp.id\_bnf\_comp\_def* *J.dtor\_ctor* *eval1\_leaf1'*)

**lemma** *tail\_fib*[simp]: *tail* *fib* = *plus* (*SCons* 1 *fib*) *fib*  
**apply** (*subst* *fib\_def*) **unfolding** *corecUU1*  
**by** (*simp* *add*: *map\_pre\_J\_def* *BNF\_Comp.id\_bnf\_comp\_def* *J.dtor\_ctor* *eval1\_leaf1'*  
*eval1\_op1* *alg1\_inr* *o\_eq\_dest[OF Abs\_Σ1\_natural]* *o\_eq\_dest[OF gg1\_natural]*  
*o\_eq\_dest[OF eval1\_gg1]* *plus\_uniform* *fib\_def*)

**lemma** *fib\_code*[code]: *fib* = *SCons* 0 (*plus* (*SCons* 1 *fib*) *fib*)  
**by** (*metis* *J.ctor\_dtor\_prod.collapse* *head\_fib* *tail\_fib*)

## 1.6 sum

**definition** *sum* :: *stream*  $\Rightarrow$  *stream* **where**  
*sum* *s* = *corecUU1* ( $\lambda s. \text{GUARD1 } (0, \text{PLS1 } (\text{END1 } s, \text{CONT1 } s))) s$ )

**lemma** *head\_sum*[simp]: *head* (*sum* *s*) = 0  
**unfolding** *sum\_def* *corecUU1*  
**by** (*simp* *add*: *map\_pre\_J\_def* *BNF\_Comp.id\_bnf\_comp\_def* *J.dtor\_ctor* *eval1\_leaf1'*)

**lemma** *tail\_sum*[simp]: *tail* (*sum* *s*) = *plus* *s* (*sum* *s*)  
**apply** (*subst* *sum\_def*) **unfolding** *corecUU1*  
**by** (*simp* *add*: *map\_pre\_J\_def* *BNF\_Comp.id\_bnf\_comp\_def* *J.dtor\_ctor* *eval1\_leaf1'*  
*eval1\_op1* *alg1\_inr* *o\_eq\_dest[OF Abs\_Σ1\_natural]* *o\_eq\_dest[OF gg1\_natural]*  
*o\_eq\_dest[OF eval1\_gg1]* *plus\_uniform* *sum\_def*)

**lemma** *sum\_code*[code]: *sum* *s* = *SCons* 0 (*plus* *s* (*sum* *s*))  
**by** (*metis* *J.ctor\_dtor\_prod.collapse* *head\_sum* *tail\_sum*)

## 1.7 alternate

**definition** *alternate* :: *stream*  $\Rightarrow$  *stream*  $\Rightarrow$  *stream* **where**  
*alternate* *s* *t* = *dtor\_corec\_J* ( $\lambda (s, t). (\text{head } s, \text{Inr } (\text{tail } t, \text{tail } s))) (s, t)$ )

**lemma** *head\_alternate*[simp]: *head* (*alternate* *s* *t*) = *head* *s*  
**unfolding** *alternate\_def* *J.dtor\_corec*  
**by** (*simp* *add*: *map\_pre\_J\_def* *BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *tail\_alternate*[simp]: *tail* (*alternate* *s* *t*) = *alternate* (*tail* *t*) (*tail* *s*)  
**apply** (*subst* *alternate\_def*) **unfolding** *J.dtor\_corec*  
**by** (*simp* *add*: *map\_pre\_J\_def* *BNF\_Comp.id\_bnf\_comp\_def* *alternate\_def*)

**lemma** *alternate\_code*[code]: *alternate* (*SCons* *m s*) (*SCons* *n t*) = *SCons* *m* (*alternate* *t s*)  
**by** (*metis* *J.ctor\_dtor J.dtor\_ctor fst\_conv snd\_conv prod.collapse head\_alternate tail\_alternate*)

## 1.8 interleave

**definition** *interleave* :: *stream*  $\Rightarrow$  *stream*  $\Rightarrow$  *stream* **where**  
*interleave* *s t* = *dtor\_corec*-*J* ( $\lambda(s, t). (head\ s, Inr\ (t, tail\ s))$ ) (*s, t*)

**lemma** *head\_interleave*[simp]: *head* (*interleave* *s t*) = *head* *s*  
**unfolding** *interleave\_def J.dtor\_corec*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *tail\_interleave*[simp]: *tail* (*interleave* *s t*) = *interleave* *t* (*tail* *s*)  
**apply** (*subst interleave\_def*) **unfolding** *J.dtor\_corec*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def interleave\_def*)

