Chapter 1

The Signature Selection Problem

Despite the clever search strategies employed by theory exploration tools, they all rely on enumerating combinations of definitions in some way, which can cause infeasible running times on larger inputs (based on experiments with our Theory Exploration Benchmark, this happens above a few dozen definitions). To avoid this, users of these systems must carefully select only small subsets of their definitions to explore at a time.

We dub such cherry-picking the *signature selection problem* for theory exploration, and we analyse its effect on the running time of theory exploration tools on the Theory Exploration Benchmark, and on the quality of the statements they are able to generate. An automated solution to this signature selection problem would accept a large number of definitions and efficiently select subsets which are both small enough to reasonably explore, whilst still enabling interesting properties to be discovered.

In this chapter we will:

1. Define the *signature selection problem* for theory exploration.

2. Analyse system performance on the Theory Exploration Benchmark, to demonstrate the effect of "toxic" definitions leading to timeouts.

3. Analyse the effect of signature selection on the quantity of desirable properties attainable by theory exploration on this problem set.

4. Demonstrate how the signature selection problem is distinct from the well-known *clustering problem* in machine learning, and how clustering algorithms cannot be directly applied to signature selection.

1.1 The Signature Selection Problem

Although complete, the enumeration approaches used by tools like QuickSpec are wasteful: many terms are unlikely to appear in theorems, which requires careful choice by the user of what to include in the signature. For example, we know that addition and multiplication are closely related, and hence obey many algebraic laws; it would be prudent to explore these functions together. On the other hand, functions such as HTTP parsers and spell-checkers will not be related in many interesting ways, so exploring their combinations is wasteful.

The signature selection problem is that of choosing sub-sets of a large signature, such that:

- Few subsets are selected
- Each sub-set can be explored quickly
- Many of the interesting properties of the original signature can be found in the union of properties of the subsets

There is a clear trade-off to be made between speed (the first two requirements) and "quality" (the final requirement): we can find all of the interesting properties of a signature if we explore the whole thing at once; whilst it is fastest to explore no definitions at all. In between are opportunities for a Pareto-optimal balance to be struck. Whilst the number of subsets can be determined simply by counting, the others require more thorough investigation.

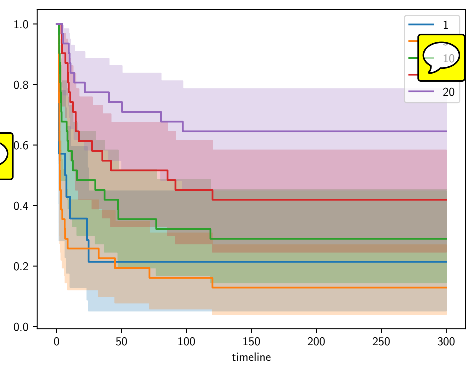
1.2 Exploration Speed

The time taken to explore a subset of definitions from a signature can be easily measured, but only in hindsight. Choosing each subset must be done with only a *prediction* of how long it will take to explore. Here we investigate the time taken by QuickSpec on signatures chosen from our Theory Exploration Benchmark (containing definitions from Tons of Inductive Problems (TIP) benchmark [1]), and how this is affected by their content.

Figure 1.1 shows a Kaplan-Meier survival plot of QuickSpec running times, when given inputs containing different numbers of definitions. Many runs finish quickly, with the remainder occupying a "long tail" which didn't finish within our 300 second timeout[1].

The height of these "tails" is linearly correlated with the number of definitions in the signature, as shown in Figure 1.2. One hypothesis to explain this is the existence of "toxic" definitions, whose presence in the input always leads to the exploration failing (with larger inputs being more likely to have sampled a toxic definition).

