

## lib/main/integer-plus.ath

```

1 # .....
2 #
3 # Integer datatype, Z, and an addition operator, Z.Plus
4 #
5
6 load "nat-less"
7
8 load "nat-minus"
9
10 structure Z := (pos N) | (neg N)
11
12 expand-input pos, neg [int->nat]
13
14 set-precedence (pos neg) (plus 1 (get-precedence *))
15
16 assert* Z-structure-axioms :=
17   [(pos x = pos y <==> x = y)
18    (neg x = neg y <==> x = y)
19    ((exists y . x = pos y) |
20     (exists y . x = neg y))]
21
22 set-precedence (pos neg) (plus (get-precedence *) 1)
23
24 module Z {
25   open N
26   declare zero, one: Z
27   assert zero-definition := (zero = pos N.zero)
28   assert zero-property := (zero = neg N.zero)
29   assert one-definition := (one = pos N.one)
30
31   define [a b c] := [?a:Z ?b:Z ?c:Z]
32
33   declare +: [Z Z] -> Z
34
35   module Plus {
36     overload + N.+
37     define [x y] := [?x:N ?y:N]
38
39     assert axioms :=
40       (fun [(pos x + pos y) = (pos (x + y)) #pos-pos
41            (pos x + neg y) =
42              [(neg (y - x)) when (x < y) #pos-neg-case1
43               (pos (x - y)) when (~ x < y)] #pos-neg-case2
44            (neg x + pos y) =
45              [(pos (y - x)) when (x < y) #neg-pos-case1
46               (neg (x - y)) when (~ x < y)] #neg-pos-case2
47            (neg x + neg y) = (neg (x + y))] #neg-neg
48
49     # Alternatively:
50     assert* axioms :=
51       [(pos x + pos y = pos (x + y)) #pos-pos
52        ( x < y ==> pos x + neg y = neg (y - x)) #pos-neg-case1
53        (~ x < y ==> pos x + neg y = pos (x - y)) #pos-neg-case2
54        ( x < y ==> neg x + pos y = pos (y - x)) #neg-pos-case1
55        (~ x < y ==> neg x + pos y = neg (x - y)) #neg-pos-case2
56        (neg x + neg y = neg (x + y))] #neg-neg
57
58     define [pos-pos pos-neg-case1 pos-neg-case2
59            neg-pos-case1 neg-pos-case2 neg-neg] := axioms
60   } # close module Plus
61
62   declare negate: [Z] -> Z [120]
63
64   module Negate {
65     define x := ?x:N
66     assert positive := (forall x . negate pos x = neg x)
67     assert negative := (forall x . negate neg x = pos x)

```

```

68 } # close module Negate
69
70 declare -: [Z Z] -> Z
71
72 module Minus {
73   assert definition := (forall a b . a - b = a + negate b)
74 } # close module Minus
75
76 extend-module Plus {
77   open Minus
78   overload - N.-
79
80   define Right-Inverse := (forall a . a + negate a = zero)
81
82   datatype-cases Right-Inverse {
83     (pos x) =>
84       conclude (pos x + negate pos x = zero)
85       let { _ := (!chain-> [true ==> (~ x < x) [Less.irreflexive]]) }
86         (!chain [(pos x + negate pos x)
87           --> (pos x + neg x)           [Negate.positive]
88           --> (pos (x - x))           [pos-neg-case2]
89           --> (pos Top.zero)         [N.Minus.second-equal]
90           <-- zero                     [zero-definition]])
91   | (neg x) =>
92     conclude (neg x + negate neg x = zero)
93     let { _ := (!chain-> [true ==> (~ x < x) [Less.irreflexive]]) }
94       (!chain [(neg x + negate neg x)
95         --> (neg x + pos x)           [Negate.negative]
96         --> (neg (x - x))           [neg-pos-case2]
97         --> (neg Top.zero)         [N.Minus.second-equal]
98         <-- zero                     [zero-property]])
99   }
100
101   define Right-Identity := (forall a . a + zero = a)
102   define Left-Identity := (forall a . zero + a = a)
103
104   datatype-cases Right-Identity {
105     (pos x) =>
106       conclude (pos x + zero = pos x)
107       (!chain [(pos x + zero)
108         --> (pos x + pos Top.zero)   [zero-definition]
109         --> (pos (x + Top.zero))     [pos-pos]
110         --> (pos x)                 [N.Plus.right-zero]])
111   | (neg x) =>
112     conclude (neg x + zero = neg x)
113     let { _ := (!chain-> [true ==> (~ x < Top.zero) [Less.not-zero]]) }
