Joseph Morgan

Homework 4

CISP440

1 *Source Code*

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

char Universe[8][10] = { "Bat", "Cat", "Chimp", "Dog", "Fish", "Liger", "Snake", "
    Turtle" };
typedef unsigned char set;

char BigUniverse[32][20] = {
  "Bat", "Cat", "Chimp", "Dog", "Fish", "Liger", "Snake", "Turtle",
  "Bear", "Dragon", "Horse", "Wolf", "Rat", "Gerbil", "Rabbit", "Monkey",
  "Donkey", "Llama", "Zebra", "Hippopotamus", "Rhiceros", "Gecko", "Frog", "Sloth",
  "Deer", "Kangaroo", "Gorilla", "Alligator", "Panda", "Squirrel", "Duck", "Platypus
      " };
typedef unsigned long int set32;

void printSet(set);
void printSet(set32);
void print8bits(unsigned char);
void print16bits(unsigned long int);
void insert(set&, char[]);
unsigned long hash_slinger (char[], int[]);
void insert(set32&, char[]);
int my_pow(int, int);
set Union(set, set);
set32 Union(set32, set32);
set Intersection(set, set);
set32 Intersection(set32, set32);
set Complement(set);
set32 Complement(set32);
set Difference(set, set);
set32 Difference(set32, set32);
int Cardinality(set);
int Cardinality(set32);
void printPowerSet(set);
void printPowerSet(set32);
bool IsSubset(set, set);
bool IsSubset(set32, set32);
bool IsProperSubset(set, set);
bool IsProperSubset(set32, set32);
void test_operations(set, set, set);
void test_operations(set32, set32, set32);

int main(void)
{
  set A = 0, B = 0, C = 0;

  // Fill A
  insert(A, "Cat");
  insert(A, "Dog");
  insert(A, "Fish");

  // Fill B
  insert(B, "Cat");
```

```
53    insert(B, "Dog");
54    insert(B, "Liger");
55
56    // Fill C
57    insert(C, "Dog");
58    insert(C, "Liger");
59    insert(C, "Snake");
60    insert(C, "Turtle");
61
62    printf("8 Bit Universe Operations: \n");
63    printf("**********************************************\n");
64    printf("Set A: ");
65    printSet(A);
66    printf("\nSet B: ");
67    printSet(B);
68    printf("\nSet C: ");
69    printSet(C);
70    printf("\n**********************************************\n");
71    printf("\n");
72
73    test_operations(A, B, C);
74
75    A = 0;
76    B = 0;
77    C = 0;
78
79    // Fill A
80    insert(A, "Bat");
81    insert(A, "Chimp");
82    insert(A, "Liger");
83    insert(A, "Snake");
84    insert(A, "Turtle");
85
86    // Fill B
87    insert(B, "Bat");
88    insert(B, "Cat");
89    insert(B, "Chimp");
90    insert(B, "Dog");
91    insert(B, "Fish");
92
93    // Fill C
94    insert(C, "Dog");
95    insert(C, "Fish");
96    insert(C, "Liger");
97    insert(C, "Snake");
98    insert(C, "Turtle");
99
100   printf("\n\n**********************************************\n");
101   printf("Set A: ");
102   printSet(A);
103   printf("\nSet B: ");
104   printSet(B);
105   printf("\nSet C: ");
106   printSet(C);
107   printf("\n**********************************************\n");
108   printf("\n");
109
```

```
110     test_operations(A, B, C);
111
112     printf("\n\n32 Bit Universe Operations: \n");
113     set32 X = 0, Y = 0, Z = 0;
114
115     // Fill X
116     insert(X, "Bat");
117     insert(X, "Dragon");
118     insert(X, "Hippopotamus");
119     insert(X, "Gecko");
120     insert(X, "Sloth");
121     insert(X, "Deer");
122     insert(X, "Kangaroo");
123
124     // Fill Y
125     insert(Y, "Hippopotamus");
126     insert(Y, "Gecko");
127     insert(Y, "Sloth");
128     insert(Y, "Bat");
129     insert(Y, "Rhinoceros");
130     insert(Y, "Squirrel");
131     insert(Y, "Platypus");
132
133     // Fill Z
134     insert(Z, "Gecko");
135     insert(Z, "Sloth");
136     insert(Z, "Bat");
137     insert(Z, "Rhinoceros");
138     insert(Z, "Dog");
139     insert(Z, "Fish");
140     insert(Z, "Horse");
141     insert(Z, "Snake");
142     insert(Z, "Turtle");
143     insert(Z, "Donkey");
144     insert(Z, "Gorilla");
145     insert(Z, "Llama");
146
147     printf("************************************************\n");
148     printf("Set A: ");
149     printSet(X);
150     printf("\nSet B: ");
151     printSet(Y);
152     printf("\nSet C: ");
153     printSet(Z);
154     printf("\n************************************************\n");
155     printf("\n");
156
157     test_operations(X, Y, Z);
158 }
159
160 void printSet(set A)
161 {
162     printf("{ ");
163
164     bool commaflag = false;
165     int i = 0;
166     unsigned char mask = 0x80;
```