[1]Chosen based on preliminary experiments, which showed little difference in survival between 300 seconds and 1 hour



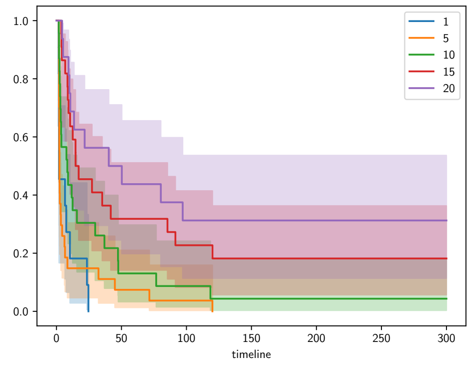


Figure 1.1: Kaplan-Meier survival plot for running QuickSpec on inputs containing various numbers of definitions, sampled from TIP. The x axis denotes running time, which we cut short after 300 seconds. The height of each line shows the proportion of QuickSpec runs which were still going at that time (lower is better). First plot is for all TIP definitions, second plot removes runs given "toxic" definitions.

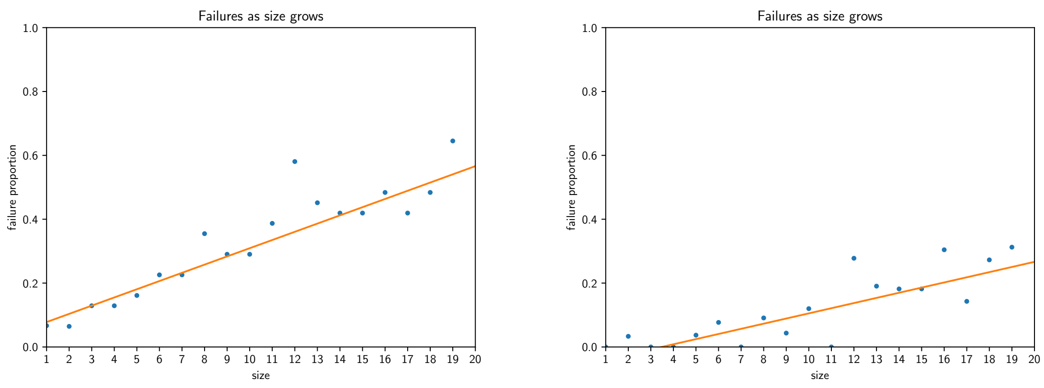


Figure 1.2: Proportion of samples which timed out per size, with least-squares linear regression. First plot is for all TIP definitions, second removes runs given “toxic” definitions.

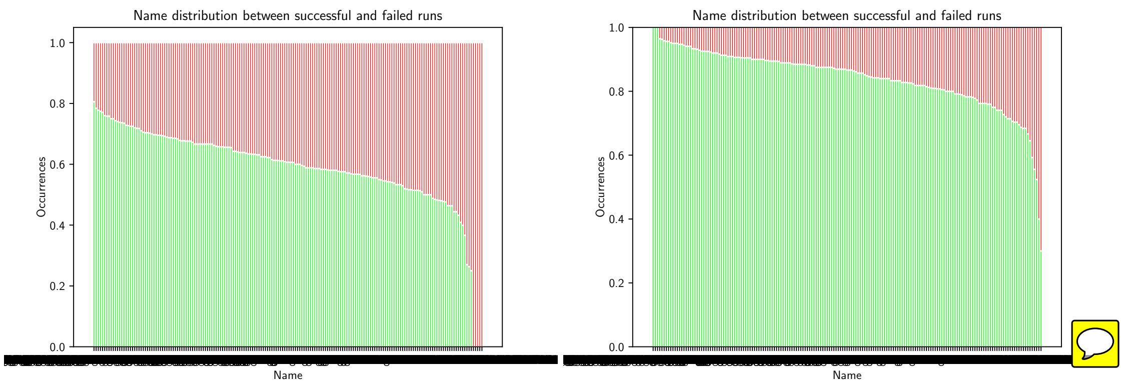


Figure 1.3: Definitions, ordered by the ratio of successes to failures of the runs they appeared in. First graph contains all TIP definitions, showing “toxic” definitions which always failed. Second graph only contains runs without any toxic definitions.

```
(define-fun-rec mult2 ((x Nat) (y Nat) (z Nat)) Nat
  (match x
    (case Z      z)
    (case (S x2) (mult2 x2 y (plus y z)))))

(define-fun-rec qexp ((x Nat) (y Nat) (z Nat)) Nat
  (match y
    (case Z     z)
    (case (S n) (qexp x n (mult x z)))))

(define-fun-rec op ((x Nat) (y Nat) (z Nat) (x2 Nat)) Nat
  (match x
    (case Z
      (match z
        (case Z      x2)
        (case (S x3) (op Z  y x3 (S x2)))))
    (case (S x4)
      (match z
        (case Z      (op x4 y y  x2))
        (case (S c ) (op x  y c  (S x2))))))))
```

