lib/basic/tp.ath

```
## An implementation of semantic tableaux for first-order logic
## using infinite streams rather than unification. The method
## uses weaving of infinite streams to get a fair and seemingly
## relatively efficient strategy for choosing terms with which
## to instantiate the universal generalizations.

(define (shuffle L k)
  (let (([x L'] (decompose-nth L (random-int k))))
    (add x L')))

(define (multiple-of? a k)
  (equal? (mod a k) 0))

(define (s L) (shuffle L (length L)))

(define (consistent? literals)
  (match literals
    ((split _ (list-of A (split _ (list-of (not A) _)))) false)
    ((split _ (list-of (not A) (split _ (list-of A _)))) false)
    (_ true)))

(define (dual L)
  (match L
    ((some-atom A) (not A))
    ((not (some-atom A)) A)))

(define (make-var n) (string->var (join "a" (symbol->string n))))

(define (safe-uspec ugen t)
  (!uspec ugen t))

(define (safe-uspec ugen t)
    (dcheck ((less? (ref (second ugen)) 8)
             (dlet ((_ (inc (second ugen))))
               (!uspec (first ugen) t)))))

(define (safe-uspec* p terms)
  (dletrec ((loop (method (p terms)
                    (dmatch terms
                      ([] (!claim p))
                      ((list-of t more) (!loop (!uspec p t) more))))))
     (dtry (!loop p terms)
           (!claim p))))

(define (has-equants? p)
  (match p
    ((exists (some-var _) _) true)
    (((some-prop-con _) (some-list props)) (for-some props has-equants?))
    ((forall (some-var _) body) (has-equants? body))
    (_ false)))

(define (sort L)
  (letrec ((loop (lambda (L cprops dprops)
                   (match L
                     ((list-of (bind p (and (some-list _))) rest) (loop rest (add p cprops) dprops))
                     ((list-of (bind p (iff _ _)) rest) (loop rest (add p cprops) dprops))
                     ((list-of (bind p (not (not _))) rest) (loop rest (add p cprops) dprops))
                     ((list-of (bind p (not (or (some-list _)))) rest) (loop rest (add p cprops) dprops))
                     ((list-of (bind p (not (if _ _))) rest) (loop rest (add p cprops) dprops))
                     ((list-of (bind p (forall (some-var _) _)) rest) (loop rest (add p cprops) dprops))
                     ((list-of (bind p (not (forall (some-var _) _))) rest) (loop rest (add p cprops) dprops))
                     ((list-of (bind p (not (exists (some-var _) _))) rest) (loop rest (add p cprops) dprops))
                     ((list-of (bind p (exists (some-var _) _)) rest) (loop rest (add p cprops) dprops))
                     ((list-of (bind p (if _ _)) rest) (loop rest cprops (add p dprops)))
                     ((list-of (bind p (or (some-list _))) rest) (loop rest cprops (add p dprops)))
                     ((list-of (bind p (not (iff _ _))) rest) (loop rest cprops (add p dprops)))
                     ((list-of (bind p (not (and (some-list _)))) rest) (loop rest cprops (add p dprops)))
```

```
68                      ((list-of L rest) (loop rest (add L cprops) dprops))
69                      ([] [(rd cprops) (rd dprops)])))))
70      (loop L [] [])))
71
72
73  (define (refute props terms)
74    (dlet ((i (cell 0))
75           (limit 50000)
76           (writeln-val1 (lambda (x y) ()))
77           (writeln-val1 (lambda (x y)
78                           (match y
79                             ((some-list _) (seq (print (join "\n" x)) (map write y)))
80                             (_ (writeln-val x y)))))
81           (print1 print)
82           (print1 (lambda (x) ()))
83           (continue1 (lambda () ()))
84           (continue1 continue)
85           (_ ()))
86      (dletrec
87        ((M (method (props literals evars terms ugens)
88              (dlet ((_ (print1 "\n================================================================================\n"))
89                     (_ (writeln-val1 "props: " props))
90                     (_ (writeln-val1 "literals: " literals))
91                     (_ (writeln-val1 "ugens: " ugens))
92                     (_ (writeln-val1 "evars: " evars))
93                     (_ (writeln-val1 "terms: " terms))
94                     (_ (continue1))
95                     (_ (check ((greater? (inc i) limit)
96                                (let ((_ (print "\nAbout to call ATP...\n"))
97                                      (_ (writeln-val1 "props: " props))
98                                      (_ (writeln-val1 "ugens: " ugens))
99                                      (_ (writeln-val1 "literals: " literals))
100                                     (_ (continue1))
101                                     (res (!spf false (join props literals ugens)))
102                                     (_ (match res (false (mark 'S)) (_ (mark 'F)))))
103                                  (halt)))
104                           (else ())))
105                    (_ (check ((null? props) (seq (print "\nEmpty prop list, can't close branch...")