114       (!chain [(neg x + zero)
115         --> (neg x + pos Top.zero)   [zero-definition]
116         --> (neg (x - Top.zero))     [neg-pos-case2]
117         --> (neg x)                 [N.Minus.zero-right]])
118   }
119
120   datatype-cases Left-Identity {
121     (pos x) =>
122       conclude (zero + pos x = pos x)
123       (!chain [(zero + pos x)
124         --> (pos Top.zero + pos x)   [zero-definition]
125         --> (pos (Top.zero + x))     [pos-pos]
126         --> (pos x)                 [N.Plus.left-zero]])
127   | (neg x) =>
128     conclude (zero + neg x = neg x)
129     (!chain [(zero + neg x)
130       --> (neg Top.zero + neg x)     [zero-property]
131       --> (neg (Top.zero + x))     [neg-neg]
132       --> (neg x)                 [N.Plus.left-zero]])
133   }
134
135   define associative := (forall a b c . (a + b) + c = a + (b + c))
136
137   # A direct proof of Z.Plus.associative would be very long and tedious,

```

```

138 # as there are three variables to consider in combinations of positive
139 # and negative values and the pos-neg and neg-pos
140 # cases require lemmas about reassociating Minus terms. Instead we
141 # transform to a representation where the proof is easy (though some of
142 # the proofs about the tranformation are somewhat long themselves).
143
144 } # close module Plus
145
146 structure NN := (nn N N)
147
148 module NN {
149   overload + N.+
150
151   define [a b c] := [?a:NN ?b:NN ?c:NN]
152
153   declare +': [NN NN] -> NN [110]
154
155   module Plus {
156     define [a1 a2 b1 b2] := [?a1:N ?a2:N ?b1:N ?b2:N]
157     assert definition :=
158       (forall a1 a2 b1 b2 .
159         (nn a1 a2) +' (nn b1 b2) = (nn (a1 + b1) (a2 + b2)))
160
161     define associative := (forall a b c . (a +' b) +' c = a +' (b +' c))
162
163     datatype-cases associative {
164       (nn a1 a2) =>
165         datatype-cases
166           (forall b c . ((nn a1 a2) +' b) +' c =
167             (nn a1 a2) +' (b +' c)) {
168         (nn b1 b2) =>
169           datatype-cases
170             (forall c . ((nn a1 a2) +' (nn b1 b2)) +' c =
171               (nn a1 a2) +' ((nn b1 b2) +' c)) {
172         (nn c1 c2) =>
173           (!chain
174             [((nn a1 a2) +' (nn b1 b2)) +' (nn c1 c2))
175             --> ((nn a1 + b1 a2 + b2) +' (nn c1 c2)) [definition]
176             --> (nn (a1 + b1) + c1 (a2 + b2) + c2) [definition]
177             --> (nn a1 + (b1 + c1) a2 + (b2 + c2)) [N.Plus.associative]
178             <-- ((nn a1 a2) +' (nn b1 + c1 b2 + c2)) [definition]
179             <-- ((nn a1 a2) +' ((nn b1 b2) +' (nn c1 c2)))
180                   [definition]])
181           }
182       }
183   }
184 } # close module Plus
185 } # close module NN
186
187 declare Z->NN: [Z] -> NN
188 declare NN->Z: [NN] -> Z
189
190 module Z-NN {
191   overload (+ N.+) (- N.-)
192   define [x y] := [?x:N ?y:N]
193   assert to-pos := (forall x . Z->NN pos x = nn x Top.zero)
194   assert to-neg := (forall x . Z->NN neg x = nn Top.zero x)
195   assert from-pos :=
196     (forall x y . ~ x < y ==> NN->Z (nn x y) = pos (x - y))
197   assert from-neg :=
198     (forall x y . x < y ==> NN->Z (nn x y) = neg (y - x))
199
200   define inverse := (forall a . NN->Z Z->NN a = a)
201
202   datatype-cases inverse {
203     (pos x) => {
204       (!chain-> [true ==> (~ x < Top.zero) [N.Less.not-zero]]);
205       (!chain [(NN->Z Z->NN pos x)
206         --> (NN->Z (nn x Top.zero)) [to-pos]
207         --> (pos (x - Top.zero)) [from-pos]

```

```

208     --> (pos x)                [N.Minus.zero-right]])
209   }
210 | (neg x) =>
211   (!two-cases
212     assume (x = Top.zero)
213     let {_ := (!chain-> [true ==> (~ Top.zero < Top.zero)
214                           [N.Less.irreflexive]])}
215     (!chain [(NN->Z Z->NN neg x)
