

lib/main/nat-times-less.ath

```

1  load "nat-times.ath"
2  load "nat-less.ath"
3
4  extend-module N {
5  extend-module Times {
6
7  define ==cancellation :=
8    (forall y z x . zero < x & x * y = x * z ==> y = z)
9
10 by-induction ==cancellation {
11   zero =>
12     pick-any z x
13     assume (zero < x & x * zero = x * z)
14     conclude (zero = z)
15     let {D := (!chain-last [(x * z)
16                             = (x * zero)      [(x * zero = x * z)]
17                             = zero            [right-zero]
18                             ==> (x = zero | z = zero)
19                             [no-zero-divisors]]]}
20     (!cases D
21      assume (x = zero)
22      (!from-complements (zero = z)
23       (x = zero)
24       (!chain-last
25        [(zero < x) ==> (x /= zero) [Less=.not-equal1]]))
26      assume (z = zero)
27      (!sym (z = zero)))
28 | (S y) =>
29   let {ind-hyp := (forall ?z ?x . zero < ?x & ?x * y = ?x * ?z ==> y = ?z)}
30   datatype-cases (forall ?z ?x .
31     zero < ?x & ?x * (S y) = ?x * ?z ==> (S y) = ?z)
32   {
33     zero =>
34       conclude (forall ?x .
35         zero < ?x & ?x * (S y) = ?x * zero ==> (S y) = zero)
36       pick-any x
37       assume (zero < x & x * (S y) = x * zero)
38       let {C1 := (!chain-last
39                 [(x * (S y))
40                  = (x * zero) [(x * (S y) = x * zero)]
41                  = zero      [right-zero]
42                  ==> (x = zero | (S y) = zero)
43                  [no-zero-divisors]]]}
44       (!cases C1
45        assume (x = zero)
46        (!from-complements ((S y) = zero)
47         (x = zero)
48         (!chain-last
49          [(zero < x) ==> (x /= zero) [Less=.not-equal1]]))
50        assume ((S y) = zero)
51        (!claim ((S y) = zero)))
52 | (S z) =>
53   conclude (forall ?x . zero < ?x & ?x * (S y) = ?x * (S z)
54     ==> (S y) = (S z))
55   pick-any x
56   assume (zero < x & x * (S y) = x * (S z))
57   (!chain-last
58    [(x * y + x)
59     = (x * (S y)) [right-nonzero]
60     = (x * (S z)) [(x * (S y) = x * (S z))]
61     = (x * z + x)
62     ==> (x * y = x * z) [Plus.--cancellation]
63     ==> (zero < x & x * y = x * z) [augment]
64     ==> (y = z) [ind-hyp]
65     ==> ((S y) = (S z)) [injective]])
66   }
67 }
68

```

```

69 define <-cancellation :=
70   (forall y z x . zero < x & x * y < x * z ==> y < z)
71
72 by-induction <-cancellation {
73   zero =>
74     pick-any z x
75     assume (zero < x & x * zero < x * z)
76     (!by-contradiction (zero < z)
77       assume A := (¬ zero < z)
78       let { _ := (!chain-last [A ==> (z = zero) [Less.=zero]]) }
79       (!absurd
80         (!chain-last
81           [(x * zero < x * z)
82             ==> (zero < zero) [right-zero (z = zero)]]
83         (!chain-last
84           [true ==> (¬ zero < zero) [Less.irreflexive]])))
85 | (S y) =>
86   let {ind-hyp := (forall ?z ?x . zero < ?x & ?x * y < ?x * ?z ==> y < ?z)}
87   datatype-cases (forall ?z ?x . zero < ?x & ?x * (S y) < ?x * ?z
88     ==> (S y) < ?z)
89   {
90     zero =>
91       pick-any x
92       assume (zero < x & x * (S y) < x * zero)
93       (!from-complements ((S y) < zero)
94         (!chain-last [(x * (S y) < x * zero)
95           ==> (x * (S y) < zero) [right-zero]]
96         (!chain-last [true ==> (¬ x * (S y) < zero) [Less.not-zero]]))
97 | (S z) =>
98   pick-any x
99   assume (zero < x & x * (S y) < x * (S z))
100   conclude ((S y) < (S z))
101   (!chain-last
102     [(x * (S y) < x * (S z))
103       ==> (x * y + x < x * z + x) [right-nonzero]
104       ==> (x * y < x * z) [Less.Plus-cancellation]
105       ==> (y < z) [(zero < x) ind-hyp]
106       ==> ((S y) < (S z)) [Less.injective]])
107   }
108 }
109
110 define <-cancellation-conv :=
111   (forall x y z . zero < x & y < z ==> x * y < x * z)
112
113 conclude <-cancellation-conv
114 pick-any x y z
115 assume A1 := (zero < x & y < z)
116 let {goal := (x * y < x * z)}
117 (!by-contradiction goal
118   assume (¬ goal)
119   let {D := (!chain-last
120     [ (¬ goal)
121       ==> (x * z <= x * y) [Less.trichotomy2]
122       ==> (x * z < x * y | x * z = x * y) [Less.definition]]})
123   (!cases D
124     assume A2 := (x * z < x * y)
125     (!chain-last
126       [A2 ==> (z < y) [<-cancellation]
127         ==> (¬ y < z) [Less.asymmetric]
128         ==> (y < z & ¬ y < z) [augment]
129         ==> false [prop-taut]])
130     assume A3 := (x * z = x * y)
131     (!absurd
132       (!chain-last
133         [(zero < x & A3) ==> (z = y) [=-cancellation]]
134       (!chain-last [(y < z)
135         ==> (¬ z = y) [Less.not-equal]])))
136   )
137
138 define <==cancellation-conv :=

