A Record of the Proceedings of SIGBOVIK 2012

March 30th, 2012

Carnegie Mellon University

Pittsburgh, PA 15213

Association for Computational Heresy

Advancing computing as Tomfoolery & Distraction

SIGBOVIK

A Record of the Proceedings of SIGBOVIK 2012

ISSN 2155-0166

March 30, 2012

A lost and rediscovered message from the organizers of SIGBOVIK 2010...

Durnken note frome the SIBOVIK Org committee:

Hey guys,

I love you, man! SIGBOVIK could not be so awesme without your contributios, and thats what it's all about, right? Woooo!

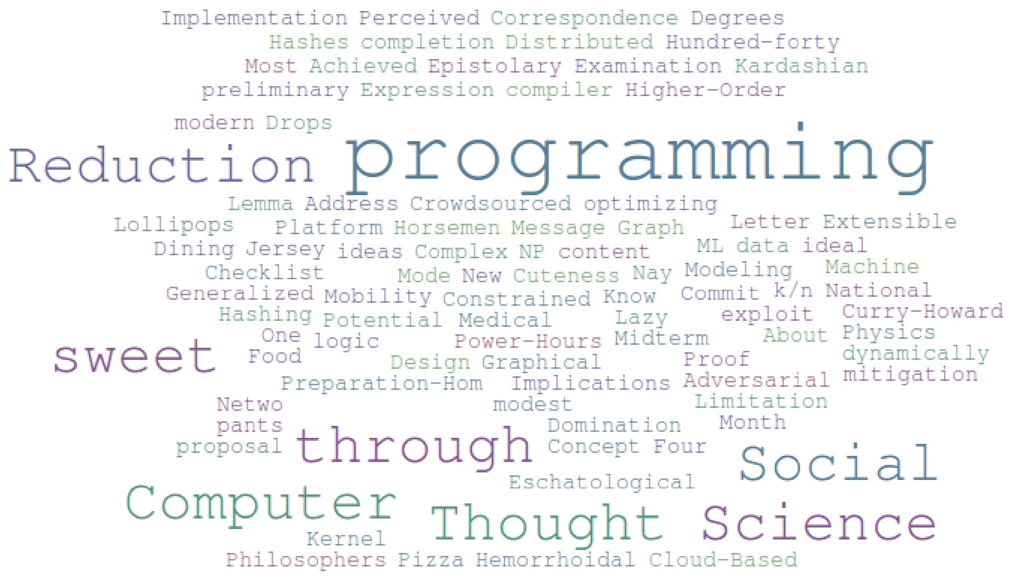
love,
Us

A preemptively lost and rediscovered message from the organizers of SIGBOVIK 2012...

The Association for Computational Heresy (ACH) Special Interest Group (SIG) on Harry Q. Bovik (BOVIK) is ecstatic to host our flagship conference, the first-ever sixth SIGBOVIK, kicking off our series of SIGBOVIK 2012 conference.

In this year's SIGBOVIK, we hoped to return to our roots in Computational and/or Heresy, and we were whelmed by the number of quality papers submitted this year covering one or both of these topics. These papers pioneer new interdisciplinary fields, like computational eschatology, celebrity systems, and drinking game theory. Others provide fresh perspectives on security, logic, and patent law.

Despite the monoculture of diversity that SIGBOVIK represents, several themes emerge from this year's conference, including: Reduction, programming, Thought, Science, and sweet. Because of visualizations, we provide a tag cloud to illustrate these themes:



Implementation Perceived Correspondence Degrees
Hashes completion Distributed Hundred-forty
Most Achieved Epistolary Examination Kardashian
preliminary Expression compiler Higher-Order
modern Drops
Reduction programming
Lemma Address Crowdsourced optimizing
Lollipops Platform Horsemen Message Graph Letter Extensible
Dining Jersey ideas Complex NP content ML data ideal
Checklist Mode New Cuteness Nay Modeling Machine
Generalized Mobility Constrained Know Commit k/n National
Hashing Potential Medical Lazy exploit Curry-Howard
One logic Power-Hours Midterm About Physics
sweet Food Design Graphical Proof dynamically
Preparation-Hom Implications Adversarial mitigation
Netwo modest Limitation
pants Domination Month
proposal through Concept Four Social
Eschatological
Computer Thought Science
Kernel
Philosophers Pizza Hemorrhoidal Cloud-Based

This year's tracks have been laid down by our expert Proceedings Jockeys, and a karaoke version may be obtained by omitting the middle two tracks. To avoid inadvertent favoritism due to the order these tracks appear, we explicitly state that the tracks have been ordered alphabetically from good to non-good.

TABLE OF CONTENTS

Track 1: Official 2012 Apocalypse Track

- *An Eschatological Approach: The Four Horsemen of Computer Science* 3
  The Reverend Nicolas A. Feltman and Joan E. Chao, Mother Supremum
- *World Domination Through the Use of a Graphical Reduction of the Six Degrees of Separation Concept with Potential Robot Design for Mode of Implementation* 11
  L. S. Berg and P. J. Barton

Track 2: Comestibility theory

- *Food for Thought: Dining Philosophers* 15
- *Higher-Order Generalized Algebraic Pizzas* 19
  ~~Rose Bohrer~~ and Samir Jindel
- *Lollipops and Lemma Drops: the sweet, sweet logic of candy* 23
  William Lovas
- *Algorithms for $^k/_n$ Power-Hours* 29
  Ben Blum, Dr. William Lovas, Ph.D., Chris Martens, and Dr. Tom Murphy VII, Ph.D.

Track 3: Brought to you by the letter...

- *TBD* 35
  Taus Brock-Nannestad and Gian Perrone
- *The Letter* 37
  Frederick J. Mespütchy
- *Proof of P = NP* 41
  Samir Jindel and Rose Bohrer
- *An Epistolary Reconstruction of the Curry-Howard Correspondence* 43
  Ben Blum and Michael Sullivan
- *The Kardashian Kernel* 49
  David F. Fouhey and Daniel Maturana

Track 4: Did you bring enough to reshare with the class?

- *Implications of Constrained Thought Expression Achieved via a One Hundred-forty Character Message Limitation Applied to Complex Social Netwo* 57
  Nathan Brooks and Tommy Liu
- *The Spineless Tagless Tweet Machine: Distributed Cloud-Based Social Crowdsourced Lazy Graph Reduction on the Web 2.0* 59
  Michael Sullivan
- *SIGBOVIK 2012 Take-Home Midterm Examination* 61
  James McCann
- *The National Month Of Pushing Spacebar* 65
  Tom Murphy VII
- *What Most Medical Students Know About Computer Science* 75
  Brian R. Hirshman

Track 5: Programming languages research and other games for children

- *Modeling Perceived Cuteness* 79
  Nathan Brooks and Evelyn Yarzebinski
- *i-PHONE app stor : where is my pants* 81
  Dr. Tom Murphy VII, Ph.D. and Dr. Abigale G. Lade, Ph.D.
- *An Extensible Platform for Upwards and Sidewards Mobility* 83
  David Renshaw
- *A modern optimizing compiler for a dynamically typed programming language: Standard ML of New Jersey (preliminary report)* 85
  Ian Zerny
- *Programming Language Checklist* 95
  Colin McMillen, Jason Reed, and Elly Jones

Track 6: Protect yo' shit

- *Cryptographically-Strong Hashing with Physics* 101
  James McCann and Fake Otherauthor
- *The Physics-Based Hashes of Twelve Really Good Ideas* 105
  Patent Troll
- *A modest proposal for the purity of programming* 107
  Robert J. Simmons and Tom Murphy VII
- *Preparation-Hom as the ideal completion of a Hemorrhoidal category* 115
  Sitsin Cømfort and Minrøv Gørondt
- *A Randomized Commit Protocol for Adversarial - Nay, Supervillain - Workloads* 119
  Ben Blum
- *Address space content randomization: exploit mitigation through data randomization* 123
  Carlo Angiuli

Track 1

Official 2012 Apocalypse Track

1. An Eschatological Approach: The Four Horsemen of Computer Science

   The Reverend Nicolas A. Feltman and Joan E. Chao, Mother Supremum

   Keywords: Eschatology, Horsies, Indulgences

2. World Domination Through the Use of a Graphical Reduction of the Six Degrees of Separation Concept with Potential Robot Design for Mode of Implementation

   L. S. Berg and P. J. Barton

   Keywords: Six Degrees of Separation, Kevin Bacon, Turtle Power, World Domination, Bad Cartoon Jokes, Topology, Reduction, Depression, Graphical Model Abuse

An Eschatological Approach: The Four Horsemen of Computer Science

The Reverend* Nicolas A. Feltman
Joan E. Chao, Mother Supremum†

March 30, 2012

1 Abstract

In this paper, we propose a novel eschatological framework for analysis of the endtimes in computer science. Based on citations from the Subroutine of Dijkstra, we identify and postulate the coming arrival of four horsemen: Noise, Compatibility, Bottlenecking, and Exponential Explosion.

2 Introduction and Background

It has been well observed that Computer Science has somewhat of an unhealthy reliance on assymptotics (having previously eschewed Mathematics' solicitations to try amphetamines). Applying this pattern to the temporal dimension yields the field of Eschatological Computer Science. In general, eschatological approaches have been steadily gaining ground, mostly because they all sound pretty cool and usually promise deliverence from the toils of our individual problems.

Piousbottom's seminal[1] work in Eschatological Computer Science laid the foundation for the style of analysis to be used by the rest of the field. It also put forth The Cthulhu Hypothesis, that the endtimes will involve something

*Ordained by the Universal Life Church Monastery.
†Currently at Our Lady of Perpetual Computation.
[1]It was developed at a seminary.

big, angry, and otherworldly. This paper supports a variant of that theory, the Multi-Cthulhu Hypothesis, that the endtimes will involve multiple things, each of which are medium-sized, angry, and otherworldly.

3 Methods and Texts

This paper involves analysis of the Subroutine of Dijkstra, which was originally written in ALGOL, then tranlated to COBOL, then FORTRAN, then λ-calculus, then COBOL again, then C, then C++, then Java, and finally to English[2]. The text itself was originally discovered in punch card form in the basement of the Alamo, and thereafter adopted into cannon and canon (in that order) by the Association for Computational Heresy[3].

Figure 1: In the future, everything's in the cloud.

[2]Ha et al. "MLA-style Programming," SIGBOVIK 2011.
[3]a surpisingly reverent organization

110:10	I saw that the Lambda acquired one of the seven mutexes, and I heard one of the four daemons saying, as with a voice of thunder, “Come and see!”
110:20	And behold, a fuzzy white horse, and he who sat on it had a sensor. An intricately-detailed crown was given to him, and he came forth obfuscating, and to obfuscate.
110:30	When he acquired the second mutex, I heard the second daemon saying, “Come!”
110:40	Another came forth, a wooden red horse. To him who sat on it was given power to take elegance from the field, and that they should code against one another. There was given to him a great codex of specifications.
110:50	When he acquired the third mutex, I heard the third daemon saying, “Come and see!” And behold, an asymmetric black horse, and he who sat on it had a balance in his hand.
110:60	I heard a voice in the midst of the four daemons saying, “A megabyte from disk for a millisecond, and three megabytes from RAM for a millisecond! Don’t damage the FLOPS!”
110:70	When he acquired the fourth mutex, I heard the fourth daemon saying, “Come and see!”
110:80	And behold, a pale horse, and he who sat on it, his name was Exponential Explosion. Hueristics followed with him. Authority over one fourth of the field, to intractablize with the combination, with integer optimization, with SAT, and by all the evaluation states of the program was given to him.

4 Analysis

Above is excerpt from the Subroutine of Dijkstra, Punched Card 110_2. Some scholars argue for a literal interpretation of the text, whereas others posit that the horsemen are symbolic references to concepts defined elsewhere. We will employ the latter approach. The text suggests four horsemen. For the less imaginative, graphical aids have been provided.

- **Noise**, riding a white horse. This horseman likely symbolizes the sensor noise that often arises with any real-world measurement, as well as the generally difficult problem of generating believable natuural clutter in computer graphics.

Forward, not backward!

- **Compatibility**, riding a red horse. This horseman represents the difficulty of supporting various standards, protocols, and interfaces while simultaneously seeking progress and innovation.

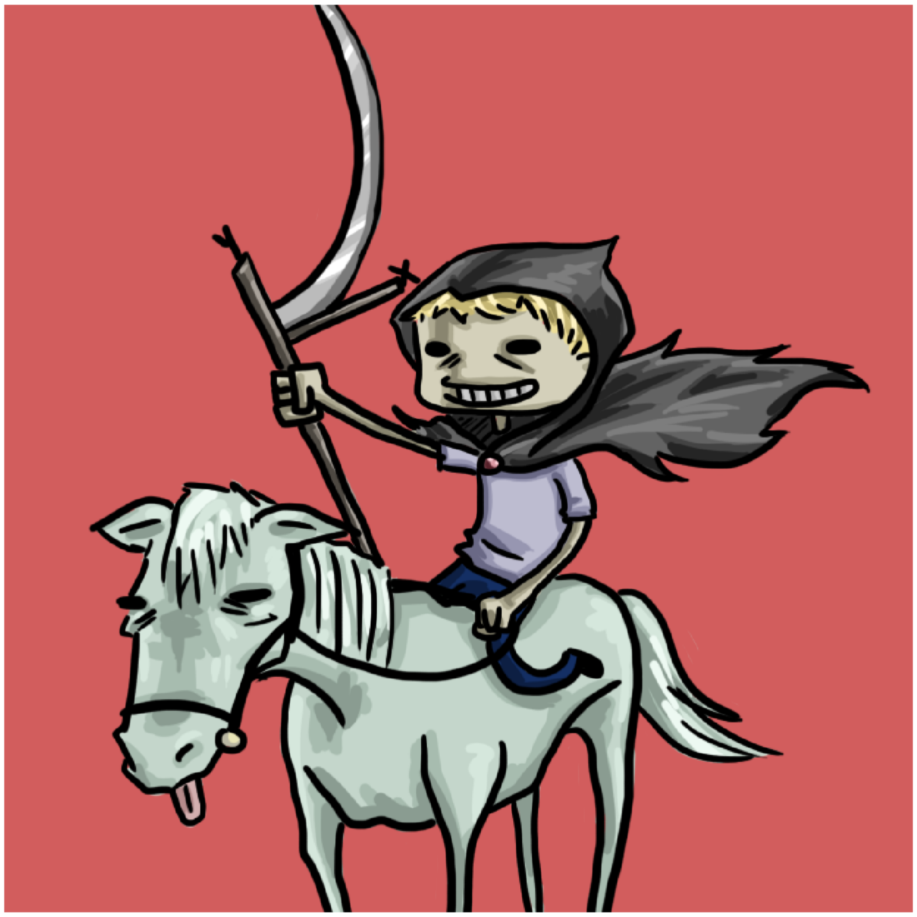
He's angry at you right now. You know what you did.

- **Bottlenecking**, riding a black horse. This horseman alludes to the process by which a few limiting bottlenecks have arisen and completely determine the performance of their systems[4].

Looks like *someone* missed a cache.

[4]Urban Dictionary has a competing definition.

- **Exponential Explosion**, riding a pale horse. This horseman is a symbol for the tendency towards exponential growth, stemming from the conjunction of combinatorial growth and non-composability.



All problems start small.

At best, it is unclear what happens once the horsemen arive. You'd think we could have just read more of the source text, but we were using this open source interpretter, and it kept hitting some error on line 90. We tried to report it, but the guys were all, "Known bug. *Why don't you fix it?*" So we gave up.

5 Simultaneous Work

Simultaneous work in Computational Ancient Poetry has given rise to a unified Ragnarok theory involving the destruction of nearly all of modern computer science. This is, of course, all a bunch of superstitious hokum designed to deter you from the true path. If you've fallen prey to this devilish mode of thought, consult my past work on Computational Indulgences.

World Domination Through the Use of a Graphical Reduction of the Six Degrees of Separation Concept with Potential Robot Design for Mode of Implementation

Berg, L.S.
Corresponding Author: Barton, P.J.
Chatham University in Communication with Carnegie Mellon University, Mellon Library, Mellon Institute, 6 ½ Floor, Office π/∂
(for directions, please ask security on 3rd floor Bellefield entrance)

Submitted for presentation March 19, 2012; Accepted for presentation March 19, 2012

ABSTRACT

Through the use of topological analysis of the Six Degrees of Separation Model a strategic action plan was developed for world domination by utilizing graph reduction through the use of religious deities, internet interaction, and Twitter. Based on survey response, we constructed an optimized robot design for world domination using the reduced graphical model. The robot designed, Chandratron (beta), shows great promise in its ability to infiltrate and dominate the human race. Future improvements to the Chandra-tron will include enhanced communication, IP logging, and retractable jet pack.

INTRODUCTION

The Six Degrees of Separation is a complex graphical model suggesting that for any two completely independent human nodes, there will be an optimized graphical path with a maximum of 6 lines between the two through acquaintances. Suggested by Frigyes Karinthy in 1929, this model requires the assumptions of a simple path forming no cycles between the two nodes, but the potential for Hamiltonian cycles to form when additional individuals are added to the network.[i] In 1961, the late Michael Gurevich conducted an empirical study on the graphical structures of social networks, concluding that "in a U.S. sized unstructured population, two individuals can contact another by means of two intermediates." In this paper he mentions though no correlation exists between number of intermediates and the age of the individual, there is a distinct correlation in regards to the religious affiliation, the religious affiliation of their acquaintances, their sex, and the sex of their acquaintances.[ii]

However, at this time, the total United States population was approximately 183.6 million, with approximately 1 million living abroad. Based on data from the U.S. State Department and U.S. Census Bureau, the current population of the United States has approximately doubled since then, with 313.2 million currently residing in the country.[iii] In 2010, no data was collected in regards to U.S. Citizens living abroad, though estimates indicate a range of 3.8 – 5.3 million.[iv] In addition, the world population has increased from 3 billion in 1960 to 7 billion in 2012. With this massive change in population,

Strategic world domination schemes have a long history of failure due to the lacking in mathematical preparation, such as the recorded attempts by the emotionally unstable Attila the Hun, the atychiphobic Ganondorf, and the spoiled and narcissistic Alexander I. The most recent failure, after many attempts, was a sentient lab rat and his cage mate during the 1990s.[v] Luckily, with the emergence of new analytical avenues for social networking theories, Stanley Milgram's "Small-World Problem" can be reviewed and corrected as a now "Too Many People Are On the Internet" problem.

For any individual hoping to strategically plan world domination, a maximally reduced model based on concatenation of all relevant data must be generated. In addition to this model, a robot must be designed to carry out a plan utilizing the maximally reduced model. In the following paper, we will demonstrate our optimally reduced model in addition to designing a robot for use in world domination.

METHODOLOGY

Through the use of a free online survey website, we distributed a non-IRB approved survey on "Personal Response to Appearance" as to ascertain what traits were maximally cute, pleasant, and generally enjoyable for people. This data was then used in planning the optimized world domination robot design.

We examined freely available world population data[vi] in order to produce a concatenated general social networking model. This model was then reduced through the use of religious prayer, unethical interrogation, and the spanish inquisition. We then strapped the model's boots because it does not know how to tie them. We reapplied our reduction methods until it became clear that each node had as many stumps as its degree, then disposed of the amputated limbs.

Upon completion of our reduced graphical model, we began incorporating the optimization personality characteristics from the survey into the Artificial Intelligence protocol for the robot. We also began world domination robot design and prototype construction utilizing the optimized physical characteristic survey results.

We tested the robot on a room full of children and childlike people aged 2-36 to check the prototype's effectiveness.

RESULTS and DISCUSSION

Our results and discussion will be presented at SIGBOVIK 2012.

iFrigyes Karinthy. “Everything is Different Now”. Hungary. 1929
iiMichael Gurevich. “The Social Structure of Acquaintanceship Relationships”. Doctorate of Philosophy Thesis. Massachusets Institute of Technology. Department of Economics and Social Science. 1961.
iiiU. S. Census Bureau (1961, 2011) “Vital Statistics in the United States” - free for public access. Google it.
ivClaire M. Smith. “These Are Our Numbers: Civilian Americans Overseas and Voter Turnout”. OVF Research Newsletter, vol. 2, issue 4 (Aug), 2010
vUncyclopedia “World Domination” 2009 Edition.
viFreely available through the use of questionable intranet access, Google, and undisclosed sources. 2012.

Track 2

Comestibility theory

1. Food for Thought: Dining Philosophers

Keywords: hunger-driven devices, morsels, monads

2. Higher-Order Generalized Algebraic Pizzas

Rose Bohrer and Samir Jindel

Keywords: Curry-Pepper Isomorphism, Grease Monad, Monoids in the Category of Delicious, $A^\flat$ Mixolydian

3. Lollipops and Lemma Drops: the sweet, sweet logic of candy

William Lovas

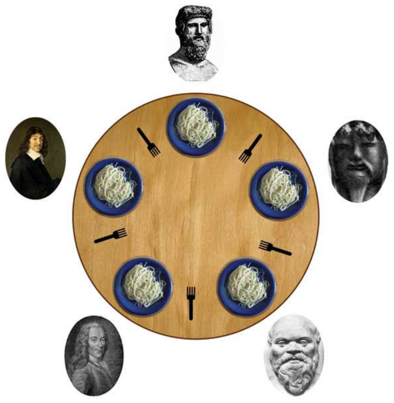
4. Algorithms for k/n Power-Hours

Ben Blum, Dr. William Lovas, Ph.D., Chris Martens, and Dr. Tom Murphy VII, Ph.D.

Keywords: alcohol in computer science, algorithms for the real world, chemically-assisted reasoning, drinking game theory, hyper-driven devices

Food for Thought: Dining Philosophers

April 1, 2012


1

Abstract

The Dining Philosophers problem is ubiquitous in concurrency theory. [1] Most approach the problem with an all-too-transparent computer-science-biased agenda. Our work offers a purely philosophical, hunger-driven solution.

[1] *via* `http://en.wikipedia.org/wiki/File:Dining_philosophers.png`

1 Appetizers

Siddharthachoke dip
Morel relativism
Tuna makiavelli
Jalapeno Poppers
Mock Gödel soup
Secular hunanism (hsun tzoup)

2 Mains

Fettuccini Alfrege
Vegetable Kormagorov
Bayesil pesto
Per Meat-Löf
Quineoa
Searleoin Steak
Cantortellini
Boeuf Sartre-tar
Texmexistentialism
Vennison
Tomato dalai lama
Cannelinihilism
Pressberger
Haskell Curry

3 A la Descartes

French Freuds
B. Russell sprouts
Transcuminism
Balsamic reductionism
Hipocratic oats

4 Dessert

Saul Kripkey lime pie
Neoplatonian ice cream
Fig Newtons and ChocoLeibniz [2]

5 Drink

Hypocratea
Gregory chai tea
Beer from L.E.J. Brouwery (served in a Wittgenstein)
Makers Marx
Three Philosophers

Appendix A: Kids' menu

Hot dogma
Metafishsticks
Aristatertotles
Plato
Maccaronietzcheese

References

[1] *C. A. R. Hoare. Communicating sequential processes. Commun. ACM, 21(8):666–677, August 1978.*

[2] *Randofu. Everything2: Choco leibniz.* "http://everything2.com/user/Randofu/writeups/Choco+Leibniz".