106                                                  (writeln-val "literals: " literals)
107                                                  (writeln-val "ugens: " ugens)
108                                                  (writeln-val "evars: " evars)
109                                                  (writeln-val "terms: " terms)
110                                                  (halt)))
111                              (else ())))
112                    (first (first props))
113                    (rest  (tail props)))
114               (dmatch first
115                 ((bind P (and P1 P2)) (!bin-conj-case (!left-and P) (!right-and P) rest literals evars terms ugens))
116                 ((not (not P)) (!unary-conj-case (!dn (not (not P))) rest literals evars terms ugens))
117                 ((bind P (not (or p1 p2))) (dlet ((q (!dm P))
118                                                   (left (!left-and q))
119                                                   (right (!right-and q)))
120                                              (!bin-conj-case left right rest literals evars terms ugens)))
121                 ((bind P (not (if _ _)))
122                   (!bin-conj-case (!neg-cond1 P) (!neg-cond2 P) rest literals evars terms ugens))
123                 ((bind P (iff _ _))
124                   (!bin-conj-case (!left-iff P) (!right-iff P) rest literals evars terms ugens))
125                 ((bind P (or _ _)) (!disj-case P rest literals evars terms ugens))
126                 ((bind P (not (and p1 p2))) (!disj-case (!dm P) rest literals evars terms ugens))
127                 ((bind P (if _ _)) (!disj-case (!cond-def P) rest literals evars terms ugens))
128                 ((bind P (not (iff _ _))) (!disj-case (!neg-bicond P) rest literals evars terms ugens))
129                 ((bind P (forall (list-of _ _) _))
130                   (!map-method (method (t) (!safe-uspec P t))
131                                (join evars terms)
132                                (method (specialized_props)
133                                  (!M (join (rd specialized_props) rest) literals evars terms (add P ugens)))))
134                  ((bind P (exists x Q))
135                   (pick-witness w P
136                     (!map-method (method (ugen) (!safe-uspec ugen w))
137                                  ugens
```

```
                                    (method (specialized_props)
                                       (!M (join [(replace-var x w Q)] specialized_props rest)
                                           literals (add w evars) terms ugens)))))
                  ((bind P (not (forall (list-of _ _) _))) (!unary-conj-case (!qn P) rest literals evars terms ugens))
                  ((bind P  (not (exists (list-of _ _) _))) (!unary-conj-case (!qn P) rest literals evars terms ugens))
                  (L (dlet ((L' (dual L)))
                       (dcheck ((| (member? L' literals) (member? L' rest))
                                   (dlet ((_ (print1 "Closing a branch!!!!\n")))
                                     (!comm-absurd L L')))
                               (else (!M rest (add L literals) evars terms ugens)))))))))
       (bin-conj-case (method (P1 P2 props literals evars terms ugens)
                        (!M (join [P1 P2] props) literals evars terms ugens)))
       (unary-conj-case (method (P props literals evars terms ugens)
                          (!M (add P props) literals evars terms ugens)))
       (disj-case (method (P props literals evars terms ugens)
                    (dmatch P
                      ((or P1 P2) (!cases P (assume P1 (!M (add P1 props) literals evars terms ugens))
                                            (assume P2 (!M (add P2 props) literals evars terms ugens)))))))
      (!M props [] [] terms [])))))



(define (taut p)
  (!by-contradiction' p
     (assume (not p)
       (dlet ((th (!refute [(not p)] (choice-prop-subterms p))))
         (!claim th)))))

(define (refute props terms)
  (dlet ((i (cell 0))
         (limit 50000)
         ([cprops dprops] (sort props))
         (writeln-val1 (lambda (x y)
                         (match y
                           ((some-list _) (seq (print (join "\n" x)) (map write y)))
                           (_ (writeln-val x y)))))
         (writeln-val1 (lambda (x y) ()))
         (print1 print)
         (print1 (lambda (x) ()))
         (continue1 continue)
         (continue1 (lambda () ()))
         (_ ()))
    (dletrec
      ((M (method (cprops dprops literals evars terms ugens)
            (dlet ((_ (print1 "\n================================================================================\n"))
                   (_ (writeln-val1 "cprops: " cprops))
                   (_ (writeln-val1 "dprops: " dprops))
                   (_ (writeln-val1 "literals: " literals))
                   (_ (writeln-val1 "ugens: " ugens))
                   (_ (writeln-val1 "evars: " evars))
                   (_ (writeln-val1 "terms: " terms))
                   (_ (continue1))
                   (_ (check ((greater? (inc i) limit)
                              (let ((_ (print "\nAbout to call ATP...\n"))
                                    (_ (writeln-val1 "props: " props))
                                    (_ (writeln-val1 "ugens: " ugens))
                                    (_ (writeln-val1 "literals: " literals))
                                    (_ (continue1))
                                    (res (!spf false (join props literals ugens)))
                                    (_ (match res (false (mark 'S)) (_ (mark 'F)))))
                                (halt)))
                             (else ())))
                   (_ (check ((null? props) (seq (print "\nEmpty prop list, can't close branch...")
