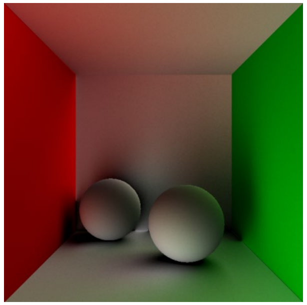
Integrating random and enumerative Property-Based Testing



DEMO

limit	Expected # tests to falsify (QC)
1	16
2	47
3	201
4	1170
5	6757
6	71429
7	526315

A Real Database Race Condition

```
Prefix:
   open_file(dets_table,[{type,bag}]) --> dets_table
   close(dets_table) --> ok
   open_file(dets_table,[{type,bag}]) --> dets_table

Parallel:
1. lookup(dets_table,0) --> []

2. insert(dets_table,{0,0}) --> ok

3. insert(dets_table,{0,0}) --> ok
```

And yet…

```
prop_Wrong = forAll (choose (1,100)) $ \n ->
  n/=67
```

And yet…

```
prop_Wrong = forAll (choose (1,100)) $ \n ->
  n/=67
```

	Expected time to find the failure
Random testing	100 tests

And yet…

```
prop_Wrong = forAll (choose (1,100)) $ \n ->
  n/=67
```

	Expected time to find the failure	Time to *exclude* such a failure (99% confidence)
Random testing	100 tests	458 tests

And yet…

```
prop_Wrong = forAll (choose (1,100)) $ \n ->
  n/=67
```

	Expected time to find the failure	Time to *exclude* such a failure (99% confidence)
Random testing	100 tests	458 tests
Enumeration	67 tests	67 tests?

And yet…

```
prop_Wrong = forAll (choose (1,100)) $ \n ->
   n/=67
```

	Expected time to find the failure	Time to *exclude* such a failure (99% confidence)
Random testing	100 tests	458 tests
Enumeration	67 tests	67 tests?
Random enumeration	50.5 tests	99 tests

DIVERSE TEST CASES MAY
FIND BUGS FASTER

Gen a [a]

Seed → a

Unions

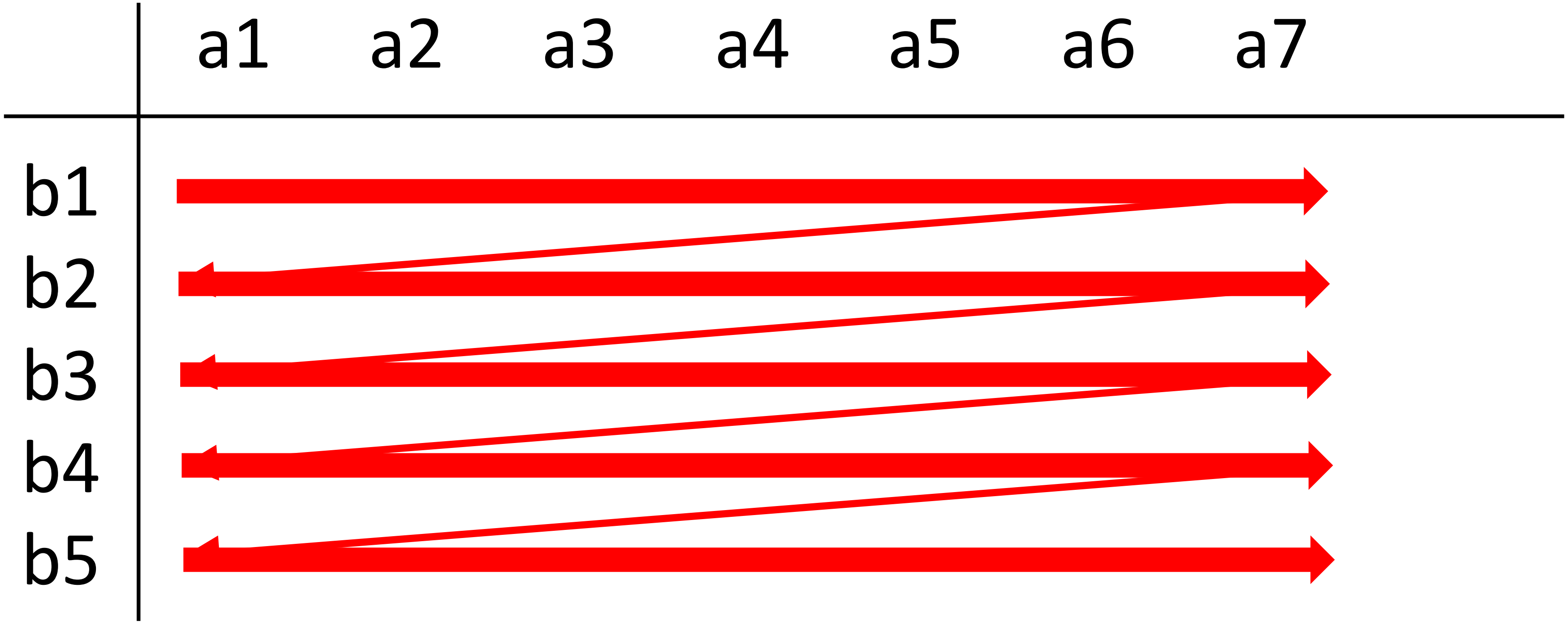
```
weightedChoice ::
  (Int, Gen [a]) ->
  (Int, Gen [a]) -> Gen [a]
```

