

lib/basic/msr-tests.ath

```

1 (load-file "msr.ath")
2 (load-file "rewriting.ath")
3
4 (define ars rewrite-search)
5 (define gr get-all-rewrites)
6
7 (define R1 (commutative +))
8 (define R2 (associative +))
9 (define R3 (commutative *))
10 (define R4 (associative *))
11
12 (define rules [R1 R2 R3 R4])
13 (assert rules)
14
15 (define (test-df s t)
16   (let ((t1 (time))
17         (_ (!drs-df s t rules rewrite-one-redex))
18         (t2 (time))
19         (time-spent (minus t2 t1)))
20     (println (join "Time: " (val->string time-spent) " seconds"))))
21
22 (define (test-rf s t)
23   (let ((t1 (time))
24         (_ (!drs-rf s t rules rewrite-one-redex))
25         (t2 (time))
26         (time-spent (minus t2 t1)))
27     (println (join "Time: " (val->string time-spent) " seconds"))))
28
29 (define (test-bf s t)
30   (let ((t1 (time))
31         (_ (!drs-bf s t rules rewrite-one-redex))
32         (t2 (time))
33         (time-spent (minus t2 t1)))
34     (println (join "Time: " (val->string time-spent) " seconds"))))
35
36
37 (define (test-pairs L)
38   (let ((failures (cell 0))
39         (sum (cell 0))
40         (pair-num (length L))
41         (_ (print "\nHave " pair-num " equations to test...\n"))
42         (index (cell 0))
43         (test-bf (lambda (s t)
44                     (let ((t1 (time))
45                           (_ (set! index (plus (ref index) 1)))
46                           (res (dtry (!drs-bf s t rules rewrite-one-redex) (!true-intro)))
47                           (t2 (time))
48                           (time-spent (minus t2 t1)))
49                       (match res
50                        (true (seq (print "\nFAILED test #" (ref index) "\n")
51                                  (set! failures (plus (ref failures) 1))))
52                        (_ (seq (print "\nFinished test #" (ref index) " in " time-spent " seconds...\n")
53                               (set! sum (plus (ref sum) time-spent)))))))
54         (_ (map-proc (lambda (pair) (match pair ([s t] (test-bf s t))) L))
55             (successes (minus pair-num (ref failures))
56                       (av-time (div (ref sum) successes)))
57             (print "\nManaged all except for " (ref failures) " tests.\nAverage proof time: " av-time "\n"))))
58
59
60 (define (dtest-pairs L)
61   (let ((failures (cell 0))
62         (sum (cell 0))
63         (pair-num (length L))
64         (_ (print "\nHave " pair-num " equations to test...\n"))
65         (index (cell 0))
66         (test-bf (lambda (s t)
67                     (let ((t1 (time))
68                           (_ (set! index (plus (ref index) 1)))

