cs113 Lab 5 By: Amuldeep Dhillon Build Script

Text written to file build.sh

```
doctex labO.doc
pptexenv latex labO.tex
dvipdf labO.dvi
```

Bourne Shell

```
chmod 777 build.sh
```

Problem 13.6

SML

```
fun element(e,[]) = false
  | element(e,(x::xs)) = x=e orelse element(e,(xs));
fun subset([],[]) = true
  | subset([],(x::xs)) = true
  | subset((x::xs),[]) = false
  | subset((x::xs),(y::ys)) = element(x,(y::ys)) andalso subset(xs,(y::ys));
fun union([],ys) = ys | union(x::xs,ys) =
                        if element(x,ys) then union(xs,ys)
                        else x::union(xs,ys);
fun onepair(x,[]) = [] | onepair(x,y::ys) = (x,y)::onepair(x,ys);
fun cartesian([],ys) = []
   |cartesian(x::xs,ys) = onepair(x,ys) @ cartesian(xs,ys);
fun equal(xs,ys) = subset(xs,ys) andalso subset(ys,xs);
```

Problem 13.6
Let A, B, and C be sets. Show that $A \times (B \cup C) = (A \times B) \cup (A \times C)$.

Problem 13.6 cont.

SML

```
val A = [1,2,3]; val B = [3,6,9]; val C = [5,10,15];
equal(cartesian(A,union(B,C)),union(cartesian(A,B),cartesian(A,C)));
val A = [3,6,4,5]; val B = [3,4]; val C = [3,3,3];
equal(cartesian(A,union(B,C)),union(cartesian(A,B),cartesian(A,C)));
val A = [45,65,78,5,4,2,11]; val B = [23,1234,5543,57]; val C = [1];
equal(cartesian(A,union(B,C)),union(cartesian(A,B),cartesian(A,C)));
```

- ∴ A x (B ∪ C) = (A x B) ∪ (A x C)

Problem 13.6
Let A, B, and C be sets. Show that $A \times (B \cup C) = (A \times B) \cup (A \times C)$.

Problem 13.6 Test Cases

```
> fun element(e,[]) = false
  | element(e,(x::xs)) = x=e orelse element(e,(xs));
fun subset([],[]) = true
  | subset([],(x::xs)) = true
  | subset((x::xs),[]) = false
  | subset((x::xs),(y::ys)) = element(x,(y::ys)) andalso subset(xs,(y::ys));
fun union([],ys) = ys | union(x::xs,ys) =
                    if element(x,ys) then union(xs,ys)
                    else x::union(xs,ys);
fun onepair(x,[]) = [] | onepair(x,y::ys) = (x,y)::onepair(x,ys);
fun cartesian([],ys) = []
   |cartesian(x::xs,ys) = onepair(x,ys) @ cartesian(xs,ys);
fun equal(xs,ys) = subset(xs,ys) andalso subset(ys,xs);
val A = [1,2,3]; val B = [3,6,9]; val C = [5,10,15];
equal(cartesian(A,union(B,C)),union(cartesian(A,B),cartesian(A,C)));
val A = [3,6,4,5]; val B = [3,4]; val C = [3,3,3];
```

```
equal(cartesian(A,union(B,C)),union(cartesian(A,B),cartesian(A,C)));
val A = [45,65,78,5,4,2,11]; val B = [23,1234,5543,57]; val C = [1];
equal(cartesian(A,union(B,C)),union(cartesian(A,B),cartesian(A,C)));
# val element = fn: ''a * ''a list -> bool
> # # # val subset = fn: ''a list * ''a list -> bool
> # # val union = fn: ''a list * ''a list -> ''a list
> val onepair = fn: 'a * 'b list -> ('a * 'b) list
> # val cartesian = fn: 'a list * 'b list -> ('a * 'b) list
> val equal = fn: ''a list * ''a list -> bool
> val A = [1, 2, 3]: int list
val B = [3, 6, 9]: int list
val C = [5, 10, 15]: int list
> val it = true: bool
> val A = [3, 6, 4, 5]: int list
val B = [3, 4]: int list
```

```
val C = [3, 3, 3]: int list
> val it = true: bool
> val A = [45, 65, 78, 5, 4, 2, 11]: int list
val B = [23, 1234, 5543, 57]: int list
val C = [1]: int list
> val it = true: bool
```

Problem 12.10

SML

```
fun count([]) = 0 | count((_,c)::xs) = c + count(xs);
fun subtractList(ys,[]) = count(ys) | subtractList(xs,ys) =
                          count(xs) - count(ys);
fun addList(ys,[]) = count(ys) | addList(xs,ys) =
                     count(xs) + count (ys);
fun subtract(ys,x) = count(ys) - x;
fun add(ys,x) = count(ys) + x;

val total = [("All",60)];
val tomato = [("tomato",45)];
val both = [("tomato&onion",30)];
val plain = [("plain",5)];
val notOnions = addList(tomato,plain);
val OnionOrTomato = add(tomato,subtract(total,notOnions));
val justOnions = subtract(total,notOnions);
val Onions = add(both,justOnions);
```

Problem 12.10
Subway prepared 60 4-inch sandwiches for a birthday party. Among these sandwiches, 45 of them had tomatoes, 30 had both tomatoes and onions, and 5 had neither tomatoes nor onions. Using a Venn diagram, how many sandwiches did he make with
(a) tomatoes or onions?
(b) onions?
(c) onions but not tomatoes?

