

Practical property-based testing in scientific software

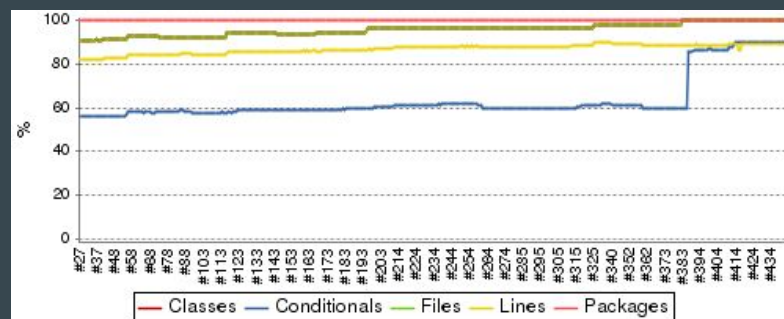


Mousumi Chattaraj
Luiz Irber
Li Li

khmer project

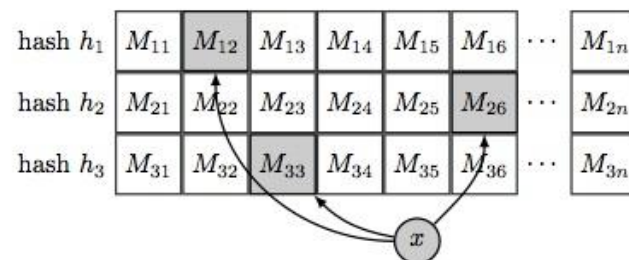


- Repository: <https://github.com/dib-lab/khmer>
- Unit testing (some integration testing too)
- ~90% coverage (lines and conditionals)
- CI server: <http://ci.oxli.org>
- Every PR is tested
- Sketches for biological data analysis
 - Diginorm
 - Partitioning
 - Read mapping/alignment



Algorithm 1 Count-Min Sketch

```
insert( $x$ ):  
  for  $i = 1$  to  $d$  do  
     $M[i, h_i(x)] \leftarrow M[i, h_i(x)] + 1$   
  end for  
  
query( $x$ ):  
   $c = \min \{M[i, h_i(x)] \text{ for all } 1 \leq i \leq d\}$   
  return  $c$ 
```



Technical approach

- Progressive coverage and exploration
 - Hashing
 - HyperLogLog
 - Bloom filters
 - Count-min sketch

Hypothesis: Practical Property-based testing

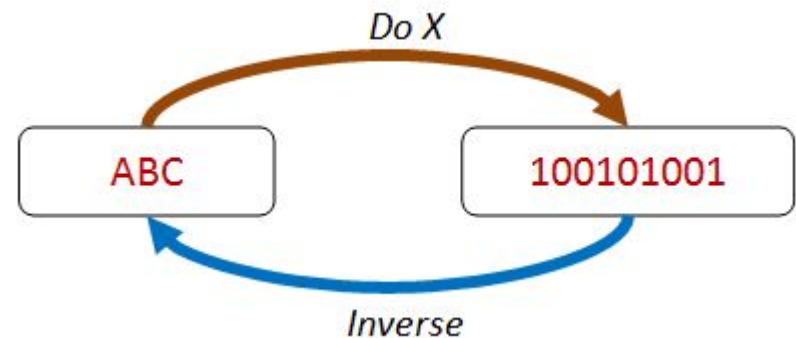
- Property-based testing works better with languages with no side effects
 - Haskell -> QuickCheck
- Hypothesis: a mix of
 - Unit testing
 - Property-based testing
 - Fuzzing
- Properties = testable specification
- Automatic input generation
 - Random
 - Customized generators
- Shrinking