```

```

69         (res (dtry (!drs-df s t rules rewrite-one-redex) (!true-intro)))
70         (t2 (time))
71         (time-spent (minus t2 t1)))
72     (match res
73       (true (seq (print "\nFAILED test #" (ref index) "\n")
74                  (set! failures (plus (ref failures) 1))))
75       (_ (seq (print "\nFinished test #" (ref index) " in " time-spent " seconds...\n")
76                (set! sum (plus (ref sum) time-spent))))))
77     (_ (map-proc (lambda (pair) (match pair ([s t] (test-bf s t))) L)
78                 (successes (minus pair-num (ref failures)))
79                 (av-time (div (ref sum) successes)))
80     (print "\nManaged all except for " (ref failures) " tests.\nAverage proof time: " av-time "\n"))
81
82 (define (btest-pairs L)
83   (let ((failures (cell 0))
84         (sum (cell 0))
85         (pair-num (length L))
86         (_ (print "\nHave " pair-num " equations to test...\n")
87            (index (cell 0))
88            (test-bf (lambda (s t)
89                      (let ((t1 (time))
90                            (_ (set! index (plus (ref index) 1)))
91                                (res (dtry (!drs-rf s t rules rewrite-one-redex) (!true-intro)))
92                                (t2 (time))
93                                (time-spent (minus t2 t1)))
94                        (match res
95                          (true (seq (print "\nFAILED test #" (ref index) "\n")
96                                       (set! failures (plus (ref failures) 1))))
97                          (_ (seq (print "\nFinished test #" (ref index) " in " time-spent " seconds...\n")
98                                   (set! sum (plus (ref sum) time-spent))))))
99                      (_ (map-proc (lambda (pair) (match pair ([s t] (test-bf s t))) L)
100                                  (successes (minus pair-num (ref failures)))
101                                  (av-time (div (ref sum) successes)))
102                      (print "\nManaged all except for " (ref failures) " tests.\nAverage proof time: " av-time "\n"))))
103
104
105 (define (test s t)
106   (let ((res (ars s t [R1 R2] "depth-first" 50000 "silent")))
107     res))
108
109 #####
110 #                               Some tests:
111 #####
112
113 (define s1 (+ ?a (+ ?b ?c)))
114 (define t1 (+ ?a (+ ?c ?b)))
115 (define t2 (+ (+ ?b ?c) ?a))
116 (define t3 (+ (+ ?a ?b) ?c))
117 (define t4 (+ (+ ?b ?a) ?c))
118 (define t5 (+ ?c (+ ?b ?a)))
119
120 (rewrite-search s1 t1 rules 'best-first)
121
122 (define t6 (+ (+ ?c ?a) ?b))
123
124
125 # (gr s1 rules)
126
127 # (load-file "msr-tests.ath")
128
129
130
131 (test-bf s1 t1)
132 (test-bf s1 t1)
133
134 (test-bf s1 t2)
135
136
137 (test-bf s1 t3)
138

```

```

139
140 (test-bf s1 t4)
141
142 (test-bf s1 t5)
143
144 (test-bf s1 t5)
145
146
147 (define A (+ ?g (+ ?e (+ (+ ?a ?b) (+ ?d ?c)))))
148 (define B (+ (+ ?c (+ ?d (+ ?e (+ ?b ?a)))) ?g))
149 (define C (+ ?a (+ ?b (+ ?c (+ ?d (+ ?e ?g)))))
150 (define D
151   (+ ?g
152     (+ ?a
153       (+ ?e
154         (+ (+ ?c ?b)
155           ?d))))))
156
157 (define E (+ (+ (+ ?g (+ ?c ?e)) (+ ?b ?a)) ?d))
158
159
160 (define (test-bf a b) ())
161
162 (test-bf A B)
163 (test-bf B A)
164
165 (test-bf B C)
166 (test-bf C B)
167
168 (test-bf A C)
169
170 (test-bf C A)
171
172 (test-bf A D)
173 (test-bf D A)
174
175 (test-bf B D)
176 (test-bf D B)
177
178 (test-bf C D)
179 (test-bf D C)
180
181
182 (define A1 (+ ?e (+ (+ ?a ?b) (+ ?d ?c))))
183 (define B1 (+ ?c (+ ?d (+ ?e (+ ?b ?a)))))
184
185 (test-bf A1 B1)
186 (test-bf B1 A1)
187
188
189 # Best-first search with some randomness succeeds in interconverting any two of
190 # the terms A, B, C, and D below (on the basis of the comm/assoc properties of + and *).
191 # Depth-first search fails on all of them with a max depth of 4000:
192
193 (test-bf B C)
194 (test-bf C B)
195
196 (test-bf A C)
197 (test-bf A C)
198
199 (test-bf A D)
200 (test-bf D A)
201
202 (test-bf B D)
203 (test-bf D B)
204
205 (test-bf C D)
206 (test-bf D C)
207
208 (test-bf D A)