                                                 (writeln-val "literals: " literals)
                                                 (writeln-val "ugens: " ugens)
                                                 (writeln-val "evars: " evars)
                                                 (writeln-val "terms: " terms)
                                                 (halt)))
                             (else ())))))))
```

```
208             (dmatch cprops
209               ((list-of (bind P (and P1 P2)) crest)
210                 (!bin-conj-case (!left-and P) (!right-and P) crest dprops literals evars terms ugens))
211               ((list-of (not (not P)) crest)
212                 (!unary-conj-case (!dn (not (not P))) crest dprops literals evars terms ugens))
213               ((list-of (bind P (not (or p1 p2))) crest)
214                  (dlet ((q (!dm P))
215                         (left (!left-and q))
216                         (right (!right-and q)))
217                    (!bin-conj-case left right crest dprops literals evars terms ugens)))
218               ((list-of (bind P (not (if _ _))) crest)
219                 (!bin-conj-case (!neg-cond1 P) (!neg-cond2 P) crest dprops literals evars terms ugens))
220               ((list-of (bind P (iff _ _)) crest)
221                 (!bin-conj-case (!left-iff P) (!right-iff P) crest dprops literals evars terms ugens))
222               ((list-of (bind P (forall (list-of _ _) _)) crest)
223                 (!map-method (method (t) (!safe-uspec P t))
224                              (join evars terms)
225                              (method (specialized_props)
226                                (dlet (([cprops' dprops'] (sort specialized_props)))
227                                  (!M (join (rd cprops') crest) (join (rd dprops') dprops)
228                                      literals evars terms (add P ugens))))))
229                ((list-of (bind P (exists x Q)) crest)
230                  (pick-witness w P
231                    (!map-method (method (ugen) (!safe-uspec ugen w))
232                                 ugens
233                                 (method (specialized_props)
234                                   (dlet (([cprops' dprops'] (sort (add (replace-var x w Q) specialized_props))))
235                                     (!M (join (rd cprops') crest) (join (rd dprops') dprops)
236                                         literals (add w evars) terms ugens))))))
237                ((list-of (bind P (not (forall (list-of _ _) _))) crest)
238                  (!unary-conj-case (!qn P) crest dprops literals evars terms ugens))
239                ((list-of (bind P  (not (exists (list-of _ _) _))) crest)
240                  (!unary-conj-case (!qn P) crest dprops literals evars terms ugens))
241                ((list-of L crest) (dlet ((L' (dual L)))
242                                     (dcheck ((member? L' literals)
243                                              (dlet ((_ (print1 "Closing a branch!!!!\n")))
244                                                (!comm-absurd L L')))
245                                             (else (!M crest dprops (add L literals) evars terms ugens)))))
246                ([] (dmatch dprops
247                      ((list-of (bind P (or _ _)) drest)
248                        (!disj-case P [] drest literals evars terms ugens))
249                      ((list-of (bind P (not (and p1 p2))) drest)
250                        (!disj-case (!dm P) [] drest literals evars terms ugens))
251                      ((list-of (bind P (if _ _)) drest)
252                        (!disj-case (!cond-def P) [] drest literals evars terms ugens))
253                      ((list-of (bind P (not (iff _ _))) drest)
254                        (!disj-case (!neg-bicond P) [] drest literals evars terms ugens))
255                      ([] (dlet ((fvar (fresh-var)))
256                            (!map-method (method (ugen) (!safe-uspec ugen fvar))
257                                         ugens
258                                         (method (specialized_props)
259                                           (dlet (([cprops' dprops'] (sort specialized_props)))
260                                             (!M (rd cprops') (rd dprops')
261                                                 literals evars (add fvar terms) ugens))))))))))))