Higher-Order Generalized Algebraic Pizzas

~~*Rose Bohrer*~~ *Samir Jindel*

March 5, 2012

Abstract

Herein we investigate the under-appreciated and probably undercooked field of Higher-Order Generalized Algebraic Pizza. We provide an elegant proof of the Curry-Pepper Isomorphism, show the general case of pizza construction is undecidable, and come to groundbreaking conclusions about the evaluational semantics of Dominos that lead us to propose a different model.

1 Keywords

Curry-Pepper Isomorphism, Grease Monad, Monoids in the Category of Delicious, $A^\flat$ Mixolydian

2 Motivation

One may ask why we choose to investigate the field of pizzas. In fact, many people have asked this very question when ordering Vocelli's late at night. There are several convincing reasons to pursue this field of research:

1. I'm hungry.
2. Judging by how often I see the Domino's guy at Gates, this field is of great interest to our academic community.
3. Most research into the field so far has been by amateurs. Turning a professional eye could bring light to currently unsolved problems.
4. I'm hungry.

3 Curry-Pepper Isomorphism

You may be aware of the Curry-Howard isomorphism relating proofs and programs. You may be less aware of the Curry-Pepper isomorphism, relating pizza toppings to each other. The first known proof is presented below.

We first define the universes of pizzas and ingredients as $\mathbb{P}$ and $\mathbb{I}$ respectively, and a garnishing function $G : \mathbb{P} \times \mathbb{I} \Rightarrow \mathbb{P}$. We also have the sets of curries and peppers, $\mathbb{C} \subset \mathbb{I}, \mathbb{P}^* \subset \mathbb{I}$.

We can now state the Curry-Pepper Isomorphism as follows:

Theorem 1. *The Curry-Pepper Isomorphism.*

$$\forall i_1, i_2 \in \mathbb{C} \cup \mathbb{P}^*, p \in \mathbb{P}, G(p, i_1) = G(p, i_2).$$

Proof. First consider the case where $i_1, i_2 \in \mathbb{C}$. Observe that I'm *not* white and I'm ~~not~~ *very* used to eating curry, so I can't actually tell the difference between the different kinds. That concludes this case.

Next consider the case $i_1, i_2 \in \mathbb{P}^*$. This case may appear complicated, because the universe $\mathbb{P}^*$ includes many types of pepper, including banana, bell, black, and habañero. However we can apply **Axiom 1**: after a while, it all tastes like ~~cholesterol~~ *MSG and late days.* That concludes this case.

Now consider the final case: $i_1 \in \mathbb{C} \oplus i_2 \in \mathbb{C}$. We have already shown that there at most two equivalence classes: curry and pepper. Thus by finding a single curry and a single pepper that are equivalent, we can reduce this to one equivalence class.

We now proceed by example. Consider Sgt. Pepper $\in \mathbb{P}^*$ and Ravi Shankar's Grandma's cooking $\in \mathbb{C}$. Because George Harrison is a Beatle he is equivalent to Sgt. Pepper, and also equivalent to Ravi Shankar since they did the Concert for Bangladesh together. You are what you eat, so Ravi Shankar is equivalent to his grandma's cooking. By transitivity of ~~yum~~ *pun*, our theorem is complete. □

4 Pizza-Paper Isomorphism

To establish an isomorphism between papers and pizzas we find a mapping from paper titles to pizzas and one from pizzas to papers. We will handle the trivial case first:

4.1 Mapping from Paper Titles to Pizzas

Here we show that it is possible to find a mapping from paper titles that are syntactically valid under reversal to pizzas.

Proof. To construct a pizza P from a paper title T, construct a graph G in which the vertices V are the buzzwords B of T, and two buzzwords B_1 and B_2 are adjacent $\{B_1, B_2\} \in E$ iff they are adjacent in the T. Next, we choose a set of ingredients I and map each buzzword B to a set $S \subseteq I$ of the ingredients. We then construct the pizza P by mapping the each buzzword B to a particular slice K such that if $\{B_\alpha,B_\beta\} \in E$, the slices of pizza K_α and K_β they map to are adjacent. We then decorate each pizza slice K with the set of ingredients S to which its corresponding buzzword B maps. □

4.2 Mapping from Pizzas to Paper Titles

Likewise, for any non-empty pizza, we can find a corresponding unordered pair of paper titles that remain syntactically valid under reversal.

Proof. Assume an injection $b : \mathbb{N} \to \mathbb{B}$, where every element of $\mathbb{B}$ is a buzzword.

As a simplification, we assume a syntax under which all titles are valid. This reduces the problem to finding one title that corresponds to a given pizza.

First we convert our pizza to a connected, simple, weighted, undirected graph. Each set of ingredients is represented by a vertex v, and two vertices v_1, v_2 are connected by an edge $e(v_1, v_2)$ iff there exist two adjacent slices s_1, s_2 such that s_1 contains exactly the ingredients for v_1 and s_2 contains exactly the ingredients for v_2. The weight of the edge is the ingredient distance $d(e(v_1, v_2))$, which is the number of ingredients you must add or remove to get from one set to the other. Note we can treat the graph as undirected because adjacency is symmetric and $d \circ e$ is commutative. Furthermore we can reduce it to a simple graph because d depends only on v_1, v_2, not s_1, s_2 (i.e. any two edges between a given pair of vertices would have the same weight). The graph is connected by the obvious observation that pizzas are also connected, at least before you eat most of them.

Given this graph, we find a title. We know the graph has at least one vertex because the pizza was non-empty by assumption. If the graph contains one vertex, we map it to the empty title. Otherwise, since the graph is connected and undirected, there will exist at least one cycle. We take an arbitrary cycle of minimal weight (there may be more than one). We then choose an arbitrary starting vertex, and iterate through all the other vertices in the cycle. Each time we traverse an edge $e(v_1, v_2)$, we append the buzzword $b(d(e(v_1, v_2)))$ to our partial title t_p. When we reach the end of the cycle we have a title t. Because we assumed a grammar under which all titles are valid, its reverse t_r is also valid, giving us the desired pair of reversible titles (t, t_r). $\square$

5 Non-Strict Semantics of Domino's

We represent the state of Domino's kitchen as a *priority* queue into which we can enqueue functions returning pizzas (orders), and dequeue the actual pizzas once Domino's says they should be ready. That is, dequeueing is the action of yelling at the pizza guy to give you your pizza, and function application is the cooking of a pizza.

We assume Domino's is implemented as immediately applying each order and storing them in the queue for retrieval. This implementation has been confirmed by the Vendor[1].

Then, under a strict evaluational semantics, the pizzas should be immediately available for consumption *biased by their price.* Given the fact that pizzas

[1] *citation needed*

take some time to execute, we cede that they might not be available immediately, but it should at least be guaranteed that ordering is preserved: If I provide a higher-order pizza, the result should be computed before computing any lower-order pizza.

Extensive experiments have proven this assumption to be false. It is often observed that pizzas can be evaluated immediately, way after other pizzas, or even never *because you only get cheese, you cheap bastard*.

We thus conclude that Domino's has a non-strict semantics, and is implemented in Haskell *(because Haskell is the only lazy language)*.

Furthermore, we note that CM Café has the same behavior, and this being CMU, it should not be implemented in Haskell, but *OCa*~~S~~ML. Not only is it good for ~~school~~ *PL* spirit, but I would be a lot less hungry.

Lollipops and Lemma Drops

the sweet, sweet logic of candy

William Lovas

March 20, 2012

Abstract

Based on observations of real-world candy practice, we derive real-math candy theory, with the hopes of spawning a decades-long research programme into newer and tastier forms of confection. Along the way, we cite several relevant tweets and play merry hell with BibTeX.

1 Motivation

There is a delicious children's candy that takes the form of brightly-colored sugar shells succulently and lovingly spray-wrapped onto highly eccentric oblate spheroids of grade "A" dark chocolate, known in the common parlance as `m&m`'s. The candies are identified by being stamped with a white letter `m` on one side of the shell, primarily to distinguish them from a very similar candy that only exists in Canada (*i.e.*, a "canady") called Smarties, not to be confused with the wafer sugar confectionery made by Ce De Candy [con11]. This of course begs the question: *what happened to the other* `m`*?*

We propose a logical solution to the problem, expositing a theory whereby candymakers can save time and space by exploiting logical equivalences [LH12], thus bridging the gap between logical theory and logical practice, with delicious consequences. We furthermore explore a variety of future avenues where this strategy can be fruitfully applied towards the invention of newer and tastier candies, thus placing the entire confection industry on a sound, productive, and logically-motivated foundation not unlike the Curry-Howard correspondence for programming languages [How80, aPCCaeNFL12].

2 The M&M-M correspondence

Observe that logically, $m\&m \equiv m$. This is easy to prove, whether & refers to the ordinary "and" of classical and intuitionistic logics or the hipper and more modern "with" of the 1980's power-ballad *linear logic*. The projection-retraction pair so-formed is typically known as the *manufacturing-eating* cycle, and it keeps the M&M-Mars company operating lucratively.[1]

2.1 Correspondence... or more?

Many would-be academes like to bicker and argue about whether Curry-Howard is a mere *correspondence* or a true *isomorphism*. The question of nomenclature is both historical and technical in nature, and although its importance is undeniable, a full discussion of all the relevant positions and their consequences is unfortunately outside the scope of the present work. But in keeping with the highest standards of academic integrity commensurate with a publication venue as prestigious as SIGBOVIK, we here evaluate our own work in this regard.

The M&M-M correspondence is a correspondence between the logical propositions $M\&M$ and M, parametric in M. For it to be a proper *isomorphism*, it must be the case that the two directions of the correspondence when composed together in either order form an identity map. The calculations are routine and uninteresting, so we leave them to future work, but suffice it to be said that it is difficult to manufacture a particular M&M, once eaten.

3 Whence `m`? And a potential new favorite

So we know by the M&M-M correspondence that $M\&M \equiv M$, parametric in M, and thus that $m\&m \equiv m$. But why *exploit* this equivalence in the printing of candies?

The answer is simple: to save space and manufacturing costs. Oblate spheroids are not only pleasing to both the touch and the tongue,[2] but also quite easy to manufacture! By contrast, *prolate spheroids* are neither a proper fit for the mouth—it would have to balance precariously on one end (see Figure 1)—nor for production by machine—and yet, that precarious shape would be required to provide sufficient surface area for printing the full name of the candy! Logical equivalence both improves aesthetic value and enables mass production.

To a similar end, we propose a new candy—not spherical nor even spheroidical, but rectangular, and in fact, square. The complete and proper name of the candy is $A \supset A$, but to enable manufacturability and improve the consumer experience, the candies will simply be printed with the equivalent proposition $\top$. An added benefit not enjoyed by M&M's is that via rotation, $A \supset A$'s may be experienced in at least four distinct ways (see Figures 2 and 3).

[1]It is conjectured that the singular `m` found on the M&M candies actually refers to *the other m, i.e.,* Mars, but these rumors are mostly baseless, as M&M's come in many colors besides red.

[2]Not to mention ideally suited to the behavior of melting in the one and not in the other!

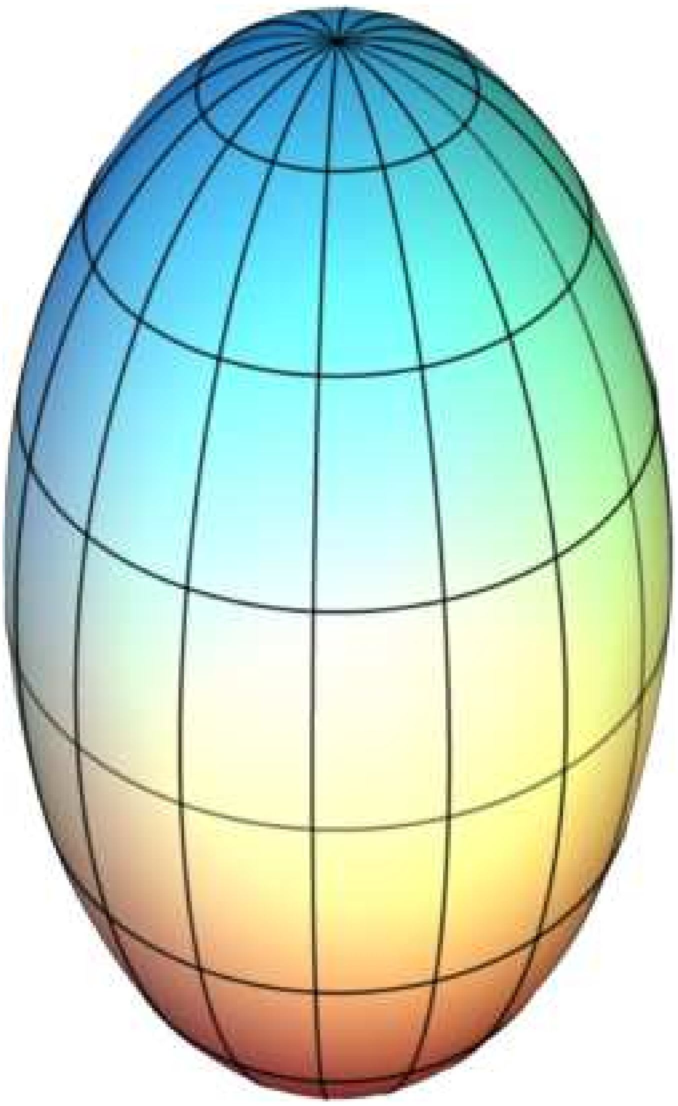


Figure 1: a 3&3 standing on end (image: Wikipedia; made by AugPi).

$$\boxed{A \supset A}$$

Figure 2: $A \supset A$'s: discarded concept sketch.

$$\boxed{\top} \quad \boxed{\dashv} \quad \boxed{\bot} \quad \boxed{\vdash}$$

Figure 3: $A \supset A$'s exploiting logical equivalence, in four pleasing orientations.

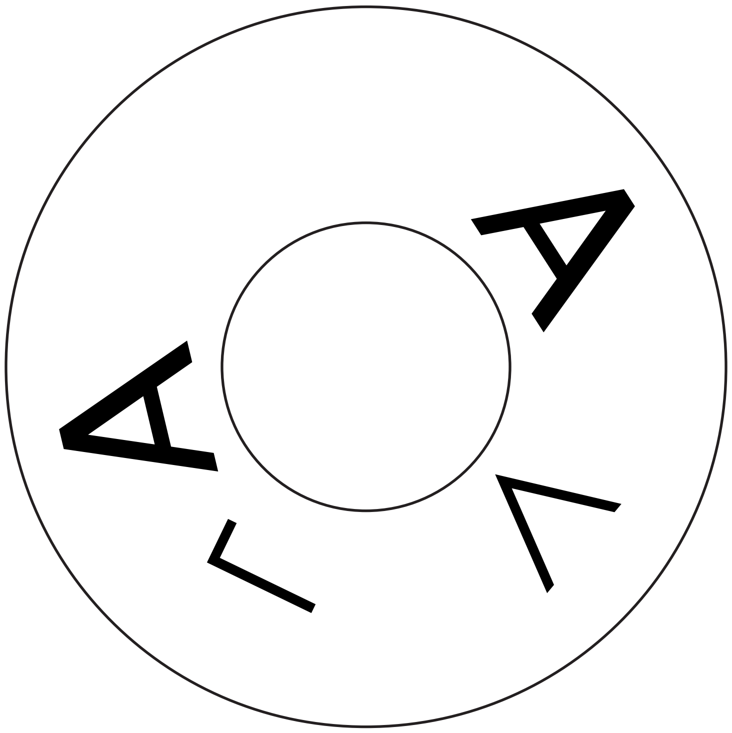


Figure 4: Would you eat that doughnut? Really?

4 Further extrusions: A classic sweet

Not all enjoyable sweets are candy: many favored confections existed before the advent of the modern industrial era of Big Candy. One classic sweet dating back to the mid-19th century, and popularized in modern form by such household alliterations as Krispy Kreme and Dunkin' Donuts, is the *dough of the excluded middle*. Logically, this breakfast favorite can be represented as $A \vee \neg A$, the eponymous law, but if one attempts to print the cake's name on the cake itself, one finds that the name is just too short—there's all kinds of extra space, which is *really* hard on the eyes, and assuming the printing is done in frosting or something, fully half the bites would be dry and tasteless (see Figure 4). I mean, just look at it! It's preposterous.

Employing logical equivalence in the opposite of the usual direction, though, we can *expand* $A \vee \neg A$ to the well-known equivalent, the heavily left-nested *Peirce's law*: $((A \supset B) \supset A) \supset A$. The expanded proposition now fits on the confection in an aesthetically pleasing way, with an even distribution of frosting throughout (see Figure 5). Succulent!

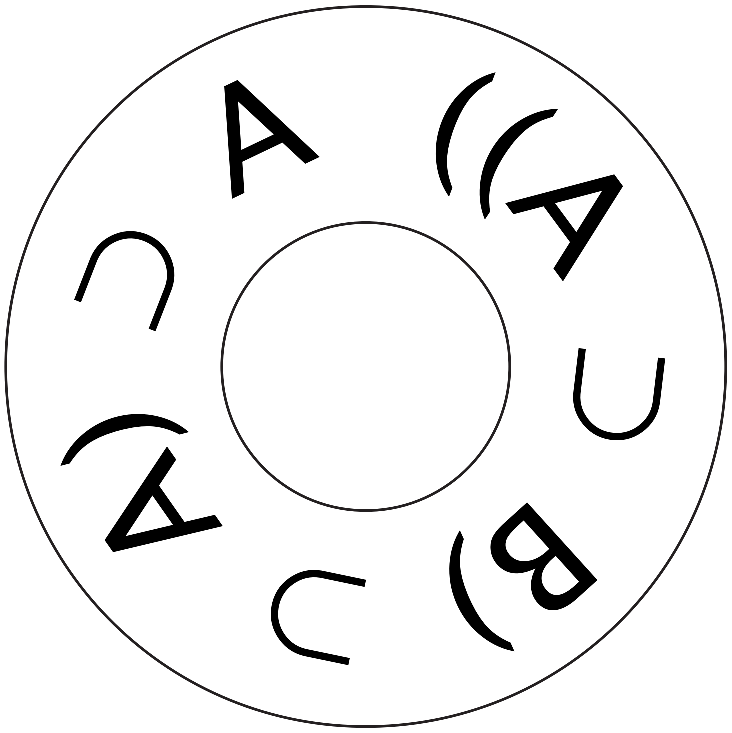


Figure 5: Mmm... heretical and higher-order.

5 Previous work

Earlier work in this area [cc, (@p] fell prey to the extraordinary fineness of logical distinctions typified by linear logic: it was irreparably unsound, due to the two different forms of truth in linear logic—**1** and ⊤—neither of which is actually equivalent to provable propositions. We have corrected and expanded on the underlying idea in Section 3, above.

6 Future work

The world is full of logics, so there are many fruitful directions we envision taking this work in the future. For instance, where did the hole go?[3] What about those lollipops?[4] And can we actually employ fruit in candy?[5]

[3] Answer: constructive logic!

[4] Formally, ⫯ [Ph.10]

[5] An earlier draft of this paper was entitled, "... *Towards* Lemma Drops", but hey, no one ever got tenure by *underselling* their work...

7 Acknowledgements

Dr. Tom 7, Ph.D generously served as classical concept artist. XLNT!

References

[aPCCaeNFL12] Ben Blum as Priority Class Continuations and Michael Sullivan as η Normal Form Letters. An epistolary reconstruction of the Curry-Howard correspondence. In *Proceedings of the 6th Annual Intercalary Workshop about Symposium on Robot Dance Party of Conference in Celebration of Harry Q. Bovik's 0x40th Birthday (SIGBOVIK 2012)*, Pittsburgh, PA, 2012. Association for Computational Heresy, Verlag & Sons Publishing Co.

[cc] chrisamaphone (@chrisamaphone). "*"I think I'm going to start a line of candies called A -o A's, but just print '1' on the candy." –@xwjl*" 7 Feb 2012, 11:14 pm. Tweet.

[con11] Wikipedia contributors. Smarties (disambiguation). *Wikipedia, The Free Encyclopedia*, 2011. [Online; accessed 18-March-2012].

[How80] William A. Howard. The formulae-as-types notion of construction. In Jonathan P. Seldin and J. Roger Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, Boston, MA, 1980.

[LH12] Daniel R. Licata and Robert Harper. Canonicity for 2-dimensional type theory. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '12, pages 337–348, New York, NY, USA, 2012. ACM.

[(@p] Popular Logic (@popularlogic). "*"I think I'm going to start a line of candies called A -o A's, but just print '1' on the candy." –@xwjl*" 7 Feb 2012, 11:14 pm. Retweet.

[Ph.10] Dr. Tom Murphy VII Ph.D. You keep dying. In *Proceedings of the 4th Annual Intercalary Workshop about Symposium on Robot Dance Party of Conference in Celebration of Harry Q. Bovik's 0x40th Birthday (SIGBOVIK 2010)*, Pittsburgh, PA, 2010. Association for Computational Heresy, Verlag & Sons Publishing Co.

Algorithms for $^k/_n$ Power-Hours

Ben Blum Dr. William Lovas, Ph.D.
Chris Martens Dr. Tom Murphy VII, Ph.D.

1 April 2012

Abstract

Drinking games, while a fun activities, can lead to memory leaks if performed to excess. In particular the Power Hour, in which a shot of beer is drunk every minute for an hour, may be modified to allow potentially arbitrarily customizably safely enjoyable consumption. We sketch some known solutions and avenues for future research. ALSO WE ARE DRINKING RIGHT NOW AND THIS PAPER WAS COMPLETED IN ONE HOUR

Keywords: alcohol in computer science, algorithms for the real world, chemically-assisted reasoning, drinking game theory, hyper-driven devices

1 Introduction

A *power hour*[1] is a drinking game in which participants drink a shot of beer every minute for an hour, usually based on musical cues.