**lemma** *interleave\_code*[code]: *interleave* (*SCons* *m s*) (*SCons* *n t*) = *SCons* *m* (*interleave* (*SCons* *n t*) *s*)  
**by** (*metis* *J.ctor\_dtor J.dtor\_ctor fst\_conv snd\_conv prod.collapse head\_interleave tail\_interleave*)

**lemma** *interleave\_uniform*: *interleave* *s t* = *alg<sub>Q6</sub>* (*s, t*)  
**unfolding** *interleave\_def*  
**apply** (*rule fun\_cong*[*OF sym*[*OF J.dtor\_corec\_unique*]])  
**unfolding** *alg<sub>Q6</sub> o\_def*[*symmetric*] *o\_assoc*  
**apply** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def fun\_eq\_iff J.dtor\_ctor*  
*Q6\_def Let\_def convol\_def eval6\_op6 eval6\_leaf6'*  
*o\_eq\_dest*[*OF Abs. $\Sigma$ <sub>6</sub>-natural*] *alg $\Lambda$ <sub>6</sub>Inr alg<sub>Q6</sub>-def*)  
**done**

## 1.9 merge

**definition** *merge* **where**  
*merge* *s t* = *dtor\_corec*-*J* ( $\lambda(s, t). (if\ head\ s \leq head\ t\ then\ (head\ s, Inr\ (tail\ s, t))\ else\ (head\ t, Inr\ (s, tail\ t)))$ ) (*s, t*)

**lemma** *head\_merge*[simp]: *head* (*merge* *s t*) = (*if* *head* *s*  $\leq$  *head* *t* *then* *head* *s* *else* *head* *t*)  
**unfolding** *merge\_def J.dtor\_corec*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *tail\_merge*[simp]: *tail* (*merge* *s t*) = (*if* *head* *s*  $\leq$  *head* *t* *then* *merge* (*tail* *s*) *t* *else* *merge* *s* (*tail* *t*)  
**apply** (*subst merge\_def*) **unfolding** *J.dtor\_corec*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def merge\_def*)

**lemma** *merge\_code*[code]:  
*merge* (*SCons* *m s*) (*SCons* *n t*) =  
(*if* *m*  $\leq$  *n* *then* *SCons* *m* (*merge* *s* (*SCons* *n t*)) *else* *SCons* *n* (*merge* (*SCons* *m s*) *t*)  
**by** (*smt2* *J.ctor\_dtor J.dtor\_ctor fst\_conv snd\_conv prod.collapse head\_merge tail\_merge*)

**lemma** *merge\_uniform*: *merge* *s t* = *alg<sub>Q8</sub>* (*s, t*)  
**unfolding** *merge\_def*  
**apply** (*rule fun\_cong*[*OF sym*[*OF J.dtor\_corec\_unique*]])  
**unfolding** *alg<sub>Q8</sub> o\_def*[*symmetric*] *o\_assoc*  
**apply** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def fun\_eq\_iff J.dtor\_ctor*  
*Q8\_def Let\_def convol\_def eval8\_op8 eval8\_leaf8'*  
*o\_eq\_dest*[*OF Abs. $\Sigma$ <sub>8</sub>-natural*] *alg $\Lambda$ <sub>8</sub>Inr alg<sub>Q8</sub>-def*)  
**done**