merge the enumerations from the two generators ***randomly****, according to their weights*

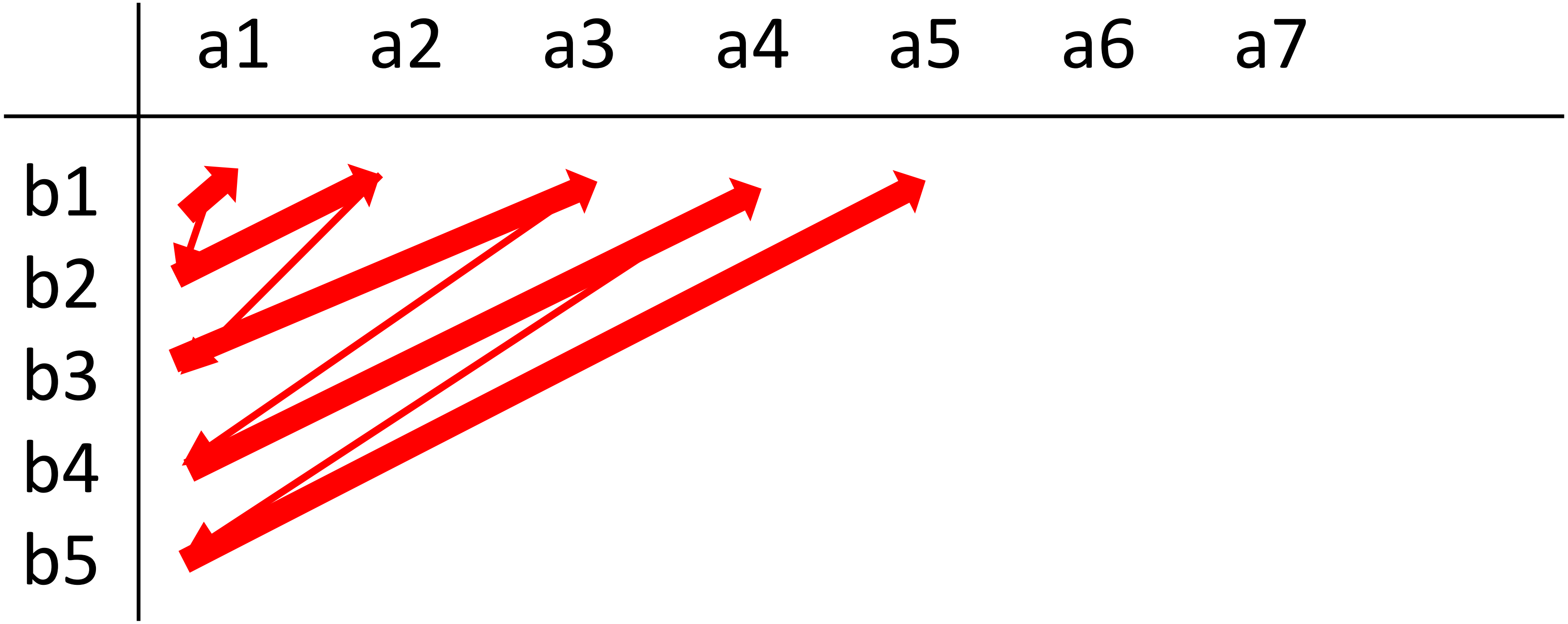
What about products?

```
(***) ::
   Gen [a] ->
   Gen [b] -> Gen [(a,b)]
```

What about products?