Using the standard of one shot = one fluid ounce of 4% alcohol, a power hour equals approximately five beers total. For an average-mass human with a well-developed alcohol tolerance, the game results in a pleasant level of inebriation.[2]

[-1]Copyright © 2012 the Regents of the Wikiplia Foundation. Appears in SIGBOVIK 2012 with the permission of the Association for Computational Heresy; *IEEEEEE!* press, Verlag-Verlag volume no. 0x40-2A. £0.00

[1]Not to be confused with power sets, Powerade, Powerpuff Girls, or Mighty Morphin' Power Rangers

[2]Anonymous personal correspondance

However, in some cases, the game may result in uncomfortable levels of inebriation. Participants therefore may wish to reduce the total amount of alcohol consumed while still experiencing the process of collaborative inebration.

If, say, a participant wants to drink half as much as everyone else, what options do they have available? One possibility is to simply drink once every other song. This is problematic for two reasons. The soft reason is that in the spirit of active participation, they would ideally like to take some action progressing their drunkenness at every song change. See 3 for a formal discussion of this condition. A more compelling reason is that humans in a state of ever-increasing inebration probably cannot remember whether or not they drank last time due to impaired reasoning abilities[3] Therefore, they must, as a finite state automoton, determine their course of action based solely on current state.

A known solution for the *half* power hour is to take a different action based on shot glass state: *fill* when the glass is empty and *drink* if the glass is full. This elegant solution achieves the goal of consuming half as much alcohol by the end but requires the participant only to observe the most recent state, then change that state in a single action.

The aim of this work is to generalize the 1/2 power hour to general k/n (with m participants).

We provide some preliminary results, but primarily we pose a call to action suggesting various avenues of research.

[3]A non-judgmental reconstruction of drunken logic. Robert J. Simmons. SIGBOVIK 2007. April 2007.

2 Desiderata

3 Desiderata

There are several properties we would like an algorithm to satisfy in order to be considered a proper power hour algorithm. In this section, we enumerate these desiderata along with illustrative examples that violate—and thus motivate—them. Without constraints, the space of potential power hour algorithms is too large to be meaningfully analyzed and understood; these desiderata serve to limit the space of possible algorithms to those that are sufficiently simple and adequate to be implemented in a real-world context.

In what follows, a power hour *player* is tasked with taking an *action* each *turn*. Typically, a *turn* occurs every minute. Each *action* involves some observation of the current state and some change of state. The classic power hour algorithm is for each action to be: *observe* the empty shot glass in front of you, *fill* that shot glass with beer, and *drink* that shot glass, reestablishing the state invariant for the next round.

3.1 Discretion

The first and most important desideratum is that of *discretion*: each player should drink only *one* shot per turn. Anyone who has participated in a power hour realizes the difficulty of drinking even a single shot late in the game, and we wish to exclude algorithms that require a player to exceed these natural human limits.

An obvious violation is the 2-power hour (= 120/60-power hour), where a player must pour and drink two shots each turn.

3.2 Simplicity

Relatedly, a power hour algorithm should consist of only *simple* actions. There is some leeway in defining precisely what it means for an action to be "simple", but the purpose of this constraint is to ensure maximal physical safety and minimal broken glass (desires which are in synergistic harmony), so for example, inverting a full cup is disallowed, since the spilled beer causes a sense of alarm, and stacking a cup on an up-cup is unpermitted (*e.g.*, $\overset{\cap}{\cup}$), since an up-cup does not provide sufficient foundation for safe stacking.

3.3 Locality

For practical purposes, a player participating in a proper power hour can perform only very simple *observations*: to that end, we posit the desideratum of *locality*: a player's action may depend only upon observing the state of the shot glass directly in front of them, and not, say, the state of some other player's glass or some written memory. This desideratum represents a kind of "memory safety": players need not have too much memory from turn to turn, since in our experience, they won't.

3.4 Singularity

Simplicity and locality together suggest the desideratum of *singularity*: a player should have at most one shot glass in front of them at any given moment. Generalizations are possible: for instance, every shot glass in front of a player must have the same state (*e.g.*, all filled, all empty, all inverted, etc.), but the resultant protocols become prohibitively complex due to the explosion of possible states and the necessity of maintaining state invariants on each action.

3.5 Liveness

Another desideratum is the property of *liveness*: we would like every player to drink infinitely often, in the limit. The goal here is to ensure that every player continues to enjoy in the camaraderie at all times, and to a certain extend to maintain their buzz at a smooth and constant level. Liveness does, however, rule out many potential interesting algorithms like the 0/60-power hour, the 1/60-power hour, etc.

3.6 Extensibility

In addition to bounding the lower limit of a player's drinking (liveness, above), we would like to bound their upper limit: the desideratum of *extensibility* captures this idea. Extensibility dictates that for any generalized extension of the hour to $n' > n$ minutes, there must be some further extension to $n'' \geq n'$ minutes such that after n'' minutes, a player has consumed k'' drinks such that $k''/n'' = k/n$, exactly. In other words, in the limit, a k/n-player must always have drunk k/n times, or be on the way to drinking k/n times. Extensibility rules out algorithms like the 58/60-power hour, the 59/60-power hour, and the 61/60-power hour, where the player has some initial sequence of actions before they enter their main loop.

In the case of non-trivial (*i.e.*, non-zero) power hours, extensibilty is a strictly stronger constraint than liveness, since for any $0 < x \leq 1$, in the limit, a player will eventually have to drink to maintain a fraction of x drinks per turn.

3.7 Asynchrony

In the case of distributed power hour algorithms (as in Section 5.1 and 4), we posit the desideratum of *asynchrony*: a player's action should not depend on any coordination with other players. Formally, and practically speaking, asynchrony requires that a player's *action* during a turn depend only upon the *observations* that player could have made at the beginning of the turn. Otherwise, you know, things just get too, uh... complicated.

4 Known Results

Solo arrangements. In the solo case, we know exactly what power hours are possible and which are not. Let us exhaust these before moving onto the more difficult distributed case.

$^{0}/_{60}$	Trivial, with multiple solutions. Start with $\cup$, never fill it, and never drink.
$^{1}/_{60}$	Many solutions. For example, start with $\uplus$. Drink on $\uplus$, leave $\cup$upon seeing $\cup$.
$^{2}/_{60}$	Start with $\uplus$. On $\uplus$, drink and leave $\cup$. On $\cup$, fill and drink, and leave $\cap$. On $\cap$, do nothing.
$^{3}/_{60}$	Impossible. Would require four distinct states, but there are only three.
$^{4...19}/_{60}$	Even more impossible. XXX is 19 actually impossible?
$^{20}/_{60}$	Drinking one shot out of three. Start with $\uplus$. On $\uplus$, drink and leave $\cup$. On $\cup$, flip to $\cap$. On $\cap$, flip, fill, and leave $\uplus$.
$^{21...29}/_{60}$	Even more impossible. XX are 21, 29 actually impossible?
$^{30}/_{60}$	Drinking every other shot. Start with $\uplus$. On $\uplus$, drink and leave $\cup$. On $\cup$, fill and leave $\uplus$.
$^{31...39}/_{60}$	Super impossible.
$^{40}/_{60}$	Drinking two shots out of three. Start with $\uplus$. On $\uplus$, drink and leave $\cup$. On $\cup$, fill, drink, and flip to $\cap$. On $\cap$, flip, fill, and leave $\uplus$.
$^{41...57}/_{60}$	Totally impossible!
$^{58}/_{60}$	Symmetric to the $^{2}/_{60}$ case. Start with $\cap$. On $\cap$, flip and leave $\cup$. On $\cup$, fill and leave $\uplus$. On $\uplus$, drink, fill, and leave $\uplus$.
$^{59}/_{60}$	Symmetric to the $^{1}/_{60}$ case. Start with $\cup$. On $\cup$, fill and leave $\uplus$. On $\uplus$, drink, fill, and leave $\uplus$.
$^{60}/_{60}$	Easy; this is a normal power-hour. Start with $\cup$. On $\cup$, fill, drink, and leave $\cup$.

Distributed algorithms. Generalizing to the distributed case unlocks many more possibilities. Even for just a small number of participants, it becomes quite difficult to exhaustively explore the possibilities. These algorithms are a class of finite state machines, probably excluding analytical approaches (note that it is not even simple to count the number of possible strategies, since some are illegal because they put more than one cup in front of a player, perhaps in a rare configuration). Here we give some known results to give a sense of what solutions look like.

A problem in the distributed case consists of players $P_1, \ldots P_m$. Each P_i wants to perform a ${}^{k_i}\!/_{n}$ power hour for the same global n, coordinating with the other players.

First, observe that any participant in a distributed setting can use a solo strategy and not interact with the rest of the group. Thus, if k_i is one of the the possible solo cases above, this player can use that strategy if the remaining players are able to solve the smaller distributed problem. This if course includes the case that every player wants to perform a ${}^{k}\!/_{n}$ power hour that is one of the possible solo cases.

With two players it is possible to perform power hours that are not possible solo, however. For example, two simultaneous ${}^{10}\!/_{60}$ performances are achievable as follows. Only use one shotglass. Each player does the ${}^{20}\!/_{60}$ strategy, transitioning $\cap$ to $\cup$, and $\cup$ to $\uplus$. A player receiving $\uplus$ drinks and transitions to $\cap$. In each case, the single shotglass is passed to the other player, cutting the normal ${}^{20}\!/_{60}$ in half by splitting it evenly. $\cap$ to $\cup$ and place it in front of the other player. This is the power of teamwork!!

More complex arrangements are possible, like where you pass to a different dude depending on what orientation the cup is in, and who knows what happens?!

5 Future Work

5.1 Distributed Algorithms

In future research we plan to study distributed algorithms involving more than one person and/or more than one shot-glass. With multiple people cooperating during one power hour, we observe many additional possibilities for tracking the state. Assume that instead of 1 participant with one shot-glass, we have p participants with q shot-glasses among them.

5.1.1 Rotation

5.1.2 Shotglass Interactions

There are also many combinations that may result if we allow for a state to be represented by multiple (presumed indistinguishable) shotglasses. As a basic example, using two empty shotglasses, we can represent three states: $\cup\cup$, and $\cup\cap$, and $\cap\cap$.

If we allow filling one or both of the cups in the former two states, this allows for even more states, but with less possibilities to transition between states without drinking.

If we allow stacking of shotglasses, we enable even more states: $\overset{\cup}{\cup}$, $\overset{\cup}{\cap}$, $\overset{\cap}{\cup}$, and $\overset{\cap}{\cap}$. In total, this allows for seven states with two shotglasses.

This can be extended to arbitrarily high stacks, with absurd consequences. We plan to hire a set theorist to study the interactions of countably infinite and possibly even uncountably infinite stacks.

5.2 Controversy

As discussed in section 3, there are several constraints on the legitimacy of power hour algorithms. In the future we will consider potential algorithms that may result if we relax these constraints.

5.2.1 Multiple Shots per Minute

If we extend the set of possible state transitions to allow the participant to drink multiple times per minute, we enable algorithms in which $k > n$. We write $(A, \ldots Z)^m$ to denote performing the actions A through Z in sequence m times repeated.

The most basic example is to extend the classic algorithm to enable a $m * n$ power hour, as follows. On each tick, (fill, drink)^m, and leave $\cup$.

We can also write algorithms for non-integral irregular fractional power hours. For a $m/3$ power hour (with $m \geq 1$): If $\cap$, flip and leave $\cup$. If $\cup$, fill and leave $\uplus$. If $\uplus$, drink, (fill, drink)$^{m-1}$, flip, leave $\cap$.

However, the algorithm described above provides poor load-balancing in the case of large m. We can solve this problem by distributing the multiple-drinks, as follows. (In the following description, we assume $m \cong 1\mathsf{mod}3$, for simplicity.) If $\cap$, flip, (fill, drink)$^{(m-1)/3}$ and leave $\cup$. If $\cup$, fill, (drink, fill)$^{(m-1)/3}$ and leave $\uplus$. If $\uplus$, drink, (fill, drink)$^{(m-1)/3}$, flip, leave $\cap$.

We postulate that a similar load-balancing algorithm can be applied to any existing conventional algorithm for $k \leq n$.

5.2.2 Non-Extensibility

As discussed in section 4, there are certain algorithms that provide results for k additively dependent on n. The possibilities for these are also greatly expanded given the techniques described above. One example algorithm is shown below.

If $\cap\cap$, stack and leave $\overset{\cap}{\cap}$. If $\overset{\cap}{\cap}$, flip the top cup and leave $\overset{\cup}{\cap}$. If $\overset{\cup}{\cap}$, flip both cups and leave $\overset{\cap}{\cup}$. If $\overset{\cap}{\cup}$, flip the top cup and leave $\overset{\cup}{\cup}$. If $\overset{\cup}{\cup}$, un-stack and leave $\cup\cup$. If $\cup\cup$, fill both glasses and drink, leaving $\cup\cup$.

With one participant using two cups, this causes a 110/60 power hour. With two participants, one drinking from each cup in the final step, this causes a 55/60 power hour.

6 Conclusion

We have presented some algorithms for k/n Power Hours, woohoo!

We wrote this paper in one hour while drinking beer.

7 Cheers

Cheers to Rob Simmons for spreading the knowledge of the original Half Power Hour formulation; to Jamie Morgenstern, Rob Arnold, and Anders "POWAH HOWAH" Schack-Nielsen for providing inspiration in the form of Power Hour Participation; to Ali Spagnola for providing musical accompaniment to our writing sprint.

Track 3

Brought to you by the letter...

1. TBD
Taus Brock-Nannestad and Gian Perrone
Keywords: all is well here, send money, love to you and yours

2. The Letter
Frederick J. Mespütchy

3. Proof of P = NP
Samir Jindel and Rose Bohrer

4. An Epistolary Reconstruction of the Curry-Howard Correspondence
Ben Blum and Michael Sullivan
Keywords: Corre-Howard-Spondence, Simply Typed Lambada Calculus, Supernatural Deduction

5. The Kardashian Kernel
David F. Fouhey and Daniel Maturana
Keywords: Kardashian, Kim, Kourtney, Khloe, Kernel, Kuadratic, Konvex, Koncave, Krylov, Kronecker, Kolmogorov, Karush-Kuhn-Tucker, K-Means, K-Armed-Bandit, Kohonen, Karhunen-Loeve, Kriging, Kalman, Kinect, Kovariance, Kurse-of-dimensionality, Kurvature, K-Nearest-Neighbor

The Letter

Frederick J. Mespütchy
CarnegieMellonTrump University
Hitler College of Barely Understandable Scientific... Stuff*
Bieber Hall, 1001110001000 Forbius Avenue, Pittsblerg

(*FYI: not *that* Hitler - his third clone was actually quite a nice guy)

Dear past future PhDs,

First, I would like to apologize to the unfortunate "victim" of this communication. Indeed, in order to ensure my submission was properly noticed (and hopefully submitted in a timely manner) to your deeply respected conference, I was forced to overwrite the most frequently accessed file in your Programme Chair's hard drive. It is my honest and sincerest wish for this inconvenience to be merely temporary, as I hope a complete and fully restoration of said file is possible so as he/she may continue to enjoy the unquestionable value of the content contained in "`Jane really loves horses - stretched sore holes - part 2.flv`".

With that, I will now take the time to briefly describe how I am reaching you. Although time-traveling is now technically possible, the energy requirements for sending information back in time grow exponentially in the size of the message and the distance in time. Furthermore, the operation changes the state of the carrier medium at that particular point in time. Consequently, we are limited to relaying this message by changing the magnetic properties of your PC's hard drive, we cannot create completely "new" and complex matter. Thus, although very experimental and not fully reliable, this message *should* have appeared in a quite noticeable location so as to be detected on time for the conference.

In case you are wondering as to why your hard drive was selected, well it has important historic value. However, you do not need to worry, for this procedure will cause no harm to it and, in fact, it will be returned this very same afternoon to the museum where your Facebook profile is on permanent display to educate today's people on how pathetic and meaningless your lives actually were.

The whole process of time-traveling is somewhat complex (and slightly itchy, with byproducts that may cause hallucination and silly behavior) and I will have to elide the details for sake of brevity, you can see the companion Technical Report[1] if you are curious about it. Interestingly, that research spawned a very insightful observation that, just like the human brain can only comprehend a single instant in time it has also evolved to only perceive a single position and ignore all the other superpositions of quantum states in a similar way that our eyes cannot see other wavelengths of light - but completely different, obviously. The mechanisms of evolution and natural selection pushed the development of brains that are adequate to observe a single quantum state at a time, even though all other possibilities continue to exist, these are shadows of the same object in dimensions that we lack a proper perception mechanism to comprehend simultaneously. Thus, the whole operation is more in changing our view of the present than your past - but calling it "quantum superposition re-collapse traveling" is probably inaccurate, imprecise, vague and apparently redundant.

Besides such obviously groundbreaking and history result, the real purpose of this message is to further guide your research paths by giving you a glimpse of today's society. Thus, contributing to the ever increasing pride and prestige of our institution by advising you on what advancements had true meaningful impact.

Similarly, perhaps the best glimpse of excitement I can give you about the future is that we have reached such an advanced state of reasonably compromise and educated discussions that even the most complex issues, such as abortion, have finally been solved. Thus, it is legal to technically do an abortion although the "aborted" embrions continue their development in an in-vitro womb. Furthermore, to ensure a proper up-bringing of these family-less individuals, they then automatically join the military and enlist in the famous *1st Single Victory Squad*, America's first suicide squad. As you can see such compromise has not only completely solved the issue once and for all, while pleasing both sides of the argument, and it was also responsible to create a group of people that played a crucial role in finally ending world hunger.

In, somewhat, unrelated news cannibalism is now socially acceptable.

Odd events have occurred since your time. For instance, the full size recreation of the Schrödinger's cat experiment resulted in a surprising outcome. Indeed, the cat was neither alive nor dead, but zombified. However, unlike what is portrayed in movies, zombies are surprisingly not all that harmful as rotting flesh has trouble keeping usable teeth. Thus, the subsequent zombie invasion produced very few but still very boring and excessively drooly deaths and some embarrassingly disturbing sexual behaviors that make necrophilia look like banging a dead corpse. Well, except in Britain. Apparently bad teeth were no impediment for British zombies. Indeed, the NHS is truly universal and the up-surge in sales of prosthetic teeth more than compensated for the economical impact of the whole zombie situation. But I digress...

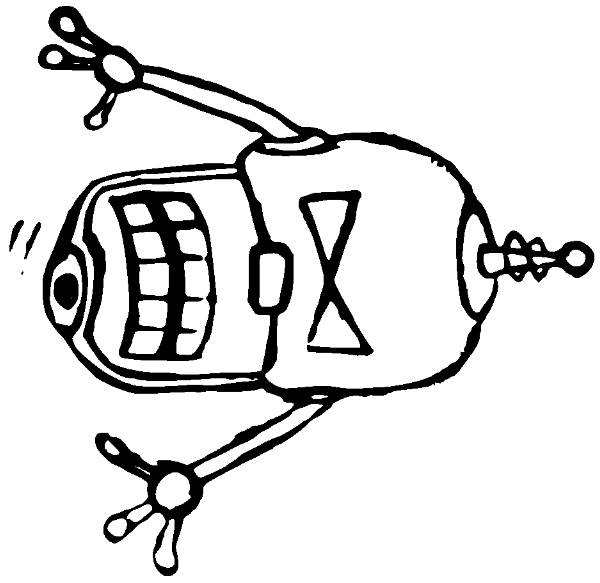
Video games have long became the norm for both education, art and recreation. Although, if our calculations are correct, you should be experiencing the Kinect controller "revolution", its impact as a long term input method was very limited. Indeed, once brain computer interfaces reached a reasonable resolution, someone immediately figure out a way they could combine the video stream from a Kinect controller and a simple BCI to accurately train a computer to recognized the brain patterns that trigger a specific kind of movement - without having to actually move. Thus, fear not, all that motion controller hype and stuff is purely a short lived fad. The true gamers of the future shall remain mostly motionless.

Perhaps the most interesting application of brain-scanner technology was that which spawned the Pavlovian FMRI brain-scanning experiment that is at the heart of today's Automatic Learning Programs. The core principle is to very quickly teach an association-based skill (such as reading, memorizing multiplication tables, etc.) by reducing the whole learning-feedback loop to a few milliseconds by reading the answer from a person's brain (even before they realize it) and conditioning their reaction to the correct result without them even having to think about it (initial versions gave you a small shock if you though of the wrong answer, later the conditioning improved by quickly re-flashing the correct answer in a subliminal-message kind of way). Later extensions to more mechanical like skills make it possible to learn the equivalent of 10 years of experience with almost any musical instrument in just a few days and make basic math a simple afternoon nuisance.

Another interesting consequence of this revolution, was its impact on programming languages. Although the usual functional versus object-orient argument continues to this day, even less people care about it. Indeed, very few people actually program in an "old-school" way. Most is done through the brain-scanners that are sufficiently smart to deduce what exactly you want in a way that resembles a 20-questions game - but played in an infinitely faster way such that the machine is capable of almost immediately create what you want or ask the precisely exactly questions or show you a situation to remove any potential source of ambiguity. Consequently, computers have become the last tool anyone needs and can even produce an unlimited amount of content to keep you entertained, base on what you would like to see. Yes, it's usually porn, but sometimes... it's *very sick* porn.

Perhaps the most interesting and surprisingly simple result in Computer Science was in the domain of the P vs NP problem. The solution becomes quite apparent onc#&$@2and perhaps some boots for me. However, suddenly my eyes crossed with a strangely familiar face. Obviously, all my instincts pushed me to try to get my picture taken with this hopefully very famous person, thus gaining and endless stream of Reddit karma, countering my previous pathetic attempt at a 'Look who I ran into today' post which - in retrospective - was quite misplaced since it involved a very *slimmified* cat. Thus, I approached the young man, who I honestly though was at least a minor character in the Game of Thrones, and proceeded to commend him o his endless and energetic performance (while shaking his hand as he was oddly uncomfortable and shy about the whole thing) at the same time as I tried to explain to my grandmother how to use the camera on my phone. She immediately also joined in on the complements, although I doubt she ever saw the TV show. And as we were both drowning him with such encouragements as 'your performance truly grips me to the screen', 'I admire your impressive dexterity in handling such a large and heavy sword' his girlfriend appeared. At that instant I immediately realized from where I had seen him (well, her too) before... On a very lustful but unfortunately lonely night, I stumbled upon this 'amateur' movie that, for some reason, captured my intensive and repetitive interest for many months. Although the guy's face only actually appears in less than a milisecond, my 'personal routine' caused me to see him more than my father and wrongly thinking he was TV famous. At this point, my grandmother - who obviously had no clue about this - just assumed the girl was also in the TV show. Which resulted in the most uncomfortable remarks such as 'oh, you two just work so well together', 'we just love your work', 'I only wish my grandson was as hardworking as you'... Their look of horror and shock persists with me to this day, as I continue my desperate quest to re-fin()?'2I+%expectable rise of robots. Again, unlike what is portrayed in movies, such uprise was generally unopposed. They simply carefully explained their point of view with such a logical simplicity and clarity that it was simply illogical to resist. Their arguments were so well devised that any real opposition was logically forced to jump off a cliff to even try to counter their reasoning. Thus, such logic bomb caused a swift end to all opposition. And guidos.