Hashing example

```
1 from hypothesis import given, assume
2 from hypothesis.specifiers import strings
3 from nose.plugins.attrib import attr
4
5 from khmer import reverse_hash, forward_hash, forward_hash_no_rc
6
7 #from hypothesis import Settings, Verbosity
8 #Settings.default.verbosity = Verbosity.verbose
9 TRANSLATE = {'A': 'T', 'C': 'G', 'T': 'A', 'G': 'C'}
10
11
12 @attr('hypothesis')
13 @given(strings("ACGT"))
14 def test_forward_hash(kmer):
15     ksize = len(kmer)
16     assume(ksize > 0)
17     assume(ksize < 32)
18
19     rh = reverse_hash(forward_hash(kmer, ksize), ksize)
20     assert rh == kmer or rh == "".join(TRANSLATE[c] for c in kmer[::-1])
21
22
23 @attr('hypothesis')
24 @given(strings("ACGT"))
25 def test_forward_hash_no_rc(kmer):
26     ksize = len(kmer)
27     assume(ksize > 0)
28     assume(ksize < 32)
29
30     rh = reverse_hash(forward_hash_no_rc(kmer, ksize), ksize)
31     assert rh == kmer
```

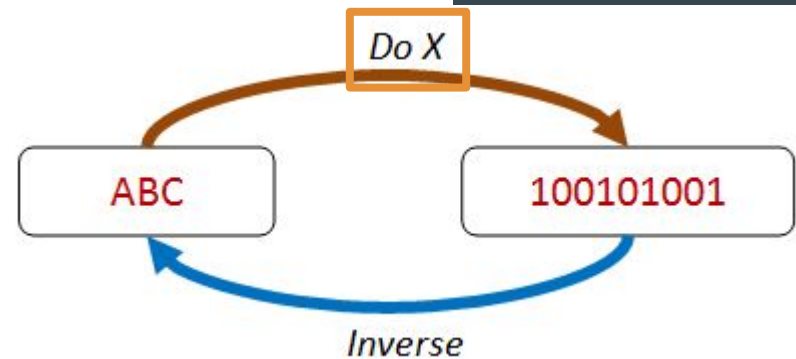
Hashing example

```
1 from hypothesis import given, assume
2 from hypothesis.specifiers import strings
3 from nose.plugins.attrib import attr
4
5 from khmer import reverse_hash, forward_hash, forward_hash_no_rc
6
7 #from hypothesis import Settings, Verbosity
8 #Settings.default.verbosity = Verbosity.verbose
9 TRANSLATE = {'A': 'T', 'C': 'G', 'T': 'A',
10
11
12 @attr('hypothesis')
13 @given(strings("ACGT"))
14 def test_forward_hash(kmer):
15     ksize = len(kmer)
16     assume(ksize > 0)
17     assume(ksize < 32)
18
19     rh = reverse_hash(forward_hash(kmer, ksize), ksize)
20     assert rh == kmer or rh == "".join(TRANSLATE[c] for c in kmer[::-1])
21
22
23 @attr('hypothesis')
24 @given(strings("ACGT"))
25 def test_forward_hash_no_rc(kmer):
26     ksize = len(kmer)
27     assume(ksize > 0)
28     assume(ksize < 32)
29
30     rh = reverse_hash(forward_hash_no_rc(kmer, ksize), ksize)
31     assert rh == kmer
```



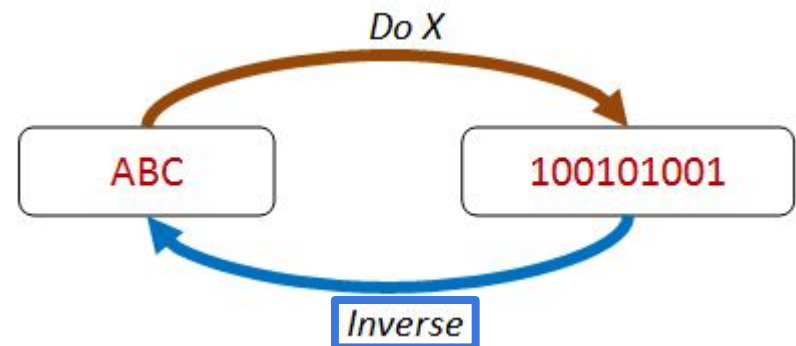
Hashing example

```
1 from hypothesis import given, assume
2 from hypothesis.specifiers import strings
3 from nose.plugins.attrib import attr
4
5 from khmer import reverse_hash, forward_hash, forward_hash_no_rc
6
7 #from hypothesis import Settings, Verbosity
8 #Settings.default.verbosity = Verbosity.verbose
9 TRANSLATE = {'A': 'T', 'C': 'G', 'T': 'A',
10
11
12 @attr('hypothesis')
13 @given(strings("ACGT"))
14 def test_forward_hash(kmer):
15     ksize = len(kmer)
16     assume(ksize > 0)
17     assume(ksize < 32)
18
19     rh = reverse_hash(forward_hash(kmer, ksize), ksize)
20     assert rh == kmer or rh == "".join(TRANSLATE[c] for c in kmer[::-1])
21
22
23 @attr('hypothesis')
24 @given(strings("ACGT"))
25 def test_forward_hash_no_rc(kmer):
26     ksize = len(kmer)
27     assume(ksize > 0)
28     assume(ksize < 32)
29
30     rh = reverse_hash(forward_hash_no_rc(kmer, ksize), ksize)
31     assert rh == kmer
```