```
167   for (; mask; mask >>= 1, i++)
168   {
169     if (mask & A)
170     {
171       if (commaflag) printf(", ");
172       printf("%s", Universe[i]);
173       commaflag = true;
174     }
175   }
176   printf(" }");
177 }
178
179 void printSet(set32 A)
180 {
181   printf("{ ");
182
183   bool commaflag = false;
184   int i = 0;
185   unsigned long int mask = 0x80000000;
186   for (; mask; mask >>= 1, i++)
187   {
188     if (mask & A)
189     {
190       if (commaflag) printf(", ");
191       printf("%s", BigUniverse[i]);
192       commaflag = true;
193     }
194   }
195   printf(" }");
196 }
197
198 void print8bits(unsigned char x)
199 {
200   for (unsigned char mask = 0x80; mask; mask >>= 1) {
201     if (mask & x)
202       printf("1");
203     else
204       printf("0");
205   }
206 }
207
208 void print32bits(unsigned long int x)
209 {
210   for (unsigned long int mask = 0x80000000; mask; mask >>= 1) {
211     if (mask & x)
212       printf("1");
213     else
214       printf("0");
215   }
216 }
217
218 void insert(set& A, char str[])
219 {
220   int hash = (str[0] + str[2]) % 20;
221
222   int g[20] = {6, -1, 0, 1, -1, 4, -1, -1, -1, -1, -1, 3, 2, -1, -1, -1, -1, -1, 7,
      5};
```

```
223
224   int index = g[hash];
225
226   set mask = 0x80 >> index;
227
228   A = A | mask;
229 }
230
231 unsigned long hash_slinger (char str[])
232 { // cjb2 hashing algorithm
233   unsigned long hash = 5281;
234   int c;
235   int magic_mod = 186; // After some testing, found to be the lowest modulus that
236                        // prevented collision.
237
238   while ((c = *str++))\
239   {
240     hash = ((hash << 5) + hash) + c; /* hash * 33 + c */
241   }
242
243   return (hash % magic_mod);
244 }
245
246 void tabler(int tbl[])
247 {
248   tbl[104] = 0;
249   tbl[77] = 1;
250   tbl[154] = 2;
251   tbl[127] = 3;
252   tbl[95] = 4;
253   tbl[36] = 5;
254   tbl[101] = 6;
255   tbl[137] = 7;
256   tbl[21] = 8;
257   tbl[86] = 9;
258   tbl[110] = 10;
259   tbl[177] = 11;
260   tbl[44] = 12;
261   tbl[138] = 13;
262   tbl[89] = 14;
263   tbl[34] = 15;
264   tbl[25] = 16;
265   tbl[4] = 17;
266   tbl[31] = 18;
267   tbl[10] = 19;
268   tbl[70] = 20;
269   tbl[72] = 21;
270   tbl[91] = 22;
271   tbl[131] = 23;
272   tbl[45] = 24;
273   tbl[123] = 25;
274   tbl[79] = 26;
275   tbl[54] = 27;
276   tbl[169] = 28;
277   tbl[60] = 29;
278   tbl[98] = 30;
279   tbl[157] = 31;
```

```
280 }
281
282 void insert(set32& A, char str[])
283 {
284   int hash = hash_slinger(str);
285
286   int g[186] = {-1};
287   tabler(g);
288   int index = g[hash];
289
290   set32 mask = 0x80000000 >> index;
291
292   A = A | mask;
293 }
294
295 int my_pow(int base, int  exp)
296 {
297   int x = 1;
298   for (int i = 0; i < exp; i++)
299     x *= base;
300
301   return x;
302 }
303
304 set Union(set A, set B)
305 {
306   return (A | B);
307 }
308
309 set32 Union(set32 A, set32 B)
310 {
311   return (A | B);
312 }
313
314 set Intersection(set A, set B)
315 {
316   return (A & B);
317 }
318
319 set32 Intersection(set32 A, set32 B)
320 {
321   return (A & B);
322 }
323
324 set Complement(set A)
325 {
326   return ~A;
327 }
328
329 set32 Complement(set32 A)
330 {
331   return ~A;
332 }
333
334 set Difference(set A, set B)
335 {
336   return (A & (~B));
```