Nonetheless, the pride of Humanity remains stained with such event. Although they are - without a doubt - the most caring rulers of all Human history, perhaps my opinion on this matter has been partially biased by their logical weapon. In hopes of avoiding such a shameful defeat, I attach a sketch of these overly intelligent artificial creatures so that you can direct all of the World's resources to at least blend in and go by unnoticed among them and thus save the less logical part of Humanity from a very splashy doom. Unfortunately, due to space limitation, I was forced to rotate the drawing.



Other more mundane technological advancements include showers with adjustable heat such that a small sensor automatically computes an adequate temperature to keep the water pleasantly cozy. Buttons, such as for elevators, are all proximity based which considerably reduces transmission of diseases although door kno$\frac{2}{\gamma}$ˆfor what?? Really? Pff... if I were the last guy on the planet I could definitely do better than you. And besides, if everyone else is dead it just means you are definitely carrying some really deadly and contagious disease down there that is surely the cause of their demise. Yes, am very subtly calling you a whore. I know. This news is surely going to come as a shock to your mother. She really dislikes having competition. And I am not even *remotely* surprised that you would refuse me even if I were the last guy on the planet. After all, such constraint simply does not *significantly* reduce your domain of possible sexual partners. Now, if you had said that you would turn me down even if there were no dogs, no cats, no horses, no elephants, no snakes, no rats/mice, no anacondas, no llamas, no donkeys, no monkeys, no gorillas, no fishes of any kind, no cucumber or similarly shaped fruits or vegetables, no brooms, no fire extinguishers, no Eiffel tower, no trains, no bicycle seats, no at least partially stiff corpses, no plungers, no rockets, no door knobs, no gear sticks o4#̀own for singing "Shitting in America" to the tune of a Lady Gaga song while locked in a bathroom stall with no shoes (nor socks). Perhaps he is just weird, or the story is fiction, or he is just someone with a very unhealthy addiction. But I do request that you do try to find the person who wrote:

```
Here I am relaxed and seated,
With my ass carefully fitted
In this ceramic dome, unheated.
Ass cheeks spread and sweated,
Waiting a discharge, long pleaded.

I now reflect on what I did and see,
plants that spend more time in school than me,
but get less than a nod, not even a "Hi!"
sometimes slapped by that girl with big... eyes.

A billion "Hey!"-friends too polite to ignore,
names I forgot, people I do not wish to bore.
Stairways with suspicious smells
Who knows what below them dwells...

Stall-mates with impressively sounding excretions,
That saturate my nasal cavity to completion
Our coordinated farts as an odd way of communication
Between separate and distant anal civilizations
If only our hands could join in respectful consolation
Perhaps such friendly touch could cure my constipation.
```

Since the bastard's bowel movement caused an unprecedented and significant loss to the university both from cost of toilet paper as well as clogged pipes in the whole GHC complex. We don't even know how he managed to do that. We suspect it was either a very funny prank or just a pathetically sad health condition mixed with unusual bad luck and a poor/careless flushing discipline.

The adoption of the metric system was a catastrophe. Although not in any scientific or technical way, just socially. The transition from inches to centimeters had the unexpected consequence of boosting the self-esteem of every "hopefully-at-least-average" man, artificially pushing *them* into the domain of "slightly-above-average-but-nothing-*too*-impressive" league which lead to several outbreaks of pointless violence, mostly with baseball bats, and some less pointless ones, with acutely sharp knifes an⊎}*2(/ccidental fart that would give flashbacks to any holocaust survivor, the interview just went downhill. Somehow I just knew the fart incident would not be forgotten. After that my mind just... well, this is what I remember of the rest of the conversation:*
"And where do you see yourself in 5 years?"
"In the future."
"Ok, then lets go to a different kind of question. Why are manhole covers typically round?"
"They were designed to be squared, but they had to cut a few corners during implementation."
"How many ping-pong balls fit in a bus?"
"Exactly 3. Now, I know what you're thinking. You're probably wondering: What kind of bus can only fit 3 ping-pong balls? The answer is surprisingly simple: not a very big one."
And as the smell of harshly digested Subway food still lingered on everyone's noses and my underwear carried a very suspicious and even more inauspicious moisty feeling I began to+;δ23@W■`NO-CARRIER`■

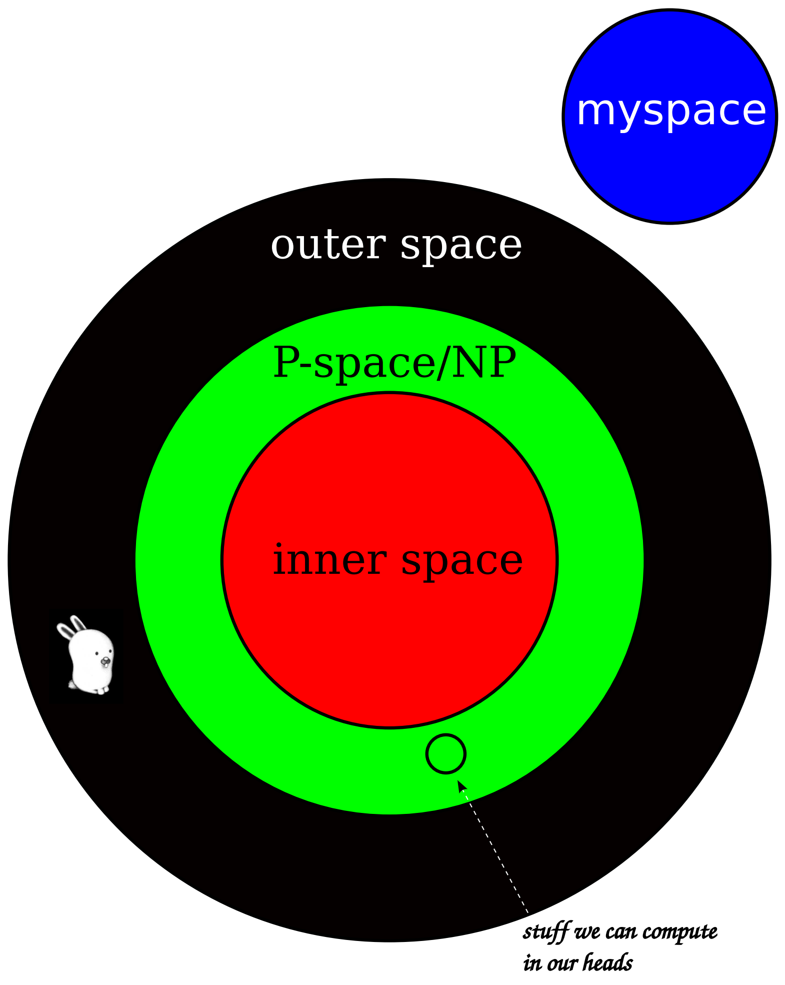
Proof of P = NP

Samir Jindel and Rose Bohrer

March 4, 2012

Abstract

We have made a revolutionary discovery that will forever change the field of computer sicence. The finest mathematicians and brightest scientists of mankind have toiled away at this problem for decades, but until now no progress has been made toward its resolution. Our solution and proof to this problem bring volumes of new insight to complexity theory. Never has mankind stumbled upon such an elegant proof to such a challenging and motivated problem before. We have dedicated years upon years of research to this topic, are simply standing upon the shoulders of giants as we present to you our asounding discovery: P = NP.



Let $N = 1$. Thus $P = 1 \cdot P = NP$. Quod erat demonstrandum, bitches.

An Epistolary Reconstruction of the Curry-Howard Correspondence

Starring
Ben Blum (`bblum@andrew.cmu.edu`) as *Priority Class Continuations*

and

Michael Sullivan (`mjsulliv@andrew.cmu.edu`) as *η-Normal Form Letters*

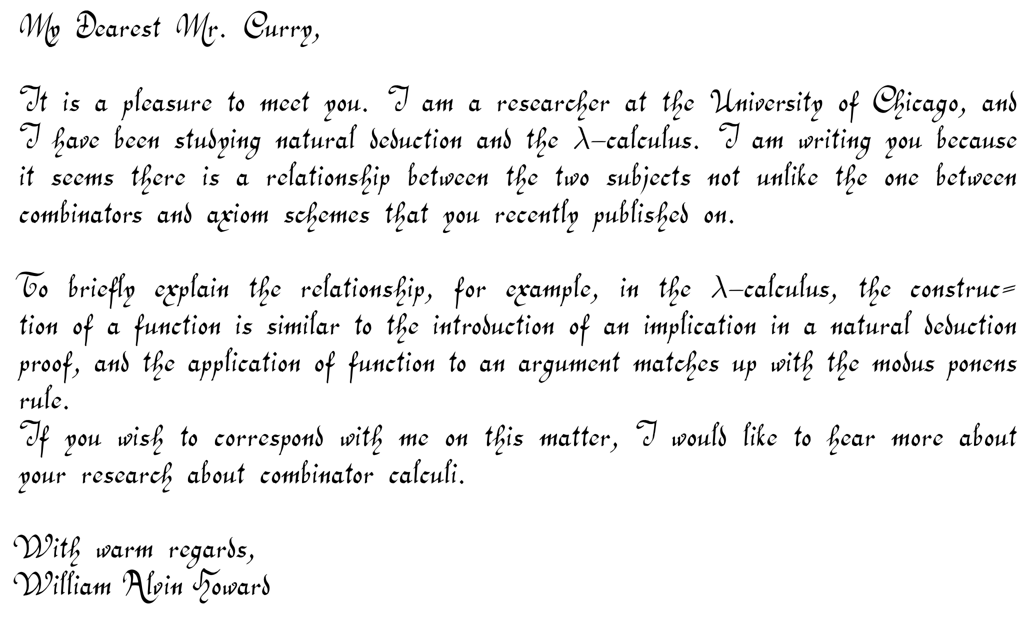
2011.04.01

Abstract

The logicians Haskell Curry and William Alvin Howard have each contributed numerous invaluable ideas to the fields of proof theory and programming theory, including the famous isomorphism between proofs and programs. Recent research, however, has uncovered a certain correspondence between the two gentlemen that contributed to the development of the famous Curry-Howard theory. We present the correspondence here in full in its original format.

Keywords: *Corre-Howard-Spondence, Simply Typed Lambada Calculus, Supernatural Deduction*

1 Introduction



My Dearest Mr. Curry,

It is a pleasure to meet you. I am a researcher at the University of Chicago, and I have been studying natural deduction and the λ–calculus. I am writing you because it seems there is a relationship between the two subjects not unlike the one between combinators and axiom schemes that you recently published on.

To briefly explain the relationship, for example, in the λ–calculus, the construc=tion of a function is similar to the introduction of an implication in a natural deduction proof, and the application of function to an argument matches up with the modus ponens rule.
If you wish to correspond with me on this matter, I would like to hear more about your research about combinator calculi.

With warm regards,
William Alvin Howard

Figure 1: Letter from Howard; Apr 1, 1946

2 Related Work

hi alvin,

good to hear from you -- you have some intriguing ideas! let me relate some of my work to give you an idea of how it all might fit together.

ive been focusing on the similarities between hilbert logic and combinator calculus. the thing about hilbert-style systems is that they are logical systems that push as much of the system into an axiom set as possible, while minimizing the number of rules of inference. so for propositional logic, the only inference rule would be modus ponens.

this corresponds to a programming model with no syntatic forms other than function application and some set of base constructors (the combinators, as you know -- S, K, I). notably, it doesn't have variable binding.

hopefully this has been helpful. let me know if you plan to publish any of your ideas, id be glad to take a look at a draft for you.

cheers,
haskell curry

Figure 2: Letter from Curry; Jul 7, 1831

3 Discussion

Greetings, Mr. Curry!

Thank you for the overview; I have found it immensely useful.

I have done some further work on my idea, and it seems that with my new way of looking at it, the idea behind natural deduction is that connectives are defined just in terms of introduction and elimination forms; not by relating connectives to each other.

It also seems that evaluation of λ–terms corresponds to reduction of proofs. A reduced proof is one with no "detours", if you will, that is to say, the introduction of a connective that is then eliminated. This turns out to match up perfectly with β–reduction.

I would love to hear what you think of this.

Best Wishes,
William A. Howard

Figure 3: Letter from Howard; May 17, 1837

```
hey alvin

this is most interesting. i wonder, what
might the implications be in the area of
classical logic? i havent yet been able
to figure out how proof by contradiction
fits in with my combinators. it seems
like it would be up your alley though?

as for me, i have discovered that there
are also translations between the lambda
calculus and the s/k calculus, much like
the isomorphism between hilbert proofs
and natural deduction proofs.

-- haskell
```

Figure 4: Letter from Curry; Nov 17, 1988

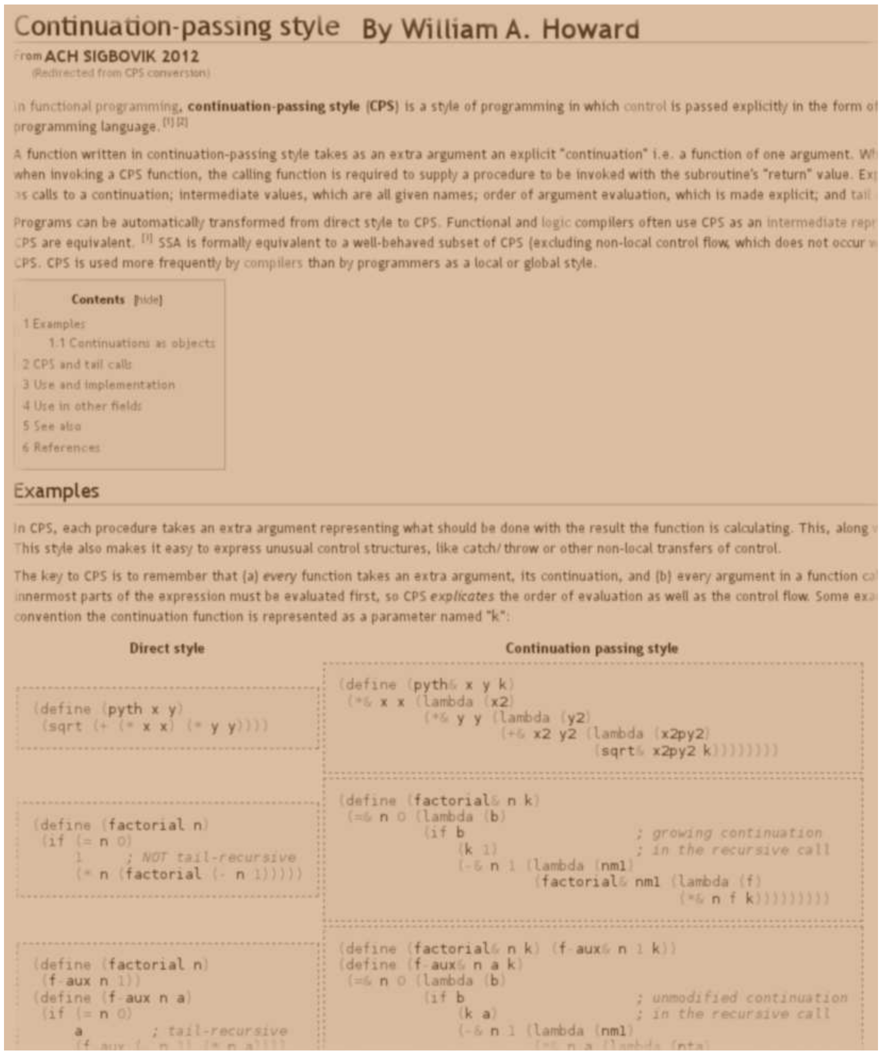
Mr. Curry,

I have made an exciting discovery. It seems as though the "continuation-passing style" invented by Sussman and Steele in 1975 is directly isomorphic to the double-negation translation from classical logic to intuitionistic logic. You may be particularly interested to note one part of this discovery, which is that Pierce's law of logic matches the "call/cc" primitive.

I have written a paper to announce this isomorphism, and intend to submit it to the upcoming SigBovik conference. I have also included a draft of the paper in this envelope, and I would be greatly obliged if you could take the time to review it.

With Much Gratitude,
William Howard

Figure 5: Letter from Howard; Mar 12, 1955



Continuation-passing style By William A. Howard

rom ACH SIGBOVIK 2012
(Redirected from CPS conversion)

n functional programming, **continuation-passing style** (**CPS**) is a style of programming in which control is passed explicitly in the form o
programming language.[1][2]

A function written in continuation-passing style takes as an extra argument an explicit "continuation" i.e. a function of one argument. W
when invoking a CPS function, the calling function is required to supply a procedure to be invoked with the subroutine's "return" value. Ex
s calls to a continuation; intermediate values, which are all given names; order of argument evaluation, which is made explicit; and tail

Programs can be automatically transformed from direct style to CPS. Functional and logic compilers often use CPS as an intermediate rep
PS are equivalent. [1] SSA is formally equivalent to a well-behaved subset of CPS (excluding non-local control flow, which does not occur
PS. CPS is used more frequently by compilers than by programmers as a local or global style.

Contents [hide]

1 Examples
1.1 Continuations as objects
2 CPS and tail calls
3 Use and implementation
4 Use in other fields
5 See also
6 References

Examples

n CPS, each procedure takes an extra argument representing what should be done with the result the function is calculating. This, along
his style also makes it easy to express unusual control structures, like catch/throw or other non-local transfers of control.

The key to CPS is to remember that (a) *every* function takes an extra argument, its continuation, and (b) every argument in a function c
nnermost parts of the expression must be evaluated first, so CPS *explicates* the order of evaluation as well as the control flow. Some ex
onvention the continuation function is represented as a parameter named "k":

Direct style

```
(define (pyth x y)
 (sqrt (+ (* x x) (* y y))))
```

```
(define (factorial n)
 (if (= n 0)
     1     ; NOT tail-recursive
     (* n (factorial (- n 1)))))
```

```
(define (factorial n)
 (f-aux n 1))
(define (f-aux n a)
 (if (= n 0)
     a        ; tail-recursive
     (f-aux (- n 1) (* n a))))
```

Continuation passing style

```
(define (pyth& x y k)
 (*& x x (lambda (x2)
          (*& y y (lambda (y2)
                   (+& x2 y2 (lambda (x2py2)
                              (sqrt& x2py2 k))))))))
```

```
(define (factorial& n k)
 (=& n 0 (lambda (b)
          (if b                    ; growing continuation
              (k 1)                ; in the recursive call
              (-& n 1 (lambda (nm1)
                       (factorial& nm1 (lambda (f)
                                        (*& n f k)))))))))
```

```
(define (factorial& n k) (f-aux& n 1 k))
(define (f-aux& n a k)
 (=& n 0 (lambda (b)
          (if b                    ; unmodified continuation
              (k a)                ; in the recursive call
              (-& n 1 (lambda (nm1)
                       (*& n a (lambda (nta)
```

Figure 6: Howard's paper on CPS conversion, rejected for lack of citations to former work.[Blu10, Ren10]

Haskell,

Did you see my last letter, about the continuation passing style? Please respond; the conference deadline is quite soon.

Thanks. . .
Will H.

Figure 7: Letter from Howard; Mar 18, 1955

```
SORRY ALVIN STOP I HAVE BEEN ON VACATION AND CANNOT READ YOUR DRAFT STOP YOU
SHOULD SUBMIT WITHOUT MY REVIEW STOP

HC STOP

SENT FROM MY CNCP TELECOMMUNICATIONS TELEGRAPH STOP
```

Figure 8: Telegram from Curry; Mar 19, 1955

4 Evaluation

$$\frac{e_1 \mapsto \lambda x.e_1' \quad e_2 \mapsto e_2'}{e_1 e_2 \mapsto [e_2'/x]e_1'}$$

5 Conclusion

Hi Mr. Curry,

It turns out that SigBovik did not accept my paper – the reviewers said something about another paper that was recently published on "CPS Conversion" that I neglected to cite in my submission.

I plan to continue developing this idea, and submit it to the Journal of Universal Rejection, whose deadline is upcoming next month.

I remain, Sir, your most Humble and Obedient Svt.,
William A. Howard

Figure 9: Letter from Howard; Apr 1, 1955

```
alvin,

sorry to hear sigbovik didnt appreciate
your discovery. hopefully youll meet
with more luck in future submissions.

i was thinking about sequent calculus
the other day, and it also might have
some place in your research. for example
cut elimination appears to represent an
abstract machine computation. i suspect
call-by-name and call-by-value semantics
play into it too.

-h
```

Figure 10: Letter from Howard; Apr 2, 1954

References

[Blu10] Ben Blum. DPS conversion: A new paradigm in higher-order compilation. In *Proceedings of the 4th Annual Intercalary Workshop about Conference in Celebration of Harry Q. Bovik's* 2^6*th Birthday*, SIGBOVIK, 2010.

[Ren10] David Renshaw. The Church-Monk Isomorphism. In *Proceedings of the 4th Symposium on Robot Dance Party of Conference in Celebration of Harry Q. Bovik's* 2^6*th Birthday*, SIGBOVIK, 2010.

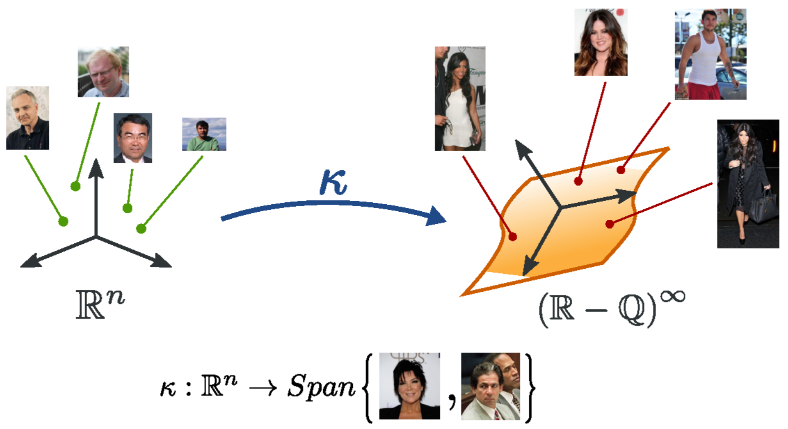
The Kardashian Kernel

David F. Fouhey
Sublime and Distinguished Grand Poobah,
Joint Appointment, Carnegie Mellon University,
Karlsruhe Inst. of Technology, Kharkiv Polytechnic Institute
`dfouhey@cs.cmu.edu`

Daniel Maturana
Distinguished Appointed Lecturer of Keeping it Real,
Joint Appointment, Carnegie Mellon University,
KAIST, Kyushu Inst. of Technology
`dimatura@cmu.edu`

Abstract

We propose a new family of kernels, based on the Kardashian family. We provide theoretical insights. We dance. As an application, we apply the new class of kernels to the problem of doing stuff.