## 1.10 dup

**definition** *dup* **where**

*dup s = dtor\_corec\_J (λs. (head s, Inl s)) s*

**lemma** *head\_dup[simp]*: *head (dup s) = head s*

**unfolding** *dup\_def J.dtor\_corec*

**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *tail\_dup[simp]*: *tail (dup s) = s*

**unfolding** *dup\_def J.dtor\_corec*

**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *dup\_code[code]*: *dup (SCons m s) = SCons m (SCons m s)*

**by** (*metis J.ctor\_dtor J.dtor\_ctor fst\_conv prod.collapse head\_dup tail\_dup*)

## 1.11 inv

**definition** *inv* :: *stream*  $\Rightarrow$  *stream* **where**

*inv s = dtor\_corec\_J (λs. (1 - head s, Inr (tail s))) s*

**lemma** *head\_inv[simp]*: *head (inv s) = 1 - head s*

**unfolding** *inv\_def J.dtor\_corec*

**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *tail\_inv[simp]*: *tail (inv s) = inv (tail s)*

**apply** (*subst inv\_def*) **unfolding** *J.dtor\_corec*

**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def inv\_def*)

**lemma** *inv\_code[code]*: *inv (SCons m s) = SCons (1 - m) (inv s)*

**by** (*metis J.ctor\_dtor J.dtor\_ctor fst\_conv snd\_conv prod.collapse head\_inv tail\_inv*)

**lemma** *inv\_uniform*: *inv s = alg<sub>Q</sub>7 (K7.I s)*

**unfolding** *inv\_def*

**apply** (*rule fun.cong[OF sym[OF J.dtor\_corec\_unique]]*)

**unfolding** *alg<sub>Q</sub>7 o\_def[symmetric] o\_assoc*

**apply** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def fun\_eq\_iff J.dtor\_ctor*  
*q7\_def Let\_def convol\_def eval7\_op7 eval7\_leaf7'*

*o\_eq\_dest[OF Abs.Σ7\_natural] algΛ7\_Inr alg<sub>Q</sub>7\_def*)

**done**

## 1.12 thue

**definition** *thue'* :: *stream* **where**

*thue' = corecUU7 (λ\_. GUARD7 (1, INTERLEAVE7 (CONT7 (), INV7 (I (CONT7 ()))))) ()*

**lemma** *head\_thue'[simp]*: *head thue' = 1*

**unfolding** *thue'\_def corecUU7*

**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def J.dtor\_ctor eval7\_leaf7'*)

**lemma** *tail\_thue'[simp]*: *tail thue' = interleave thue' (inv thue')*

**apply** (*subst thue'\_def*) **unfolding** *corecUU7*

**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def J.dtor\_ctor eval7\_leaf7' eval7\_op7*  
*algΛ7\_Inl algΛ7\_Inr algΛ6\_Inr o\_eq\_dest[OF Abs.Σ7\_natural] o\_eq\_dest[OF Abs.Σ6\_natural]*  
*interleave\_uniform inv\_uniform thue'\_def*)

**lemma** *thue'\_code[code]*: *thue' = SCons 1 (interleave thue' (inv thue'))*

**by** (*metis J.ctor\_dtor prod.collapse head\_thue' tail\_thue'*)

**definition** *thue* :: *stream* **where**

*thue* = *SCons* 0 *thue*'

### 1.13 times

**definition** *times* :: *nat*  $\Rightarrow$  *stream*  $\Rightarrow$  *stream* **where**  
*times* *n* *s* = *dtor\_corec\_J* ( $\lambda s. (n * \text{head } s, \text{Inr } (\text{tail } s)))$  *s*

**lemma** *head\_times*[*simp*]: *head* (*times* *n* *s*) = *n* \* *head* *s*  
**unfolding** *times\_def* *J.dtor\_corec*  
**by** (*simp* *add*: *map\_pre\_J\_def* *BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *tail\_times*[*simp*]: *tail* (*times* *n* *s*) = *times* *n* (*tail* *s*)  
**apply** (*subst* *times\_def*) **unfolding** *J.dtor\_corec*  
**by** (*simp* *add*: *map\_pre\_J\_def* *BNF\_Comp.id\_bnf\_comp\_def* *times\_def*)

**lemma** *times\_code*[*code*]: *times* *n* (*SCons* *m* *s*) = *SCons* (*n* \* *m*) (*times* *n* *s*)  
**by** (*metis* *J.ctor\_dtor* *J.dtor\_ctor* *fst\_conv* *snd\_conv* *prod.collapse* *head\_times* *tail\_times*)

**lemma** *times\_uniform*: *times* *m* *s* = *alg9* (*m*, *s*)  
**unfolding** *times\_def*  
**apply** (*rule* *fun\_cong*[*OF* *sym*[*OF* *J.dtor\_corec\_unique*]])  
**unfolding** *alg9* *o\_def*[*symmetric*] *o\_assoc*  
**apply** (*simp* *add*: *map\_pre\_J\_def* *BNF\_Comp.id\_bnf\_comp\_def* *fun\_eq\_iff* *J.dtor\_ctor*  
*9* *9* *def* *Let\_def* *convol\_def* *eval9\_op9* *eval9\_leaf9'*  
*o\_eq\_dest*[*OF* *Abs\_9\_natural*] *alg9* *Inr* *alg9* *9* *def*)  
**done**