```
337 }
338
339 set32 Difference(set32 A, set32 B)
340 {
341   return (A & (~B));
342 }
343
344 int Cardinality(set A)
345 {
346   set mask = 0x01;
347   unsigned int count = 0;
348
349   while (mask) {
350     if (A & mask) ++count;
351     mask <<= 1;
352   }
353
354   return count;
355 }
356
357 int Cardinality(set32 A)
358 {
359   set32 mask = 0x01;
360   unsigned int count = 0;
361
362   while (mask) {
363     if (A & mask) ++count;
364     mask <<= 1;
365   }
366
367   return count;
368 }
369
370 void printPowerSet(set A)
371 {
372   int CardOfP = my_pow(2, Cardinality(A));
373   set setB = 1;
374   set subA = 0;
375
376   for (; setB < CardOfP; ++setB)
377   {
378     unsigned char Amask = 0x01;
379     unsigned char Bmask = 0x01;
380
381     for (; Amask > 0; Amask <<= 1)
382     {
383       if (Amask & A)
384       {
385         if (Bmask & setB)
386         {
387           subA |= Amask;
388         }
389         Bmask <<= 1;
390       }
391       // Amask shifts left
392     }
393     printSet(subA);
```

```
394       printf("\n");
395       subA = 0;
396     }
397   }
398
399   void printPowerSet(set32 A)
400   {
401     int CardOfP = my_pow(2, Cardinality(A));
402     set32 setB = 1;
403     set32 subA = 0;
404
405     for (; setB < CardOfP; ++setB)
406     {
407       unsigned long int Amask = 0x01;
408       unsigned long int Bmask = 0x01;
409
410       for (; Amask > 0; Amask <<= 1)
411       {
412         if (Amask & A)
413         {
414           if (Bmask & setB)
415           {
416             subA |= Amask;
417           }
418           Bmask <<= 1;
419         }
420         // Amask shifts left
421       }
422       printSet(subA);
423       printf("\n");
424       subA = 0;
425     }
426   }
427
428   bool IsSubset(set ASubset, set ASet)
429   {
430     return ((ASubset & ASet) == ASubset);
431   }
432
433   bool IsSubset(set32 ASubset, set32 ASet)
434   {
435     return ((ASubset & ASet) == ASubset);
436   }
437
438   bool IsProperSubset(set ASubset, set ASet)
439   {
440     return ((ASet != ASubset) && (ASubset != 0) && ((ASubset & ASet) == ASubset));
441   }
442
443   bool IsProperSubset(set32 ASubset, set32 ASet)
444   {
445     return ((ASet != ASubset) && (ASubset != 0) && ((ASubset & ASet) == ASubset));
446   }
447
448   void test_operations(set A, set B, set C)
449   {
450     set R = 0;
```

```
451
452     printf("(A U B) ^ C: ");
453     R = Intersection(Union(A, B), C);
454     printSet(R);
455     printf("\n\n");
456
457     printf("A U (B ^ C): ");
458     R = Union(A, Intersection(B, C));
459     printSet(R);
460     printf("\n\n");
461
462     printf("~(A ^ B): ");
463     R = Complement(Intersection(A, B));
464     printSet(R);
465     printf("\n\n");
466
467     printf("(~A U ~B): ");
468     R = Union(Complement(A), Complement(B));
469     printSet(R);
470     printf("\n\n");
471
472     printf("A - B: ");
473     R = Difference(A, B);
474     printSet(R);
475     printf("\n\n");
476
477     printf("Powerset A: \n\n");
478     printPowerSet(A);
479
480     printf("\nA is a proper subset of B: ");
481     IsProperSubset(A, B) ? printf("True\n\n") : printf("False\n\n");
482
483     printf("A is a subset of B: ");
484     IsSubset(A, B) ? printf("True\n\n") : printf("False\n\n");
485
486     printf("(~C U A) ^ B: ");
487     R = Intersection(Union(Complement(C), A), B);
488     printSet(R);
489     printf("\n\n");
490
491     printf("(A ^ B) is a Proper Subset of B: ");
492     IsProperSubset(Intersection(A, B), B) ? printf("True\n\n") : printf("False\n\n");
493 }
494
495 void test_operations(set32 A, set32 B, set32 C)
496 {
497     set32 R = 0;
498
499     printf("(A U B) ^ C: ");
500     R = Intersection(Union(A, B), C);
501     printSet(R);
502     printf("\n\n");
503
504     printf("A U (B ^ C): ");
505     R = Union(A, Intersection(B, C));
506     printSet(R);
507     printf("\n\n");
```