216               --> (NN->Z (nn Top.zero x))          [to-neg]
217               --> (NN->Z (nn Top.zero Top.zero))    [(x = Top.zero)]
218               --> (pos (Top.zero - Top.zero))      [from-pos]
219               --> (pos Top.zero)                   [N.Minus.zero-right]
220               <-- zero                             [zero-definition]
221               --> (neg Top.zero)                   [zero-property]
222               <-- (neg x)                          [(x = Top.zero)]]])
223     assume (~ x = Top.zero)
224     let {A := (!chain->
225               [true ==> (Top.zero <= x)  [N.Less=.zero<=]
226               ==> (Top.zero < x | Top.zero = x)
227                   [N.Less=.definition]])}
228     (!cases A
229       assume (Top.zero < x)
230       (!chain [(NN->Z Z->NN neg x)
231                 --> (NN->Z (nn Top.zero x))          [to-neg]
232                 --> (neg (x - Top.zero))            [from-neg]
233                 --> (neg x)                         [N.Minus.zero-right]])
234       assume (Top.zero = x)
235       (!from-complements (NN->Z Z->NN neg x = neg x)
236         (!sym (Top.zero = x) (x =/= Top.zero))))
237 } # datatype-cases
238 } # close module Z-NN
239
240 module NN-equivalence {
241 overload - N.-
242 define [x y] := [?x:N ?y:N]
243 assert case1 :=
244   (forall x y . x < y ==> (nn x y) = nn Top.zero (y - x))
245 assert case2 :=
246   (forall x y . ~ x < y ==> (nn x y) = nn (x - y) Top.zero)
247 } # close module NN-equivalence
248
249 extend-module Z-NN {
250 define +' := NN.+
251
252 define additive-homomorphism :=
253   (forall a b . Z->NN (a + b) = (Z->NN a) +' (Z->NN b))
254
255 ## Proof sketch (uses force):
256
257 datatype-cases additive-homomorphism {
258   (pos x) =>
259     datatype-cases
260     (forall ?b .
261       (Z->NN (pos x + ?b)) = (Z->NN (pos x)) +' (Z->NN ?b)) {
262     (pos y) =>
263       (!combine-equations
264         (!chain [(Z->NN (pos x + pos y))
265                   --> (Z->NN (pos (x + y)))          [Plus.pos-pos]
266                   --> (nn (x + y) Top.zero)         [to-pos]])
267         (!chain [(Z->NN (pos x) +' (Z->NN pos y))
268                   --> ((nn x Top.zero) NN.+ (nn y Top.zero)) [to-pos]
269                   --> (nn (x + y) (Top.zero + Top.zero)) [NN.Plus.definition]
270                   --> (nn (x + y) Top.zero)             [N.Plus.right-zero]]))
271     | (neg y) =>
272       (!two-cases
273         assume (x < y)
274         (!force (Z->NN ((pos x) + (neg y)) =
275                 (Z->NN pos x) +' (Z->NN neg y)))
276         assume (~ x < y)
277         (!force (Z->NN ((pos x) + (neg y)) =

```

```

278         (Z->NN pos x) + (Z->NN neg y)))
279     }
280 | (neg x) =>
281   datatype-cases
282     (forall ?b . Z->NN (neg x + ?b) =
283       (Z->NN neg x) + (Z->NN ?b)) {
284   (pos y) =>
285     (!two-cases
286       assume (x < y)
287       let { _ := (!chain-> [(x < y) ==> (~ y < x)
288                             [N.Less.asymmetric]]) }
289         (!force (Z->NN (neg x + pos y) =
290           (Z->NN neg x) + (Z->NN pos y)))
291       assume (~ x < y)
292       let { A := (!chain->
293         [ (~ x < y)
294           ==> (y <= x)      [N.Less=.trichotomy2]
295           ==> (y < x | y = x)
296             [N.Less=.definition]]) }
297         (!cases A
298           assume (y < x)
299             (!force (Z->NN (neg x + pos y) =
300               (Z->NN neg x) + (Z->NN pos y)))
301           assume (y = x)
302             let { _ := (!chain-> [true ==> (~ x < x)
303                                   [N.Less.irreflexive]]) }
304               (!force (Z->NN (neg x + pos y) =
305                 (Z->NN neg x) + (Z->NN pos y))))))
306   | (neg y) =>
307     (!force (Z->NN (neg x + neg y) =
308       (Z->NN neg x) + (Z->NN neg y)))
309   }
310 }
311
312 #.....