262         (bin-conj-case (method (P1 P2 cprops dprops literals evars terms ugens)
263                          (dlet (([cprops' dprops'] (sort [P1 P2])))
264                            (!M (join cprops' cprops) (join dprops' dprops) literals evars terms ugens))))
265         (unary-conj-case (method (P cprops dprops literals evars terms ugens)
266                            (dlet (([cprops' dprops'] (sort [P])))
267                              (!M (join cprops' cprops) (join dprops dprops') literals evars terms ugens))))
268         (disj-case (method (P cprops dprops literals evars terms ugens)
269                      (dmatch P
270                        ((or P1 P2) (!cases P (assume P1
271                                                (dlet (([cprops' dprops'] (sort [P1])))
272                                                  (!M (join cprops' cprops) (join dprops' dprops)
273                                                      literals evars terms ugens)))
274                                              (assume P2
275                                                (dlet (([cprops' dprops'] (sort [P2])))
276                                                  (!M (join cprops' cprops) (join dprops' dprops)
277                                                      literals evars terms ugens)))))))))
```

```
278       (!M cprops dprops [] [] terms [])))
279
280
281
282 (define (show-inconsistent props)
283   (dmatch props
284     ((split _ (list-of p (split _ (list-of (not p) rest)))) (!absurd p (not p)))
285     ((split _ (list-of (not p) (split _ (list-of p rest)))) (!absurd p (not p)))))
286
287
288 (define (refute props terms retry)
289   (dlet ((i (cell 0))
290          (limit 5000)
291          ([cprops dprops] (sort props))
292          (writeln-val1 (lambda (x y)
293                          (match y
294                            ((some-list _) (seq (print (join "\n" x)) (map write y)))
295                            (_ (writeln-val x y)))))
296          (writeln-val1 (lambda (x y) ()))
297          (print1 print)
298          (print2 print)
299          (writeln-val2 writeln-val)
300          (print1 (lambda (x) ()))
301          (continue1 continue)
302          (continue1 (lambda () ()))
303          (_ ()))
304     (dletrec
305       ((M (method (cprops dprops literals evars terms ugens)
306             (dlet ((_ (print1 "\n================================================================================\n"))
307                    (_ (print1 "\nKKKKKKKKKKKKKKKKKKK\n"))
308                    (_ (writeln-val1 "cprops: " cprops))
309                    (_ (writeln-val1 "dprops: " dprops))
310                    (_ (writeln-val1 "literals: " literals))
311                    (_ (writeln-val1 "ugens: " ugens))
312                    (_ (writeln-val1 "evars: " evars))
313                    (_ (writeln-val1 "terms: " terms))
314                    (cprops' (check ((null? cprops) cprops)
315                                    (else (shuffle cprops (length cprops)))))
316                    (cprops  cprops')
317                    (_ (continue1))
318                    (_ ())
319                    (_ (check ((null? props) (seq (print "\nEmpty prop list, can't close branch...")
320                                                  (writeln-val "literals: " literals)
321                                                  (writeln-val "ugens: " ugens)
322                                                  (writeln-val "evars: " evars)
323                                                  (writeln-val "terms: " terms)
324                                                  (halt)))
325                              (else ())))))
326              (dcheck
327               ((greater? (inc i) limit)
328                 (dtry (!show-inconsistent (join cprops dprops literals ugens))
329                       (dlet ((_ (print1 "\nAbout to call ATP...\n"))
330                              (_ (writeln-val1 "cprops: " cprops))
331                              (_ (writeln-val1 "drops: " dprops))
332                              (_ (writeln-val1 "literals: " literals))
333                              (_ (writeln-val1 "ugens: " ugens))
334                              (all-props (join cprops dprops literals ugens))
335                              ([_ all-props'] (filter-and-complement all-props has-equants?))