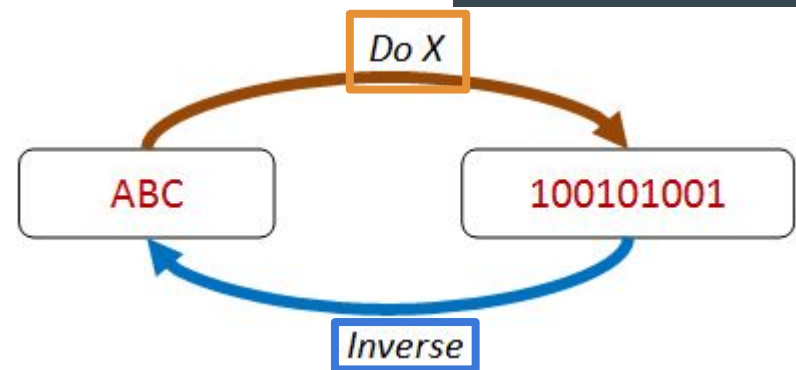
Hashing example

```
1 from hypothesis import given, assume
2 from hypothesis.specifiers import strings
3 from nose.plugins.attrib import attr
4
5 from khmer import reverse_hash, forward_hash, forward_hash_no_rc
6
7 #from hypothesis import Settings, Verbosity
8 #Settings.default.verbosity = Verbosity.verbose
9 TRANSLATE = {'A': 'T', 'C': 'G', 'T': 'A',
10
11
12 @attr('hypothesis')
13 @given(strings("ACGT"))
14 def test_forward_hash(kmer):
15     ksize = len(kmer)
16     assume(ksize > 0)
17     assume(ksize < 32)
18
19     rh = reverse_hash(forward_hash(kmer, ksize), ksize)
20     assert rh == kmer or rh == "".join(TRANSLATE[c] for c in kmer[::-1])
21
22
23 @attr('hypothesis')
24 @given(strings("ACGT"))
25 def test_forward_hash_no_rc(kmer):
26     ksize = len(kmer)
27     assume(ksize > 0)
28     assume(ksize < 32)
29
30     rh = reverse_hash(forward_hash_no_rc(kmer, ksize), ksize)
31     assert rh == kmer
```