313 ## The complete proof:
314
315 datatype-cases additive-homomorphism {
316   (pos x) =>
317     datatype-cases
318       (forall ?b .
319         (Z->NN (pos x + ?b)) = (Z->NN (pos x)) + (Z->NN ?b)) {
320     (pos y) =>
321       (!combine-equations
322         (!chain [(Z->NN (pos x + pos y))
323           --> (Z->NN (pos (x + y)))          [Plus.pos-pos]
324           --> (nn (x + y) Top.zero)         [to-pos]])
325         (!chain [(Z->NN pos x) + (Z->NN pos y)
326           --> ((nn x Top.zero) + (nn y Top.zero)) [to-pos]
327           --> (nn (x + y) (Top.zero + Top.zero)) [NN.Plus.definition]
328           --> (nn (x + y) Top.zero)             [N.Plus.right-zero]))
329     | (neg y) =>
330       (!two-cases
331         assume (x < y)
332           (!combine-equations
333             (!chain [(Z->NN (pos x + neg y))
334               --> (Z->NN neg (y - x))          [Plus.pos-neg-case1]
335               --> (nn Top.zero (y - x))      [to-neg]])
336             (!chain [(Z->NN pos x) + (Z->NN neg y)
337               --> ((nn x Top.zero) + (nn Top.zero y)) [to-pos to-neg]
338               --> (nn (x + Top.zero) (Top.zero + y)) [NN.Plus.definition]
339               --> (nn x y)                          [N.Plus.right-zero N.Plus.left-zero]
340               --> (nn Top.zero (y - x))            [NN-equivalence.case1]))
341         assume (~ x < y)
342           (!combine-equations
343             (!chain [(Z->NN (pos x + neg y))
344               --> (Z->NN pos (x - y))          [Plus.pos-neg-case2]
345               --> (nn (x - y) Top.zero)       [to-pos]])
346             (!chain [(Z->NN pos x) + (Z->NN neg y)
347               --> ((nn x Top.zero) + (nn Top.zero y)) [to-pos to-neg]

```

```

348     --> (nn (x + Top.zero) (Top.zero + y)) [NN.Plus.definition]
349     --> (nn x y) [N.Plus.right-zero N.Plus.left-zero]
350     --> (nn (x - y) Top.zero) [NN-equivalence.case2]]))
351   }
352 | (neg x) =>
353   datatype-cases
354     (forall ?b . Z->NN (neg x + ?b) =
355       (Z->NN neg x) +' (Z->NN ?b)) {
356   (pos y) =>
357     (!two-cases
358       assume (x < y)
359       let { _ := (!chain-> [(x < y) ==> (~ y < x)
360                             [N.Less.asymmetric]]) }
361       (!combine-equations
362         (!chain [(Z->NN (neg x + pos y))
363                 --> (Z->NN pos (y - x)) [Plus.neg-pos-case1]
364                 --> (nn (y - x) Top.zero) [to-pos]])
365         (!chain [(Z->NN neg x) +' (Z->NN pos y))
366                 --> ((nn Top.zero x) +' (nn y Top.zero)) [to-neg to-pos]
367                 --> (nn (Top.zero + y) (x + Top.zero))
368                       [NN.Plus.definition]
369                 --> (nn y x) [N.Plus.right-zero N.Plus.left-zero]
370                 --> (nn (y - x) Top.zero) [NN-equivalence.case2]]))
371       assume (~ x < y)
372       let { A := (!chain->
373                 [ (~ x < y)
374                   ==> (y <= x) [N.Less=.trichotomy2]
375                   ==> (y < x | y = x)
376                     [N.Less=.definition]]) }
377       (!cases A
378         assume (y < x)
379         (!combine-equations
380           (!chain [(Z->NN (neg x + pos y))
381                   --> (Z->NN neg (x - y)) [Plus.neg-pos-case2]
382                   --> (nn Top.zero (x - y)) [to-neg]])
383           (!chain [(Z->NN neg x) +' (Z->NN pos y))
384                   --> ((nn Top.zero x) +' (nn y Top.zero))
385                   [to-neg to-pos]
386                   --> (nn (Top.zero + y) (x + Top.zero))
387                       [NN.Plus.definition]
388                   --> (nn y x) [N.Plus.left-zero
389                               N.Plus.right-zero]
390                   --> (nn Top.zero (x - y)) [NN-equivalence.case1]]))
391         assume (y = x)
392         let { _ := (!chain-> [true ==> (~ x < x)