κ
$\mathbb{R}^n$
$(\mathbb{R} - \mathbb{Q})^\infty$
$\kappa : \mathbb{R}^n \to Span\{\;,\;\}$

$$K_K(x, x') = \kappa(x)^T \kappa(x')$$

Figure 1: A motivation for the Kernel Trick. κ maps the real world of sane people into the subset of $(\mathbb{R} - \mathbb{Q})^\infty$ spanned by Robert Kardashian, Sr. and Kris Jenner (formerly Kardashian). We would like to avoid having our data appear on Keeping Up with the Kardashians, and so we use the Kardashian Kernel K_K to compute an inner product in the Kardashian Space without ever having to go there.

Kernel machines are popular because they work well and have fancy math, pleasing practitioners and theoreticians alike. The Kardashians are popular because (**TODO**). Why not combine them?

1 The Kardashian Kernel

Let $\mathcal{X}$ be an instance space. The Kardashian Kernel is an inner product operator $K_K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Applying the Kernel trick [14] we express it as $K_K(x, x') = \kappa(x)^T \kappa(x)$, with $\kappa : \mathcal{X} \to \mathfrak{K}$. Here $\mathfrak{K}$ represents a possibly infinitely-dimensional feature space. In Fig. 1, we provide the best (to our knowledge) motivation of the Kernel Trick: by using the Kardashian Kernel, we can leverage the Kardashian Feature space without suffering the Kurse of Dimensionality. This kurse is similar in nature to the better-known Curse of Dimensionality (c.f., [3]); however, the motivation for avoiding such a space is different: here, we wish to avoid having our data be associated with the Kardashian shenanigans[1].

1.1 Related Work

It is common in the literature to cite work that is thematically related; here, we explore an totally better style, in which we cite work that is alphabetically related.

Historically, we are of course motivated by the Kronecker product and delta, and by the fundamental impact had by Andrey Kolmogorov on probability theory. In more recent work, our work is motivated by Kalman Filters [9], especially in application to active stereo sensors such as Kinect or circular distributions such as the Kent distribution. We are also inspired by the ingenuity of David Lowe in using an modified K-d tree search ordering to perform fast keypoint retrieval in computer vision [11]; however, we believe that our approach is provably k-optimal, as our paper has significantly more k's and substantially more pictures of the Kardashians. We feel that it is important to note that several machine learning techniques are in fact, special cases of our k-themed approach: Gaussian process regression also known as Kriging [12], and Self-Organizing Maps [10] are also known as Kohonen maps, and thus both are of interest; in contrast to this work, however, we place our k's up-front systematically rather than hide them in complex formulations and semantics.

1.2 On Some Issues Raised by the Kardashian Kernel

1.2.1 On Reproducing Kardashian Kernels

A natural question is, does K_K define a Reproducing Kernel Hilbert Space (RKHS)? In other words, are the Kardashians Reproducing Kernels? We conjecture that it may be the case: see Fig. 2, as well as [16] and [13]. Nonetheless this has only been proven for a special case, Kourtney [5].

Figure 2: Are the Kardashians Reproducing Kernels? So far this conjecture has only been proven in the case of Kourtney (left figure), but many authors have argued that figures such as (right) may suggest it is also true for Kim.

[1]We thank the anonymous reviewer at *People* for correcting an error which occurred in the previous version: Kris Humphries is no longer considered a member of the Kardashian feature space.

1.2.2 On Divergence Functionals

Our paper naturally raises a number of questions. Most importantly of all, one must ask whether the space induced by κ has structure that is advantageous to minimizing the f-divergences (e.g., see [15])? We provide a brief analysis and proof sketch. Note that

$$D_\phi(\mathbb{Z}, \mathbb{Q}) = \int p_0 \phi(q_0/p_0) d\mu$$

with ϕ convex. The following result follows fairly straight-forwardly from the standard definitions:

$$\min_w = \frac{1}{n}\sum_{i=1}^{n} \langle w, \kappa(x_i)\rangle - \frac{1}{n}\sum_{j=1}^{n} \log\langle w, \kappa(y_j)\rangle + \frac{\lambda_n}{2}||w||^2_{\mathfrak{K}}$$

A complete proof is omitted due to space considerations, but should be fairly straight-forward for even an advanced undergraduate; it is made much easier by the use of the Jensen-Jenner Inequality [8].

2 Kardashian SVM

2.1 Problem setting

SVMs are very popular, and provide a great way to plug in our new kernel and demonstrate the importance of being Kardashian [18]. We propose to solve the following optimization problem, which is subject to the Kardashian-Karush-Kuhn-Tucker (KKKT) Conditions[2]

$$\min_{\mathbf{w},\xi,\mathbf{b}} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n} \xi_i$$

such that

$$\begin{aligned} y_i(\mathbf{w}^T\kappa(\mathbf{x}_i) - \mathbf{b}) &\geq 1 - \xi_i && 1 \leq i \leq n \\ \xi_i &\geq 0 && 1 \leq i \leq n \\ \zeta_j &= 0 && 1 \leq j \leq m. \end{aligned}$$

κ is the mapping of datapoints into the Kardashian feature space; $\mathbf{x}_i$ and $\mathbf{y}_i$ are data points $1, \ldots, n$ and their labels (-1 or 1); ξ_i are slack variables; and ζ_i are the sentences imposed upon O.J. Simpson, defended by Robert Kardashian, Sr., for charges $1, \ldots, m$. It can be proven that for $n = 3$, each ξ_i has the psychological interpretation of the expected relative proportion of attention given to Kardashian daughter i by the American public.

2.2 Learning algorithm

Learning the optimal classifier involves finding an optimal $\mathbf{w}$. A common approach is to use standard Kuadratic Programming (KP) methods; see [4] for an summary of relevant techniques[3].

However, the optimization manifold has a highly characteristic kurvature (see fig. 3). We use an alternative approach that takes advantage of the structure of the problem (c.f., our earlier discussion regarding minimal f-divergences in RKHS).

It is clear[4] that the problem meets the conditions to be considered "Konvex". Analogously, its dual is "Koncave". The solution for our problem is bounded by relaxed versions of both; therefore we use a Koncave-Konvex (KKP) procedure [19] to solve the problem.

2.3 Experiments - Kardashian or Cardassian?

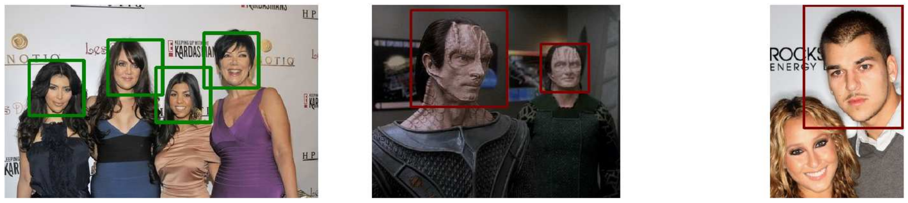
In this experiment, we use the proposed Kardashian-Support Vector Machine (K-SVM) to learn a classifier for "Cardassian" or "Kardashian" given a window of a humanoid face. The

[2]Conditions under which our function will converge to a global optimum and stay there for at least 72 days.

[3]N.B. Although popular, readers should note significant spelling deficiencies in [4]; notably, "konvex" is misspelled as "convex [sic]"

[4]Unless you're dumb.

Figure 3: Kurvature of the optimization manifold



(a) Kardashian - (l. to r.) Kim, Khloé, Kourtney, Kris (b) Cardassian - (l. to r.) Gul Dukat, Elim Garak (c) A failure case (or is it?): Kardashian - Rob

Figure 4: Example detections on our "Kardashian or Cardassian" dataset. After detection by a standard Humanoid Face detector, subjects are classified as Cardassian or Kardashian. Green bounding boxes indicate Kardashians; red ones indicate Cardassians.

Kardashians are a family of American socialites made famous by the purported reality show "Keeping up with the Kardashians"; the Cardassians are a species of humanoid extraterrestial beings from Cardassia prime, depicted in "Star Trek", a science fiction franchise.

We train and test on "Kardashian or Cardassian", a dataset which contains approximately 2000 examples of each class [5] Given a bounding box of a subject, we classify subjects into either Cardassian or Kardashian using HOG features. We demonstrate state-of-the art results, slightly out-performing a state-of-the-art latent SVM formulation in average precision, while beating it in Kardashianity[6]. It significantly beats both 1-NN and K-NN (not to be confused with Kardashian-Nearest Neighbors [1]) in performance as well. When paired with a standard humanoid facial detector, we can quickly detect and accurately localize and characterize persons of terrestial or interstellar origin as one of the two classes.

3 Kardashian Kernel Karhunen-Loève Transform

Just as the Kardashians have been used to deceptively market predatory credit cards [6], we use them to deceptively market an ineffective dimensionality reduction technique. Taking inspiration from their "famous-because-they're-famous" nature, we present a "low-dimension-because-it's-low dimension" dimensionality reduction technique, the Kardashian Kernel Karhunen-Loève Transform (KKKLT). Note that the Karhunen-Loève Transform (KLT) is also known as Principal Component Analysis (PCA), but we prefer the former nomenclature for its *k*-optimality. Our approach is highly general and works for arbitrary objective functions.

[5] Some may argue our ground truth is not be completely accurate, since there have been recent claims that Khloe is not a biological Kardashian [7]. However, we believe that this is not true and you should leave Khloe alone.

[6] Roughly equivalent to Rademacher complexity, measured in the Kardashian space $\mathfrak{K}$ induced by K_K, except for various constraints on the name of the algorithm and package used to implement it. We have yet to explore connections to Kolmogorov complexity.

Method	K-SVM	Latent SVM	K-NN	1-NN
AP	**80.1%**	78.1%	66.5%	60.9%
Kardashianity	∞	0	1	0

Table 1: Performance results on "Kardashian or Cardassian" dataset, with K-fold validation

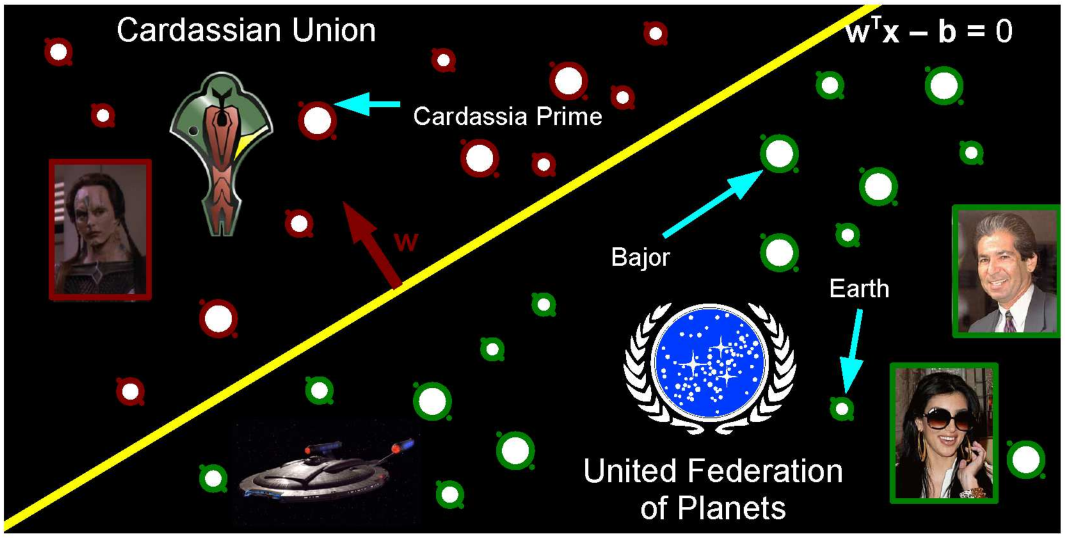


Figure 5: Schematic for Kardashian SVM: in the feature space $\mathfrak{K}$ induced by κ, the decision boundary between Cardassian and Kardashian lies approximately 5 light years from Cardassia Prime.

3.1 Method

Input: Datapoints $\mathbf{X} = \{\mathbf{x}_i : 1 \le i \le n, \mathbf{x}_i \in \Re^m\}$; objective function $F : \Re^{m\times n} \times \Re^{k\times n} \to \Re$; target dimension k; and k-dimensional representation $\mathbf{Y} = \{\mathbf{y}_i : 1 \le i \le n, \mathbf{y}_i \in \Re^k\}$ s.t. $F(\mathbf{X}, \mathbf{Y}) \le F(\mathbf{X}, \mathbf{Y}') \;\forall\; \mathbf{Y}' \in \Re^{k\times n}$.

Output: k-dimensional representation of $\mathbf{X}$ (i.e., some $\mathbf{Z} \in \Re^{k\times n}$).

Procedure:

1. Compute the Kernelized Kovariance (see [17] for an illuminating discussion of the identical, but non-k-optimal, concept of Kernelized *Covariance*).
2. Compute the eigenvectors of the Kernelized Kovariance using the Kardashian-Krylov Subspace method; following k-optimality, this is preferred to Lanczos or Power Iteration.
3. Treat eigenvectors as a basis and form an infinite-dimensional rotation matrix R_R^R.
4. Compute the Kronecker product $R_R^R \otimes (R_R^R)^{-1}$, and take the SVD of the resulting matrix, again using Kardashian-Krylov methods for maximal k-optimality.
5. **return** $\mathbf{Y}$.

3.2 Experiments

Our approach is provably optimal for any objective function F (see Appendix A); no experiments are necessary. However, we use this space to pontificate, as is common in the literature. We believe that the Kardashian-inspired approach suggests underlying connections to the classic K-armed bandit problem. Further research will be needed. Please send cash (USD) to the authors to fund a reality show/NSF REU on this.

4 Graph Kardashiancian

In Kardashian Kernelized space the weight of data points is defined by the weight of the data points to which they are linked to. In this situation the graph connectivity is essential in the ranking of celebrities; e.g., see Fig. 6, which explains the prominence of Kim Kardashian[7] To capture this connectivity we define an analog of the so-called Laplacian on graph similarity matrices, the Kardashiancian. For details on the Kardanshiancian, and its applications, see our forthcoming paper on our Kardashiancian-powered algorithm, KardashianRank [2].

Figure 6: KardashianRank: Kardashiancian-based celebrity ranking

5 Conclusions and future work

We consider the Kardashian Kernel the foundational breakthrough of a new field, Celebrity-based Machine Learning (CML). We believe that we have already exhausted the potential for Kardashianity in this, and other K-Themed papers (e.g., [1, 2]); however, while we have closed the door on any new Kardashian papers by our thorough and complete treatment of the topic, various avenues of new research are now wide open - we recommend young researchers to hop on the bandwagon as soon as possible. We are currently working on various new groundbreaking results:

- The Tila Tequila Transform (T_{T_T})
- Johnny Depp Belief Nets ($JDBN$s)
- The Kardashian-Kulback-Leibler (KKL) divergence and its generalization, the Jensen-Shannon-Jersey-Shore (JS^2) divergence
- Miley Cyrus Markov Chain Monte Carlo ($MCMCMC$) methods for inference
- The Orlando Bloom Filter
- The Carrie Fisher Information Matrix

Our work suggests the monetization of research via branding: for instance, in exchange for a significant monetary compensation, we suggest that the NIPS and ICML insist upon the convention of referring to "Tikhonov regularization" as "Jamie Lee Curtis Regularization"®, sponsored by Activia Yogurt^{TM}.

[7] Please see supplementary video for an insightful and intuitive explanation of the connection between Kim Kardashian and Ray J.

Acknowledgments

This paper is brought to you by the letter K, the restaurants on Kraig Street, and contributions from viewers like you.

A Proof that Kardashian Kernel KLT/PCA is optimal

Suppose $\mathbf{Z} \in \Re^{k \times n}$ is the output of KKKLT$(\mathbf{X}, \mathbf{Y}, k, F)$; then $\mathbf{Z}$ satisfies $F(\mathbf{X}, \mathbf{Z}) \leq F(\mathbf{X}, \mathbf{Z}') \; \forall \; \mathbf{Z}' \in \Re^{k \times n}$.

Proof. Trivially follows from the input conditions. □

References

[1] Anonymous. Kardashian Nearest Neighbors - toward a totally new era of nonparametric methods. *In Submission to Transactions on Pattern Analysis and Machine Intelligence*, 2012.

[2] Anonymous. KardashianRank: A random STD model for celebrity ranking. *In Submission to Transactions on Computer Networks and ISDN Systems*, 2012.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer, 1st ed. 2006. corr. 2nd printing edition, 2007.

[4] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[5] E!Online. *The Internet*, 2012. URL: `http://www.eonline.com/news/kourtney_kardashian_pregnant_with_baby/277501`.

[6] Gawker. *The Internet*, 2010. URL: `http://gawker.com/5693964/kim-kardashians-credit-card-may-be-the-worst-credit-card-ever`.

[7] Perez Hilton. *The Internet*, 2012. URL: `http://perezhilton.com/2012-01-11-khloe-kardashian-reportedly-not-a-biological-kardashian`.

[8] K. Jenner and R. Kardashian. The Jensen-Jenner inequality. *IEEE Transations on Los Angeles-Themed Statistics*, 13(6):1352–1368, mar. 1998.

[9] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45.

[10] Teuvo Kohonen. *Self-Organizing Maps.* Springer, 2001.

[11] David G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.

[12] G. Matheron. Principles of geostatistics. *Economic Geology*, (58):1246–1266, 1963.

[13] MediaTakeout. *The Internet*, 2010. URL: `http://mediatakeout.com/45282/mto_world_exclusive_kim_kardashian_is_reportedly_pregnant____and_youll_never_guess_who_the_father_is.html`.

[14] John Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philos. Trans. Roy. Soc. London*, 1909.

[15] XuanLong Nguyen, M.J. Wainwright, and M.I. Jordan. Estimating divergence functionals and the likelihood ratio by convex risk minimization. *Information Theory, IEEE Transactions on*, 56(11):5847 –5861, nov. 2010.

[16] RadarOnline. *The Internet*, 2012. URL: `http://www.radaronline.com/exclusives/2012/01/khloe-kardashian-fertility-treatments-lamar-odom-dallas`.

[17] L. Song, A. Gretton, D. Bickson, Y. Low, and C. Guestrin. Kernel belief propagation. 2011.

[18] O. Wilde. *The Importance of Being Kardashian.* Random House, 1895.

[19] A.L. Yuille, A. Rangarajan, K. Kardashian, K. Kardashian, and K. Kardashian. The koncave-konvex procedure. *Neural Komputation*, 15(4):915–936, 2003.

Track 4

Did you bring enough to reshare with the class?

0. Implications of Constrained Thought Expression Achieved via a One Hundred-forty Character Message Limitation Applied to Complex Social Netwo
Nathan Brooks and Tommy Liu
Keywords: #socialmedia, #sm, #140

1. The Spineless Tagless Tweet Machine: Distributed Cloud-Based Social Crowd-sourced Lazy Graph Reduction on the Web 2.0
Michael Sullivan

2. SIGBOVIK 2012 Take-Home Midterm Examination
James McCann

3. The National Month Of Pushing Spacebar
Tom Murphy VII
Keywords: C++harles D:\ickens, fanfic, repetitive stress injury, 2012: A Space Odyssey

4. What Most Medical Students Know About Computer Science
Brian R. Hirshman
Keywords: computer science, electronic medical records, all-knowing doctors, important stuff, take two kilobytes and see me in the morning, dude we're too busy learning other things

The Spineless Tagless Tweet Machine

Distributed Cloud-Based Social Crowdsourced Lazy Graph Reduction on the Web 2.0

Michael Sullivan

Carnegie Mellon University
mjsulliv@cs.cmu.edu

Abstract

Lazy graph reduction is a common technique for implementing non-strict functional programming languages. We present the Spineless Tagless Tweet Machine, a distributed lazy graph reduction system that uses Twitter to communicate and store unevaluated expressions.

1. Introduction

Over the last few decades, there has been a large amount of work discussing methods for efficient implementation of non-strict functional programming languages [3] [4]. A much touted benefit of non-strict, pure functional languages is that computations can be easily parallelized without worrying about concurrency issues.

We propose that the lazy graph reduction model combines well with a number of other major recent developments in the computing world: the emergence of cloud computing and the social web. Cloud computing refers to the offloading of computation and storage to unaccountable and untrusted software-as-a-service providers, while the social web refers to the "sharing" of statuses, photographs, personal information, and other data with friends, acquaintances, and enemies (frequently in the form of "frenemies") through the Web. One of the most popular cloud applications on the social web is Twitter [1], which allows users to publish 140-character "tweets" (which include images), search for tweets by "hashtags", and other such "social" activities.

Our proposal is to take advantage of of the parallelizability of lazy pure functional programs by using the tools of cloud computation and the social web. We present the Spineless Tagless Tweet Machine, a distributed lazy graph reduction system that uses Twitter to share unevaluated expressions with the user's friends, acquaintances, and frenemies.

2. Description

Lazy computation is generally implemented by creating "thunks" containing computations that might be needed later and evaluating the thunks (or "pulling on" them) only when the result is needed. Pulling on a thunk may involve creating new thunks representing computations for subparts of the value computed.

The Spineless Tagless Tweet Machine evaluates a internal language called `STTM`. The `STTM` is an austere untyped internal language with support for lazy evaluation. `STTM` is essential a more syntactically restricted variant of the untyped lambda calculus. Algebraic data-types are represented using the Scott encoding; that is, as functions that take case functions for each of their branches and then call the matching one. Integers and integer operations are included in the language for efficiency.

In the Spineless Tagless Tweet Machine model, interested users run Spineless Tagless Tweet Machine computation nodes on their machines. These nodes monitor twitter for `STTM` tweets containing unevaluated thunks and upon seeing them may choose to do some evaluation of them and tweet the results. This may involve creating additional thunks. Note that this does not necessarily need to been done by the `STTM` software: individual users should feel encouraged to evaluate thunks by hand and tweet the results, allowing a crowdsourced graph reduction.

Each unevaluated expression will be associated with a unique hash-tag [1] To pull on thunks, a client searches for tweets with the appropriate hashtag, in order to find an evaluated version. (This search may need to be repeated until the thunk has been evaluated.) Since the size of expressions is likely to exceed the strict 140-character limit, expressions will be encoded as images and included with the tweets.

