The Asymptotic Behaviour of the Proportion of Hard Instances of the Halting Problem

Antti Valmari
Tampere University of Technology
FINLAND

1 My Motivation
2 Incomplete Testers
3 Proportions of Easy and . . .
4 Varied Asymptotics
5 Literature Survey
6 Domain-frequency
7 A Typical Hardness Result
8 A Model-Independent Easiness . . .
9 Anomalies Stealing the Results
10 A Difference Between A- and . . .
11 Discussion
1 My Motivation

Does program \( P \) halt on input \( I \)?

The classic (\textit{and correct!}) undecidability proof

- assume that a \textbf{halting tester} exists
- using it, build a \textbf{program} that predicts its own future behaviour and does precisely the opposite to the prediction

\( \Rightarrow \) the prediction is incorrect by construction
\( \Rightarrow \) \textit{halting tester does not exist}

```
void nasty(string P){
  if(!halts(P,P)){
    while(true){}
  }
}
```

Many people feel this proof is cheating, “a rabbit out of the magician’s hat”

- comp.theory noisemakers — ignore them
- Eric C.R. Hehner — serious scientist
- some of my not worst students — \textit{I can’t ignore them!}

I wanted to write another proof that would not create such feelings

- (Halting tester proof for software engineers ..., comp.theory 2012-06-15)
- got interested in this problem area

\( \Rightarrow \) found some new small results in this very classic field
2 Incomplete Testers

Fail on some instances \((P, I)\)

- **3-way tester** — replies “I don’t know”
- **generic-case tester** — the tester fails to halt
- **approximating tester** — gives a wrong “yes” or “no” answer

**Hard instance** = tester fails on it

Examples

- always reply “I don’t know” — absolutely useless but meets the definition
- simulate \(9^9n\) steps, reply “I don’t know” if did not halt or . . . by then

Any 3-way tester can be trivially converted to a generic or approximating tester

For each incomplete tester, the classic proof constructs a hard instance of it

- the tester can be **modified** to handle the instance . . .
- . . . but an accordingly modified **nasty** is hard for the modified tester

\[\text{Every tester has } \infty \text{ many hard instances}\]

\[\text{No instance is hard for every tester}\]
3 Proportions of Easy and Hard Instances

Notation for the number of instances of size $n$ (of tester $T$)

<table>
<thead>
<tr>
<th></th>
<th>easy</th>
<th>hard</th>
<th>altogether</th>
</tr>
</thead>
<tbody>
<tr>
<td>halting</td>
<td>$\bar{h}_T(n)$</td>
<td>$\bar{h}_T(n)$</td>
<td>$h(n)$</td>
</tr>
<tr>
<td>non-halting</td>
<td>$\bar{d}_T(n)$</td>
<td>$\bar{d}_T(n)$</td>
<td>$d(n)$</td>
</tr>
<tr>
<td>altogether</td>
<td>$\bar{p}_T(n)$</td>
<td>$\bar{p}_T(n)$</td>
<td>$p(n)$</td>
</tr>
</tbody>
</table>

*Failure rate* $= \frac{\bar{p}_T(n)}{p(n)} = \frac{\bar{h}_T(n) + \bar{d}_T(n)}{p(n)} = \text{the proportion of hard instances}$

The hope

- the failure rate cannot be made 0, but . . .
- . . . perhaps it can be made small?

It proved interesting to investigate separately $\frac{\bar{h}_T(n)}{p(n)}$ and $\frac{\bar{d}_T(n)}{p(n)}$ as $n \rightarrow \infty$

Why asymptotic?

- failure rate can be made 0 for *any finite set of instances* with a look-up table
  – absolutely impractical and uninformative, but rules out interesting results
4 Varied Asymptotics

Most results in the paper are of the following kinds, with varying assumptions

**Easiness formulae**
- a single ever-improving tester \( \exists T : \forall c > 0 : \exists n_c \in \mathbb{N} : \forall n \geq n_c : \frac{\overline{p}_T(n)}{p(n)} \leq c \)
  - that is, \( \frac{\overline{p}_T(n)}{p(n)} \to 0 \) as \( n \to \infty \)
- a family of better and better testers \( \forall c > 0 : \exists T_c : \forall n \in \mathbb{N} : \frac{\overline{p}_{T_c}(n)}{p(n)} \leq c \)
  - no \( n_c \), because small inputs solved with a look-up table