Figure 1.4: "Toxic" definitions, which consistently cause QuickSpec to fail. Two other definitions (`mul3` and `mul3acc`) are ommitted due to their verbosity.

To investigate the plausibility of such definitions, the ratio of successful and timed-out runs ~~is~~ plotted for each name in Figure 1.3; showing that five definitions appeared *only* in failing inputs, and seem likely to be "toxic". These definitions are named `mul3`, `mul3acc`, `mult2`, `op` and `qexp`; their definitions appear in Figure 1.4. All of these are functions of Peano-encoded natural numbers (`Nat`), and they cause exploration to time out by either exhausting the RAM with too many expressions to explore, or by generating such deeply-nested outputs that comparing them takes too long.

The `mul3` and `mul3acc` definitions are rather pathological implementations of multiplication with an accumulator parameter, with many (non-tail) recursive calls. The `op` function appears in files named `weird_nat_op`, which assert its commutativity and associativity. Finally, the `mult2` and `qexp` functions are standard tail-recursive definitions of multiplication and exponentiation, respectively. All of these functions have an extra "accumulator" argument, which increases the number of possible expressions to explore compared to those without.

Note that Haskell is lazily evaluated, which allows large structures like Peano numerals to be compared using little memory: the `S` constructors are successively unwrapped from each side, being calculated on-demand and immediately garbage collected in a tight loop. However, this still takes a lot of CPU time and hence causes timeouts. We confirmed this hypothesis by exploring with a custom data generator which only generates the values `Z`, `S Z` and `S (S Z)` (0, 1 and 2), which caused the exploration to finish quickly in these cases. Other interventions, like making the accumulator arguments strict (to prevent space leaks), did not prevent timeouts.

To assess the impact of these toxic definitions, we removed any samples containing them and repeated our analysis: the results are shown on the right hand side of the previous figures. Whilst timeouts remain, they are substantially reduced, and completely eliminated for the smallest sample sizes. Some definitions never appeared in a failing signature.

Our analysis of signature contents on theory exploration performance leads us to two recommendations for signature selection methods:

- Avoid "toxic" definitions: those producing exponentially large outputs, and/or taking many arguments (leading to a large number of combinations). In particular, tail-recursive functions which expose their accumulator argument are harder to explore.

- Smaller signatures are faster to explore, so should be preferred if possible.

It also seems prudent for theory exploration tools to limit time and memory usage, both at the global level (to abort if it becomes clear we are in the "long tail") and at the level of individual evaluations. In particular such fine-grained limits can abort evaluation of toxic definitions, whilst still being lenient enough to allow the majority of non-toxic definitions to be explored unhindered.

1.3 Maintaining Quality

Our final criterion concerns the "quality" of the selected subsets. This depends crucially on how we define whether a property is "interesting". We can use the same ground truth definition as our Theory Exploration Benchmark: a property is "interesting" if it appears in a particular corpus (in our case TIP [1]), and "uninteresting" if not. Whilst naïve, this approach allows the criterion to be quantified: the quality of the selected subsets is the proportion of the whole signature's interesting properties that also apply to the subsets taken individually.

Since signature selection is largely independent of which tool we choose to explore the resulting subsets, we do not require that such exploration actually *finds* all of the interesting properties; only that we have not *prevented* them from being found. For example, we may be given a signature containing addition and multiplication functions and may consider commutativity and distributivity properties to be "interesting". A subset containing both would be a high quality selection, since all of these properties are *available* to find, even if our chosen tool doesn't spot them. In contrast, splitting addition and multiplication into *separate* subsets is lower quality, since this stops *any* tool from finding the distributivity property (since it involves both definitions).