What about products?



a1 a2 a3 a4 a5 a6 a7

b1

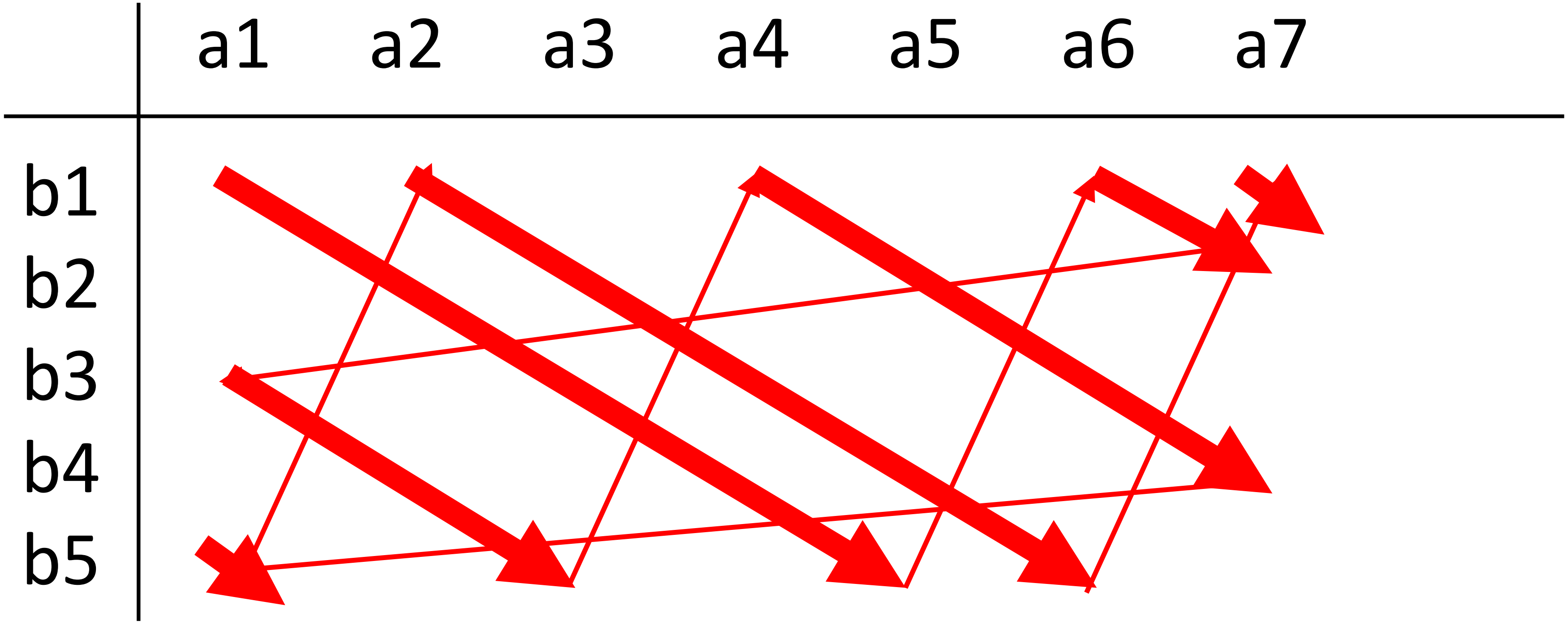
b2

b3

b4

b5

What about products?