336                              (_ (writeln-val1 "all-props': " all-props'))
337                              (_ (print2 "\nCalling external ATP...\n"))
338                              (th (dtry (dlet ((th (!spf false all-props'))
339                                               (_  (print2 "\nSUCCESS on NO EXISTENTIALS...\n")))
340                                          (!claim th))
341                                        (dlet ((_ (writeln-val2 "\nATP failed on: " all-props'))
342                                               (th' (dmatch retry
343                                                      (true (!refute all-props (choice-prop-subterms* all-props) false)
344                                                      (_ (!true-intro))))
345                                               (_ (check ((equal? th' true) (print2 "\nRefute failed as well..."))
346                                                         (else (print2 "\nBut refute succeeded!")))))
347                                          (dcheck ((equal? th' true) (!vpf false all-props))
```

```
                                          (else (!claim th'))))))
                          (_ (check ((equal? th false) (print1 "\nSuccess!!\n"))
                                    (else (print1 "\nFailure...\n"))))
                          (_ (continue1)))
                    (!claim th))))
          (else
           (dmatch cprops
             ((list-of (bind P (and P1 P2)) crest)
               (!bin-conj-case (!left-and P) (!right-and P) crest dprops literals evars terms ugens))
             ((list-of (not (not (some-prop P))) crest)
               (!unary-conj-case (!dn (not (not P))) crest dprops literals evars terms ugens))
             ((list-of (bind P (not (or p1 p2))) crest)
                (dlet ((q (!dm P))
                       (left (!left-and q))
                       (right (!right-and q)))
                  (!bin-conj-case left right crest dprops literals evars terms ugens)))
             ((list-of (bind P (not (if _ _))) crest)
               (!bin-conj-case (!neg-cond1 P) (!neg-cond2 P) crest dprops literals evars terms ugens))
             ((list-of (bind P (iff _ _)) crest)
               (!bin-conj-case (!left-iff P) (!right-iff P) crest dprops literals evars terms ugens))
             ((list-of (bind P (forall (list-of _ _) _)) crest)
               (!map-method (method (t) (!safe-uspec P t))
                            (join evars terms)
                            (method (specialized_props)
                              (dlet ((['cprops 'dprops'] (sort specialized_props)))
                                (!M (join (rd cprops') crest) (join (rd dprops') dprops)
                                    literals evars terms (add P ugens))))))
              ((list-of (bind P (exists (some-var x) (some-prop Q))) crest)
                (pick-witness w P
                  (!map-method (method (ugen) (!safe-uspec ugen w))
                               ugens
                               (method (specialized_props)
                                 (dlet ((_ ())
                                        (['cprops' dprops'] (sort (add (replace-var x w Q) specialized_props))))
                                   (!M (join crest (rd cprops')) (join dprops (rd dprops'))
                                       literals (add w evars) terms ugens))))))
             ((list-of (bind P (not (forall (list-of _ _) _))) crest)
               (!unary-conj-case (!qn P) crest dprops literals evars terms ugens))
             ((list-of (bind P  (not (exists (list-of _ _) _))) crest)
               (!unary-conj-case (!qn P) crest dprops literals evars terms ugens))
             ((list-of L crest) (dlet ((L' (dual L)))
                                  (dcheck ((member? L' literals)
                                           (dlet ((_ (print1 "Closing a branch!!!!\n")))
                                             (!comm-absurd L L')))
                                          (else (!M crest dprops (add L literals) evars terms ugens)))))
             ([] (dmatch dprops
                   ((list-of (bind P (or _ _)) drest)
                     (!disj-case P [] drest literals evars terms ugens))
                   ((list-of (bind P (not (and p1 p2))) drest)
                     (!disj-case (!dm P) [] drest literals evars terms ugens))
                   ((list-of (bind P (if _ _)) drest)
