lib/basic/pairs.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")

   define (lst->pair-general x pre-left pre-right) :=
     match x {
       [v1 v2] => (pair (pre-left v1) (pre-right v2))
     | _ => x
     }

   define (lst->pair x) :=  (lst->pair-general x id id)

   define (pair->lst-general x pre-left pre-right) :=
     match x {
       (pair v1 v2) => [(pre-left v1) (pre-right v2)]
     | _ => x
     }

   define (pair->lst x) := (pair->lst-general x id id)

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)]])
    }

define pair-converter-2 :=
  method (premise)
    match premise {
      (exists p:(Pair 'S 'T) body) =>
         pick-witness pw for premise premise-inst
           let {lemma := (!chain-> [true ==> (exists ?a ?b . pw = ?a @ ?b) [(datatype-axioms "Pair")]])}
            pick-witnesses a b for lemma eq-inst
              let {#_ := (print "\ninst:\n" eq-inst "and premise-inst:\n" premise-inst);
                   premise-inst' := (replace-var pw (a @ b) premise-inst);
                   S1 := (!chain-> [premise-inst ==> premise-inst'  [eq-inst]])}
                (!egen* (exists ?a ?b (replace-vars [a b] [?a ?b] premise-inst')) [a b])
    }

conclude swap-theorem-1b := (forall p . swap swap p = p)
  (!pair-converter
     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)
    (!pair-converter swap-theorem-1)

} # close module Pair

open Pair
```