3. Characteristics

One major advantage of this system is that caching of results is provided by Twitter. If a computation has been performed before, the result will be immediately found when pulling on the thunk.

Some potential downsides in this system are the potential weakness in privacy and correctness. The system provides no mechanism to assess the correctness of evaluated tweets or to hide tweets containing private computational data from unwanted observers. In practice, users of cloud and social services seem to not mind these limitations.

Deciding which tweets should be evaluated remains a major open problem in this work. Only evaluating tweets that are directly needed fails to gain any parallelism, while evaluating all tweets will result in chasing down infinite data structures and rapidly being rate-limited by Twitter.

4. Conclusion

In this paper, we presented a novel way to violate Twitter's Terms of Service [2] by using it as the communication medium and backing store for distributed lazy graph reduction. We would have evaluated the performance, but will be implementing the system sometime between now and the conference.

We're no strangers to love. You know the rules and so do I. A full commitment's what I'm thinking of. You wouldn't get this from any other guy. I just wanna tell you how I'm feeling. Gotta make you understand. Never gonna give you up. Never gonna let you down. Never gonna run around and desert you. Never gonna make you cry. Never gonna say goodbye. Never gonna tell a lie and hurt you. We've known each other for so long. Your heart's been aching but. You're too shy to say it. Inside we both know what's been going on. We know the game and we're gonna play it. And if you ask me how I'm feeling. Don't tell me you're too blind to see. Never gonna give you up. Never gonna let you down. Never gonna run around and desert you. Never gonna make you cry. Never gonna say goodbye. Never gonna tell a lie and hurt you.

[1] For this reason, "tagless" is inaccurate. "Spineless" also is inaccurate.

References

[1] Twitter. `http://twitter.com`, .

[2] Twitter terms of service. `http://twitter.com/tos`, .

[3] S. L. P. Jones. Implementing lazy functional languages on stock hardware: the spineless tagless g-machine - version 2.5. *Journal of Functional Programming*, 2:127–202, 1992.

[4] M. Naylor and C. Runciman. The reduceron reconfigured. *SIGPLAN Not.*, 45(9):75–86, Sept. 2010. ISSN 0362-1340. doi: 10.1145/1932681.1863556. URL `http://doi.acm.org/10.1145/1932681.1863556`.

SIGBOVIK 2012 Take-Home Midterm Examination

James McCann[*]
Adobe Systems, Inc.

Abstract

This exam is 3 pages long and has 5 questions. Please check to be sure that you have all the pages before leaving class today.

This midterm is closed-book, closed-internet, open-proceedings; you may refer to other material in this packet, but please don't use outside references or discuss any of the questions with others until the examination period has elapsed.

Typeset your answers and any supporting material neatly, label each page with your name and section number, staple the resulting pages firmly, and slip them under my office door. The hard deadline for this exam is tomorrow at 4pm. Any exams not turned in at this time will be counted as failing grades.

I will be able to answer questions on the exam during my normal office hours or via e-mail. I will not answer questions received after 3pm, however.

Good luck.

CR Categories: 5.Q.a [Exams]: SIGBOVIK—2012

1 Optimal ordering [10pts]

Given a list of the first n non-negative integers in some arbitrary order, place them in sorted order by using as few calls to move(p, t) as possible. (Where move(p, t) moves the element at position p in the list to just before the element at position t.)

Example: Given $(1, 0, 5, 2, 4, 3)$, one optimal sequence of calls is:

Call	resulting list
-	$(1, 0, 5, 2, 4, 3)$
move(1, 0)	$(0, 1, 5, 2, 4, 3)$
move(2, 6)	$(0, 1, 2, 4, 3, 5)$
move(4, 3)	$(0, 1, 2, 3, 4, 5)$

Exercises:

1. How – in an abstract, mathematical sense – can you determine the minimum number of calls to `move` required? [2pts]
2. Give an efficient algorithm for determining the number of calls. [4pts]
3. Give an efficient algorithm which outputs a minimum-length list of calls. [4pts]

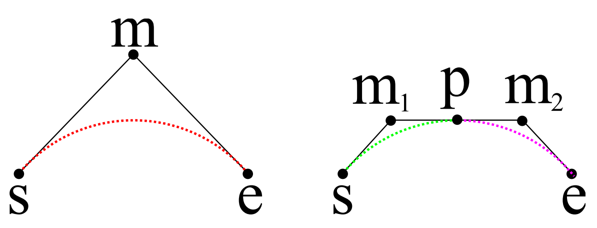
[*]e-mail: jmccann@adobe.com

2 Arc subdivision [8pts]

Given non-coincident start and end points – s, e – and a midpoint – m – equidistant from these, one can draw exactly one circular arc that starts at s tangent to $\overline{sm}$, ends at e tangent to $\overline{me}$, and is contained in the triangle s, e, m. (For the purposes of this problem, I consider straight lines to be circular arcs of infinite radius.)

When drawing or colliding against arcs, it is convenient to be able to subdivide them – that is, produce points m_1, p, m_2 such that s, m_1, p and p, m_2, e each describe half of the arc.

Illustration:



Exercises:

1. Write an arc-subdivision subroutine. You do not need trigonometry to solve this problem. [6pts]
2. What shape results when your routine is applied with m not equidistant from s and e? [2pts]

3 Enumeration of sums [12pts]

Given a set of integers, one may wish to enumerate through subsets of those integers in order of their sum.

Example: Given set $\{-1, 2, 3\}$, one ordering of the subsets by their sum is:

Subset	Sum
$\{-1\}$	-1
$\{\}$	0
$\{-1, 2\}$	1
$\{2\}$	2
$\{-1, 3\}$	2
$\{3\}$	3
$\{-1, 2, 3\}$	4
$\{2, 3\}$	5

Exercises:

1. Write a time-efficient subroutine that returns the next subset every time it is called. This subroutine may store data between calls. [8pts]
2. (**Extra Credit**) Is your subroutine memory efficient? [10pts]
3. Why is it difficult to write a time-efficient subroutine that does not store data between calls? [3pts]
4. Describe a sufficient condition on the set such that one can write an efficient subroutine that does not store extra data. Your condition should admit an infinite number of sets. [1pts]

4 In-place rearrangement [10pts]

Sometimes, it is desirable to reorder an array of items in place. That is, given an array $A = (a_1, \ldots, a_n)$ of generic items and an array $T = (i_1, \ldots, i_n)$ of target indices for those items, you wish to overwrite A with $(a_{i_1}, \ldots, a_{i_n})$.

Note: For this problem, the only operations available on generic items are creation and assignment.

Example: Given array (a, d, f, c, e, b) and target indices $(0, 2, 4, 1, 3, 5)$, your subroutine should over-write the input array with (a, c, d, e, f, b).

Exercises:

1. Given an array of n items and an array of target indices, write a subroutine that reorders these items in place in linear time and constant memory. You may clobber the array of indices. [4pts]
2. Write a routine to rotate (in-place) a rectangular image stored in scanline order by 90 degrees; the routine should operate in linear time and constant memory. Note that there is no target indices array. [5pts]
3. Describe a sufficient condition on the structure of the target indices which will allow you to perform remapping operations without clobbering the target indices array. [1pts]

5 Lights Out [9pts]

Lights Out is an electronic puzzle game featuring a grid of lit buttons. Every time a button is pressed, that button (and its 4-way neighbors) toggle from lit to unlit or unlit to lit. The goal is to toggle all buttons to the unlit state.

We can generalize Lights Out in two ways. First, we can get rid of the grid, and instead work over a general directed graph, where activating vertex v toggles all vertices v' such that there is an edge (v, v'). Notice that this definition allows us to chose whether activating a vertex toggles itself. Second, we can begin to talk about having S states instead of only 2. So every vertex of our graph is assigned a number $0, \ldots, S-1$, and toggling that vertex increases the number by one (with $S-1$ wrapping back to 0).

Exercises:

1. Describe a method for finding a solution in this general setting which works in $O(v^3)$ time. [3pts]
2. When is this solution optimal? [3pts]
3. Given a non-optimal solution, how much time is required to make it optimal? [3pts]

End of Exam

The National Month Of Pushing Spacebar

by Tom Murphy VII

First edition, April 2012

PRIOR ATTEMPTS at writing a novel had been unsuccessful. Think of all the obstacles: There's the blinking green cursor of the tele-type that you can't figure out how to turn off. And when you put tape on the screen to cover it up, then it keeps moving whenever you make progress on the novel, except if you count progress like deleting some piece of text and adding new, better text of equal length. Or like, making the font of the whole book smaller with each added letter, like when you're typing your name onto a sticky name badge and you decide to add the ceremonial "Ph.D." and "D.D.S." without realizing that due to horizontal space constraints, they will diminish the gravitas of the rest of your name, as measured in point size, except you can't turn back now. And also you can't do that on a tele-type on account of it only has one font, built into the Rondom Occess Memory, called Times New Rondom, which is also green and looks like it was invented for other computers to read, which makes you feel like a robot-man or -woman ("wo-bot"), who are known to not be able to write novels except like "1001010101: A Tale Of Two Bitties" by C++harles D:\ickens. So that is stymying.

Then you discovered the National Novel Writing Month a.k.a. NaNoWriMo,[1] the creatively capitalized internet web page that encourages stymied novel writers to risk their jobs, romantic entanglements, and friendships for the chance to self-publish horrendous exigent fantasy pastiche, thinly-veiled Twilight, Harry Potter, and Dr. Who crossover fanfic, or offensively self-referential poo-poop kinda like this. All you need to do is set aside 120 hours in the month of November to jot down 50,000 words that have something to do with each other, and declare victory. Maybe even pay a small fee for a publishing service that provides you with a UPC-like number precomputed to have a

[1]http://nanowrimo.org/

valid check digit, and an obligation to purchase too many copies of your work at nearly novel-low prices.

And you did that too, but what shame! After customizing your profile and packing the fridge up with the right snacks (things you once saw someone who you consider health-conscious and knowledgeable about food things bring to a party), and writing a purple description of your main character, who was just by the way a faint simulacrum of either you (but cooler), your imaginary girl/boyfriend, your World of Warcraft character, or something like that, but anyway doesn't matter because after writing that beginning bit and a little bit of not-thought-through plot thickening, it all dried up again.

But enough about that because, sitting afore the pale computer glow or perhaps with a hardcopy in hand patiently awaiting a talk to finish at a prestigious academic conference, you are now discovering the solution: The National Month of Pushing Spacebar. This annual competition, fresh as angel diapers, the spring chicken of massively-singleplayer forced-creativity suicide pacts, challenges you to achieve your dream of producing a novel-sized document without the creative stress and feelings of inadequacy that come from having that document also contain original content.

The premise is simple: During the National Month, push spacebar. The National Month begins on Friday 30 March 2012 to coincide with the prestigious academic conference SIGBOVIK, and ends at 23:59:59.$\overline{9}$ eastern-jingoist-time on Sunday 29 April 2012, not including the final day of the only-partially-national month of April, for a nice round 31 days. Due to no bullshits having to do with leap anything or 2$^{\text{nd}}$ extended deadline, this comprises exactly 44,640 minutes. Success constitutes pressing the spacebar 100,000 times, which yields a novella of approximately 33 pages, consisting only of whitespace. The best part is you don't need to think about anything hard or worry that it won't turn out good, because it can only be spaces. 100,000 elegantly simple, stress-relieving spaces.

Your eyebrows perk up with interest. Actually one eyebrow goes up and the other goes down. Can the NaMoOfPuSp be the real deal Holyfield? Indeed it may, sir or madam. With one finger in the air politely to indicate *pause*, you wonder, "What more about the logistics shall I know?"

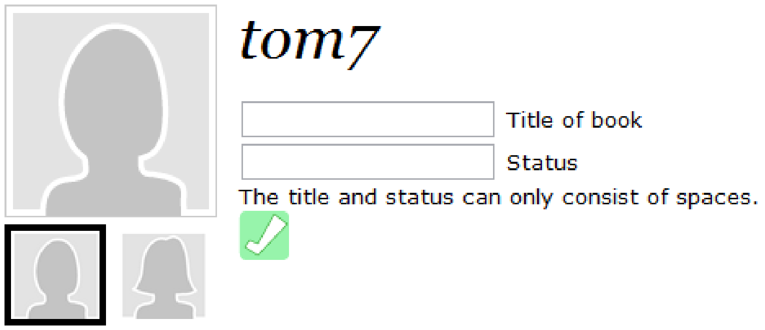
Well, first things first, get this in your web browser's location indicator box thing pronto:

http://national.month.of.pushing.spacebar.org/

Next you can do the usual stuff involving making an account and customizing the profile. To avoid the distractions and body dysmorphic stressors having to do with selecting a profile picture that adequately captures your on-line persona while attracting potential mates without seeming too self-involved or pimply (if male) or confusing potential creepers as to attractiveness status or gender appropriateness (if female), you can only select one of two faceless grey line drawings as your profile avatar.

"Endless customization options totaling 1 bit of entropy!" you exclaim. "What other fields can I type in?"

Well, don't get too excited but you can also modify the title of your book, and you can set your status message, which allows you to do social networking. These can only consist of spaces, but beware, for *they do not count towards your total number of spaces pressed.* To prevent cardiac involvement, a preview of the profile customization interface is presented in Figure 1, which should reduce arousal upon seeing it for the first time.


tom7

Title of book

Status

The title and status can only consist of spaces.

Figure 1: This is what it looks like when you're customizing your profile (well, my profile).

You're looking peppier already, and you say zestfully, "Aw right! Profile customized. Networks socialed. Now what?"

The next thing is to push spacebar. When on the proper page, pushing spacebar records the action and instantly apprises you of your progress. A bunch of data boxes and graph things show math entertainment for you as you press, and the counter indicates your tally front and center. How you type spaces is up to you. Some people prefer to *push*, others to *press*. Of course you can't just hold down the spacebar, duh.

Warning: Repeatedly pressing the spacebar, or any key, can cause repetitive stress injury. Just kidding! Nobody needs to worry about that. It's a fake disease for hypochondriacs like fibromyalgia. Just kidding about kidding! It's totally real. So's fibromyalgia! So's hypochondriasis! Just kidding! You totally have that! You're dying inside! Actually your hands are going to fall off from all the pressing! But seriously, press the spacebar gingerly and without repetitive stress, or RSI can be yours, truly. FALSE! DOUBLE FALSE! I was kidding! Hehe but really. Quotes around the whole thing.

Warning: If you try to do crazytimes stuff like have multi different computer devices all pushing spacebars at the same account at the same simultaneous, then you might lose record of some spacebars. Don't do that. It's crazy!

Hippies must turn on Javascript.

You can press spacebar all day and night, as far as I'm concerned, but the best strategies probably involve doing a bit each day until you get pretty sick of it. 3,226 spaces a day will get you to 100,000 just on time. The graphs help you see how you've been performing on a daily basis and what your pace needs to be for the rest of the national month, in order to reach the goal, pro-rated based on how much you've done. This thing is totally fancy. Also if you have the home page open, you can watch the progress of yourself and your "friends", and the numbers just like change right before your eyes like some kind of fucking wizard did that.

Oh by the way, I wanted to point out a little funny mystery here which was curious. Take a looksee at Figure 2.

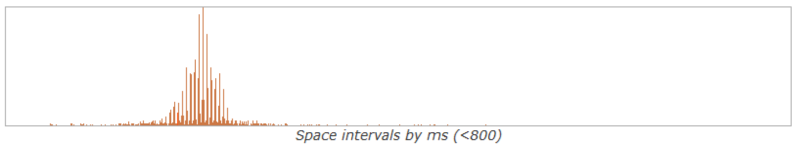


Figure 2: Someone has taken a comb filter to our spacebars. What is that about?

What we have here, muffin, is the distribution of intervals between space pushes in one practice session of pushing by the author.[2] What the *f*? We see the expected Gaussian distribution centered around 200 milliseconds. We also see fairly inexplicable regularly-spaced gaps in the otherwise pretty nicely shaped bell curve: A gap of 202 milliseconds occurs about a hundred times, 203 milliseconds about 20, 204 milliseconds only 4, 205 milliseconds only 6, then back up to 90 times for 206 ms. Why no love for milliseconds 204 & 205? It's like even though my spacebar presses have some random variance in them centered around the mean, there is much less randomness in the lowest digit. That don't make no sense, sugar, which is to say that it does not make sense. At least for a moment and then you realize there's probably some discretization thingy going on inside the Chrome browser that causes events to be more likey to be processed at certain times at certain interval-intervals which is not that disturbing after all, except that it also happens in the Safari browser? This may be a mystery that we never solve, candypants, and that's just how the world works sometimes.

In this paperwork, I told the tale of NaMoOfPuSp, in an attempt to engage you in its competitive spirit. Combining compressive art movements like NaNoWriMo and Album-a-Day with the increment-operator-based gameplay of World of Warcraft and Battlefield 3 (themselves popular topics of NaNoWriMo and Album-a-Day works), the National Month of Pushing Spacebar provides a way to achieve your creative dreams without actu-

[2]Don't get worried. Although the author is participating in this year's NaMoOfPuSp, and expects to whoop all y'all, he is not starting early or nuthin', he's just workin' out the kinks in the web-site. Everybody starts from scratch at the beginning of the National Month.

ally being creative, a way as fresh as celery from the crisper & princes of Bel-Air. So let's get on with it! Right now, even while you read this, you could be pressing spacebar. And who knows, maybe you could be the next whatever that chick is that wrote Twilight, or John Cage? It's never too late to join! (Unless it's after April 30, 2012.)

You can't click, but you could type:

http://national.month.of.pushing.spacebar.org/

Only you, or perhaps someone with your druthers, can prevent forest fires.

What Most Medical Students Know About Computer Science

Brian R. Hirshman
School of Medicine
University of California, San Diego
La Jolla, CA 92093
hirshman@ucsd.edu

Track 5

Programming languages research and other games for children

1. Modeling Perceived Cuteness
Nathan Brooks and Evelyn Yarzebinski
Keywords: cute, puppy, kitten, turtle, spider, snake, dog, cat, gecko, hamster

2. i-PHONE app stor : where is my pants
Dr. Tom Murphy VII, Ph.D. and Dr. Abigale G. Lade, Ph.D.
Keywords: pants, where, is my, juvenilia

3. An Extensible Platform for Upwards and Sidewards Mobility
David Renshaw

4. A modern optimizing compiler for a dynamically typed programming language: Standard ML of New Jersey (preliminary report)
Ian Zerny

5. Programming Language Checklist
Colin McMillen, Jason Reed, and Elly Jones

Modeling Perceived Cuteness

Nathan Brooks
Carnegie Mellon University

Evelyn Yarzebinski
Carnegie Mellon University

***Abstract*—In this paper, we empirically investigate factors which contribute to a creatures perceived cuteness (PC). For a variety of lifeforms that could be found in a home as a pet or occupant, we review both its physical factors as well as lifestyle choices of the homeowner for possible influences on PC. Data was collected from volunteer participants using an online submission form and several strong correlations were identified. In particular, strong trends in creatures leg-length-to-torso-volume (LL:TV) ratio were realized, as well as correlations with participant eye color. This work will serve as the basis for future work for providing better pet-owner matches and designing genetically engineered pets optimized for cuteness.**

I. Methods

We created a survey that involved a wide range of species one might reasonably keep as a pet. Although extremely finicky in allowing consistent access, the image sharing site Flickr was invaluable for our research. In order to avoid bias towards one species over another, great care was taken to attempt to keep the types of pictures uniform. When possible, we chose images that displayed a lateral perspective of the species in a natural light, which aligned closely with our hypothesis regarding the legs and torso. Additionally, we filtered the image results by skin or coat color to avoid a confound of subject color preference. In total, fifteen different images comprised our survey. Figure 2 contains the pictures provided for review by our subjects.

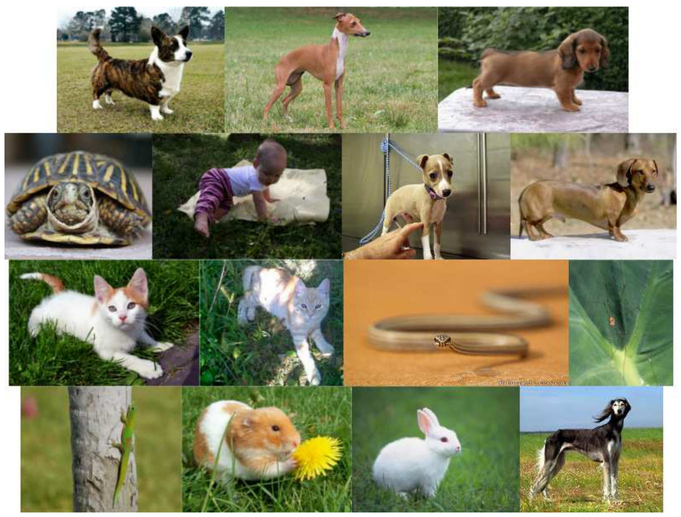


Fig. 1. Images used in the survey: Corgi, adult greyhound, baby dachshund, turtle, human baby, baby greyhound, adult dachshund, kitten, adult cat, snake, daddy long-legs, gecko, hamster, rabbit, adult saluki

Opinions about these fifteen species were collected, along with 5 demographic questions asking about expected things, such as gender, eye color, hair color, toothpaste preference, and favored Ninja Turtle. We created a 10 point Likert scale ranging from 1 (My lawyers will contact you about this life scarring image) to 10 (I am drooling from the cuteness) to aid in standardizing opinions. We collected data from 37 anonymized participants over a 6 hour period by posting our survey to social media websites.

II. Discussion

After collecting the data, we laboriously calculated the LL:TV ratio for each represented species through exact approximation. These data were then graphed in order of their increasing LL:TV ratio (on a scale from Snake to Daddy Longlegs) on the x axis and the PC rating on the y axis. No strong pattern was found, which we suspected was due to different PC factors for different types of animals. The data was then repartitioned to form results for three categories: Dogs, Furry Creatures, and Non-Furry Creatures. The Dog graph (Figure 2) followed a somewhat bell-curve shape; that is, a golden LL:TV ratio for dogs appears to exist. Another interesting pattern was observed for the non-furry creatures graph (Figure 3), which clearly is a table-shaped graph. It seems that for non-furry creatures, definite negative feelings regarding extreme values of the LL:TV ratio exist. However, the unimodel features of the dog and non-furry creature data is not reflected in the furry creature data (Figure 4), which is bimodal. A second peak is introduced by the inclusion of hamster data, which breaks the low-cuteness trend for very small LL:TV ratios found in the other data sets.

