Cryptol Crib Sheet[1]

1 To Use Cryptol:

1. From the linux command line: `prompt> cryptol` to get this:

```
Cryptol version 1.8.4, Copyright (C) 2004-2008 Galois, Inc.
                                               www.cryptol.net
Type :? for help
Cryptol>
```

2. To load a source file, in this case `tests.cry`, do this:

```
Cryptol> :l tests.cry
Loading "tests.cry".. Checking types.. Processing.. Done!
tests>
```

3. To set the base to 10 do this:

```
tests> :s base=10
```

4. To invoke a function, in this case `rev(..)` (reverse a list) do this:

```
tests> rev([1 2 3])
[3 2 1]
tests>
```

5. To enter symbolic mode (for proving assertions) and be verbose do this:

```
tests> :s symbolic
tests> :s +v
```

6. To switch to a different backend, in this case `yices`, do this:

```
tests> :s yices
:set sbv
:set sbv_solver=yices
tests>
```

7. All source files are edited using any simple text editor. The following examples are assumed to be written to file then loaded as above. Where `prompt>` shows up, the functions in the file are invoked.

2 Data Structures

Some variables:

cryptol	c

`v:[32];`
`v = 45;`
	`unsigned int v = 45;`

`x:[64];`
`x = 72625;`
	`unsigned long x = 72625;`

[1] These are notes collected from experiments by John Franco and do not necessarily represent solutions as they would be coded by experts in cryptol.

Arrays:

```
          cryptol                                      c
--------------------------     ------------------------------------------------------------
v:[8][32];                     unsigned int v[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
v = [1 2 3 4 5 6 7 8];
prompt> v
[1 2 3 4 5 6 7 8]              for (int i=0 ; i < 8 ; i++) cout << v[i] << " ";
--------------------------
v = [1..8];
prompt> v
[1 2 3 4 5 6 7 8]
--------------------------     ------------------------------------------------------------
prompt> v@1;                   printf("%d",v[1]);
2
--------------------------     ------------------------------------------------------------
x:[8];                         unsigned char x = 23;
x = 23;
prompt> :s base=2
prompt> x                      f(x,0,7);       void f(unsigned char z, int s, int e) {
0b00010111                     00010111            for (int i=e ; i >= s ; i--)
prompt> x@3                    f(x,3,3);               printf("%d",(int)((z>>i)&1));
False                          0                   printf(\n);
prompt> x@4                    f(x,4,4);       }
True                           1
prompt> x@6                    f(x,6,6);
False                          0
prompt> x@@[0..3]              f(x,0,3);
0b0111                         0111
```

Operations:

```
          cryptol                          c
--------------------------     --------------------------
x:[8];                         unsigned char x = 23;
x = 23;                        unsigned char y = 65;
y:[8];
y = 65;
prompt> :s base=2
prompt> x+y                    f(x+y,0,7);
0b01011000                     01011000
prompt> x-y                    f(x-y,0,7);
0b11010110                     11010110
prompt> x*y                    f(x*y,0,7);
0b11010111                     11010111
prompt> y/x                    f(y/x,0,7);
0b00000010                     00000010
prompt> y%x                    f(y%x,0,7);
0b00010011                     00010011
prompt> y&x                    f(y&x,0,7);
0b00000001                     00000001
prompt> y|x                    f(y|x,0,7);
0b01010111                     01010111
prompt> x>>1                   f(x>>1,0,7)
0b00001011                     00001011
prompt> x<<1                   f(x<<1,0,7)
0b00101110                     00101110
--------------------------     -------------
x:[8][32];
x = [2 3 6 4 3 2 7 8];
y:[8][32];
y = [1 8 3 4 2 1 1 9];
prompt> x+y
[3 11 9 8 5 3 8 17]
```

Function examples:

Simple `for` loop, using `[|..||..|]` operator. Type of function is inferred by cryptol to be `sf:{a b} (fin a,b >= 2) => [a][b] -> [a][b]` (maps any finite list of `a` numbers of width `b`≥ 2 to a list of the same type). Symbol `@` allows array indexing.

