How To Efficiently Process 2^{100} Different Lists

Lukas Lazarek,[1] Chu-Pan Wong,[2] Jens Meinicke,[2,3] Christian Kästner[2]
[1]University of Massachusetts Lowell
[2]Carnegie Mellon University
[3]University of Magdeburg

Handling the Exponential Explosion

Configurable systems have too many features to test all configurations. There is an exponential number of combinations with the number of configurable features available. Yet, most executions and data are redundant across configurations.[1][3] Exploiting this redundancy provides a means of soundly reasoning about configurable systems over all configurations. For example, it enables testing that a system works in every possible configuration without having to actually run the system for each one. Alternatively, it can help reveal and diagnose unintended feature interactions within a system.

Variational Execution

Variational Execution is an analysis technique that exploits sharing of executions and data to make testing **all** configurations feasible. It works by mapping concrete values to sets of configurations called contexts.

This method is general and sound, but not necessarily efficient. Data structures present performance problems.[2]

Building a list with Variational Execution

```
True  List<Integer> x = new List<>();
True  if (A)
A       x.add(1);
True  if (B)
B       x.add(2);
...   // ...
```

Iterating over list

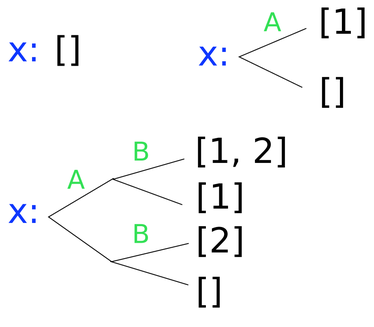
```
for (element: x)
  // ...
```

V<List>

V<List> is the simplest variational list representation with a seperate copy of the list for every configuration. Exponential growth of feature configurations creates exponential copies of the list!

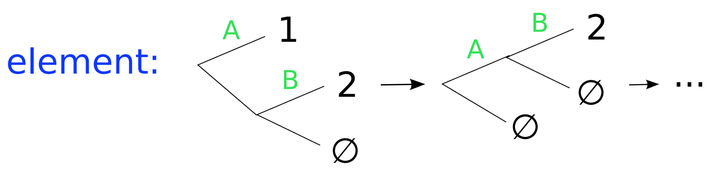
Iterating over V<List> processes the elements of *every copy* of the list; one element may be visited many times.

Building a V<List>



Iterating over V<List>

```
for (i = 0; i < x.length; i++){
  for (element: x.getChoice(i))
    // ...
}
```



CtxList

V<List> is automatically replaced with CtxList, a specialized list storing pairs of values with contexts so that every element occurs only once. The explosion of configurations does not lead to redundant values in the list.

Iteration over CtxList is transformed to visit each element *once* in its associated context.

Building a CtxList

x: []

x: $[1_A]$

x: $[1_A, 2_B]$

Iterating over CtxList

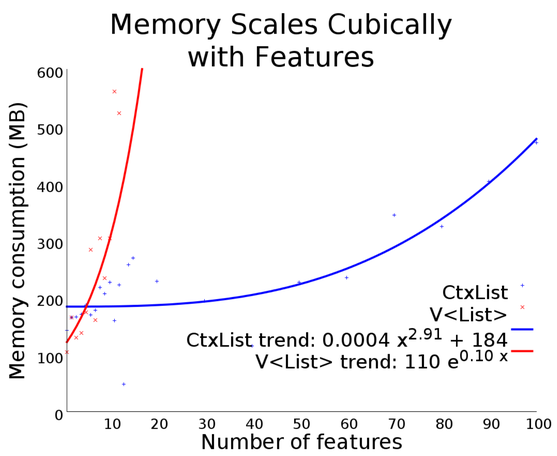
```
for ((ctx, element): x){
  if (ctx.isSatisfiable)
    // ...
}
```

