lib/main/pair.ath

```
datatype (Pair S T) := (pair pair-left:S pair-right:T)

define (alist->pair inner1 inner2) :=
  lambda (L)
    match L {
      [a b] => (pair (inner1 a) (inner2 b))
    | _ => L
    }

module Pair {

set-precedence pair 260

define @ := pair

define [x y z w p p1 p2] :=
       [?x ?y ?z ?w ?p:(Pair 'S 'T) ?p1:(Pair 'S1 'T1)
        ?p2:(Pair 'S2 'T2)]

assert pair-axioms :=
  (datatype-axioms "Pair" joined-with selector-axioms "Pair")

conclude pair-theorem-1 :=
     (forall p . p = (pair-left p) @ (pair-right p))
  datatype-cases pair-theorem-1 {
    (P as (pair x y)) =>
      (!chain [P = ((pair-left P) @ (pair-right P)) [pair-axioms]])
  }

conclude pair-theorem-2 :=
       (forall x y z w . x @ y = z @ w <==> y @ x = w @ z)
  pick-any x y z w
    (!chain [(x @ y = z @ w) <==> (x = z & y = w)    [pair-axioms]
                              <==> (y = w & x = z)     [prop-taut]
                              <==> (y @ x = w @ z)  [pair-axioms]])

declare swap:  (S, T) [(Pair S T)] -> (Pair T S)

assert* swap-def := (swap x @ y = y @ x)

conclude swap-theorem-1 :=
  (forall x y . swap swap x @ y = x @ y)
pick-any x y
  (!chain [(swap swap x @ y) = (swap y @ x) [swap-def]
                             = (x @ y)      [swap-def]])

conclude swap-theorem-1b := (forall p . swap swap p = p)
 pick-any p
  let {E := (!chain-> [true ==> (exists x y . p = x @ y) [pair-axioms]])}
    pick-witnesses x y for E
      (!chain-> [(swap swap x @ y)
                 = (swap y @ x)      [swap-def]
                 = (x @ y)           [swap-def]
              ==> (swap swap p = p)  [(p = x @ y)]])

define (pair-converter premise) :=
  match premise {
    (forall u:'S (forall v:'T body)) =>
      pick-any p:(Pair 'S 'T)
        let {E := (!chain-> [true ==> (exists ?x:'S ?y:'T .
                                       p = ?x @ ?y) [pair-axioms]])}
          pick-witnesses x y for E
            let {body' := (!uspec* premise [x y]);
                 goal := (replace-term-in-sentence (x @ y) body' p)}
#                _ := (print "\nBody: " (val->string body) "\nand goal:\n" (val->string goal))}
              (!chain-> [body' ==> goal  [(p = x @ y)]])
    }
```

```
68
69 define pair-converter-2 :=
70   method (premise)
71     match premise {
72       (exists p:(Pair 'S 'T) body) =>
73         pick-witness pw for premise premise-inst
74           let {lemma := (!chain-> [true ==> (exists ?a ?b . pw = ?a @ ?b) [(datatype-axioms "Pair")]])}
75            pick-witnesses a b for lemma eq-inst
76              let {#_ := (print "\ninst:\n" eq-inst "and premise-inst:\n" premise-inst);
77                   premise-inst' := (replace-var pw (a @ b) premise-inst);
78                   S1 := (!chain-> [premise-inst ==> premise-inst' [eq-inst]])}
79                (!egen* (exists ?a ?b (replace-vars [a b] [?a ?b] premise-inst')) [a b])
80    }
81
82 conclude swap-theorem-1b := (forall p . swap swap p = p)
83   (!pair-converter
84      pick-any x y
85        (!chain [(swap swap x @ y) = (swap y @ x) [swap-def]
86                                   = (x @ y)      [swap-def]]))
87
88 conclude swap-theorem-1b := (forall p . swap swap p = p)
89     (!pair-converter swap-theorem-1)
90
91 } # close module Pair
```