lib/basic/prop-tab.ath

```
define (literal? p) :=
  match p {
    (|| (some-atom _) (~ (some-atom _)) (_ where (quant? p)) ((~ (some-sent q)) where (quant? q))) => true
  | _  => false
  }

define (non-literal? p) := (negate literal? p)


define (conjunctive-case p K) :=
  match p {
    (and (some-list _)) => (!decompose p K)

  | (q <==> r) => let {p1 := (!left-iff p);
                       p2 := (!right-iff p)}
                    (!K [p1 p2])

  | (~ (~ _)) => let {q := (!dn p)}
                   (!K [q])

  | (~ (or (some-list _))) => (!decompose (!dm p) K)

  | (~ (q ==> r)) => let {p1 := conclude q
                                  (!neg-cond1 p);
                          p2 := conclude (~ r)
                                  (!neg-cond2 p)}
                       (!K [p1 p2])
  }

define (disjunctive-case p K) :=
  match p {
    (or (some-list L)) => (!K p L)
  | (q ==> r) => let {q'|r := (!cond-def p)}
                   (!K q'|r [(complement q) r])
  | (~ (and (some-list L))) => (!K (!dm p) (map complement L))
  | (~ (q <==> r)) => (!K (!negated-bicond p) [(q & (complement r))
                                              ((complement q) & r)])
  }

define (inconsistent-literals L) :=
 #let {_ := (print "\nCalling inconsistent-literals on this:\n" L "\n")}
   (!find-some L method (p)
                  match p {
                    false => (!claim false)
                  | (not true) => (!absurd (!true-intro) p)
                  | _ => (!from-complements false p (complement p))
                  }
               fail)

define (refute L) :=
  match L {
    (split L1 [(p where (non-literal? p))] L2) =>
      try { (!conjunctive-case p
               method (components)
                 (!refute (join L1 components L2)))
          | (!disjunctive-case p
              method (disjunction disjuncts)
                (!map-method method (d)
                               assume d
                                 (!refute (join L1 [d] L2))
                             disjuncts
                             method (conds)
                               (!cases disjunction conds)))
          | #let {_ := (print "\nFailed on this non-literal: " (val->string p) "\n")}
            (!fail)
          }
  | _ => (!inconsistent-literals L)
```

```
68    }
69
70  (define (prop-taut p)
71    (!by-contradiction' p
72       (assume (not p)
73         (!refute [(not p)]))))
74
75  (define (prop-taut-core p)
76    (!by-contradiction' p
77       (assume (not p)
78         (!refute [(not p)]))))
79
80  (primitive-method (sat-derive p)
81    (match (sat-solve [(not p)])
82      ('Unsat p)))
83
84  (define (sat-prop-taut p)
85    (!sat-derive p))
86
87  (define (sat-prop-taut-from goal premises)
88     (dlet ((single-premise (dmatch premises
89                              ([] (!true-intro))
90                              ([p] (!claim p))
91                              (_ (!conj-intro premises))))
92            (goal' (if single-premise goal))
93            (conditional (!sat-derive goal')))
94       (!mp conditional single-premise)))
95
96  (define (prop-taut-from goal premises)
97     (dlet ((single-premise (!conj-intro premises))
98            (goal' (if single-premise goal))
99            (conditional (!prop-taut goal')))
100      (!mp conditional single-premise)))
101
102
103 (define (prop-taut premise goal)
104    (dtry (!prove-components-of goal)
105          (!prop-taut-from goal [premise])))
106
107 EOF
```