Problem 12.10 cont.

SML

```
val total = [("All",20)];
val tomato = [("tomato",5)];
val both = [("tomato&onion",5)];
val plain = [("plain",0)];
val notOnions = addList(tomato,plain);
val OnionOrTomato = add(tomato,subtract(total,notOnions));
val justOnions = subtract(total,notOnions);
val Onions = add(both,justOnions);
val total = [("All",100)];
val tomato = [("tomato",50)];
val both = [("tomato&onion",40)];
val plain = [("plain",10)];
val notOnions = addList(tomato,plain);
val OnionOrTomato = add(tomato,subtract(total,notOnions));
val justOnions = subtract(total,notOnions);
val Onions = add(both,justOnions);
```

Problem 12.10
Subway prepared 60 4-inch sandwiches for a birthday party. Among these sandwiches, 45 of them had tomatoes, 30 had both tomatoes and onions, and 5 had neither tomatoes nor onions. Using a Venn diagram, how many sandwiches did he make with
(a) tomatoes or onions?
(b) onions?
(c) onions but not tomatoes?

Problem 12.10 Test Cases

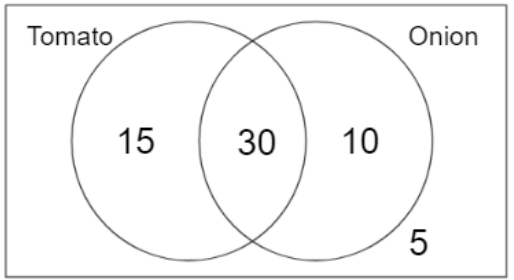
- If you have 60 sandwhiches, 45 with tomato, 30 with tomato and onion, and 5 with neither
  - Then there are 55 with tomatoes or onions, 40 with onions, and 10 with only onion
- If you have 20 sandwhiches, 5 with tomato, 5 with tomato and onion, and 0 with neither
  - Then there are 20 with tomatoes or onions, 20 with onions, and 15 with only onion
- If you have 100 sandwhiches, 50 with tomato, 40 with tomato and onion, and 10 with neither
  - Then there are 90 with tomatoes or onions, 80 with onions, and 40 with only onion

```
val count = fn: ('a * int) list -> int
> # val subtractList = fn: ('a * int) list * ('b * int) list -> int
> # val addList = fn: ('a * int) list * ('b * int) list -> int
> val subtract = fn: ('a * int) list * int -> int
> val add = fn: ('a * int) list * int -> int
> > val total = [("All", 60)]: (string * int) list
> val tomato = [("tomato", 45)]: (string * int) list
> val both = [("tomato&onion", 30)]: (string * int) list
> val plain = [("plain", 5)]: (string * int) list
> val notOnions = 50: int
> val OnionOrTomato = 55: int
> val justOnions = 10: int
> val Onions = 40: int
> > val total = [("All", 20)]: (string * int) list
> val tomato = [("tomato", 5)]: (string * int) list
> val both = [("tomato&onion", 5)]: (string * int) list
```

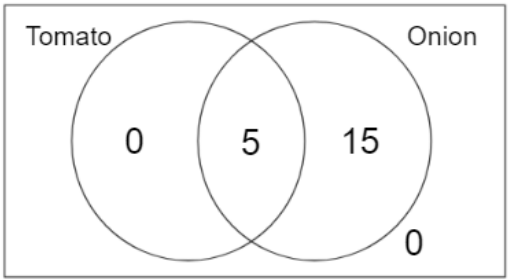
```
> val plain = [("plain", 0)]: (string * int) list
> val notOnions = 5: int
> val OnionOrTomato = 20: int
> val justOnions = 15: int
> val Onions = 20: int
> > val total = [("All", 100)]: (string * int) list
> val tomato = [("tomato", 50)]: (string * int) list
> val both = [("tomato&onion", 40)]: (string * int) list
> val plain = [("plain", 10)]: (string * int) list
> val notOnions = 60: int
> val OnionOrTomato = 90: int
> val justOnions = 40: int
> val Onions = 80: int
```