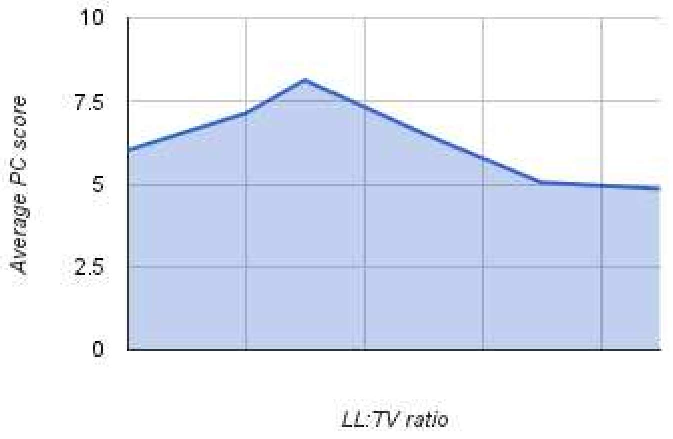


Fig. 2. Dog PC versus LL:TV ratio

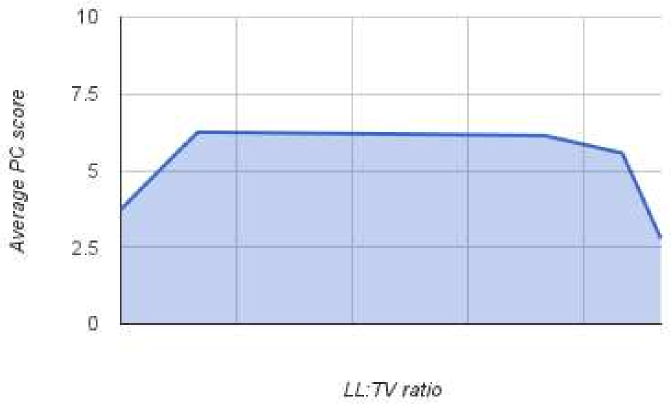


Fig. 3. Non-furry creature PC versus LL:TV ratio

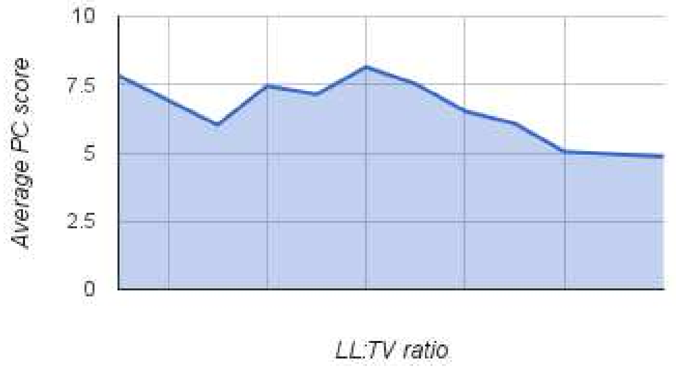


Fig. 4. Furry creature PC versus LL:TV ratio

One surprising finding was the relationship between eye color and a favorable PC rating of cats (Figure 5). A one-way between subjects ANOVA was conducted with Adult Cat rating as the dependent variable and Eye Color as the independent variable. There was a significant effect of Eye Color on the Adult Cat cuteness rating at p¡.05 level, F(3,33) = 8.550, p = .000. Bonferroni post hoc tests revealed that the mean Adult Cat score for blue-eyed participants (M = 8.00, SD = 1.70) was significantly different than the mean Adult Cat cuteness rating for brown-eyed participants (M = 5.00, SD = 1.08), p = .000; hazel-eyed participants (M = 5.50, SD = 1.31), p = .006; and green-eyed participants (M = 5.83, SD = 1.94), p = .043. Brown-eyed, hazel-eyed, and green-eyed participants did not give significantly different cuteness ratings from each other on a p¡.05 scale.

We also found a strong correlations between cuteness ratings for spiders and cuteness ratings for snakes (ie, creepy-crawlies); r(37) = .610, p = .000. Those who give low ratings to spiders also tend to give low ratings to snakes; those who give high ratings to spiders also give high ratings to snakes and obviously need to seek professional help.

We had hypothesized that there would be a relationship between PC of turtles and favored Ninja Turtle, but this was not confirmed with an ANOVA, F(3,33) = 1.488, p = .236. This is truly upsetting. Other expected results/dreams that were crushed include: Correlation between preference for younger TMNT (Rafael, Michelangelo) and higher PC ratings for younger animals Correlation between eye or hair color and favorite TMNT (Not enough blue-haired people out there) Knowledge of who the TMNT are (Really, Nitram[1]? Really?) And many more! (Its hard being a researcher sometimes...)

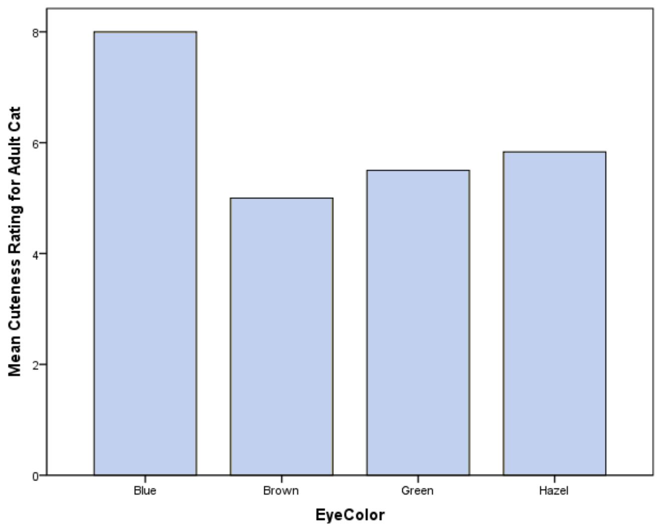


Fig. 5. Adult cat PC versus participant eye color

III. Future Directions

Despite our proposed LL:TV ratio, there is still some uncertainty in our model. A followup study seeking to further reduce bias through a finer animal picture and participant demographic selection could offer more accurate data. Specifically, two additional studies using exclusively cats or dogs are planned to avoid bias from animal preference. In order to collect consistent imagery, the authors may be required to perform extensive field research at local animal shelters, dog parks, and cat palaces. In addition, the bimodalness of Figure 4 presents interesting opportunities for genetic engineers and adventurers. Future work may include design of a furry animal with a LL:TV ratio of zero, or a rainforest safari to find creatures at LL:TV points where data is insufficient.

IV. Conclusion

We have discussed our preliminary analyses that provide some data-driven ways in which future pet ownership can be specifically tailored to meet a certain demographics preferences. We have shown, introduced, and discussed preferences regarding the LL:TV ratio and the implications this previously undiscovered ratio could have for future pet ownership. We have also discussed findings that show the natural relationship between certain species and genetic demographics. These metrics provide the basis for additional work in many areas to increase model accuracy and areas of application.

[1]Purely hypothetical name. The authors do not know any of the participants real names as per IRB[2]regulations.

[2]Irish Republican Brotherhood.

i-PHONE app stor :
where is my pants

Dr. Tom Murphy VII, Ph.D.   Dr. Abigale G. Lade, Ph.D.

1 April 2012

Abstract

Dear I–PHONE app stor please accept my applicatian for publishment of my game

Keywords: pants, where, is my; juvenilia

1 Introduction

Dear app stor I want you to have my game . My game is *where is my pants*. Where is my Pants is a newwest game for the i-touch (and I-PAD, ect.). I believe it should be publiced it the app stor. For every1 to play the game and find my pants. Because

1.1 Reasons because it should be publiced

1. This game it SO FUN!!
2. ∪ can play it any time or any place
3. I believe that if you put your mind to it, ∪ can accomplish anything
4. There are one thousand people with i-touch (and ¡-PAD ect.) . If evry touch and/or PAD bought this game. Then I would have at least RICH!!!!!!!
5. I don't know where did my pants went

[-1] Copyright © 2012 the Regents of the Wikiplia Foundation. Appears in SIGBOVIK 2012 with the permission of the Association for Computational Heresy; *IEEEEEE!* press, Verlag-Verlag volume no. 0x40-2A. Distribute free-ily. ♡ 0.00

1.2 Price of game

1. The price of the *where is my pants* will be $9.99 USA.

1.3 How to play

1. get down load game in i- touch
2. lunch game
3. look in the screen to see if you find pants
4. touch pants to see where is my pants
5. lotta of people have pants! so keep touching
6. if the pants .
7. ∪ can go in the next world to find more pants

1.4 Various pants worlds

Pants can be seen in various pants worlds

1. under the sea (a.k. in the ocean)
2. at the beach (any beach)
3. wisconsin

1.5 APP

`http://spacebar.org/pants/`

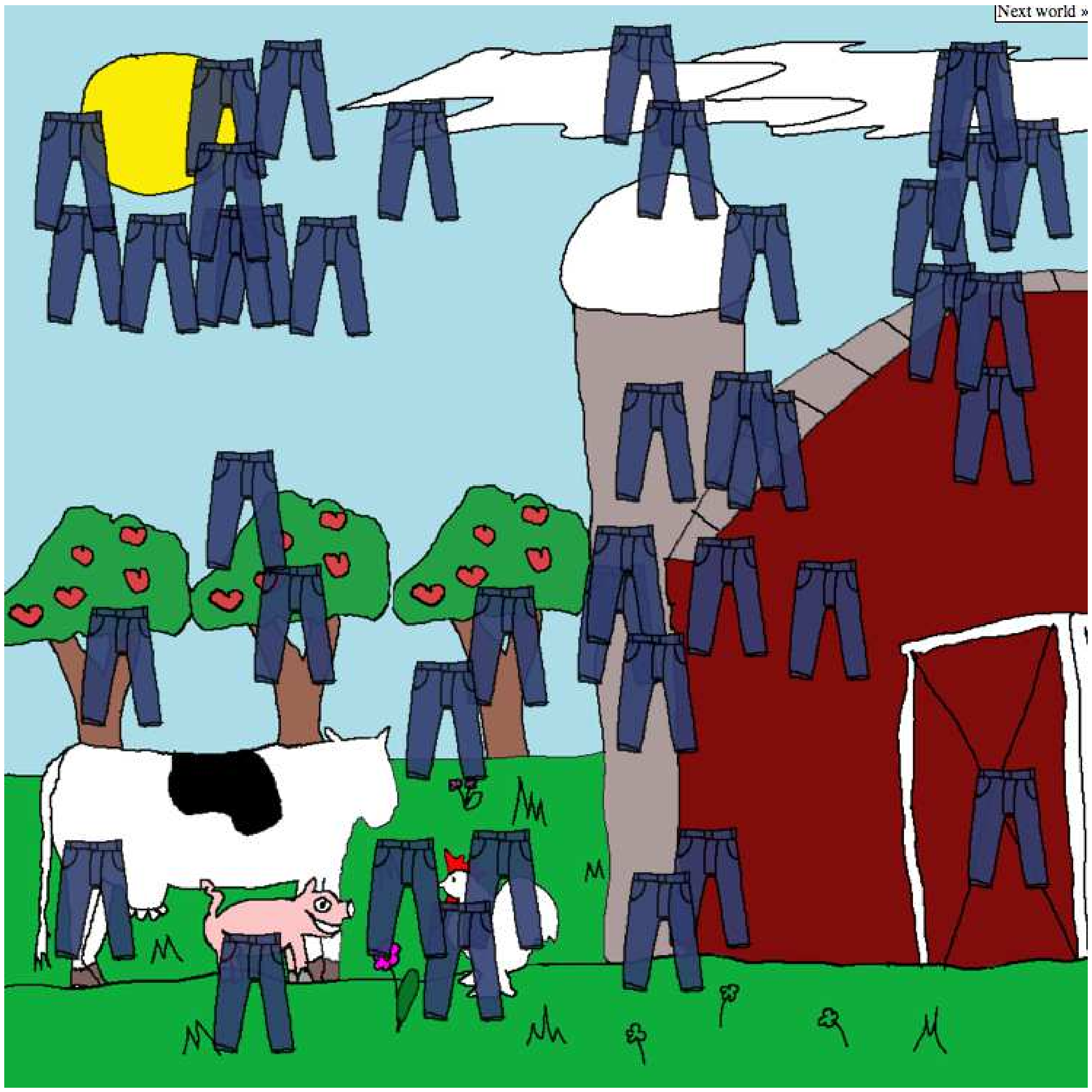


Figure 1: this is app *where is my pants*

An Extensible Platform for Upwards and Sidewards Mobility

David Renshaw

As platforms of all kinds become more and more important to our everyday lives, choosing among them becomes an increasingly difficult task. Platforms often represent an intricate set of priorities and compromises. Democratic, Republican, iOS, Android—who can say which is the least evil? And who can say which will be the least evil next year? Clearly we would all benefit if there were *extensible* platforms that could adapt to future situations and minimize the need for negative compromises today.

In the early 1980s, Shigeru Miyamoto pioneered the use of platforms for enhanced mobility in perilous environments [1], sparking innovation in an entire new field of research that continues to see remarkable developments to this day [2][3]. Extensibility, however, has never been a major focus in this field—until now. With this paper, we announce a new mobility platform that supersedes these previous contributions. Our platform is extensible in the sense that you can program it to move whatever way you want.

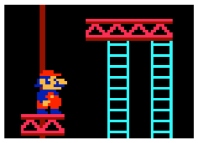


Fig. 1. A traditional platform is static or exhibits only tightly constrained motion.

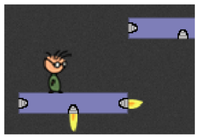


Fig. 2. Our prototype platform has three rocket boosters that can be reprogrammed.

You may try our new platform online by visiting this paper's interactive tech report at `www.cs.cmu.edu/~dwrensha/newplatform` .

References

1. S. Miyamoto. *Donkey Kong*. 1981.
2. J. Blow. *Braid*. 2008.
3. J. Carlsen, et al. *Limbo*. 2010.

A modern optimizing compiler for a dynamically typed programming language: Standard ML of New Jersey (preliminary report)

Ian Zerny `<ian@zerny.dk>`

April 1, 2012

Abstract

We present Dynamic ML: a dynamically typed language in the ML family. We show how Dynamic ML provides modern programming-language features, such as type reflection and implicit coercions. In addition, we show how the existing optimizing compiler infrastructure of Standard ML of New Jersey outperforms that of other dynamically typed programming languages currently in wide-spread use.

Contents

1 Introduction

2 Dynamic ML

3 Tools
- 3.1 The compiler: sed
- 3.2 The editor: dml-mode
- 3.3 The runtime: Standard ML of New Jersey

4 Benchmarks
- 4.1 The Fibonacci function
- 4.2 The faster Fibonacci function

5 JavaScript-style equality

6 Conclusion and perspectives

1 Introduction

Standard ML [2] is an aging language by now. It appears that most of its development has stagnated. Meanwhile, development has continued for other programming languages and among their features we can find many that Standard ML lacks: aspects, contracts, duck typing, dynamic scope, dynamic typing, implicit coercions, lvalues and real variables defined by assignment, multiple inheritance, objects, runtime code evaluation, and type reflection, just to name a few. These are all notable features provided by 2.0 programming languages.

In this work, we take a first step towards modernizing ML and consider dynamic types for an ML-like language. We follow Harper's vision of a dynamically typed language as a unityped language [1] and extend Standard ML with such a unityped embedded language. We assume the reader to be mildly familiar with functional languages (such as Standard ML) and with dynamically typed languages (such as Python or JavaScript). The entire development can be found on the author's website.[1]

2 Dynamic ML

We extend the syntax of Standard ML with constructs for dynamically typed features. The extension is defined by the following BNF:

$$\texttt{decl} \ni d ::= \mathsf{FUN}\ f[x_1, ..., x_n] = e\ \mathsf{NUF} \qquad \text{(recursive function definitions)}$$

$$\begin{array}{rll}
\texttt{exp} \ni e ::= & \mathsf{IF}\ e\ \mathsf{THEN}\ e\ \mathsf{ELSE}\ e & \text{(conditionals)} \\
\mid & \mathsf{FN}\ [x_1, ..., x_n]\ \ => \ e\ \mathsf{NF} & \text{(anonymous functions)} \\
\mid & e\$[e_1, ..., e_n] & \text{(application)} \\
\mid & \mathsf{VOID} & \text{(literal onething)} \\
\mid & \mathsf{TRUE} \mid \mathsf{FALSE} & \text{(literal booleans)} \\
\mid & \mathsf{NUM}\ n & \text{(literal numbers)} \\
\mid & \mathsf{STR}\ \text{"..."} & \text{(literal strings)} \\
\mid & \mathsf{LIST}\ [e_1, ..., e_n] & \text{(literal lists)} \\
\mid & \mathsf{as}\mathit{TYPE}\ e & \text{(type casts)} \\
\mid & \mathsf{is}\mathit{TYPE}\ e & \text{(type predicates)} \\
\mid & \mathsf{PRINT}\ e \mid \mathsf{PRINTLN}\ e & \text{(printing)}
\end{array}$$

where *TYPE* is one of the types: `VOID`, `BOOL`, `NUM`, `STR`, `LIST`, or `CLO`. Since Dynamic ML is an extension of Standard ML, `decl` and `exp` are super-sets of Standard ML declarations and expressions respectively. A program is then just a sequence of declarations (or a module or any other valid top-level declaration from Standard ML).

[1] `http://www.zerny.dk/dynamic-ml.html`

Equality. The primitive equality is the straightforward extension of structural equality in Standard ML. Two values are equal if they are of the same dynamic type and the raw data is structurally equal. For example:

```
val _ = (PRINTLN (NUM 1 == NUM 0);
         PRINTLN (NUM 0 == STR "0");
         PRINTLN (STR "0" == STR "0"));
```

will print `false`, `false`, and `true`.

Of course, this is a very simpleminded notion of equality, but luckily we can extend it to more powerful equalities if we wat. We illustrate this with a JavaScript-style semantics described in Section 5.

Type casting. Often we will need to pass the raw data of a dynamically typed value to some typed ML code. In such a case, we need explicit casts to obtain the raw value. These are done with the `as`*TYPE* type-cast expressions. For example, say we want the size of a string. We can use the Standard ML `String.size` for this:

```
FUN size[s] =
  NUM (Num.fromInt (String.size (asSTR s)))
NUF
val s = STR "Hello World!"
val n = size$[s]
```

which, when loaded, will give us the output:

```
val size = CLO fn : DML.t
val s = STR "Hello World!" : DML.t
val n = NUM 12 : DML.t
```

For the most part, such type casts will be implicit in the dynamically typed code, as can be seen here from the use of `size`. Should a type cast be applied to a dynamically typed value of a different type a `TYPE_CAST_ERROR` is raised.

Type reflection. Now that we have types at runtime we can inspect them! This is a hallmark feature of a solid dynamically typed language and Dynamic ML provides ample support for it. If we just need to check for one type we can use the `is`*TYPE* predicates. For example:

```
FUN foo[x] =
  IF isNUM x
  THEN x + NUM 42
  ELSE x
NUF
```

In other cases, we want have behaviors for several types. Here we can use the built in case analysis of Standard ML. For example, lets define a more reusable size function typical found in dynamically typed languages:

```
FUN size[x] =
  (case x
    of STR s  => NUM (Num.fromInt (String.size s))
     | LIST l => NUM (Num.fromInt (List.length l))
     | _ => VOID)
NUF
val n1 = size$[LIST [NUM 1, NUM 2]]
val n2 = size$[STR "Hello World!"]
val n3 = size$[NUM 10]
```

which yields:

```
val n1 = NUM 2 : DML.t
val n2 = NUM 12 : DML.t
val n3 = VOID : DML.t
```

In the above, we let `VOID` be the result for non-sizable values. This use of case analysis is kind of neat since we do not need to put in the `asSTR` and `asLIST` type casts in the case analysis.

3 Tools

Any real programming language comes with tools. Currently we provide a compiler (the `dmlc` program), a runtime system (the Standard ML of New Jersey system), and an editor (the dml-mode for Emacs).

3.1 The compiler: sed

The compiler for Dynamic ML is a sed-script that desugars the Dynamic ML constructs into Standard ML. This script is found in the file `dmlc`. To execute a program, simply desugar it and then run the output with Standard ML of New Jersey in the working directory containing the Dynamic ML implementation file `dml.sml`:

```
$ ./dmlc hello.dml
$ sml hello.dml.sml
...
val it = () : unit
'Hello World!'
-
```

3.2 The editor: dml-mode

The dml-mode for Emacs provides a nicer interface when working with Dynamic ML. It provides syntax highlighting and interacting with the underlying Standard ML process (which will be executing our Dynamic ML code). To load dml-mode in Emacs, hit `M-x load-file RET`, then input the path to dml-mode/dml-mode-startup.el. Now, opening a .dml file will start the *DML* mode. To load code from a buffer, simply hit C-c C-b. On the first go it will ask for the Standard ML process to use (sml should work if you have it installed). After that the code within the buffer will be compiled and then loaded and executed by Standard ML of New Jersey.

3.3 The runtime: Standard ML of New Jersey

Dynamic ML is executed by embedding it into Standard ML. This embedding defines the dynamic type as a one big recursive sum type: DML.t. Each type of dynamic runtime value is thus a summand in the dynamic type DML.t. The constructs used in the image of the embedding are contained within the Standard ML structure DML found in dml.sml.

4 Benchmarks

Having defined Dynamic ML and its execution environment we now benchmark it against Python, a popular and dynamically typed programming language.

4.1 The Fibonacci function

Consider the quintessential benchmark program: the recursively defined Fibonacci function. Here is its definition in Dynamic ML:

```
FUN fib[x] =
  IF x == NUM 0 THEN x
  ELSE IF x == NUM 1 THEN x
  ELSE fib$[x - NUM 1] + fib$[x - NUM 2]
NUF
```

and its counterpart in Python:

```
def fib(x):
    if x == 0: return x
    elif x == 1: return x
    else: return fib(x - 1) + fib(x - 2)
```

Timing these two functions applied to 35 yields a wall time of 11020 milliseconds for Dynamic ML and 20272 milliseconds for Python. That is 1.84 times slower for Python compared to Dynamic ML. Despite being defined for a functional language, the Standard ML of New Jersey implementation demonstrates its superiority as an optimizing compiler and runtime for a dynamically typed programming language.

4.2 The faster Fibonacci function

Somewhat surprisingly, we can do better yet! We can step-wise optimize our use of type reflection to improve performance of the Fibonacci function.

First, we create a "primitive" (i.e., non-dynamically typed) function inside `fib` that knows we only ever give it one argument:

```
val fib1 =
  FN [x] =>
    let fun go x =
            IF x == NUM 0 THEN x
            ELSE IF x == NUM 1 THEN x
            ELSE go (x - NUM 1) + go (x - NUM 2)
    in go x
    end
  NF
```

Here `fib1 35` runs in 6929 milliseconds.