a1 a2 a3 a4 a5 a6 a7

b1

b2

b3

b4

b5

Combinatorial testing

Imagine testing an application…

- …on Windows, Mac, and Linux
- …in Firefox and Chrome
- …over IPv4 and IPv6
- …on Intel and AMD
- …using MySQL, Sybase or Oracle

72 combinations

OS	Browser	IP version	Processor	Database
Windows	Chrome	IPv4	Intel	MySQL
Windows	Firefox	IPv6	AMD	Sybase
Windows	Chrome	IPv6	Intel	Oracle
Mac	Firefox	IPv4	AMD	MySQL
Mac	Chrome	IPv4	Intel	Sybase
Mac	Firefox	IPv4	Intel	Oracle
Linux	Chrome	IPv6	AMD	MySQL
Linux	Firefox	IPv4	Intel	Sybase
Linux	Firefox	IPv4	AMD	Oracle
Mac	Firefox	IPv6	AMD	Oracle

TESTS OFTEN FAIL BECAUSE OF A PART OF THE INPUT

The *interaction rule*

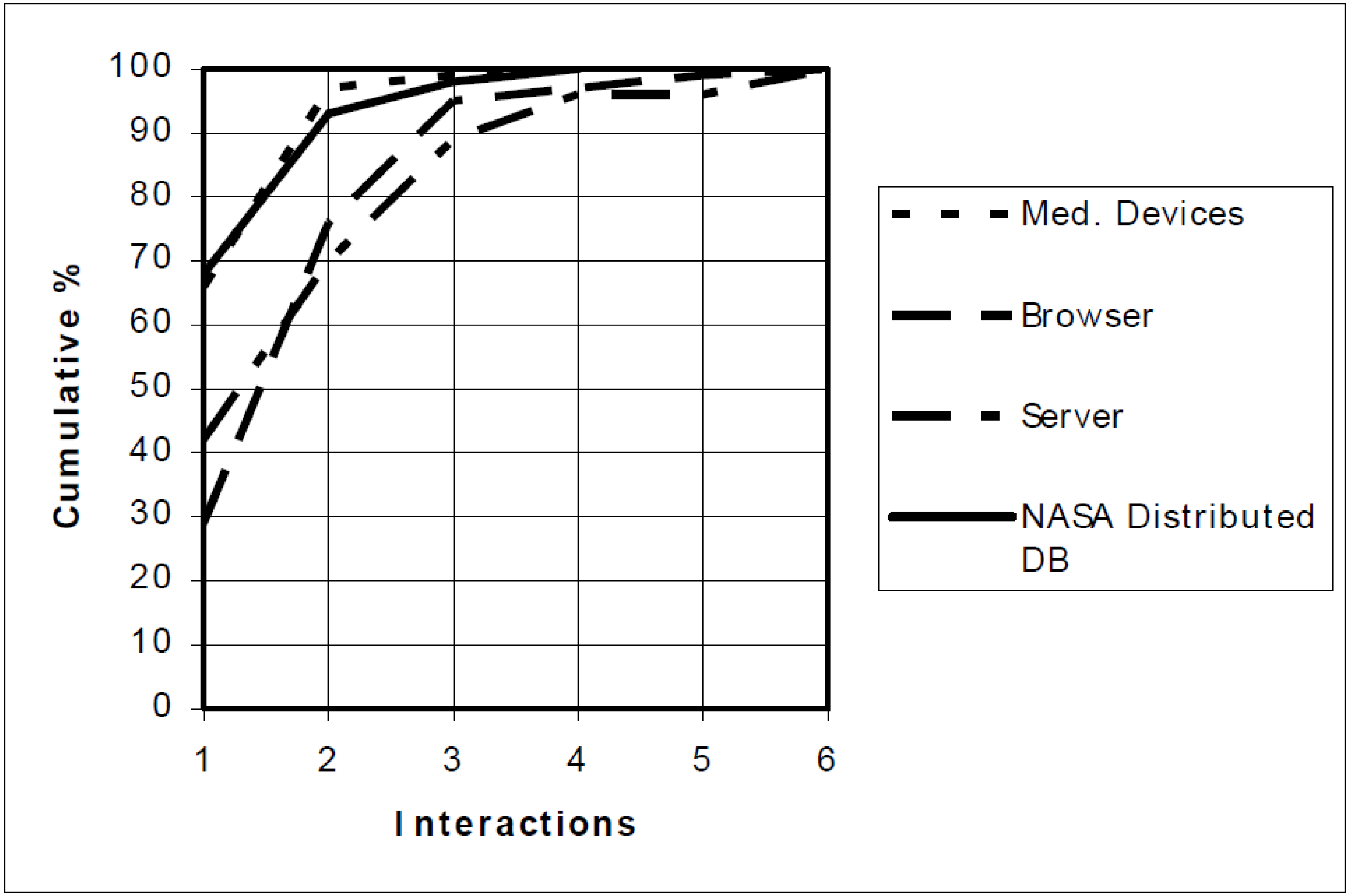
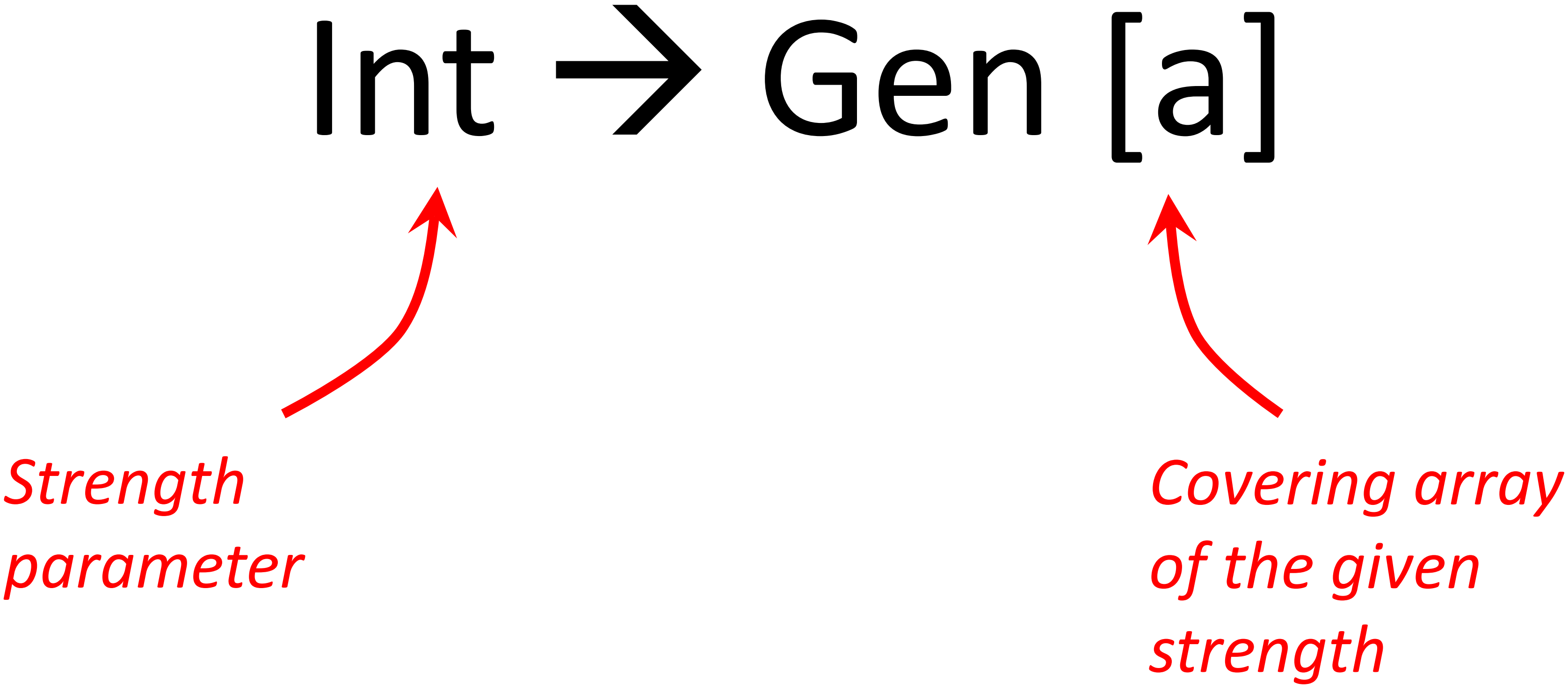