393                               [N.Less.irreflexive]]) }
394         (!combine-equations
395           (!chain [(Z->NN (neg x + pos y))
396                   --> (Z->NN neg (x - y)) [Plus.neg-pos-case2]
397                   --> (nn Top.zero (x - y)) [to-neg]
398                   --> (nn Top.zero (x - x)) [(y = x)]
399                   --> (nn Top.zero Top.zero) [N.Minus.second-equal]])
400           (!chain [(Z->NN neg x) +' (Z->NN pos y))
401                   --> ((nn Top.zero x) +' (nn y Top.zero))
402                   [to-neg to-pos]
403                   --> (nn (Top.zero + y) (x + Top.zero))
404                       [NN.Plus.definition]
405                   --> (nn y x) [N.Plus.left-zero
406                               N.Plus.right-zero]
407                   --> (nn x x) [(y = x)]
408                   --> (nn (x - x) Top.zero) [NN-equivalence.case2]
409                   --> (nn Top.zero Top.zero) [N.Minus.second-equal]]))
410   | (neg y) =>
411     (!combine-equations
412       (!chain [(Z->NN (neg x + neg y))
413               --> (Z->NN neg (x + y)) [Plus.neg-neg]
414               --> (nn Top.zero (x + y)) [to-neg]])
415       (!chain [(Z->NN neg x) +' (Z->NN neg y))
416               --> ((nn Top.zero x) +' (nn Top.zero y)) [to-neg]
417               --> (nn (Top.zero + Top.zero) (x + y)) [NN.Plus.definition]

```

```

418         --> (nn Top.zero (x + y))           [N.Plus.right-zero]))
419     }
420 } # datatype-cases
421 } # close module Z-NN
422
423 # Finally:
424
425 extend-module Plus {
426 define +' := NN.+'
427
428 conclude associative
429   pick-any a:Z b:Z c:Z
430   let {f:(OP 1) := Z->NN;
431       g:(OP 1) := NN->Z;
432       f-application :=
433         conclude ((f ((a + b) + c)) = (f (a + (b + c))))
434         (!chain
435           [(f ((a + b) + c))
436            --> ((f (a + b)) +' (f c))      [Z-NN.additive-homomorphism]
437            --> ((f a +' f b) +' f c)      [Z-NN.additive-homomorphism]
438            --> (f a +' (f b +' f c))      [NN.Plus.associative]
439            <-- (f a +' (f (b + c)))      [Z-NN.additive-homomorphism]
440            <-- (f (a + (b + c)))      [Z-NN.additive-homomorphism]])}
441   conclude ((a + b) + c) = a + (b + c)
442   (!chain [(a + b) + c)
443     <-- (g f ((a + b) + c)) [Z-NN.inverse]
444     --> (g f (a + (b + c))) [f-application]
445     --> (a + (b + c))      [Z-NN.inverse]])
446
447 # Next, the commutative property:
448
449 define commutative := (forall a b . a + b = b + a)
450
451 # Prove it by transforming to NN representation, as with associativity.
452
453 extend-module NN {
454 extend-module Plus {
455 define commutative := (forall a b . a +' b = b +' a)
456
457 datatype-cases commutative {
458   (nn a1 a2) =>
459     datatype-cases
460       (forall ?b . (nn a1 a2) +' ?b = ?b +' (nn a1 a2)) {
461         (nn b1 b2) =>
462           (!chain [(nn a1 a2) +' (nn b1 b2))
463             --> (nn (a1 + b1) (a2 + b2)) [definition]
464             --> (nn (b1 + a1) (b2 + a2)) [N.Plus.commutative]
465             <-- ((nn b1 b2) +' (nn a1 a2)) [definition]])
466       }
467   }
468 } # close module Plus
469 } # close module NN
470
471 conclude commutative
472   pick-any a:Z b:Z
473   let {f:(OP 1) := Z->NN;
474       g:(OP 1) := NN->Z;
475       f-application :=
476         conclude (f (a + b) = f (b + a))
477         (!chain [(f (a + b))
478           --> (f a +' f b) [Z-NN.additive-homomorphism]
479           --> (f b +' f a) [NN.Plus.commutative]
480           <-- (f (b + a)) [Z-NN.additive-homomorphism]])}
481   conclude (a + b = b + a)
482   (!chain [(a + b)
483     <-- (g f (a + b)) [Z-NN.inverse]
484     --> (g f (b + a)) [f-application]
485     --> (b + a)      [Z-NN.inverse]])
486
487

```

```
488 } # close module Plus  
489  
490 } # close module Z
```