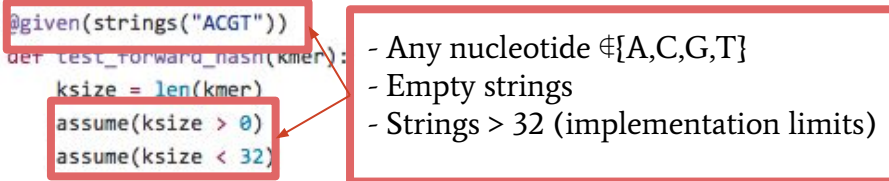
Hashing example

```
1 from hypothesis import given, assume
2 from hypothesis.specifiers import strings
3 from nose.plugins.attrib import attr
4
5 from khmer import reverse_hash, forward_hash, forward_hash_no_rc
6
7 #from hypothesis import Settings, Verbosity
8 #Settings.default.verbosity = Verbosity.verbose
9 TRANSLATE = {'A': 'T', 'C': 'G', 'T': 'A',
10
11
12 @attr('hypothesis')
13 @given(strings("ACGT"))
14 def test_forward_hash(kmer):
15     ksize = len(kmer)
16     assume(ksize > 0)
17     assume(ksize < 32)
18
19     rh = reverse_hash(forward_hash(kmer, ksize), ksize)
20     assert rh == kmer or rh == "".join(TRANSLATE[c] for c in kmer[::-1])
21
22
23 @attr('hypothesis')
24 @given(strings("ACGT"))
25 def test_forward_hash_no_rc(kmer):
26     ksize = len(kmer)
27     assume(ksize > 0)
28     assume(ksize < 32)
29
30     rh = reverse_hash(forward_hash_no_rc(kmer, ksize), ksize)
31     assert rh == kmer
```



Hashing example

```
1 from hypothesis import given, assume
2 from hypothesis.specifiers import strings
3 from nose.plugins.attrib import attr
4
5 from khmer import reverse_hash, forward_hash, forward_hash_no_rc
6
7 #from hypothesis import Settings, Verbosity
8 #Settings.default.verbosity = Verbosity.verbose
9 TRANSLATE = {'A': 'T', 'C': 'G', 'T': 'A', 'G': 'C'}
10
11
12 @attr('hypothesis')
13 @given(strings("ACGT"))
14 def test_forward_hash(kmer):
15     ksize = len(kmer)
16     assume(ksize > 0)
17     assume(ksize < 32)
18
19     rh = reverse_hash(forward_hash(kmer, ksize), ksize)
20     assert rh == kmer or rh == "".join(TRANSLATE[c] for c in kmer[::-1])
21
22
23 @attr('hypothesis')
24 @given(strings("ACGT"))
25 def test_forward_hash_no_rc(kmer):
26     ksize = len(kmer)
27     assume(ksize > 0)
28     assume(ksize < 32)
29
30     rh = reverse_hash(forward_hash_no_rc(kmer, ksize), ksize)
31     assert rh == kmer
```



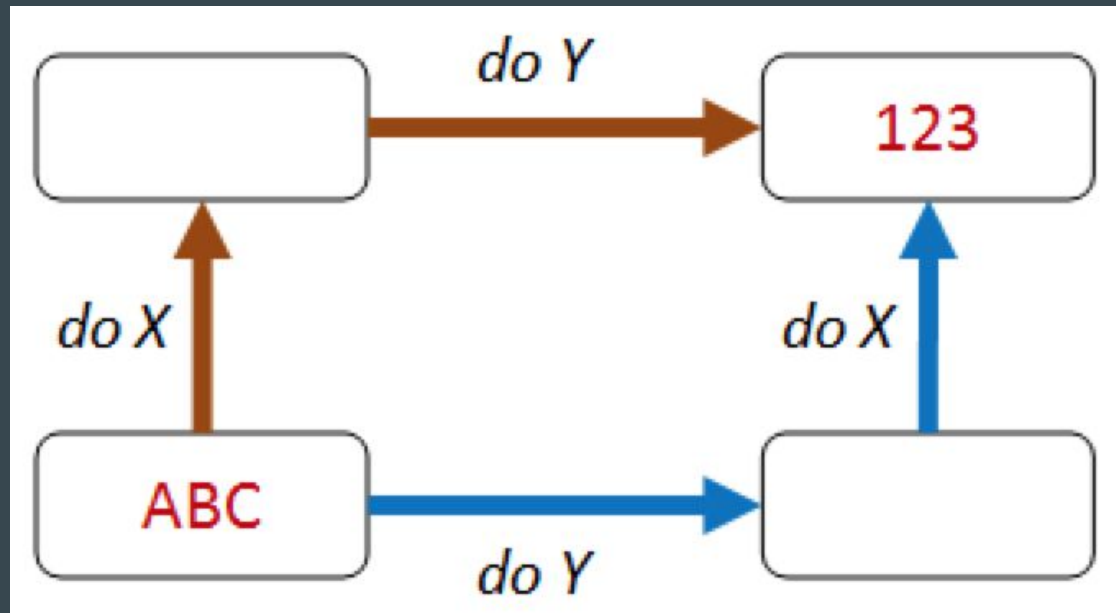
- Any nucleotide ∈ {A,C,G,T}
- Empty strings
- Strings > 32 (implementation limits)