Figure 2. Error detection rates for interaction strengths 1 to 6



Int → Gen [a]

Strength parameter

Covering array of the given strength

Enumerating

```
enumerate xs

   Strength 0: choose a random element

   Strength 1: all elements in a random
               order
```

At strength k,

every subset of k *calls to* **`enumerate`** *takes all combinations of values*

Example

```
testCase = (,,,,)
  <$> os
  <*> browser
  <*> ipversion
  <*> processor
  <*> db

os        = enumerate ["Windows","Mac","Linux"]
browser   = enumerate ["Chrome","Firefox"]
ipversion = enumerate ["IPv4", "IPv6"]
processor = enumerate ["Intel","AMD"]
db        = enumerate ["MySQL","Sybase","Oracle"]
```

Strength 0

Mac Chrome IPv6 Intel Oracle

Strength 1

Linux	Chrome	IPv4	AMD	Sybase
Mac	Firefox	IPv6	Intel	Oracle
Windows	Firefox	IPv4	Intel	MySQL

Strength 2

Linux	Chrome	IPv4	AMD	Oracle
Windows	Chrome	IPv6	AMD	MySQL
Linux	Firefox	IPv4	Intel	Sybase
Mac	Chrome	IPv6	AMD	Oracle
Windows	Firefox	IPv4	Intel	MySQL
Linux	Chrome	IPv6	AMD	Sybase
Mac	Firefox	IPv4	Intel	Oracle
Windows	Firefox	IPv6	Intel	MySQL
Mac	Chrome	IPv6	AMD	Sybase
Windows	Chrome	IPv6	Intel	Oracle
Mac	Firefox	IPv4	Intel	MySQL
Linux	Firefox	IPv4	AMD	Sybase
Mac	Firefox	IPv6	Intel	Oracle
Linux	Chrome	IPv4	AMD	MySQL
Windows	Chrome	IPv6	Intel	Sybase

15 cases

Other interpretations of strength?

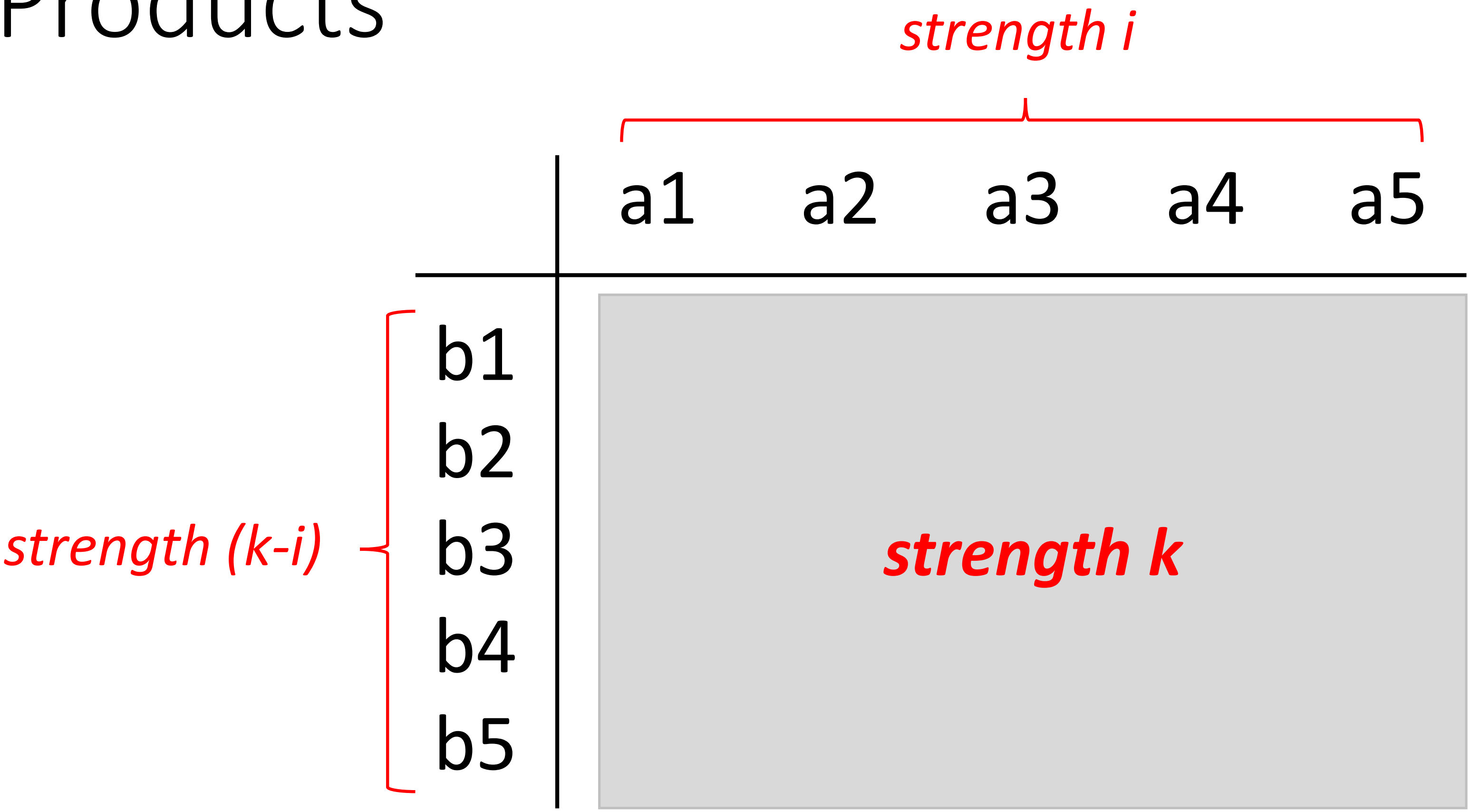
E.g. integer range m..n at strength *k*:

- First *k* elements
- Last *k* elements
- *k* random elements from the middle

Will it work?

- Combinator definitions
  - Monadic interface is a bit awkward
  - Can we automatically ”rebracket” for smallest enumerations?
- What notions of strength make sense for various datatypes?
- Will it actually find bugs faster?

Products



strength i

a1 a2 a3 a4 a5

strength (k-i)

b1
b2
b3
b4
b5

strength k

$$\bigcup_{i \leftarrow [0..k]} \text{strength a i `prod` strength b (k-i)}$$

An optimisation

$$\overbrace{\text{a1} \qquad \text{a2} \qquad \text{a3} \qquad \text{a4} \qquad \text{a5}}^{\textit{strength k}}$$

$$\overbrace{\text{b1} \qquad \text{b2} \qquad \text{b3} \qquad \text{b4} \qquad \text{b5}}^{\textit{strength k}}$$

An optimisation

$$\overbrace{(a1,\_)\qquad(a2,\_)\qquad(a3,\_)\qquad(a4,\_)\qquad(a5,\_)}^{\textit{strength k}}$$

$$\overbrace{b1\qquad b2\qquad b3\qquad b4\qquad b5}^{\textit{strength k}}$$

An optimisation

$$\overbrace{(a1,\_)\quad (a2,\_)\quad (a3,\_)\quad (a4,\_)\quad (a5,\_)}^{\textit{strength k}}$$

$$\overbrace{(\_,b1)\quad (\_,b2)\quad (\_,b3)\quad (\_,b4)\quad (\_,b5)}^{\textit{strength k}}$$

(a1,b1) (a2,b2) (a3,b3) (a4,b4) (a5,b5)

What is a Set of strength *k*?

strength i

a1	a2	a3	a4	a5	a6	a7
T	F	F	T	T	F	T
F	F	F	T	F	T	T
T	F	T	F	T	F	T
F	F	T	T	T	F	F

strength (k-i)

What is a list of strength k?

$$\overbrace{a1 \quad a2 \quad a3 \quad a4 \quad a5 \quad a6 \quad a7}^{\textit{strength k-1}}$$

$$a7 \quad a6 \quad a5 \quad a4 \quad a3 \quad a2 \quad a1$$

Cf Sequence covering arrays