```
508
509     printf("~(A ^ B): ");
510     R = Complement(Intersection(A, B));
511     printSet(R);
512     printf("\n\n");
513
514     printf("(~A U ~B): ");
515     R = Union(Complement(A), Complement(B));
516     printSet(R);
517     printf("\n\n");
518
519     printf("A - B: ");
520     R = Difference(A, B);
521     printSet(R);
522     printf("\n\n");
523
524     printf("Powerset A: \n\n");
525     printPowerSet(A);
526
527     printf("\nA is a proper subset of B: ");
528     IsProperSubset(A, B) ? printf("True\n\n") : printf("False\n\n");
529
530     printf("A is a subset of B: ");
531     IsSubset(A, B) ? printf("True\n\n") : printf("False\n\n");
532
533     printf("(~C U A) ^ B: ");
534     R = Intersection(Union(Complement(C), A), B);
535     printSet(R);
536     printf("\n\n");
537
538     printf("(A ^ B) is a Proper Subset of B: ");
539     IsProperSubset(Intersection(A, B), B) ? printf("True\n\n") : printf("False\n\n");
540 }
```

2 *Output*

```
8 Bit Universe Operations:
*********************************************
Set A: { Cat, Dog, Fish }
Set B: { Cat, Dog, Liger }
Set C: { Dog, Liger, Snake, Turtle }
*********************************************

(A U B) ^ C: { Dog, Liger }

A U (B ^ C): { Cat, Dog, Fish, Liger }

~(A ^ B): { Bat, Chimp, Fish, Liger, Snake, Turtle }

(~A U ~B): { Bat, Chimp, Fish, Liger, Snake, Turtle }

A - B: { Fish }

Powerset A:

{ Fish }
```

```
{ Dog }
{ Dog, Fish }
{ Cat }
{ Cat, Fish }
{ Cat, Dog }
{ Cat, Dog, Fish }

A is a proper subset of B: False

A is a subset of B: False

(~C U A) ^ B: { Cat, Dog }

(A ^ B) is a Proper Subset of B: True



**********************************************
Set A: { Bat, Chimp, Liger, Snake, Turtle }
Set B: { Bat, Cat, Chimp, Dog, Fish }
Set C: { Dog, Fish, Liger, Snake, Turtle }
**********************************************

(A U B) ^ C: { Dog, Fish, Liger, Snake, Turtle }

A U (B ^ C): { Bat, Chimp, Dog, Fish, Liger, Snake, Turtle }

~(A ^ B): { Cat, Dog, Fish, Liger, Snake, Turtle }

(~A U ~B): { Cat, Dog, Fish, Liger, Snake, Turtle }

A - B: { Liger, Snake, Turtle }

Powerset A:

{ Turtle }
{ Snake }
{ Snake, Turtle }
{ Liger }
{ Liger, Turtle }
{ Liger, Snake }
{ Liger, Snake, Turtle }
{ Chimp }
{ Chimp, Turtle }
{ Chimp, Snake }
{ Chimp, Snake, Turtle }
{ Chimp, Liger }
{ Chimp, Liger, Turtle }
{ Chimp, Liger, Snake }
{ Chimp, Liger, Snake, Turtle }
{ Bat }
{ Bat, Turtle }
{ Bat, Snake }
{ Bat, Snake, Turtle }
{ Bat, Liger }
{ Bat, Liger, Turtle }
{ Bat, Liger, Snake }
```

```
{ Bat, Liger, Snake, Turtle }
{ Bat, Chimp }
{ Bat, Chimp, Turtle }
{ Bat, Chimp, Snake }
{ Bat, Chimp, Snake, Turtle }
{ Bat, Chimp, Liger }
{ Bat, Chimp, Liger, Turtle }
{ Bat, Chimp, Liger, Snake }
{ Bat, Chimp, Liger, Snake, Turtle }