### 1.14 ham

**definition** *ham* :: *stream* **where**  
*ham* = *corecUU9* ( $\lambda. \text{GUARD9 } (1, \text{MERGE9 } (\text{TIMES9 } (2, \text{CONT9 } ()), \text{TIMES9 } (3, \text{CONT9 } ())))$ )

**lemma** *head\_ham*[*simp*]: *head* *ham* = 1  
**unfolding** *ham\_def* *corecUU9*  
**by** (*simp* *add*: *map\_pre\_J\_def* *BNF\_Comp.id\_bnf\_comp\_def* *J.dtor\_ctor* *eval9\_leaf9'*)

**lemma** *tail\_ham*[*simp*]: *tail* *ham* = *merge* (*times* 2 *ham*) (*times* 3 *ham*)  
**apply** (*subst* *ham\_def*) **unfolding** *corecUU9*  
**by** (*simp* *add*: *map\_pre\_J\_def* *BNF\_Comp.id\_bnf\_comp\_def* *J.dtor\_ctor* *eval9\_leaf9'* *eval9\_op9*  
*alg9* *Inl* *alg9* *Inr* *alg9* *Inr* *o\_eq\_dest*[*OF* *Abs\_9\_natural*] *o\_eq\_dest*[*OF* *Abs\_8\_natural*]  
*merge\_uniform* *times\_uniform* *ham\_def*)

**lemma** *ham\_code*[*code*]: *ham* = *SCons* 1 (*merge* (*times* 2 *ham*) (*times* 3 *ham*))  
**by** (*metis* *J.ctor\_dtor* *prod.collapse* *head\_ham* *tail\_ham*)

### 1.15 even

**definition** *even* :: *stream*  $\Rightarrow$  *stream* **where**  
*even* *s* = *dtor\_corec\_J* ( $\lambda s. (\text{head } s, \text{Inr } (\text{tail } (\text{tail } s)))$ ) *s*

**lemma** *head\_even*[*simp*]: *head* (*even* *s*) = *head* *s*  
**unfolding** *even\_def* *J.dtor\_corec*  
**by** (*simp* *add*: *map\_pre\_J\_def* *BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *tail\_even*[*simp*]: *tail* (*even* *s*) = *even* (*tail* (*tail* *s*))  
**apply** (*subst* *even\_def*) **unfolding** *J.dtor\_corec*  
**by** (*simp* *add*: *map\_pre\_J\_def* *BNF\_Comp.id\_bnf\_comp\_def* *even\_def*)

**lemma** *even\_code*[*code*]: *even* (*SCons* *n* (*SCons* *m* *s*)) = *SCons* *n* (*even* *s*)

**by** (*metis* *J.ctor\_dtor J.dtor\_ctor fst\_conv snd\_conv prod.collapse head\_even tail\_even*)

## 1.16 odd

**definition** *odd* :: *stream*  $\Rightarrow$  *stream* **where**  
*odd* *s* = *dtor\_corec*-*J* ( $\lambda s. (head (tail s), Inr (tail (tail s)))$ ) *s*

**lemma** *head\_odd[simp]*: *head* (*odd* *s*) = *head* (*tail* *s*)  
**unfolding** *odd\_def J.dtor\_corec*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *tail\_odd[simp]*: *tail* (*odd* *s*) = *odd* (*tail* (*tail* *s*))  
**apply** (*subst odd\_def*) **unfolding** *J.dtor\_corec*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def odd\_def*)

**lemma** *odd\_code[code]*: *odd* (*SCons* *n* (*SCons* *m* *s*)) = *SCons* *m* (*odd* *s*)  
**by** (*metis J.ctor\_dtor J.dtor\_ctor fst\_conv snd\_conv prod.collapse head\_odd tail\_odd*)

## 1.17 drop

**definition** *drop* :: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *stream*  $\Rightarrow$  *stream* **where**  
*drop* *i* *l* *s* = *dtor\_corec*-*J* ( $\lambda(i, l, s). \text{case } i \text{ of}$   
 $\text{Suc } i \Rightarrow (head\ s, Inr\ (i, l, tail\ s))$   
 $| 0 \Rightarrow (head\ (tail\ s), Inr\ (l - 2, l, tail\ (tail\ s)))$ ) (*i*, *l*, *s*)