```

```

209 (test-bf D A)
210
211 (test-bf B D)
212 (test-bf D B)
213
214 (test-bf D C)
215
216 (test-bf A E)
217 (test-bf E A)
218
219 (test-bf B E)
220 (test-bf E B)
221 (test-bf C E)
222 (test-bf E C)
223
224
225 (define E1 (+ ?c (+ ?b (+ ?a (+ ?d (+ ?g ?e))))))
226
227 (test-bf E E1)
228 (test-bf E1 E)
229 (test-bf E1 A)
230 (test-bf E1 C)
231 (test-bf D E1)
232
233
234 (distance D C)
235 (distance E1 C)
236
237 (define a (+ (+ ?a ?b) (+ ?c ?d)))
238 (define b1 (+ ?d (+ ?c (+ ?b ?a))))
239 (define b2 (+ ?c (+ ?d (+ ?b ?a))))
240 (define b3 (+ ?b (+ ?d (+ ?c ?a))))
241 (define b4 (+ (+ (+ ?d ?c) ?b) ?a))
242 (define b5 (+ (+ (+ ?d ?c) ?a) ?a))
243 (define start1 (+ (+ (+ ?a ?b) ?c) (+ ?d ?e)))
244 (define finish1 (+ ?e (+ ?d (+ ?c (+ ?b ?a)))))
245 (define start2 (+ ?foo (+ (+ (+ ?a ?b) ?c) (+ ?d ?e))))
246 (define finish2 (+ (+ ?e (+ ?d (+ ?c (+ ?b ?a)))) ?foo))
247 (define finish2a (+ ?e (+ ?d (+ ?c (+ ?b (+ ?foo ?a))))))
248
249 (define start3 (+ (+ ?foo ?goo) (+ (+ (+ ?a ?b) ?c) (+ ?d ?e))))
250 (define finish3 (+ (+ ?e (+ ?d (+ ?c (+ ?b ?a)))) (+ ?goo ?foo)))
251 (define finish4 (+ ?e (+ ?d (+ ?c (+ ?b (+ ?goo (+ ?a ?foo))))))
252
253 (test-bf a b1)
254 (test-bf a b1)
255 (test-bf a b2)
256 (test-bf a b3)
257 (test-bf a b4)
258
259 (test-bf start1 finish1)
260 (test-bf finish1 start1)
261
262 (test-bf start2 finish2)
263 (test-bf finish2 start2)
264
265 (test-bf start2 finish2a)
266 (test-bf finish2a start2)
267
268 (test-bf start3 finish3)
269 (test-bf finish3 start3)
270
271 (test-bf start3 finish4)
272
273 (test-bf finish4 start3)
274
275 (define X1 (+ finish4 start3))
276 (define Y1 (+ start3 finish3))
277 (define X2 (+ start2 start3))
278 (define Y2 (+ finish3 finish2))

```

```

279
280 (test-bf X1 Y1)
281
282 (test-bf X2 Y2)
283
284
285
286 (define s (+ (+ finish4 start3) (+ start2 start3)))
287 (define f (+ (+ start3 finish3) (+ finish3 finish2)))
288 (test-bf s f)
289
290 (define S1 (+ (* ?a ?b)
291               (+ (+ ?A (* ?B ?C))
292                 (* ?e (* ?d ?f)))))
293
294 (define S2 (+ (+ (* (* ?f ?d) ?e) (+ ?A (* ?C ?B)))
295               (* ?b ?a)))
296
297 (define S3 (+ (+ (+ (* ?C ?B) ?A)
298                 (* (* ?d ?e) ?f))
299               (* ?b ?a)))
300
301
302 (define pairs [[A B] [B A] [B C] [C B] [A C] [C A] [A D] [D A] [B D] [D B] [C D] [D C] [A E] [E A] [B E] [E B] [C E]
303               [E E] [E1 E] [E1 A] [E1 C] [D E1]
304               [a b1] [a b2] [a b3] [a b4] [start1 finish1] [finish1 start1] [start2 finish2] [finish2 start2]
305               [start2 finish2a] [finish2a start2] [start3 finish3] [finish3 start3] [start3 finish4] [finish4 start3]
306               [X1 Y1] [X2 Y2] [S1 S2] [S2 S1] [S1 S3] [S3 S1] [S2 S3] [S3 S2]])
307
308 # Can't do [s f]
309
310
311
312 (define (t) (test-pairs pairs))
313 EOF
314 (load-file "msr-tests.ath")
315
316 (test-pairs pairs)
317
318 (dtest-pairs pairs)
319 (btest-pairs pairs)
320
321 (test-bf S1 S2)
322
323 (test-bf S2 S1)
324 (test-bf S2 S1)
325
326 (test-bf S1 S3)
327 (test-bf S3 S1)
328
329 (test-bf S2 S3)
330 (test-bf S3 S2)
331
332 (define S4 (+ (+ (+ (+ ?A1 (+ ?A2 (* ?C ?B))) ?A)
333                 (* (* ?d ?e) ?f))
334               (* ?b ?a)))
335
336
337 (define S5 (+ (+ (* (* ?f ?d) ?e) (+ (+ ?A1 (+ ?A2 (* ?C ?B))) ?A))
338               (* ?b ?a)))
339
340 (test-bf S4 S5)
341 (test-bf S5 S4)
342
343 (define S6 (+ S4 S5))
344 (define S7 (+ S5 S4))
345
346 (test-bf S6 S7)
347 (test-bf S7 S6)
348