Problem 12.10 Visual

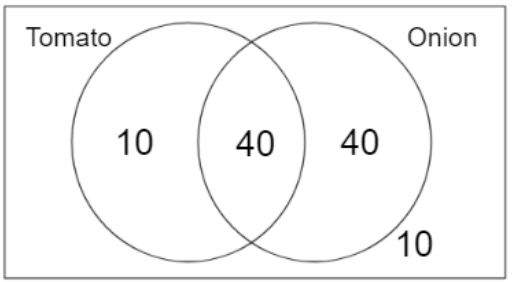
Test Case 1:



Test Case 2:



Test Case 3:



Example 13.3

SML

```
fun inter([],ys) = [] | inter(x::xs,ys) =
                        if element(x,ys) then x::inter(xs,ys)
                        else inter(xs,ys);
val A = [1,2,3,4,5,6];
val A1 = [1,2]; val A2 = [3,4]; val A3 = [5,6];
equal(union(union(A1,A2),A3),A);
inter(A1,A2);
inter(A2,A3);
inter(A1,A3);
val A = [1,5,6,7,8];
val A1 = [1,5,6]; val A2 = [7]; val A3 = [8];
equal(union(union(A1,A2),A3),A);
inter(A1,A2);
inter(A2,A3);
inter(A1,A3);
```

Example 13.3
Let $A = \{1,2,3,4,5,6\}, A_1 = \{1,2\}, A_2 = \{3,4\}, A_3 = \{5,6\}$. Show that $\{A_1, A_2, A_3\}$ is a partition of A.

Example 13.3 cont.

SML

```
val A = [1,4,5,6,7,8,12];
val A1 =[1,4]; val A2 = [5]; val A3 = [6,7]; val A4 = [8]; val A5 = [12];
equal(union(union(union(union(A1,A2),A3),A4),A5),A);
inter(A1,A2);
inter(A1,A3);inter(A1,A4);inter(A1,A5);
inter(A2,A3);inter(A2,A4);inter(A2,A5);
inter(A3,A4);inter(A3,A5);inter(A4,A5);
```

Example 13.3
Let $A = \{1,2,3,4,5,6\}, A_1 = \{1,2\}, A_2 = \{3,4\}, A_3 = \{5,6\}$. Show that $\{A_1, A_2, A_3\}$ is a partition of A.

- A1 ∪ A2 ∪ A3 ... = A
- A1 ∩ A2 = A1 ∩ A3 = A2 ∩ A3 ... =∅
- ∴ by definition A1,A2,A3 ... are a partition of A

Example 13.3 Test Cases

```
> fun inter([],ys) = [] | inter(x::xs,ys) =
                        if element(x,ys) then x::inter(xs,ys)
                        else inter(xs,ys);
val A = [1,2,3,4,5,6];
val A1 = [1,2]; val A2 = [3,4]; val A3 = [5,6];
equal(union(union(A1,A2),A3),A);
inter(A1,A2);
inter(A2,A3);
inter(A1,A3);
val A = [1,5,6,7,8];
val A1 = [1,5,6]; val A2 = [7]; val A3 = [8];
equal(union(union(A1,A2),A3),A);
inter(A1,A2);
inter(A2,A3);
inter(A1,A3);
val A = [1,4,5,6,7,8,12];
```

```
val A1 =[1,4]; val A2 = [5]; val A3 = [6,7]; val A4 = [8]; val A5 = [12];
equal(union(union(union(union(A1,A2),A3),A4),A5),A);
inter(A1,A2);
inter(A1,A3);inter(A1,A4);inter(A1,A5);
inter(A2,A3);inter(A2,A4);inter(A2,A5);
inter(A3,A4);inter(A3,A5);inter(A4,A5);
# # val inter = fn: ''a list * ''a list -> ''a list
> val A = [1, 2, 3, 4, 5, 6]: int list
> val A1 = [1, 2]: int list
val A2 = [3, 4]: int list
val A3 = [5, 6]: int list
> val it = true: bool
> val it = []: int list
> val it = []: int list
> val it = []: int list
> val A = [1, 5, 6, 7, 8]: int list
```

```
> val A1 = [1, 5, 6]: int list
val A2 = [7]: int list
val A3 = [8]: int list
> val it = true: bool
> val it = []: int list
> val it = []: int list
> val it = []: int list
> val A = [1, 4, 5, 6, 7, 8, 12]: int list
> val A1 = [1, 4]: int list
val A2 = [5]: int list
val A3 = [6, 7]: int list
val A4 = [8]: int list
val A5 = [12]: int list
> val it = true: bool
> val it = []: int list
> val it = []: int list
```

```
val A2 = [5]: int list
val A3 = [6, 7]: int list
val A4 = [8]: int list
val A5 = [12]: int list
> val it = true: bool
> val it = []: int list
> val it = []: int list
val it = []: int list
val it = []: int list
> val it = []: int list
val it = []: int list
val it = []: int list
> val it = []: int list
val it = []: int list
val it = []: int list
```