**lemma** *head\_drop[simp]*: *head* (*drop* *i* *l* *s*) = (*case* *i* *of* *Suc* \_  $\Rightarrow$  *head* *s* | 0  $\Rightarrow$  *head* (*tail* *s*))  
**unfolding** *drop\_def J.dtor\_corec*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def split: nat.splits*)

**lemma** *tail\_drop[simp]*:  
*tail* (*drop* *i* *l* *s*) = (*case* *i* *of* *Suc* *i*  $\Rightarrow$  *drop* *i* *l* (*tail* *s*) | 0  $\Rightarrow$  *drop* (*l* - 2) *l* (*tail* (*tail* *s*)))  
**unfolding** *drop\_def J.dtor\_corec*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def split: nat.splits*)

**lemma** *drop\_code[code]*:  
*drop* (*Suc* *i*) *l* (*SCons* *m* *s*) = *SCons* *m* (*drop* *i* *l* *s*)  
*drop* 0 *l* (*SCons* *m* (*SCons* *n* *s*)) = *SCons* *n* (*drop* (*l* - 2) *l* *s*)  
**by** (*smt2 J.ctor\_dtor J.dtor\_ctor fst\_conv snd\_conv prod.collapse head\_drop tail\_drop nat.case*)+

## 1.18 diff

**definition** *diff* :: *stream*  $\Rightarrow$  *stream* **where**  
*diff* *s* = *dtor\_corec*-*J* ( $\lambda s. (head (tail s) - head\ s, Inr (tail s))$ ) *s*

**lemma** *head\_diff[simp]*: *head* (*diff* *s*) = *head* (*tail* *s*) - *head* *s*  
**unfolding** *diff\_def J.dtor\_corec*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *tail\_diff[simp]*: *tail* (*diff* *s*) = *diff* (*tail* *s*)  
**apply** (*subst diff\_def*) **unfolding** *J.dtor\_corec*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def diff\_def*)

**lemma** *diff\_code[code]*: *diff* (*SCons* *m* (*SCons* *n* *s*)) = *SCons* (*n* - *m*) (*diff* (*SCons* *n* *s*))  
**by** (*metis J.ctor\_dtor J.dtor\_ctor fst\_conv snd\_conv prod.collapse head\_diff tail\_diff*)

## 1.19 Some Proofs

**theorem** *thue\_code[code]*: *thue* = *SCons* 0 (*interleave* (*inv* *thue*) (*tail* *thue*))  
**unfolding** *thue\_def J.ctor\_inject Pair\_eq*

by (rule conjI[OF refl], coinduction rule: stream\_coinduct0) (auto simp: J.dtor\_ctor)

**theorem plus\_commute:**  $plus\ s\ t = plus\ t\ s$   
 by (coinduction arbitrary: s t rule: stream\_coinduct) auto

**theorem plus\_assoc:**  $plus\ (plus\ s\ t)\ u = plus\ s\ (plus\ t\ u)$   
 by (coinduction arbitrary: s t u rule: stream\_coinduct) auto

**theorem plus\_commute\_assoc:**  $plus\ s\ (plus\ t\ u) = plus\ t\ (plus\ s\ u)$   
 by (metis plus\_assoc plus\_commute)

**theorem even\_plus[simp]:**  $even\ (plus\ s\ t) = plus\ (even\ s)\ (even\ t)$   
 by (coinduction arbitrary: s t rule: stream\_coinduct) auto

**theorem odd\_alt:**  $odd\ s = even\ (tail\ s)$   
 by (coinduction arbitrary: s rule: stream\_coinduct) auto

**theorem even\_alt:**  $even\ s = SCons\ (head\ s)\ (odd\ (tail\ s))$   
 by (coinduction arbitrary: s rule: stream\_coinduct0) (auto simp: J.dtor\_ctor odd\_alt)

**theorem sum\_odd\_fib:**  $sum\ (odd\ fib) = even\ fib$   
 by (coinduction rule: stream\_coinduct1)  
 (auto simp: J.dtor\_ctor plus\_assoc plus\_commute\_assoc odd\_alt  
 intro: genCngdd1\_alg01[folded plus\_uniform])

## 2 Binary Tree Examples

### 2.1 one

**definition one :: btree where**  
 $one = dtor_corec\_J\ (\lambda\_.\ (1, Inr\ (), Inr\ ()))\ ()$

**lemma val\_one[simp]:**  $val\ one = 1$   
 unfolding one\_def J.dtor\_corec  
 by (simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def)

**lemma left\_one[simp]:**  $left\ one = one$   
 unfolding one\_def J.dtor\_corec  
 by (simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def)