To understand the quality impact of limiting ourselves to such subsets, we applied four signature selection methods to signatures sampled from the Theory Exploration Benchmark. Signatures varied in size from a single definition up to 100; for each size, we sampled 100 signatures.

All four methods were forced to place each definition in exactly one subset, to focus on the effect of splitting apart definitions rather than e.g. duplicating definitions across several subsets or dropping some entirely. The tested methods are:

- **Optimal**: Uses knowledge of the ground truth to choose equal sized subsets with the highest quality.
- **Pessimal**: Like optimal but chooses the lowest quality subsets.
- **Pseudorandom**: Assigns definitions to subsets pseudorandomly, used as a control.
- **Clustering**: Subsets are chosen by a recurrent clustering algorithm, based on their syntactic similarity.

The optimal and pessimal methods are unrealistic, since they require access to the ground truth (which only exists for benchmark problems), but set upper and lower bounds on the impact of separating a signature's definitions. These are implemented via constraint solving, whose exponential running time became infeasible for signatures containing more than 10 definitions. To avoid extreme solutions, such as placing all definitions in one subset and leaving the others empty, we required that all of the selected subsets are non-empty and their cardinality differs by at most one. As a consequence, these methods cannot divide signatures of size N into more than N subsets (since there are not enough definitions to populate every subset without duplicating).

The pseudorandom and clustering implementations are more realistic, since they do not use any information from the ground truth. These methods may also leave some of their subsets empty (since, unlike the optimal and pessimal methods, they are less prone to extremes).

Figure 1.5 shows the impact of selection (specifically, dividing definitions between subsets, with no skipping or duplication) on the proportion of interesting properties that remain available for subsequent theory exploration tools to discover. The shaded region shows the difference between (average) quality, with optimal and pessimal selections starting out equal (when dividing signatures of N definitions into N subsets, where there is no choice other than putting each definition on its own) but quickly diverging as choices become available. Note that the optimal and pessimal methods are constrained to avoid empty or unequally-distributed sets, so they cannot divide up a signature of size N into more than N subsets. Pseudorandom selection is shown as a grey line, and although the cardinality of its subsets is not constrained, it nevertheless appears consistently between the bounds.

Two trends are visible in these charts: firstly the bounded region trends upwards as the signature size increases, which is promising for the application of signature selection to break apart large signatures. Secondly the upper bound reduces as the number of subsets increases: this is to be expected as more definitions must be separated to produce the extra sets, but it demonstrates the tradeoff between quality and subset size. The distance between the pseudorandom and optimal results shows the potential gains to be made by smarter signature selection methods.

The signature selection problem is superficially similar to the *clustering* problem in machine learning: grouping data points based on their similarity across multiple dimensions. To investigate this potential relationship we created a signature selection method based on a *recurrent clustering* algorithm for Haskell code: this flattens syntax trees into vectors, substitutes fixed numbers for keywords and recurses to determine suitable numbers for referenced expressions.

The results of this method are shown in Figure 1.6, along with the pseudorandom method as a control. The results of clustering turn out to be no better than random; in fact the curves show surprisingly similar patt[illegible].