Technical approach

- Categories of properties
 - “Different paths, same destination”
 - “There and back again”
 - “Some things never change”
 - “The more things change, the more they stay the same”
 - “Solve a smaller problem first”
 - “Hard to prove, easy to verify”
 - “Test oracle”

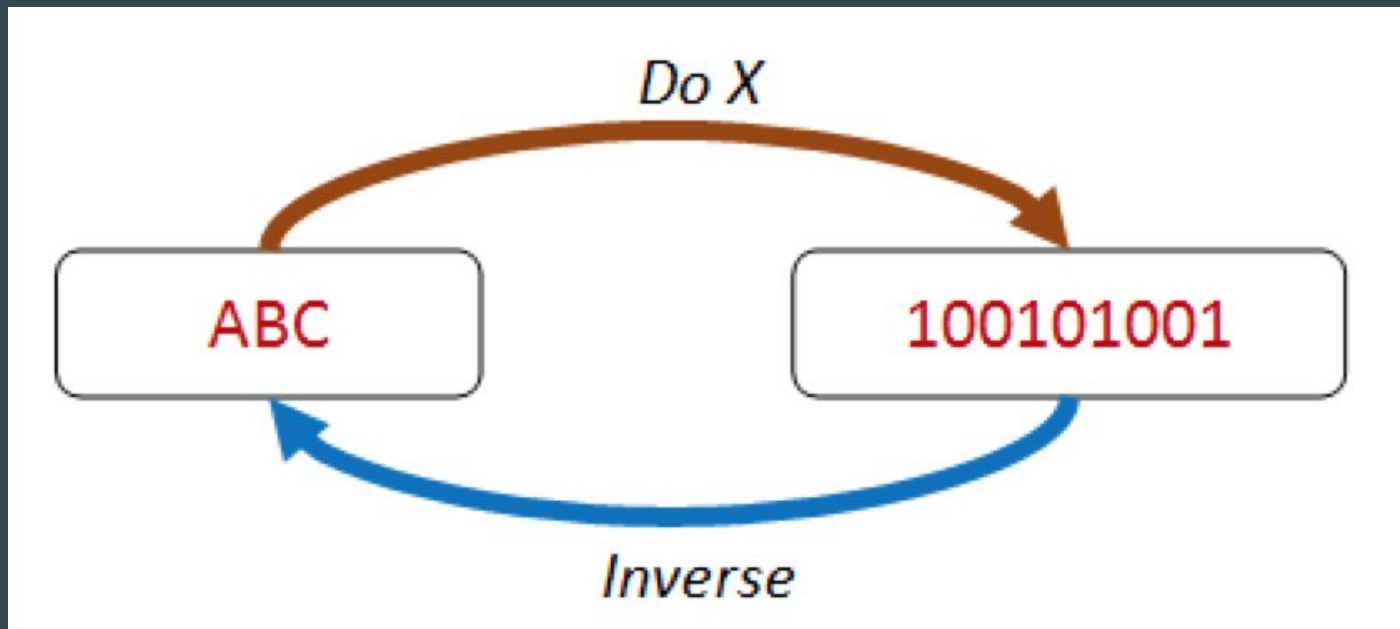
“Different paths, same destination”

Combining operations in different orders but getting the same results.



“There and back again”

Combining an operation with its inverse, ending up with the same value you started with.



“Some things never change”

A invariant that is preserved after some transformation.



Evaluation methodology

- How many method calls can we cover?
 - Start with core data structures, move to specialized methods
- How many interesting bugs can we find?
- Generate guidelines for testing similar software (bioinformatics)