                     (!disj-case (!cond-def P) [] drest literals evars terms ugens))
                   ((list-of (bind P (not (iff _ _))) drest)
                     (!disj-case (!neg-bicond P) [] drest literals evars terms ugens))
                   ([] (dlet ((fvar (fresh-var)))
                         (!map-method (method (ugen) (!safe-uspec ugen fvar))
                                      ugens
                                      (method (specialized_props)
                                        (dlet ((['cprops' dprops'] (sort specialized_props)))
                                          (!M (rd cprops') (rd dprops')
                                              literals evars (add fvar terms) ugens))))))))))))))
(bin-conj-case (method (P1 P2 cprops dprops literals evars terms ugens)
                 (dlet ((['cprops' dprops'] (sort [P1 P2])))
                   (!M (join cprops' cprops) (join dprops' dprops) literals evars terms ugens))))
(unary-conj-case (method (P cprops dprops literals evars terms ugens)
                   (dlet ((['cprops' dprops'] (sort [P])))
                     (!M (join cprops' cprops) (join dprops dprops') literals evars terms ugens))))
(disj-case (method (P cprops dprops literals evars terms ugens)
             (dmatch P
               ((or P1 P2) (!cases P (assume P1
```

```
                                                (dlet (([cprops' dprops'] (sort [P1])))
                                                  (!M (join cprops' cprops) (join dprops' dprops)
                                                      literals evars terms ugens)))
                                              (assume P2
                                                (dlet (([cprops' dprops'] (sort [P2])))
                                                  (!M (join cprops' cprops) (join dprops' dprops)
                                                      literals evars terms ugens)))))))))
      (!M cprops dprops [] [] terms [])))))


(define (taut p)
  (!by-contradiction' p
     (assume (not p)
       (dlet ((th (!refute [(not p)] (choice-prop-subterms p) true)))
         (!claim th)))))


(define (rp props)
  (assume (and props)
    (!refute props (fold join (map choice-prop-subterms props) []) true)))


(define (writeln-val1 x y)
  (match y
    ((some-list _) (seq (print (join "\n" x)) (map write y)))
    (_ (writeln-val x y))))


(define (show-state-cont cprops dprops literals evars terms ugens ccount dcount iteration)
            (let ((_ (print "\n================================================================================\n"))
                  (_ (writeln-val1 "Iteration: " iteration))
                  (_ (writeln-val1 "ccount: " ccount))
                  (_ (writeln-val1 "dcount: " dcount))
                  (_ (writeln-val1 "cprops: " cprops))
                  (_ (writeln-val1 "dprops: " dprops))
                  (_ (writeln-val1 "literals: " literals))
                  (_ (writeln-val1 "ugens: " ugens))
                  (_ (writeln-val1 "evars: " evars))
                  (_ (writeln-val1 "terms: " terms)))
              (continue)))

(define (show-state cprops dprops literals evars terms ugens ccount dcount iteration)
            (let ((_ (print "\n================================================================================\n"))
                  (_ (writeln-val1 "Iteration: " iteration))
                  (_ (writeln-val1 "ccount: " ccount))
                  (_ (writeln-val1 "dcount: " dcount))
                  (_ (writeln-val1 "cprops: " cprops))
                  (_ (writeln-val1 "dprops: " dprops))
                  (_ (writeln-val1 "literals: " literals))
                  (_ (writeln-val1 "ugens: " ugens))
                  (_ (writeln-val1 "evars: " evars))
                  (_ (writeln-val1 "terms: " terms)))
              ()))

(define (try-again cprops dprops literals ugens evars terms refute retry)
  (dlet ((ugens' (map first ugens)))
    (dtry (!show-inconsistent (join cprops dprops literals ugens'))
          (dlet ((_ (print "\nAbout to call ATP...\n"))
                 (show-state1 (lambda (cprops dprops literals evars terms ugens a b c) ()))
                 (_ (show-state1 cprops dprops literals evars terms ugens 0 0 0))
                 (all-props (join cprops dprops literals ugens'))
                 ([_ all-props'] (filter-and-complement all-props has-equants?))