**lemma right\_one[simp]:**  $right\ one = one$   
 unfolding one\_def J.dtor\_corec  
 by (simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def)

**lemma one\_code[code]:**  
 $one = Node\ 1\ one\ one$   
 by (metis J.ctor\_dtor prod.collapse val\_one left\_one right\_one)

### 2.2 plus

**definition plus :: btree  $\Rightarrow$  btree  $\Rightarrow$  btree where**  
 $plus\ t\ u = dtor_corec\_J\ (\lambda(t, u).\ (val\ t + val\ u, Inr\ (left\ t, left\ u), Inr\ (right\ t, right\ u)))\ (t, u)$

**lemma val\_plus[simp]:**  $val\ (plus\ t\ u) = val\ t + val\ u$   
 unfolding plus\_def J.dtor\_corec  
 by (simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def)



**lemma** *left\_plus[simp]*:  $\text{left } (\text{plus } t \ u) = \text{plus } (\text{left } t) (\text{left } u)$   
**unfolding** *plus\_def J.dtor\_corec*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *right\_plus[simp]*:  $\text{right } (\text{plus } t \ u) = \text{plus } (\text{right } t) (\text{right } u)$   
**unfolding** *plus\_def J.dtor\_corec*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *plus\_code[code]*:  
 $\text{plus } (\text{Node } x1 \ l1 \ r1) (\text{Node } x2 \ l2 \ r2) = \text{Node } (x1 + x2) (\text{plus } l1 \ l2) (\text{plus } r1 \ r2)$   
**by** (*smt2 J.ctor\_dtor J.dtor\_ctor fst\_conv snd\_conv prod.collapse val\_plus left\_plus right\_plus*)

**lemma** *plus\_uniform*:  $\text{plus } s \ t = \text{alg}\varrho 1 \ (s, t)$   
**unfolding** *plus\_def*  
**apply** (*rule fun\_cong[OF sym[OF J.dtor\_corec-unique]]*)  
**unfolding** *alg\varrho 1 o\_def[symmetric] o\_assoc*  
**apply** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def fun\_eq\_iff J.dtor\_ctor*  
 $\varrho 1\_def \text{Let\_def convol\_def eval1\_op1 eval1\_leaf1'}$   
 $\text{o\_eq\_dest[OF Abs\_}\Sigma 1\_natural] \text{alg}\Lambda 1\_Inr \text{alg}\varrho 1\_def$ )  
**done**

## 2.3 divide

**definition** *divide* ::  $\text{btree} \Rightarrow \text{btree} \Rightarrow \text{btree}$  **where**  
 $\text{divide } t \ u = \text{dtor\_corec\_J}$   
 $(\lambda(t, u). (\text{val } t / \text{val } u, \text{Inr } (\text{left } t, \text{left } u), \text{Inr } (\text{right } t, \text{right } u))) (t, u)$

**lemma** *val\_divide[simp]*:  $\text{val } (\text{divide } t \ u) = \text{val } t / \text{val } u$   
**unfolding** *divide\_def J.dtor\_corec*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *left\_divide[simp]*:  $\text{left } (\text{divide } t \ u) = \text{divide } (\text{left } t) (\text{left } u)$   
**unfolding** *divide\_def J.dtor\_corec*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *right\_divide[simp]*:  $\text{right } (\text{divide } t \ u) = \text{divide } (\text{right } t) (\text{right } u)$   
**unfolding** *divide\_def J.dtor\_corec*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *divide\_code[code]*:  
 $\text{divide } (\text{Node } x1 \ l1 \ r1) (\text{Node } x2 \ l2 \ r2) = \text{Node } (x1 / x2) (\text{divide } l1 \ l2) (\text{divide } r1 \ r2)$   
**by** (*smt2 J.ctor\_dtor J.dtor\_ctor fst\_conv snd\_conv prod.collapse val\_divide left\_divide right\_divide*)

**lemma** *divide\_uniform*:  $\text{divide } s \ t = \text{alg}\varrho 2 \ (s, t)$   
**unfolding** *divide\_def*  
**apply** (*rule fun\_cong[OF sym[OF J.dtor\_corec-unique]]*)  
**unfolding** *alg\varrho 2 o\_def[symmetric] o\_assoc*  
**apply** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def fun\_eq\_iff J.dtor\_ctor*  
 $\varrho 2\_def \text{Let\_def convol\_def eval2\_op2 eval2\_leaf2'}$   
 $\text{o\_eq\_dest[OF Abs\_}\Sigma 2\_natural] \text{alg}\Lambda 2\_Inr \text{alg}\varrho 2\_def$ )  
**done**

