

lib/main/nat-minus.ath

```

1 # Subtraction of natural numbers.
2
3 load "nat-less"
4
5 extend-module N {
6
7 declare -: [N N] -> N [200]
8
9 module Minus {
10
11 define [x y z] := [?x:N ?y:N ?z:N]
12
13
14 assert* axioms :=
15   [(zero - x = zero)
16    (x - zero = x)
17    (S x - S y = x - y)]
18
19 define [zero-left zero-right both-nonzero] := axioms
20
21 define Plus-Cancel := (forall y x . y <= x ==> x = (x - y) + y)
22
23 by-induction Plus-Cancel {
24   zero =>
25     conclude (forall ?x . zero <= ?x ==> ?x = (?x - zero) + zero)
26     pick-any x
27     assume (zero <= x)
28     (!sym (!chain [(x - zero) + zero)
29                  = (x + zero)           [zero-right]
30                  = x                    [Plus.right-zero]]))
31 | (S y) =>
32   let {ind-hyp := (forall ?x . y <= ?x ==> ?x = (?x - y) + y)}
33   datatype-cases
34     (forall ?x . S y <= ?x ==> ?x = (?x - S y) + S y) {
35     zero =>
36       conclude
37         (S y <= zero ==> (zero = (zero - S y) + S y))
38         assume A := (S y <= zero)
39         (!from-complements (zero = (zero - S y) + S y)
40          A
41          (!chain-> [true ==> (~ A) [Less=.not-S-zero]]))
42 | (S x) =>
43       conclude
44         (S y <= S x ==> (S x = (S x - S y) + S y))
45         assume A := (S y <= S x)
46         let {C := (!chain-> [A ==> (y <= x) [Less=.injective]])}
47         (!sym (!chain
48               [(S x - S y) + S y)
49                = ((x - y) + S y)           [both-nonzero]
50                = (S ((x - y) + y))       [Plus.right-nonzero]
51                = (S x)                   [C ind-hyp]))
52     }
53 }
54
55 define second-equal := (forall x . x - x = zero)
56
57 by-induction second-equal {
58   zero => (!chain [(zero - zero) = zero [zero-left]])
59 | (S x) =>
60   let {ind-hyp := (x - x = zero)}
61   (!chain [(S x - S x) = (x - x) [both-nonzero]
62           = zero [ind-hyp]])
63 }
64
65 #Or, without using induction:
66 conclude second-equal
67 pick-any x:N
68 (!chain-> [true

```

```

69         ==> (x <= x)                                [Less=.reflexive]
70         ==> (x = (x - x) + x)                        [Plus-Cancel]
71         ==> (zero + x = (x - x) + x)                [Plus.left-zero]
72         ==> (zero = x - x)                           [Plus.=cancellation]
73         ==> (x - x = zero)                           [sym]]
74
75 define second-greater := (forall x y . x < y ==> x - y = zero)
76
77 by-induction second-greater {
78   zero =>
79     conclude (forall ?y . zero < ?y ==> zero - ?y = zero)
80     pick-any y
81     assume (zero < y)
82     (!chain [(zero - y) = zero [zero-left]])
83 | (S x) =>
84   let {ind-hyp := (forall ?y . x < ?y ==> x - ?y = zero)}
85   datatype-cases (forall ?y . S x < ?y ==> S x - ?y = zero)
86   {
87     zero =>
88       assume A := (S x < zero)
89       (!from-complements (S x - zero = zero)
90         A
91         (!chain-> [true ==> (~ A) [Less.not-zero]]))
92 | (S y) =>
93     assume A := (S x < S y)
94     let {C := (!chain-> [A ==> (x < y) [Less.injective]])}
95     (!chain [(S x - S y)
96       = (x - y)          [both-nonzero]
97       = zero             [C ind-hyp]])
98   }
99 }
100
101 define second-greater-or-equal :=
102   (forall x y . x <= y ==> x - y = zero)
103
104 conclude second-greater-or-equal
105 pick-any x:N y
106 assume A := (x <= y)
107 let {C := (!chain-> [A ==> (x < y | x = y) [Less=.definition]])}
108   (!cases C
109     (!chain [(x < y) ==> (x - y = zero) [second-greater]]
110       assume (x = y)
111         (!chain [(x - y) = (x - x) [(x = y)]
112           = zero [second-equal]]))
113   )
114
115 define alt-<=characterization :=
116   (forall x y . x <= y <==> exists z . y = x + z)
117
118 conclude alt-<=characterization
119 pick-any x y
120 (!equiv
121   (!chain [(x <= y)
122     ==> (y = (y - x) + x) [Plus-Cancel]
123     ==> (y = x + (y - x)) [Plus.commutative]
124     ==> (exists ?z . y = x + ?z) [existence]])
125   assume A := (exists ?z . y = x + ?z)
126   pick-witness z for A witnessed
127   (!chain-> [witnessed ==> (x <= y) [Less=.k-Less=]]))
128
129 define <-left := (forall x y . zero < y & y <= x ==> x - y < x)
130
131 conclude <-left
132 pick-any x y
133 assume A := (zero < y & y <= x)
134 let {goal := ((x - y) < x)}
135   (!by-contradiction goal
136     assume (~ goal)
137     (!absurd
138       (!chain-> [(zero < y)
139         ==> (zero + x < y + x) [Less.Plus-k]