                 (_ (writeln-val1 "all-props': " all-props'))
                 (_ (print "\nCalling external ATP...\n"))
                 (th (dtry (dlet ((th (!spf false all-props'))
                                  (_  (print "\nSUCCESS on NO EXISTENTIALS...\n")))
                             (!claim th))
                           (!vpf false all-props)
                           (!true-intro)))
                 (_ (check ((equal? th false) (print "")))
```

```
                          (else (print "\nATP failed on joined props...\n")))))
            (!claim th)))))


(define (refute1 props terms retry)
  (dlet ((psize (prop-size* props))
         (i (cell 0))
         (limit 10000)
         ([cprops dprops] (sort props))
         (print1 print)
         (show-state-cont (lambda (x1 x2 x3 x4 x5 x6 x7 x8 x9) ())))
    (dletrec
      ((do-cprop (method (cprop crest dprops literals evars terms ugens ccount dcount iteration)
                   (dmatch cprop
                     ((bind p (and _ _))
                       (!bin-conj-case (!left-and p) (!right-and p) crest dprops literals evars terms ugens
                                       ccount dcount iteration))
                     ((not (not (some-prop p)))
                       (!unary-conj-case (!dn (not (not p))) crest dprops literals evars terms ugens
                                         ccount dcount iteration))
                     ((bind p (not (or _ _)))
                       (dlet ((q (!dm p))
                              (left (!left-and q))
                              (right (!right-and q)))
                         (!bin-conj-case left right crest dprops literals evars terms ugens
                                         ccount dcount iteration)))
                     ((bind p (not (if _ _)))
                       (!bin-conj-case (!neg-cond1 p) (!neg-cond2 p) crest dprops literals evars terms ugens
                                       ccount dcount iteration))
                     ((bind p (iff _ _))
                       (!bin-conj-case (!left-iff p) (!right-iff p) crest dprops literals evars terms ugens
                                       ccount dcount iteration))
                     ((bind p (forall (bind uvars (list-of _ _)) _))
                       (!map-method-non-strictly
                                    (method (term) (!uspec p term))
                                    (join evars terms)
                                    (method (specialized_props)
                                      (dlet ((cprops' dprops') (sort specialized_props)))
                                        (!M (join cprops' crest) (join  dprops' dprops)
                                            literals evars terms (add [p (cell 0)] ugens)
                                                     (plus (minus ccount 1) (length cprops'))
                                                     (plus dcount (length dprops')) (plus iteration 1))))))
                     ((bind p (exists (some-var x) (some-prop q)))
                       (pick-witness w p
                         (!map-method-non-strictly (method (ugen) (!safe-uspec ugen w))
                                      ugens
                                      (method (specialized_props)
                                        (dlet (([cprops' dprops'] (sort (add (replace-var x w q) specialized_props))))
                                          (!M (join crest cprops') (join dprops (rd dprops'))
                                              literals (add w evars) terms ugens (plus (minus ccount 1) (length cprop
                                              (plus dcount (length dprops')) (plus iteration 1)))))))))
                     ((bind p (not (forall (list-of _ _) _)))
                       (!unary-conj-case (!qn p) crest dprops literals evars terms ugens ccount dcount iteration))
                     ((bind p  (not (exists (list-of _ _) _)))
                       (!unary-conj-case (!qn p) crest dprops literals evars terms ugens ccount dcount iteration))
                     (L (dlet ((L' (dual L)))
                          (dcheck ((member? L' literals)
                                   (dlet ((_ (print1 "Closing a branch!!!!\n")))
                                     (!comm-absurd L L')))
                                  (else (!M crest dprops (add L literals) evars terms ugens (minus ccount 1) dcount
                                            (plus iteration 1)))))))))
       (do-dprop (method (dprop drest cprops literals evars terms ugens ccount dcount iteration)
                   (dmatch dprop
                     ((bind p (or _ _))
                       (!disj-case p cprops drest literals evars terms ugens ccount dcount iteration))
                      ((bind p (if _ _))
                        (!disj-case (!cond-def p) cprops drest literals evars terms ugens
                                    ccount dcount iteration))
                      ((bind p (not (and _ _)))
                        (!disj-case (!dm p) cprops drest literals evars terms ugens ccount dcount iteration))
```

```
                ((bind p (not (iff _ _)))
                  (!disj-case (!neg-bicond p) cprops drest literals evars terms ugens
                              ccount dcount iteration)))))
      (bin-conj-case (method (p1 p2 cprops dprops literals evars terms ugens ccount dcount iteration)
                       (dlet (([cprops' dprops'] (sort [p1 p2]))
                              (ccount' (length cprops'))
                              (dcount' (length dprops')))
                         (!M (join cprops' cprops) (join dprops' dprops) literals evars terms ugens
                             (plus ccount' (minus ccount 1)) (plus dcount dcount') (plus iteration 1)))))
      (unary-conj-case (method (p cprops dprops literals evars terms ugens ccount dcount iteration)
                         (dlet (([cprops' dprops'] (sort [p]))