cryptol

```
p:[6][8];
p = [1 2 3 4 5 6];
sf(z) = [| 2*(z@i)
        || i <- [0..width(z)-1] |];
prompt> sf(p)
[2 4 6 8 10 12]
```

c

```
typedef unsigned char u_int_8
u_int_8 *sf(u_int_8 z[], int sz) {
    u_int_8 *q = new u_int_8[sz];
    for (int i=0 ; i < 6 ; i++)
        q[i] = 2*z[i];
    return q;
}
u_int_8 p[]={ 1,2,3,4,5,6 };
u_int_8 *z = sf(p,6);
```

Reverse the elements of an array. Construct `where` is used to establish a value `ln` that is used twice in `rev`. Inferred type is `rev:{a b} (fin a) => [a]b -> [a]b`. Observe that `b` does not have to be a number. Note: there is a built-in function called `reverse` with the same functionality as `rev`.

cryptol

```
rev (x) = [| (x@(ln-i))
          || i <- [0 ..  ln] |]
   where {
      ln = width(x)-1;
   };
prompt> rev(p)
[6 5 4 3 2 1]
```

c

```
void **rev(void *x[], int sz) {
    void **q = new void*[sz];
    for (int i=sz-1 ; i >= 0 ; i--)
        q[sz-i-1] = x[i];
    return q;
}
void **p = new void*[6];
for (int i=0; i<=6; i++) p[i] = new u_int_8(i);
u_int_8 **s = (u_int_8**)rev(p,6);
```

Membership in a list. The function's input type is explicitly stated to be any length integer list of at least one 32 bit number and a 32 bit number. The output is a single `Bit` which is `True` if `n` is a member of list `x`. A list `s` is initialized to `False`. As `n` is tested against elements of `x` either `True` or `False` is appended to `s`. If `True` is appended, all following elements of `s` will be `True`. The last element of `s` is the output. This is a simple example using concatenation (`#`) and `if-then-else`.

cryptol

```
member:{a} (fin a, a>=1) =>
([a][32],[32]) -> Bit;
member(x,n) = s@width(x)
  where {
    s = [False]#
        [| if ((x@i) == n)
           then True else s@i
        || i <- [0..ln] |];
    ln = width(x)-1;
  };
prompt> member(p,4);
True
```

c

```
typedef unsigned int u_int_32;
bool member(u_int_32 x[], u_int_32 n, int sz) {
    bool *s = new bool[sz+1];
    s[0] = false;
    for (int i=0 ; i < sz ; i++)
        if (x[i] == n) s[i+1] = true;
        else s[i+1] = s[i];
    return s[sz];
}
```

Merge two infinite increasing streams of integers. This is defined recursively which cryptol does not object to because argument types always match (due to `inf`). First example of `tail`.

cryptol

```
mrg:([inf][32],[inf][32]) -> [inf][32];
mrg(x,y) =
   if ((x@0) < (y@0))
   then [(x@0)]#mrg(tail(x),y)
   else [(y@0)]#mrg(x,tail(y));
```

c++

```
class Stream {
 public:
   int first;
   virtual Stream *rest() { return this; }
   Stream () { first = -1; }
   bool isNull() { return first == -1 }
};
```

A recursive specification of mergesort involving finite lists is possible by coercing the lists into infinite lists then stripping off the required number of tokens at the front using `take`. Notice that the role of `width` changes from finding bits in `x@0` to finding the number of elements in `x` and `y`. For simplicity, we use 0 as a list marker - this prevents 0 from being a legal element. A more complicated solution eliminates this need.

cryptol

```
merge:{a b c} (fin a, fin b, fin c,
b+1>=width(a),b+1>=width(c)) =>
([a][b],[c][b])->[a+c][b];
merge(x,y) = take(lx+ly, mrg(ax,ay))
  where {
    mrg(px,py) =
      if ((px@0) == 0) then py
      else if ((py@0) == 0) then px
      else if ((px@0) < (py@0)) then
        [(px@0)]#mrg(tail(px),py)
      else
        [(py@0)]#mrg(px,tail(py));
    m = width(x@0);
    ax = x#zero:[inf][m];
    ay = y#zero:[inf][m];
    lx:[m+1];
    lx = width(x);
    ly:[m+1];
    ly = width(y);
  }
prompt> merge([3 6 8 10],[1 4 5 9 11])
[1 3 4 5 6 8 9 10 11]
```

c++ (continued)

```
class Merge : public Stream {
   Stream *s1, *s2;
 public:
   Merge (Stream *a, Stream *b) {
      if (a->isNull() && b->isNull()) {
         first = -1;
      } else if (a->isNull() || (!b->isNull()
                 && a->first >= b->first)) {
         s1 = b; s2 = a; first = s1->first;
      } else {
         s1 = a; s2 = b; first = s1->first;
      }
   }
   Stream *rest() {
      if (isNull()) return new Stream();
      return new Merge(s1->rest(),s2);
   }
};
```