```

```

349 (define S8 (+ S6 (+ S5 S4)))
350 (define S9 (+ (+ S4 S5) S6))
351
352 (test-bf S8 S9)
353
354 #####
355
356 (load-file "rewriting.ath")
357
358 (domain Z)
359 (datatype Nat zero (succ Nat))
360 (declare Minus (-> (Nat Nat) Nat))
361 (declare Zzero Z)
362 (declare neg (-> (Nat) Z))
363
364 (define p1 (forall ?x:Nat (= (Minus ?x ?x) zero)))
365 (define p2 (= Zzero (neg zero)))
366 (assert p1 p2)
367
368 (define a ?a:Nat)
369 (define s (neg (Minus a a)))
370 (define t Zzero)
371
372 (define rules [p1 p2])
373
374 (!drs-bf s t rules rewrite-one-redex)
375
376 #####
377 ## D: Added 1/21/2009 to document an apparent bug:
378
379 # Uncomment the following if starting here:
380 #
381
382 (load-file "rewriting.ath")
383 (datatype Nat zero (succ Nat))
384
385 (domain E)
386
387 (declare T (-> (E) Nat))
388
389 (declare (s1 s2) E)
390
391 (declare t Nat)
392
393 (define p1 ((T s1) = t))
394
395 (define p2 ((T s2) = (succ t)))
396
397 (assert p1 p2)
398
399 # This fails
400
401 (!drs-bf (T s2) (succ (T s1)) [p1 p2] rewrite-one-redex)
402
403 # This works
404 (!drs-bf (succ (T s1)) (T s2) [p1 p2] rewrite-one-redex)
405
406 # This works:
407 (!chain [(T s2)
408          = (succ t)                [p2]
409          = (succ (T s1))           [p1]])
410
411 # So does this, since chain tries both calls of drs-bf:
412
413 (!chain [(T s2)
414          = (succ (T s1))           [p1 p2]])
415
416 # But if we do essentially the same thing with a binary symbol T1
417 # instead of T, it fails:
418

```

```

419 (domain C)
420
421 (declare T1 (-> (C E) Nat))
422
423 (declare C1 C)
424
425 (declare (x1 x2) E)
426
427 (declare t1 Nat)
428
429 (define p1 ((T1 C1 x1) = t1))
430
431 (define p2 ((T1 C1 x2) = (succ t1)))
432
433 (assert p1 p2)
434
435 # This fails:
436 (!drs-bf (T1 C1 x2) (succ (T1 C1 x1)) [p1 p2] rewrite-one-redex)
437
438 # and so does this:
439 (!drs-bf (succ (T1 C1 x1)) (T1 C1 x2) [p1 p2] rewrite-one-redex)
440
441 # So, although this works:
442 (!chain [(T1 C1 x2)
443          = (succ t1) [p2]
444          = (succ (T1 C1 x1)) [p1]])
445
446 # the following fails (calls external theorem prover)
447 # since both drs-bf calls fail:
448
449 (!chain [(T1 C1 x2)
450          = (succ (T1 C1 x1)) [p1 p2]])
451
452 # This works:
453
454 (define p3 (!sym p1))
455
456 (!drs-bf (T1 C1 x2) (succ (T1 C1 x1)) [p3 p2] rewrite-one-redex)
457
458 # hence this does too:
459 (!chain [(T1 C1 x2)
460          = (succ (T1 C1 x1)) [p3 p2]])

```