A is a proper subset of B: False

A is a subset of B: False

(~C U A) ^ B: { Bat, Cat, Chimp }

(A ^ B) is a Proper Subset of B: True



32 Bit Universe Operations:
*********************************************
Set A: { Bat, Dragon, Hippopotamus, Gecko, Sloth, Deer, Kangaroo }
Set B: { Bat, Hippopotamus, Gecko, Sloth, Squirrel, Platypus }
Set C: { Bat, Dog, Fish, Snake, Turtle, Horse, Donkey, Llama, Gecko, Sloth, Gorilla
    }
*********************************************

(A U B) ^ C: { Bat, Gecko, Sloth }

A U (B ^ C): { Bat, Dragon, Hippopotamus, Gecko, Sloth, Deer, Kangaroo }

~(A ^ B): { Cat, Chimp, Dog, Fish, Liger, Snake, Turtle, Bear, Dragon, Horse, Wolf,
    Rat, Gerbil, Rabbit, Monkey, Donkey, Llama, Zebra, Rhiceros, Frog, Deer,
    Kangaroo, Gorilla, Alligator, Panda, Squirrel, Duck, Platypus }

(~A U ~B): { Cat, Chimp, Dog, Fish, Liger, Snake, Turtle, Bear, Dragon, Horse, Wolf,
     Rat, Gerbil, Rabbit, Monkey, Donkey, Llama, Zebra, Rhiceros, Frog, Deer,
    Kangaroo, Gorilla, Alligator, Panda, Squirrel, Duck, Platypus }

A - B: { Dragon, Deer, Kangaroo }

Powerset A:

{ Kangaroo }
{ Deer }
{ Deer, Kangaroo }
{ Sloth }
{ Sloth, Kangaroo }
{ Sloth, Deer }
{ Sloth, Deer, Kangaroo }
{ Gecko }
{ Gecko, Kangaroo }
{ Gecko, Deer }
{ Gecko, Deer, Kangaroo }
{ Gecko, Sloth }
{ Gecko, Sloth, Kangaroo }
```

{ Gecko, Sloth, Deer }
{ Gecko, Sloth, Deer, Kangaroo }
{ Hippopotamus }
{ Hippopotamus, Kangaroo }
{ Hippopotamus, Deer }
{ Hippopotamus, Deer, Kangaroo }
{ Hippopotamus, Sloth }
{ Hippopotamus, Sloth, Kangaroo }
{ Hippopotamus, Sloth, Deer }
{ Hippopotamus, Sloth, Deer, Kangaroo }
{ Hippopotamus, Gecko }
{ Hippopotamus, Gecko, Kangaroo }
{ Hippopotamus, Gecko, Deer }
{ Hippopotamus, Gecko, Deer, Kangaroo }
{ Hippopotamus, Gecko, Sloth }
{ Hippopotamus, Gecko, Sloth, Kangaroo }
{ Hippopotamus, Gecko, Sloth, Deer }
{ Hippopotamus, Gecko, Sloth, Deer, Kangaroo }
{ Dragon }
{ Dragon, Kangaroo }
{ Dragon, Deer }
{ Dragon, Deer, Kangaroo }
{ Dragon, Sloth }
{ Dragon, Sloth, Kangaroo }
{ Dragon, Sloth, Deer }
{ Dragon, Sloth, Deer, Kangaroo }
{ Dragon, Gecko }
{ Dragon, Gecko, Kangaroo }
{ Dragon, Gecko, Deer }
{ Dragon, Gecko, Deer, Kangaroo }
{ Dragon, Gecko, Sloth }
{ Dragon, Gecko, Sloth, Kangaroo }
{ Dragon, Gecko, Sloth, Deer }
{ Dragon, Gecko, Sloth, Deer, Kangaroo }
{ Dragon, Hippopotamus }
{ Dragon, Hippopotamus, Kangaroo }
{ Dragon, Hippopotamus, Deer }
{ Dragon, Hippopotamus, Deer, Kangaroo }
{ Dragon, Hippopotamus, Sloth }
{ Dragon, Hippopotamus, Sloth, Kangaroo }
{ Dragon, Hippopotamus, Sloth, Deer }
{ Dragon, Hippopotamus, Sloth, Deer, Kangaroo }
{ Dragon, Hippopotamus, Gecko }
{ Dragon, Hippopotamus, Gecko, Kangaroo }
{ Dragon, Hippopotamus, Gecko, Deer }
{ Dragon, Hippopotamus, Gecko, Deer, Kangaroo }
{ Dragon, Hippopotamus, Gecko, Sloth }
{ Dragon, Hippopotamus, Gecko, Sloth, Kangaroo }
{ Dragon, Hippopotamus, Gecko, Sloth, Deer }
{ Dragon, Hippopotamus, Gecko, Sloth, Deer, Kangaroo }
{ Bat }
{ Bat, Kangaroo }
{ Bat, Deer }
{ Bat, Deer, Kangaroo }
{ Bat, Sloth }
{ Bat, Sloth, Kangaroo }
{ Bat, Sloth, Deer }

{ Bat, Sloth, Deer, Kangaroo }
{ Bat, Gecko }
{ Bat, Gecko, Kangaroo }
{ Bat, Gecko, Deer }
{ Bat, Gecko, Deer, Kangaroo }
{ Bat, Gecko, Sloth }
{ Bat, Gecko, Sloth, Kangaroo }
{ Bat, Gecko, Sloth, Deer }
{ Bat, Gecko, Sloth, Deer, Kangaroo }
{ Bat, Hippopotamus }
{ Bat, Hippopotamus, Kangaroo }
{ Bat, Hippopotamus, Deer }
{ Bat, Hippopotamus, Deer, Kangaroo }
{ Bat, Hippopotamus, Sloth }
{ Bat, Hippopotamus, Sloth, Kangaroo }
{ Bat, Hippopotamus, Sloth, Deer }
{ Bat, Hippopotamus, Sloth, Deer, Kangaroo }
{ Bat, Hippopotamus, Gecko }
{ Bat, Hippopotamus, Gecko, Kangaroo }
{ Bat, Hippopotamus, Gecko, Deer }
{ Bat, Hippopotamus, Gecko, Deer, Kangaroo }
{ Bat, Hippopotamus, Gecko, Sloth }
{ Bat, Hippopotamus, Gecko, Sloth, Kangaroo }
{ Bat, Hippopotamus, Gecko, Sloth, Deer }
{ Bat, Hippopotamus, Gecko, Sloth, Deer, Kangaroo }
{ Bat, Dragon }
{ Bat, Dragon, Kangaroo }
{ Bat, Dragon, Deer }
{ Bat, Dragon, Deer, Kangaroo }
{ Bat, Dragon, Sloth }
{ Bat, Dragon, Sloth, Kangaroo }
{ Bat, Dragon, Sloth, Deer }
{ Bat, Dragon, Sloth, Deer, Kangaroo }
{ Bat, Dragon, Gecko }
{ Bat, Dragon, Gecko, Kangaroo }
{ Bat, Dragon, Gecko, Deer }
{ Bat, Dragon, Gecko, Deer, Kangaroo }
{ Bat, Dragon, Gecko, Sloth }
{ Bat, Dragon, Gecko, Sloth, Kangaroo }
{ Bat, Dragon, Gecko, Sloth, Deer }
{ Bat, Dragon, Gecko, Sloth, Deer, Kangaroo }
{ Bat, Dragon, Hippopotamus }
{ Bat, Dragon, Hippopotamus, Kangaroo }
{ Bat, Dragon, Hippopotamus, Deer }
{ Bat, Dragon, Hippopotamus, Deer, Kangaroo }
{ Bat, Dragon, Hippopotamus, Sloth }
{ Bat, Dragon, Hippopotamus, Sloth, Kangaroo }
{ Bat, Dragon, Hippopotamus, Sloth, Deer }
{ Bat, Dragon, Hippopotamus, Sloth, Deer, Kangaroo }
{ Bat, Dragon, Hippopotamus, Gecko }
{ Bat, Dragon, Hippopotamus, Gecko, Kangaroo }
{ Bat, Dragon, Hippopotamus, Gecko, Deer }
{ Bat, Dragon, Hippopotamus, Gecko, Deer, Kangaroo }
{ Bat, Dragon, Hippopotamus, Gecko, Sloth }
{ Bat, Dragon, Hippopotamus, Gecko, Sloth, Kangaroo }
{ Bat, Dragon, Hippopotamus, Gecko, Sloth, Deer }
{ Bat, Dragon, Hippopotamus, Gecko, Sloth, Deer, Kangaroo }

```
A is a proper subset of B: False

A is a subset of B: False

(~C U A) ^ B: { Bat, Hippopotamus, Gecko, Sloth, Squirrel, Platypus }

(A ^ B) is a Proper Subset of B: True
```