Sum all numbers in an arbitrarily long list. This is a recursive solution so lists are padded with `zero:[inf][m]`. Function `f` is made tail recursive to allow cryptol to complete the sum of the necessary elements.

cryptol

```
sum1:{a b} (fin a, fin b, a==b, b>=a)
=> [a][b] -> [b];
sum1(x) = take(width(x),f(ax,0,0))
  where {
    f(y,acc,n) =
        if (n == width(x))
        then acc
        else f(tail(x),acc+y@0,n+1);
    m = width(x@0);
    ax = x#zero:[inf][m];
};
prompt> sum1([5 3 4 2])
14
```

c

```
u_int_32 sum2(u_int_32 x[],
             u_int_32 acc, int n) {
    if (n == 0) return acc;
    return sum2(x, acc+x[n-1], n-1);
}
u_int_32 sum1(u_int_32 x[], int n) {
    return sum2(x,0,n);
}
u_int_32 p[] = { 1,2,3,4,5,6 };
printf("%d",sum1(p,6));
```

Sort a list of numbers using mergesort. Mergesort splits a list into two roughly equal sized lists - the odd indexed elements go to one list and the even indexed elements go to the other, recursively sorts both lists, then merges the two now sorted lists. The variable `i` in `srt` is needed to allow cryptol to show termination.

cryptol

```
mrgsrt:{a b} (fin a, fin b, a>=1, b>=1)
=> [a][b] -> [a][b];
mrgsrt(x) = take(width(x),srt(ax,0))
  where {
    srt(x,i) =
      if (((x@1) == 0)|(i >= lx))
      then x
      else mrg(srt(splite(x,0),i+1),
               srt(splito(x,0),i+1));
    splite(x,i) =
      if (((x@0) == 0)|(i >= lx))
      then x
      else if ((x@1) == 0)
      then drop(1,x)
      else [(x@1)]#splite(drop(2,x),i+1);
    splito(x,i) =
      if (((x@0) == 0)|(i >= lx))
      then x
      else [(x@0)]#splito(drop(2,x),i+1);
    mrg(x,y) =
      if ((px@0) == 0) then py
      else if ((py@0) == 0) then px
      else if ((px@0) < (py@0)) then
        [(px@0)]#mrg(tail(px),py)
      else
        [(py@0)]#mrg(px,tail(py));
    m = width(x@0);
    ax = x#zero:[inf][m];
    lx = width(x);
  };
prompt> mrgsrt([7 3 4 2 9 6 2 1 10])
[1 2 2 3 4 6 7 9 10]
```

c++

```
class Split : public Stream {
   Stream *s;
 public:
   Split(Stream *str) {
      first = str->first; s = str;
   }
   Stream *rest() {
      if (isNull() ||
          s->rest()-isNull() ||
          s->rest()->rest()->isNull())
         return new Stream();
      return new Split(s->rest()->rest());
   }
};
class MergeSort : public Stream {
   Stream *s;
 public:
   MergeSort(Stream *str) {
      Stream *s1 = new Split(str);
      Stream *s2 = new Split(str->rest());
      if (s1->isNull() && s2->isNull()) {
         s = new Stream(); first = -1;
      } else if (s2->isNull() &&
                 s1->rest()->isNull()) {
         s = new Stream(); first = s->first;
      } else {
         s = new Merge(new MergeSort(s1),
                       new MergeSort(s2));
         first = s->first;
      }
   }
   Stream *rest() { return s->rest(); }
};
```

Returns `True` if and only if elements of list `x` are in increasing order.