**Hardness formulae**
- every tester suffers a lower bound \( \forall T : \exists c_T > 0 : \exists n_T \in \mathbb{N} : \forall n \geq n_T : \frac{\overline{p}_T(n)}{p(n)} \geq c_T \)
- there is a common lower bound for all testers \( \exists c > 0 : \forall T : \exists n_T \in \mathbb{N} : \forall n \geq n_T : \frac{\overline{p}_T(n)}{p(n)} \geq c \)

Infinitely often
- the dark blue part is replaced by \( \forall n_0 \in \mathbb{N} : \exists n \geq n_0 \)
- important, because \( \neg \text{ “from some } n \text{ on” } \varphi \iff \text{ “infinitely often” } \neg \varphi \)
5 Literature Survey

Surprisingly few papers were found!

• and many are surprisingly recent
• many are unaware of most others

Diversity of the problem

• model of computation
  – Turing machine — which variant?
  – programming language — frequency/density assumption (next slide)
  – Gödel numbers of recursive functions — \( \approx \) indices of programs

• type of halting problem
  (A) \( T(P) \) tells if \( P \) halts on the empty input
  (B) \( T(P) \) tells if \( P \) halts on the input \( P \), i.e., given itself as its input
  (C) \( T(P, I) \) tells if \( P \) halts on the input \( I \)
    – until now we have discussed (C)
    – with (A) and (B), \( p(n) = \) number of programs of size \( n \)

• failure mode: 3-way, generic-case, approximating
6 Domain-frequency

It is trivial to make many longer identically behaving copies of a program

- if( 000000000 == 123456789 ){ /* put any code here */ }
- bool b = ... something very complicated yielding false ... ; if(b){...}

⇒ we cannot even recognize all the copies nor design a language avoiding them

Many results assume that for any bigger size, many enough copies can be made, but they need not necessarily be fully identically behaving

Example: domain-frequency

∀π ∈ programs : ∃nπ ∈ N : ∃cπ > 0 :
∀n ≥ nπ : π(n)/p(n) ≥ cπ

• here π(n) = # programs of size n that halt on precisely the same inputs as π

• the esoteric minimalistic programming language BF is domain-frequent

• end-of-program maximum density raw data block implies domain-frequency
  – even if inaccessible to the actual code

• whether C++ is domain-frequent has been too difficult to find out!

{char*s="σ"} ⇝ {char*s="σ",*t="ρ"}
7 A Typical Hardness Result

If the programming language is domain-frequent, then
\[ \forall T \in \text{three-way}(B) : \exists c_T > 0 : \exists n_T \in \mathbb{N} : \forall n \geq n_T : \frac{\overline{h_T}(n)}{p(n)} \geq c_T \land \frac{\overline{d_T}(n)}{p(n)} \geq c_T \]
\[ \forall T \in \text{generic}(B) : \exists c_T > 0 : \exists n_T \in \mathbb{N} : \forall n \geq n_T : \frac{\overline{h_T}(n)}{p(n)} \geq c_T \]
\[ \forall T \in \text{approx}(B) : \exists c_T > 0 : \exists n_T \in \mathbb{N} : \forall n \geq n_T : \frac{\overline{h_T}(n) + \overline{d_T}(n)}{p(n)} \geq c_T \]

That is, the proportion of hard instances does not vanish as \( n \to \infty \)

The proof is a modification of the classical one

- given \( T \), all copies of \( \text{nasty}_T \) are hard instances

A generic-case tester with \( \overline{h_T}(n) = 0 \) exists

- simulate the instance until it halts

\( \Rightarrow \) cannot generalize \( \frac{\overline{h_T}(n)}{p(n)} \geq c_T \) to the generic case

A (useless) approximat. tester with \( \overline{h_T}(n) = 0 \) exists, and another with \( \overline{d_T}(n) = 0 \)