```

```

139          ==> (x < y + x)                [Plus.left-zero]])
140      (!chain-> [(~ goal)
141          ==> (x <= x - y)                [Less=.trichotomy1]
142          ==> (x + y <= (x - y) + y) [Less=.Plus-k]
143          ==> (x + y <= x)    [(y <= x) Plus-Cancel]
144          ==> (~ x < x + y)    [Less=.trichotomy4]
145          ==> (~ x < y + x)    [Plus.commutative]]]))
146
147 define Plus-Minus-property :=
148   (forall x y z . x = y + z ==> x - y = z)
149
150 conclude Plus-Minus-property
151   pick-any x y z
152     assume A := (x = y + z)
153     let {C1 :=
154         (!chain->
155           [A ==> (y <= x)                [Less=.k-Less=]
156           ==> (x = (x - y) + y) [Plus-Cancel]]);
157         C2 := (!chain-> [A ==> (x = z + y)    [Plus.commutative]])}
158     (!chain->
159       [(x - y) + y = x          [C1]
160        = (z + y)              [C2]
161        ==> ((x - y) = z)      [Plus.--cancellation]])
162
163 conclude Plus-Minus-property-1 :=
164   (forall x y z . x = y + z ==> x - z = y)
165 pick-any x:N y:N z:N
166   (!chain [(x = y + z)
167     ==> (x = z + y)    [Plus.commutative]
168     ==> (x - z = y)    [Plus-Minus-property]])
169
170 conclude Plus-Minus-property-2 :=
171   (forall x y z . x + y = z ==> x = z - y)
172 pick-any x:N y:N z:N
173   (!chain [(x + y = z)
174     ==> (z = x + y)    [sym]
175     ==> (z - y = x)    [Plus-Minus-property-1]
176     ==> (x = z - y)    [sym]])
177
178 conclude Plus-Minus-property-3 :=
179   (forall x y z . x + y = z ==> y = z - x)
180 pick-any x:N y:N z:N
181   (!chain [(x + y = z)
182     ==> (z = x + y)    [sym]
183     ==> (z - x = y)    [Plus-Minus-property]
184     ==> (y = z - x)    [sym]])
185
186 define Plus-Minus-properties :=
187   [Plus-Minus-property Plus-Minus-property-1
188    Plus-Minus-property-2 Plus-Minus-property-3]
189
190 define cancellation := (forall x y . (x + y) - x = y)
191
192 conclude cancellation
193   pick-any x y
194     (!chain->
195       [(x + y = x + y) ==> ((x + y) - x = y) [Plus-Minus-property]])
196
197 define comparison :=
198   (forall x y z . z < y & y <= x ==> x - y < x - z)
199
200 conclude comparison
201   pick-any x y z
202     let {A1 := (z < y);
203          A2 := (y <= x)}
204     assume (A1 & A2)
205     let {u := (x - y);
206          v := (x - z);
207          B1 := (!chain->
208            [A2 ==> (x = u + y)    [Plus-Cancel]]);

```

```

209     B2 := (!chain->
210           [(A1 & A2)
211             ==> (z < x)           [Less=.transitive1]
212             ==> (z <= x)          [Less=.Implied-by-<]
213             ==> (x = v + z)       [Plus-Cancel]
214             ==> (x = z + v)       [Plus.commutative]
215             ==> (u + y = z + v) [B1]]])
216 (!by-contradiction (u < v)
217   assume (~ u < v)
218   let {C1 := (!chain->
219             [(~ u < v) ==> (v <= u) [Less=.trichotomy2]]);
220       C2 := (!chain->
221             [(z < y) ==> (z + v < y + v) [Less.Plus-k]
222              ==> (z + v < v + y) [Plus.commutative]]);
223       C3 := (!chain->
224             [(v <= u)
225              ==> (v + y <= u + y) [Less=.Plus-k]
226              ==> (z + v < v + y & v + y <= u + y) [augment]
227              ==> (z + v < u + y) [Less=.transitive1]
228              ==> (u + y =/= z + v) [Less.not-equal1]]])
229   (!absurd B2 C3))
230
231 define Times-Distributivity :=
232   (forall x y z . x * y - x * z = x * (y - z))
233
234 conclude Times-Distributivity
235 pick-any x y z
236 (!two-cases
237   assume A := (z <= y)
238   (!chain->
239     [(x * y)
240      = (x * ((y - z) + z)) [Plus-Cancel]
241      = (x * (y - z) + x * z) [Times.left-distributive]
242      = (x * z + x * (y - z)) [Plus.commutative]
243      ==> (x * y - x * z = x * (y - z))
244          [Plus-Minus-property]])
245   assume A := (~ z <= y)
246   let {C := (!chain-> [A ==> (y < z) [Less=.trichotomy1]])}
247   (!combine-equations
248     (!chain->
249       [C ==> (C | y = z) [alternate]
250        ==> (y <= z) [Less=.definition]
251        ==> (x * y <= x * z) [Times.<=-cancellation-conv]
252        ==> (x * y - x * z = zero)
253            [second-greater-or-equal]])
254     (!chain
255       [(x * (y - z))
256        = (x * zero) [second-greater]
257        = zero [Times.right-zero]]))
258 } # N.Minus
259 } # N

```