```

```

139 (forall x y z . y <= z ==> x * y <= x * z)
140
141
142 conclude <--cancellation-conv
143 pick-any x y z
144   assume A := (y <= z)
145   let {goal := (x * y <= x * z)}
146     (!two-cases
147       assume A1 := (zero < x)
148       (!by-contradiction goal
149         assume (~ goal)
150           (!chain-last
151             [ (~ goal)
152               ==> (x * z < x * y) [Less=.trichotomy1]
153               ==> (z < y) [A1 <-cancellation]
154               ==> (~ y <= z) [Less=.trichotomy4]
155               ==> (A & ~ A) [augment]
156               ==> false [prop-taut]]))
157       assume A2 := (~ zero < x)
158       let {C := (!chain-last
159         [true ==> (~ x < zero) [Less.not-zero]
160           ==> (~ x < zero & A2) [augment]
161           ==> (x = zero) [Less.trichotomy1]]})
162         (!chain-first [goal <== (zero * y <= zero * z) [C]
163           <== (zero <= zero) [left-zero]
164           <== true [Less=.reflexive]]))
165
166 define identity-lemmal :=
167   (forall x y . zero < x & x * y = x ==> y = one)
168 define identity-lemma2 :=
169   (forall x y . x * y = one ==> x = one)
170
171 conclude identity-lemmal
172 pick-any x y
173   assume (zero < x & x * y = x)
174   (!chain-last
175     [(x * y = x)
176       ==> (x * y = x * one) [right-one]
177       ==> (y = one) [(zero < x) ==cancellation]])
178
179 conclude identity-lemma2
180 pick-any x y
181   assume A := (x * y = one)
182   let {C1 := (!by-contradiction (x /= zero)
183     assume (x = zero)
184       (!absurd
185         (!chain-last
186           [true ==> (zero * y = zero) [left-zero]
187             ==> (x * y = zero) [(x = zero)]
188             ==> (one = zero) [A]])
189         (!chain-last
190           [true ==> (one /= zero) [one-not-zero]]))));
191     C2 := (!by-contradiction (y /= zero)
192       assume (y = zero)
193         (!absurd
194           (!chain-last
195             [true ==> (x * zero = zero) [right-zero]
196               ==> (x * y = zero) [(y = zero)]
197               ==> (one = zero) [A]])
198           (!chain-last
199             [true ==> (one /= zero) [one-not-zero]]))));
200     C3 := (!by-contradiction (~ one < x)
201       assume (one < x)
202       let {_ := (!chain-last
203         [C2 ==> (zero < y) [Less.zero<]])}
204       (!absurd
205         (!chain-last
206           [(one < x)
207             ==> (zero < y & one < x) [augment]
208             ==> (y * one < y * x) [<-cancellation-conv]

```

```

209         ==> (one * y < x * y) [commutative]
210         ==> (one * y < one) [A]
211         ==> (y < (S zero)) [left-one one-definition]
212         ==> (y < (S zero) & y /= zero) [augment]
213         ==> (y < zero) [Less.S-step]]
214         (!chain-last
215           [true ==> (~ y < zero) [Less.not-zero]]));
216 C4 := (!chain-last
217       [C3 ==> (~ (S zero) < x) [one-definition]
218         ==> (x <= (S zero)) [Less=.trichotomy2]
219         ==> (x < (S zero) | x = (S zero)) [Less=.definition]]])
220 (!by-contradiction (x = one)
221   assume A := (x /= one)
222   (!absurd
223     (!chain-last
224       [A ==> (C4 & A) [augment]
225         ==> (C4 & x /= (S zero)) [one-definition]
226         ==> (x < (S zero)) [prop-taut]
227         ==> (x /= zero & x < (S zero)) [augment]
228         ==> (x < zero) [Less.S-step]])
229     (!chain-last
230       [true ==> (~ x < zero) [Less.not-zero]])))
231
232 } # Times
233 } # N

```