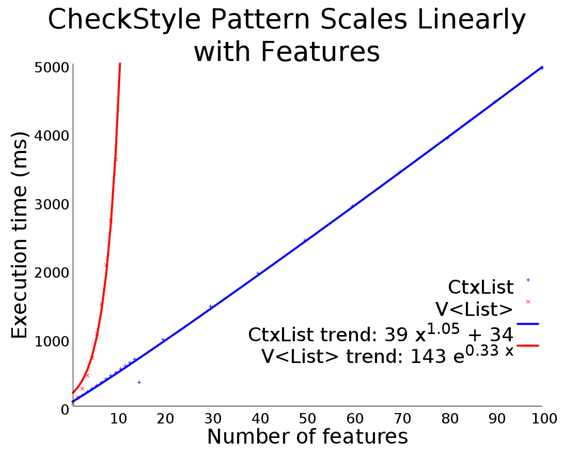
element: $1_A \rightarrow 2_B \rightarrow \cdots$

Evaluation

Building a list with Variational Execution

```
List<Integer> x = new ...
if (A)
  x.add(1);
if (B)
  x.add(2);
// ...
```



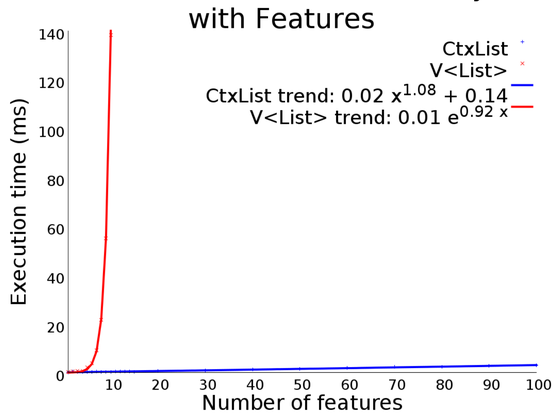


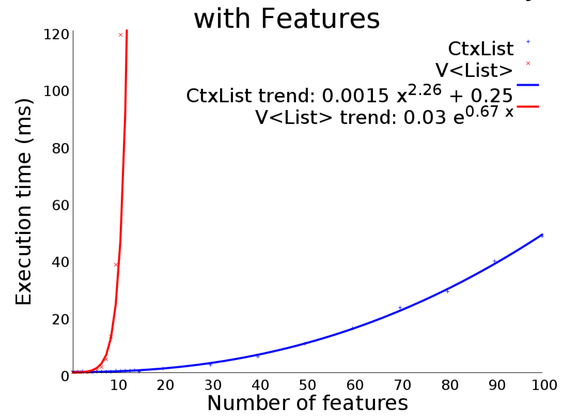
CheckStyle Iteration Pattern

```
List<Object> l = new ...
if (A)
  l.add(new Object());
if (B)
  l.add(new Object());
// ...
for (Object el: l)
  el.expensiveOperation();
```

Basic Iteration Pattern

```
for (Integer el: x)
  System.out.println(el)
```





Random Access

```
x.get(6)
```

Future Work

CtxList can be automatically substituted for lists processed using iterators. Future work will explore ways to avoid poor random access performance by transforming index-based loops to use iterators instead.
Another avenue is to evaluate the performance gain of this optimization in larger, real-world programs.

[1] J. Meinicke. VarexJ: A Variability-Aware Interpreter for Java Applications. Master's thesis, University of Magdeburg, 2014.
[2] E. Walkingshaw, C. Kästner, M. Erwig, S. Apel, and E. Bodden. Variational Data Structures: Exploring Tradeoffs in Computing with Variability. In Proceedings of the 13rd SIGPLAN Symposium on New Ideas in Programming and Reflections on Software at SPLASH (Onward!), pages 213--226, New York, NY: ACM Press, 2014.
[3] J. Meinicke, C. Wong, C. Kästner, T. Thüm, and G. Saake. On Essential Configuration Complexity: Measuring Interactions In Highly-Configurable Systems. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 483--494, New York, NY: ACM Press, September 2016.