                                (ccount' (length cprops'))
                                (dcount' (length dprops')))
                           (!M (join cprops' cprops) (join dprops dprops') literals evars terms ugens
                               (plus ccount' (minus ccount 1)) (plus dcount dcount') (plus iteration 1)))))
      (disj-case (method (p cprops dprops literals evars terms ugens ccount dcount iteration)
                   (dmatch p
                     ((or p1 p2) (!cases p (assume p1
                                             (dlet (([cprops' dprops'] (sort [p1]))
                                                    (ccount' (length cprops'))
                                                    (dcount' (length dprops')))
                                               (!M (join cprops' cprops) (join dprops' dprops)
                                                   literals evars terms ugens (plus ccount ccount')
                                                   (plus dcount' (minus dcount 1)) (plus iteration 1))))
                                           (assume p2
                                             (dlet (([cprops' dprops'] (sort [p2]))
                                                    (ccount' (length cprops'))
                                                    (dcount' (length dprops')))
                                               (!M (join cprops' cprops) (join dprops' dprops)
                                                   literals evars terms ugens (plus ccount ccount')
                                                   (plus dcount' (minus dcount 1)) (plus iteration 1)))))))))
      (do-dprops (method (dprops cprops literals evars terms ugens ccount dcount iteration)
                   (dcheck ((multiple-of? iteration 1)
                            (dlet (((list-of dprop drest) (shuffle dprops dcount)))
                              (!do-dprop dprop drest cprops literals evars terms ugens ccount dcount iteration)))
                           (else (!do-dprop (head dprops) (tail dprops) cprops literals evars terms ugens
                                            ccount dcount iteration)))))
      (do-cprops (method (cprops dprops literals evars terms ugens ccount dcount iteration)
                   (dcheck ((multiple-of? iteration 1)
                            (dlet (((list-of cprop crest) (shuffle cprops ccount)))
                              (!do-cprop cprop crest dprops literals evars terms ugens ccount dcount iteration)))
                           (else (!do-cprop (head cprops) (tail cprops) dprops literals evars terms ugens
                                            ccount dcount iteration)))))
      (M (method (cprops dprops literals evars terms ugens ccount dcount iteration)
           (dlet ((_ (show-state-cont cprops dprops literals evars terms ugens ccount dcount iteration))
                  (evar-count (length evars))
                  (stop-cond1 (& (less? psize 11) (greater? evar-count 5) (greater? iteration 50)))
                  (stop-cond2 (& (less? psize 25) (greater? evar-count 10)))
                  (stop-cond3 (& (greater? psize 30) (greater? evar-count 20))))
             (dcheck ((| (greater? (inc i) limit) stop-cond1 stop-cond2 stop-cond3 (greater? iteration 500))
                      (!try-again cprops dprops literals ugens evars terms refute1 retry))
                     (else
                       (dmatch [cprops dprops]
                         ([(list-of _ _) (list-of _ _)]
                            (dcheck ((multiple-of? iteration 5)
                                     (!do-dprops dprops cprops literals evars terms ugens ccount dcount iteration))
                                    (else (!do-cprops cprops dprops literals evars terms ugens ccount dcount iteratio
                         ([_ (list-of dprop drest)]
                           (dcheck ((multiple-of? iteration 1)
                                    (dlet (((list-of dprop' drest') (shuffle dprops dcount)))
                                      (!do-dprop dprop' drest' [] literals evars terms ugens ccount dcount iteration)
                                   (else (!do-dprop dprop drest [] literals evars terms ugens ccount dcount iteration
                         ([(list-of cprop crest) _]
                           (dcheck ((multiple-of? iteration 1)
                                    (dlet (((list-of cprop' crest') (shuffle cprops ccount)))
                                      (!do-cprop cprop' crest' [] literals evars terms ugens ccount dcount iteration)
                                   (else (!do-cprop cprop crest [] literals evars terms ugens ccount dcount iteration
                         ([[] []] (dlet ((fvar (fresh-var)))
                                    (!map-method-non-strictly (method (ugen) (!safe-uspec ugen fvar))
                                                              ugens
```

```
628                                                    (method (specialized_props)
629                                                      (dlet (([cprops' dprops'] (sort specialized_props))
630                                                             (ccount' (length cprops'))
631                                                             (dcount' (length dprops')))
632                                                        (!M cprops' dprops' literals evars (add fvar terms) ugens
633                                                            ccount' dcount' (plus iteration 1)))))))))))))))
634       (!M cprops dprops [] [] terms [] (length cprops) (length dprops) 1))))
635
636
637  (define (taut1 p)
638    (!by-contradiction' p
639       (assume (not p)
640         (dlet ((th (!refute1 [(not p)] (choice-prop-subterms p) true)))
641           (!claim th)))))
642
643  (define taut taut1)
644
645  ## (load-file "tableaux-tests.ath")
```