- always reply “yes”, always reply “no”
8 A Model-Independent Easiness Result

For each programming model and variant $X \in \{A, B, C\}$ of the halting problem,

$$\forall c > 0 : \exists T_c \in \text{approx}(X) : \forall n_0 \in \mathbb{N} : \exists n \geq n_0 : \frac{h_{T_c}(n)}{p(n)} \leq c \wedge \overline{d}_{T_c}(n) = 0$$

$$\forall c > 0 : \exists T_c \in \text{three-way}(X) : \forall n_0 \in \mathbb{N} : \exists n \geq n_0 : \frac{h_{T_c}(n)}{p(n)} \leq c$$

$$\exists T \in \text{generic}(X) : \forall n \in \mathbb{N} : \overline{h}_T(n) = 0$$

In the approximating case, that means it is infinitely often as easy as you want

Proof for approximating testers [Köhler & al. 05]

- divide $0 \leq y \leq 1$ to strips
- there is the lowest strip $i$ that visits infinitely many times
- for small $n$, reply “no”
- for big $n$, simulate instances until so many have halted that strip $i$ is met, reply “yes” iff given instance halted

We already saw the (trivial) proof for generic-case testers
9 Anomalies Stealing the Results

For Turing machines with one-way infinite tape, it is very easy [Hamkins & al. 06]

- the probability of falling off the left end of the tape → 1, as \(|Q| \to \infty\)

⇒ simulate the machine until it falls off (reply “yes”) or repeats a local state (reply “I don’t know”) 

⇒ \(\exists T \in \text{three-way}(X) : \forall c > 0 : \exists n_c \in \mathbb{N} : \forall n \geq n_c : \frac{\bar{h}_T(n) + \bar{d}_T(n)}{p(n)} \leq c\)

If compile-time errors are counted, it is very easy [Köhler & al. 05], [this paper]

- the probability of syntax error → 1, as \(n \to \infty\)

⇒ reply “I don’t know” if compilation succeeds, otherwise “no”

By tampering the progr. lang., it can be made very easy and very hard [Lynch 74]

Each one is an anomaly stealing the result

- formally true, but does not tell anything about the interesting programs!
- they seem common in this research field
- make it difficult to formulate interesting results
- make it necessary to be very careful with the details of the language, etc.
10 A Difference Between A- and B-types

A program may have lots of information that it cannot access
• comments, junk after the end of a self-delimiting program, ...

If the language allows dense junk, an arbitrarily good empty-input tester exists
\[ \forall c > 0 : \exists T_c \in \text{three-way}(A) : \forall n \in \mathbb{N} : \frac{\overline{h}_{T_c}(n) + \overline{d}_{T_c}(n)}{p(n)} \leq c \]
• reason: as \( n \) grows, a growing proportion of big programs are copies of programs of size \( \leq n \) (yet another anomaly)
• (the claim for B in the paper is wrong, sorry . . .)

A modified proof (not in the paper) of Theorem 7 yields
\[ \exists c > 0 : \forall T \in \text{three-way}(B) : \forall n_0 \in \mathbb{N} : \exists n \geq n_0 : \frac{\overline{h}_T(n)}{p(n)} \geq c \land \frac{\overline{d}_T(n)}{p(n)} \geq c \]
• \( T \) is not in the program, but is obtained from the size of the input
  – if \( |I| \in \{0, 1, 3, 6, 10, \ldots\} \), then \( T \) is \( P_1 \)
  – if \( |I| \in \{2, 4, 7, 11, \ldots\} \), then \( T \) is \( P_2 \), and so on (\( \overline{h}_T(n) \) of any good B-tester oscillates)

So with dense junk, A is strictly easier than B
• intuitive reason: with B, the program gets the junk as part of its input
11 Discussion

There are more theorems in the paper
• even so, the results leave many questions open
⇒ lots of room for future work

Hardness proofs rely on the ability to pack raw data densely
• string constants do not seem dense enough!
⇒ theorems assumed, e.g., any byte string as the input or at the end of program

Many known easiness results arise as anomalies
• uninteresting in themselves, but make it hard to find interesting results

Ideas for future work
• perhaps it would be better to study $\frac{H_T(n)}{h(n)}$ and $\frac{D_T(n)}{d(n)}$?
• [Lynch 74] gives a very strong result, how do its assumptions relate to ours?

\[ \exists c > 0 : \forall T \in \text{three-way}(B) : \exists n_T \in \mathbb{N} : \forall n \geq n_T : \frac{H_T(n) + D_T(n)}{P(n)} \geq c \]

Thank you for attention!