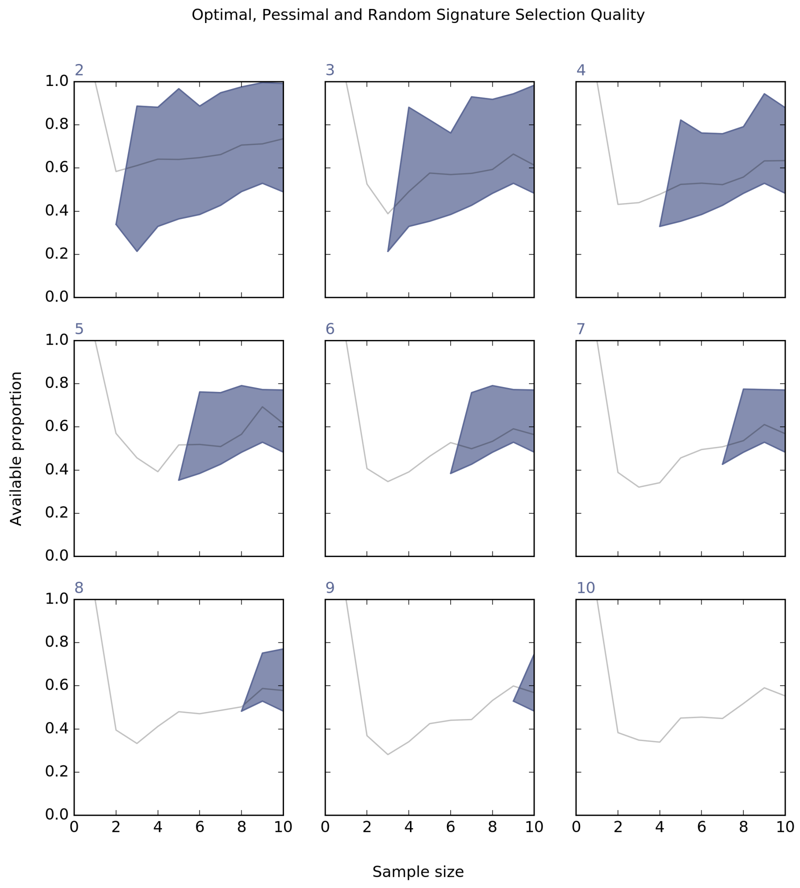


Figure 1.5: The average quality (proportion of ground-truth properties available) when splitting signatures sampled using the Theory Exploration Benchmark into subsets. Each plot shows division into a different number of subsets. The shaded area is bounded by the optimal and pessimal methods, which are can only produce N subsets for signatures of size $\geq N$. Grey lines show pseudorandom selection, for comparison; these extend into smaller signature sizes since empty subsets are permitted.

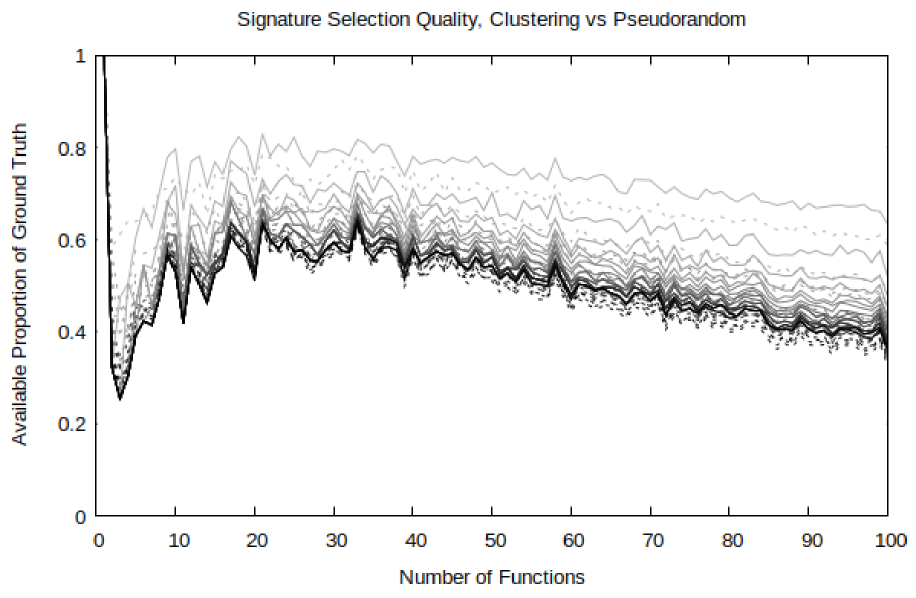


Figure 1.6: Comparison of subset quality when selected by recurrent clustering (solid lines) versus pseudorandom selection (dashed lines). Darker lines show division into a greater number of subsets (between 1 and 20). The solid and dashed lines follow each other closely, indicating that clustering performs no better than random at signature selection.

Bibliography

[1] Koen Claessen, Moa Johansson, Dan Rosén, and Nicholas Smallbone. TIP: Tons of Inductive Problems. In *Conferences on Intelligent Computer Mathematics*, pages 333–337. Springer, 2015.