## 2.4 bird

**definition** *bird* **where**  
 $\text{bird} = \text{corecUU2 } (\lambda_. \text{GUARD2 } (1,$   
 $\text{DIV2 } (\text{END2 one}, \text{PLS2 } (\text{CONT2 } (), \text{END2 one})),$   
 $\text{PLS2 } (\text{DIV2 } (\text{END2 one}, \text{CONT2 } ()), \text{END2 one}))) ()$

**lemma** *val\_bird*[simp]: *val bird = 1*  
**unfolding** *bird\_def corecUU2*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def J.dtor\_ctor eval2\_leaf2'*)

**lemma** *left\_bird*[simp]: *left bird = divide one (plus bird one)*  
**apply** (*subst bird\_def*) **unfolding** *corecUU2*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def J.dtor\_ctor eval2\_leaf2' eval2\_op2*  
*algΛ2\_Inl algΛ2\_Inr algΛ1\_Inr o\_eq\_dest[OF Abs\_Σ2\_natural] o\_eq\_dest[OF Abs\_Σ1\_natural]*  
*plus\_uniform divide\_uniform bird\_def*)

**lemma** *right\_bird*[simp]: *right bird = plus (divide one bird) one*  
**apply** (*subst bird\_def*) **unfolding** *corecUU2*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def J.dtor\_ctor eval2\_leaf2' eval2\_op2*  
*algΛ2\_Inl algΛ2\_Inr algΛ1\_Inr o\_eq\_dest[OF Abs\_Σ2\_natural] o\_eq\_dest[OF Abs\_Σ1\_natural]*  
*plus\_uniform divide\_uniform bird\_def*)

**lemma** *bird\_code*[code]: *bird = Node 1 (divide one (plus bird one)) (plus (divide one bird) one)*  
**by** (*metis J.ctor\_dtor prod.collapse val\_bird left\_bird right\_bird*)

## 2.5 mirror

**definition** *mirror* :: *btree*  $\Rightarrow$  *btree* **where**  
*mirror t = dtor\_corec\_J* ( $\lambda t. (val\ t, Inr\ (right\ t), Inr\ (left\ t))$ ) *t*

**lemma** *val\_mirror*[simp]: *val (mirror t) = val t*  
**unfolding** *mirror\_def J.dtor\_corec*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *left\_mirror*[simp]: *left (mirror t) = mirror (right t)*  
**unfolding** *mirror\_def J.dtor\_corec*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *right\_mirror*[simp]: *right (mirror t) = mirror (left t)*  
**unfolding** *mirror\_def J.dtor\_corec*  
**by** (*simp add: map\_pre\_J\_def BNF\_Comp.id\_bnf\_comp\_def*)

**lemma** *mirror\_code*[code]:  
*mirror (Node x l r) = Node x (mirror r) (mirror l)*  
**by** (*metis J.ctor\_dtor J.dtor\_ctor fst\_conv snd\_conv prod.collapse val\_mirror left\_mirror right\_mirror*)

## 2.6 Some Proofs

**theorem** *mirror\_one*[simp]: *mirror one = one*  
**by** (*coinduction rule: btree\_coinduct*) *auto*

**theorem** *mirror\_plus*[simp]: *mirror (plus r s) = plus (mirror r) (mirror s)*  
**by** (*coinduction arbitrary: r s rule: btree\_coinduct*) *auto*

**theorem** *mirror\_divide*[simp]: *mirror (divide r s) = divide (mirror r) (mirror s)*  
**by** (*coinduction arbitrary: r s rule: btree\_coinduct*) *auto*

**theorem** *divide\_divide\_one*[simp]: *divide one (divide one r) = r*  
**by** (*coinduction arbitrary: r rule: btree\_coinduct*) *auto*

**theorem** *mirror\_bird*: *mirror bird = divide one bird*  
**by** (*coinduction rule: btree\_coinduct2*)  
*(auto intro!: genCngdd2\_algo1[folded plus\_uniform] intro: genCngdd2\_algo2[folded divide\_uniform]*  
*genCngdd2\_trans[OF genCngdd2\_algo2[folded divide\_uniform], of \_ \_ \_ divide one bird])*