cryptol

```
ordered:{a b} (fin a, fin b, a>=1, b>=1)
=> [a][b] -> Bit;
ordered x = s@(width(x)-1)
  where {
    s = [True]#
        [| if ((x@i)<=(x@(i+1)))
           then (s@i & True)
           else False
        || i <- [0..(width(x)-2)] |];
  };
prompt> ordered([4 8 10 23 66])
True
prompt> ordered([4 8 10 23 4 66])
False
```

c

```
bool ordered (u_int_32 x[], int sz) {
   bool *s = new bool[sz+1];
   s[0] = true;
   for (int i=0 ; i < sz-1 ; i++)
      s[i+1] = (x[i] < x[i+1]) ?
               s[i] & true : false;
   return s[sz-1];
}
```

Returns True if and only if list x contains only positive numbers.

cryptol

```
valid_list:{a b} (fin a, fin b, a>=1, b>=1) => [a][b] -> Bit;
valid_list x = val(0)
  where {
    val(i) =
      if (i == lx) then True
      else if ((x@i)<=0) then False
      else val(i+1);
    lx = width(x);
  };
prompt> valid_list([4 5 2 3 9])
True
prompt> valid_list([4 5 2 0 3 9])
False
```

Returns list x with one occurrence of number n removed.

cryptol

```
remove:{a b c} (fin a, fin b, fin c, a>=1, b>=1, c>=1, c==a-1, b>=width(a)) =>
([a][b],[b]) -> [c][b];
remove(x,n) = take(lx-1, rem(ax,lx,lx-1))
  where {
    rem(x,l,i) =
      if ((x@i) == n) then ins(x@(l-1),i,l,0)
      else if (i == 0) then x
      else rem(x,l,i-1);
    ins(p,i,l,j) =
      if (j == l-1) then zero:[inf][w]
      else if (j == i) then [p]#ins(p,i,l,j+1)
      else [(x@j)]#ins(p,i,l,j+1);
    w = width(x@0);
    lx = width(x);
    ax = x#zero:[inf][w];
  };
prompt> remove([8 4 3 5 6 4 2], 4)
[8 4 3 5 6 2]
```

Returns True if and only if list x is a permutation of list y. Note: the member function used in perm is as above but typed as: member:{a b} (fin a, fin b,a>=1,b>=1) => ([a][b],[b]) -> Bit;

cryptol

```
perm:{a b} (fin a, fin b, a>=1, b>=1, b>=width(a+1)) =>
([a][b],[a][b]) -> Bit;
perm(x,y) = if (lx != ly) then False else isperm(x,y,0)
  where {
    isperm(x,y,i) = if (i == lx) then True
                    else if (~member(y,(x@0))) then False
                    else isperm(tail(x)#[0],remove(y#[0],(x@0)),i+1);
    lx = width(x);
    ly = width(y);
  };
prompt> perm([1 2 3],[3 2 1])
True
prompt> perm([1 1 2],[2 2 1])
False
```

3 Proofs of properties

Correctness of mrgsrt:

Prove that any list of four 5 bit numbers is correctly sorted by `mrgsrt`. A list `x` is correctly sorted to list `y` if `y` is a permutation of `x` and all elements of `y` are in increasing order.

cryptol

```
prompt> :s symbolic
prompt> :sat (\(x,y) -> (((y:[4][5]) == mrgsrt(x:[4][5])) &
                         ~(perm(y,x) & ordered(y)) & valid_list(x)))
No variable assignment satisfies this function
```

Alternative check - the theorems are placed in the file.

cryptol

```
mergeSortIsCorrect :  [3][5] -> Bit;
theorem mergeSortIsCorrect:  {x}.  (ordered(y) & perm(x,y)) | ~valid_list(x)
  where y = mrgsrt(x);
prompt> :s +v
prompt> :s sbv
prompt> :s symbolic
prompt> :prove mergeSortIsCorrect
Q.E.D.
prompt> :sat mergeSortIsCorrect
mergeSortIsCorrect [0x00 0x00 0x00]
         = True

msCorrectBySat :  [3][5] -> Bit;
theorem msCorrectBySat:  {x}.  ~(ordered(y) & perm(x,y)) & valid_list(x)
  where y = mrgsrt(x);
prompt> :sat msCorrectBySat
No variable assignment satisfies this function
```