Second, we call the inner function with the raw number data:

```
val fib2 =
  FN [x] =>
    let fun go x =
            if x = 0 then NUM x
            else if x = 1 then NUM x
            else go (Num.- (x,1)) + go (Num.-(x,2))
    in go (asNUM x)
    end
  NF
```

Here we have to be careful not to mix the raw numbers (which we do subtraction on) from the dynamically typed numbers (which we do addition on). Now `fib2 35` runs in 1487 milliseconds.

Third, we make sure we return the raw number from the inner function too:

```
val fib3 =
  FN [x] =>
    let fun go x =
            if x = 0 then x
            else if x = 1 then x
            else Num.+ (go (Num.- (x,1)),
                        go (Num.- (x,2)))
    in NUM (go (asNUM x))
    end
  NF
```

Now `fib3 35` runs in 617 milliseconds.

For each definition, Table 1 lists its total running time in milliseconds and its relative speed compared to original Dynamic ML definition. Our conclusion: not only is Dynamic ML faster than other popular dynamically typed languages, we can actually make programs even more efficient if we care to do so.

Language	RT in ms	Rel RT
Python	20272	1.84
Dynamic ML: fib	11020	1
Dynamic ML: fib1	6929	0.63
Dynamic ML: fib2	1487	0.14
Dynamic ML: fib3	617	0.06

Table 1: Running time for the Fibonacci function applied to 35

5 JavaScript-style equality

As mentioned in Section 2, the equality operator is not really what we find in most dynamically typed languages. For example, it requires the user to add annoying type conversions by hand. To avoid this we show how to define alternative interpretations of the dynamic types entirely within the language. This allows defining more user friendly operators.

In the Dynamic ML distribution, we provide a JavaScript structure, `JS`, that implements JavaScript-style conversions and operators. To redefine the Fibonacci function using JavaScript-style semantics we simply open up the JS-structure locally to the function:

```
local open JS in
  FUN fibjs[x] =
    IF x == NUM 0
    THEN x
    ELSE IF x == NUM 1
         THEN x
         ELSE fibjs$[x - NUM 1] + fibjs$[x - NUM 2]
  NUF
end
```

We have effectively extended the domain of the Fibonacci function to any dynamic value that can be interpreted as a number in a JavaScript-like way:

```
- fibjs$[STR "0"];
val it = STR "0" : t
- fibjs$[STR "1"];
val it = STR "1" : t
- fibjs$[STR "2"];
val it = NUM 1 : t
- fibjs$[STR "10"];
val it = NUM 55 : t
- fibjs$[STR ""];
val it = STR "" : t
- fibjs$[STR "-1"];
  C-c C-c
Interrupt
```

The JavaScript-like equality allows us to easily compare values of distinct types:

```
- open JS;
- STR "" == NUM 0;
val it = BOOL true : t
- NUM 0 == STR "0";
val it = BOOL true : t
```

That might look disturbing, but fret not, we cannot draw bogus conclusions such as the empty string being equal to a non-empty string:

```
- STR "" == STR "0";
val it = BOOL false : t
```

That would be nonsense.

6 Conclusion and perspectives

We have presented Dynamic ML, a dynamically typed language in the ML family. Dynamic ML shows that even though Standard ML has a static type system we don't need to use it. The language is a simple embedding into Standard ML and allows us to reuse the existing optimizing compiler and runtime infrastructure. Indeed, the execution of dynamically typed programs in Standard ML of New Jersey is faster than the execution of programs in Python.

To illustrate the ease of use and performance of Dynamic ML, we would have liked to finish with a more real-world program than the Fibonacci function. Unfortunately, our current map/reduce program fails with a `TYPE_CAST_ERROR` exception and we have not been able to debug the cause. We are now looking at building a debugger.

Our next step is to add objects. It is a bit of a stretch to call a language without an object system for a dynamically typed (or modern) language.

As illustrated in the optimizations of the Fibonacci function, a lot of time can be saved by optimizing out dynamic type casts. Future work should consider automatically optimizing these cases. We are looking at a refinement-type analysis to this effect.

We are also excited about Ohori et al.'s resent work on SML# [3], and hope that it will open new vistas for Bovik's use of thaumaturgic data bases.

Acknowledgments: This work was done while visiting CMU in the winter 2011/2012. Many thanks to Bob Harper for his lecture series on the principles of programming languages in which it became clear to the author that we can avoid the static types of Standard ML altogether.

References

[1] Robert Harper. Programming languages: Theory and practice. Working Draft v1.27. Available at `http://www.cs.cmu.edu/ rwh/plbook/`, 2012.

[2] Robert Harper, Robin Milner, and Mads Tofte. The definition of Standard ML, version 2. Report ECS-LFCS-88-62, University of Edinburgh, Edinburgh, Scotland, August 1988.

[3] Atsushi Ohori and Katsuhiro Ueno. Making Standard ML a practical database programming language. In Manuel M T Chakravarty, Olivier Danvy, and Zhenjiang Hu, editors, *ICFP*, pages 307–319, Tokyo, Japan, September 2011. ACM Press. Invited talk.

Programming Language Checklist

Colin McMillen Jason Reed Elly Jones

March 19, 2012

1 Introduction

You appear to be advocating a new:
☐ functional ☐ imperative ☐ object-oriented ☐ procedural ☐ stack-based
☐ "multi-paradigm" ☐ lazy ☐ eager ☐ statically-typed
☐ dynamically-typed ☐ pure ☐ impure ☐ non-hygienic ☐ visual
☐ beginner-friendly ☐ non-programmer-friendly
☐ completely incomprehensible
programming language. Your language will not work.
Here is why it will not work.

2 Motivation

You appear to believe that:
☐ Syntax is what makes programming difficult
☐ Garbage collection is free ☐ Computers have infinite memory
☐ Nobody really needs:
  ☐ concurrency ☐ a REPL ☐ debugger support ☐ IDE support ☐ I/O
  ☐ to interact with code not written in your language
☐ The entire world speaks 7-bit ASCII
☐ Scaling up to large software projects will be easy
☐ Convincing programmers to adopt a new language will be easy
☐ Convincing programmers to adopt a language-specific IDE will be easy
☐ Programmers love writing lots of boilerplate
☐ Specifying behaviors as "undefined" means nobody will rely on them
☐ "Spooky action at a distance" makes programming more fun

3 Problem Statement

Unfortunately, your language (has/lacks):
☐ comprehensible syntax ☐ semicolons ☐ significant whitespace ☐ macros
☐ implicit type conversion ☐ explicit casting ☐ type inference
☐ goto ☐ exceptions ☐ closures ☐ tail recursion ☐ coroutines
☐ reflection ☐ subtyping ☐ multiple inheritance ☐ operator overloading

☐ algebraic datatypes ☐ recursive types ☐ polymorphic types
☐ covariant array typing ☐ monads ☐ dependent types
☐ infix operators ☐ nested comments ☐ multi-line strings ☐ regexes
☐ call-by-value ☐ call-by-name ☐ call-by-reference ☐ call-cc

4 Theoretical Considerations

The following philosophical objections apply:

- ☐ Programmers shouldn't need category theory to write "Hello, World!"
- ☐ Programmers shouldn't develop RSI from writing "Hello, World!"
- ☐ The most significant program written in your language is its own compiler
- ☐ . . . isn't even its own compiler
- ☐ No language spec
- ☐ "The implementation is the spec"
    - ☐ The implementation is closed-source
    - ☐ covered by patents
    - ☐ not owned by you
- ☐ Your type system is unsound
- ☐ Your language cannot be unambiguously parsed
    - ☐ a proof of same is attached
    - ☐ invoking this proof crashes the compiler
- ☐ The name of your language makes it impossible to find on Google
- ☐ Interpreted languages will never be as fast as C
- ☐ Compiled languages will never be "extensible"
- ☐ Writing a compiler that understands English is AI-complete
- ☐ You rely on an optimization which has never been shown possible
- ☐ There are fewer than 100 people alive smart enough to use your language
- ☐ ____________ takes exponential time
- ☐ ____________ is known to be undecidable

5 Implementation

Your implementation has the following flaws:

- ☐ CPUs do not work that way
- ☐ RAM does not work that way
- ☐ VMs do not work that way
- ☐ Compilers do not work that way
- ☐ Compilers cannot work that way
- ☐ Shift-reduce conflicts in parsing seem to be resolved using rand()
- ☐ You require the compiler to be present at runtime
- ☐ You require the language runtime to be present at compile-time
- ☐ Your compiler errors are completely inscrutable
- ☐ Dangerous behavior is only a warning
- ☐ The compiler crashes if you look at it funny
- ☐ The VM crashes if you look at it funny

- ☐ You don't seem to understand basic optimization techniques
- ☐ You don't seem to understand basic systems programming
- ☐ You don't seem to understand pointers
- ☐ You don't seem to understand functions

6 Challenges

Additionally, your marketing has the following problems:

- ☐ Unsupported claims of increased productivity
- ☐ Unsupported claims of greater "ease of use"
- ☐ Obviously rigged benchmarks
  - ☐ Graphics, simulation, or crypto benchmarks where your code just calls handwritten assembly through your FFI
  - ☐ String-processing benchmarks where you just call PCRE
  - ☐ Matrix-math benchmarks where you just call BLAS
- ☐ No one really believes that your language is faster than:
  - ☐ assembly ☐ C ☐ FORTRAN ☐ Java ☐ Ruby ☐ Prolog
- ☐ Rejection of orthodox programming-language theory without justification
- ☐ Rejection of orthodox systems programming without justification
- ☐ Rejection of orthodox algorithmic theory without justification
- ☐ Rejection of basic computer science without justification

7 Related Work

Taking the wider ecosystem into account, I would like to note that:

- ☐ Your complex sample code would be one line in: ____________
- ☐ We already have an unsafe imperative language
- ☐ We already have a safe imperative OO language
- ☐ We already have a safe statically-typed eager functional language
- ☐ You have reinvented Lisp but worse
- ☐ You have reinvented Javascript but worse
- ☐ You have reinvented Java but worse
- ☐ You have reinvented C++ but worse
- ☐ You have reinvented PHP but worse
- ☐ You have reinvented PHP better, but that's still no justification
- ☐ You have reinvented Brainfuck but non-ironically

8 Conclusion

In conclusion, this is what I think of you:

- ☐ You have some interesting ideas, but this won't fly.
- ☐ This is a bad language, and you should feel bad for inventing it.
- ☐ Programming in this language is adequate punishment for inventing it.

Track 6

Protect yo' shit

1. Cryptographically-Strong Hashing with Physics
   James McCann and Fake Otherauthor

2. The Physics-Based Hashes of Twelve Really Good Ideas
   Patent Troll

3. A modest proposal for the purity of programming
   Robert J. Simmons and Tom Murphy VII
   Keywords: the next 700 terrible programming languages, hyper-driven devices, types

4. Preparation-Hom as the ideal completion of a Hemorrhoidal category
   Sitsin Cømfort and Minrøv Gørondt
   Keywords: hemorrhoids, fistulads, co-fistulads, push outs

5. A Randomized Commit Protocol for Adversarial - Nay, Supervillain - Workloads
   Ben Blum
   Keywords: two-phase, two-face, randomized algorithms, acid

6. Address space content randomization: exploit mitigation through data randomization
   Carlo Angiuli
   Keywords: security, optimization, nondeterministic memory, lower is better

Cryptographically-Strong Hashing with Physics

James McCann*
TCHOW

Fake Otherauthor
Justifying Plural Pronouns, Inc.

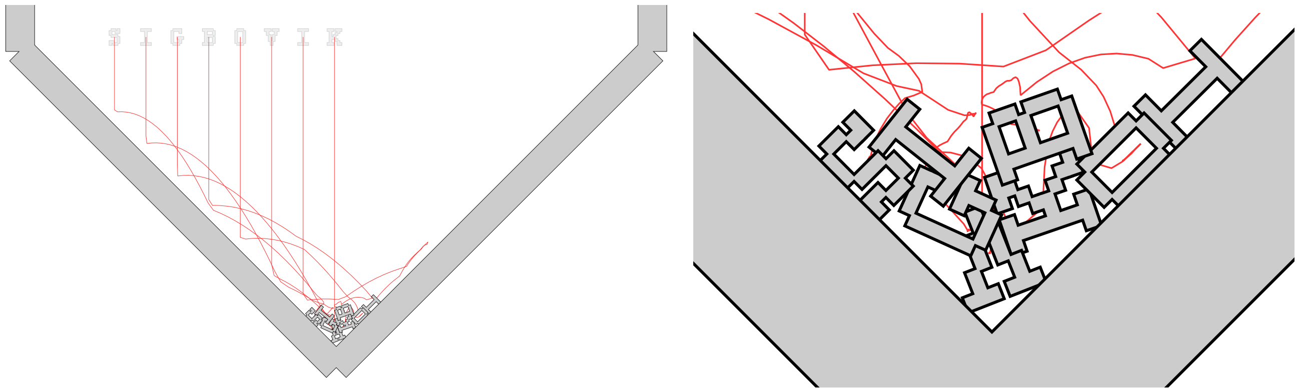


Figure 1: *Using physical simulation to hash the string "SIGBOVK". The resulting hash is* `0xe5aefadd258abc59a22b23-ab0a29723c37dac787decf8c46`. *The close-up view (right) highlights the complex interaction between the bytes in the string.*

Abstract

Good cryptographic hash functions must have the trapdoor property. Often, this property is interpreted figuratively – functions are constructed that are easy to run forward but hard to invert. In this work, we follow the definition more literally, constructing a hash function wherein the data is simulated as it falls into a heap, as if dropped through a trapdoor (e.g. Figure 1).

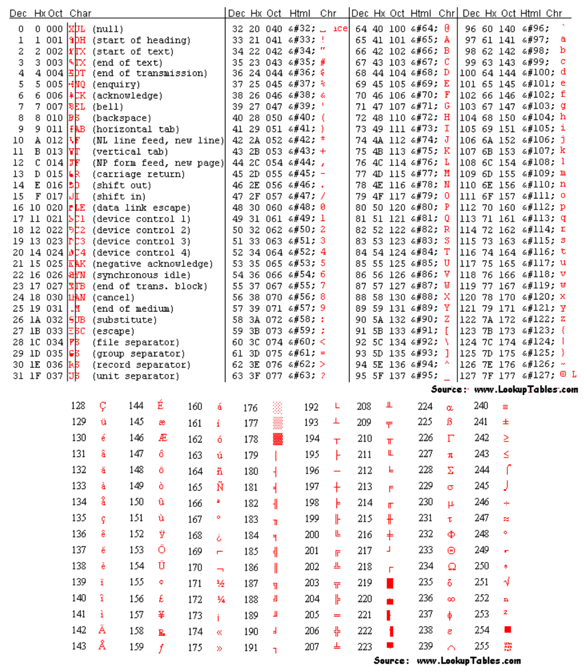
CR Categories: 4.2.O [Hashing]: Functions—Misc

1 Construction

Our hash function operates on between 1 and 256 bytes of data, and this operation proceeds in three phases*: first, the data bytes are converted into rigid bodies by assigning them shapes; second, the shapes are positioned in a simulated world which guarantees that they interact; second-and-a-half, the physical world is simulated forward in time; finally, positions of the bodies are read out of the simulation and used to construct the digest value. Each of these steps is described in more detail in the following subsections.

1.1 Byte-Shaping

Each byte is shaped by referring to the slightly modified ASCII and Extended-ASCII tables shown in Figure 2. These tables were copied from a canonical source [ASCII TABLE 1979], then modified to improve security.



Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	XUL	(null)	32	20	040	&#32;	␣ ace	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH	(start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX	(start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX	(end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT	(end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ	(enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK	(acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL	(bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS	(backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB	(horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT	(vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR	(carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO	(shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI	(shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE	(data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1	(device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2	(device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3	(device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4	(device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK	(negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN	(synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB	(end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN	(cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM	(end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB	(substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC	(escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS	(file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS	(group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS	(record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US	(unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	⊠ L

Source: www.LookupTables.com

128	Ç	144	É	160	á	176	░	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	[illegible]

Source: www.LookupTables.com

Figure 2: *The modified ASCII tables used to shape each byte in our simulation.*

*e-mail: ix@tchow.com

*Just like residential 220-volt power in the United States.

```
0: Read-Back($B$):
1:     $hash \leftarrow 0$
2:     for ($body \in B$):
3:         $hash \leftarrow (hash >> (23 \cdot 8)) \oplus (hash << 8)$
4:         $hash \leftarrow hash \oplus \text{state}(body)$
```

Figure 3: *Pseudo-code for the read-back procedure which converts rigid body position and velocity information into the final hashed value.*

All non-printing characters were replaced with printing variants[†]. Any identical characters were modified to be unique, as identical characters lead to trivial collisions. Finally, character 255 was replaced with a pixelated version of the phrase "u mad bro?" in order to "troll" potential attackers.

The character shape from these modified tables is used both as collision geometry and to specify the mass and moment of inertia of each byte.

1.2 Rigid Body Positioning

Once bytes have been transformed to rigid bodies, they are arranged in a 16-wide grid above a v-shaped pit. The v-shaped pit ensures that the bytes interact as they plummet by forcing them to bounce into each-other. Because all characters in the stream are guaranteed to interact, we don't need to pad short streams to guarantee high-quality results.

1.3 Simulation

The actual simulation of the system is handled through the open source Box2D engine [Catto 2001]. Box2D is ideal for this situation because it is a deterministic physics engine; that is, no random values are used in the simulation, the warm-starting of the solvers, or elsewhere. The engine is set to run 500 steps of length 1/50th of a second.

1.4 Read-back

Once the simulation has been run, the final 24-byte hash is constructed by reading back the computed position, angle, velocity, and angular velocity of each body (all stored as 4-byte floats) and xor-ing the values together, rotating the hash by one byte as each body is read. Pseudo-code for this procedure is given in Figure 3.

2 Examples

We show the results of our hash function on several inputs. Hashing the string "SIGBOVIK" (Figure 1) takes only 105ms. A longer string consisting of bytes 0-255 (in order) takes slightly longer – 10416ms (Figure 4. Longer inputs than this must be chained, by using several runs of the hash

[†]Our chosen simulator does not handle noncorporeal bodies.

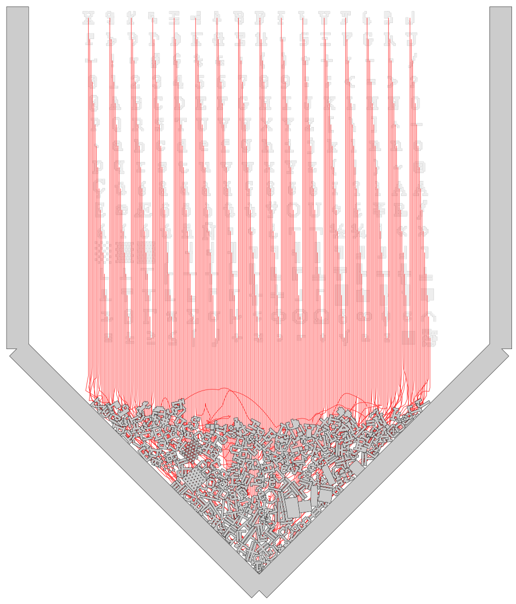


Figure 4: *Hashing bytes 0-255 with our function. The resulting hash is* `0x63477ee68bafc0021d895f8-baebd0f877d21965744a5c686`.

function, each of which covers the previous hash value and up to 232 bytes of new input. Examples of chained hash computations are shown in Figures 5 and 6.

3 Conclusion

We have demonstrated that a literal trapdoor function is a reasonable implementation for a trapdoor function.

Our simulation operates in 2D, but there is no reason that a higher-dimensional physical simulation couldn't be used to perform a similar hash. This would provide more degrees of freedom and – thus – a potentially longer digest value.

Our function takes on the order of 1 minute / kilobyte to process data. This may seem slow, but it is actually quite fast (considering that it can take *hours* to compute the SHA1 of similar data by hand). As our hash function is widely adopted, we believe that special-purpose hardware or compute accelerators (e.g. [nVIDIA 1981]) will be adapted to support it adeptly.

References

ASCII TABLE, 1979. Ascii table. http://www.asciitable.com.

CATTO, E., 2001. Box2D. http://box2d.org.

NVIDIA, 1981. PhysX. http://www.not-actually-the-url-for-physics.fake.

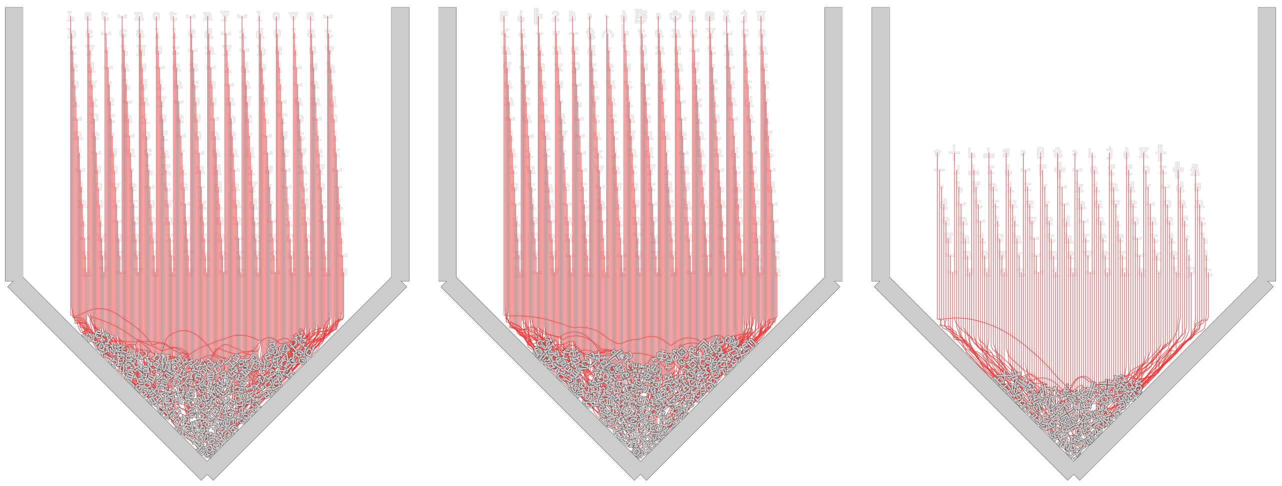


Figure 5: *Hashing Shakespeare's Sonnet 105 by chaining together several hash computations. The final hash value (that is, the hash computed in the last hash computation) is* `0x4512b25b71e73d989b9f1802a75d46b2aa14354c155596a8`*. The total computation time for the hash to be computed in the hash computation's computation of the hash is* $8048 + 8946 + 2634$ *ms.*

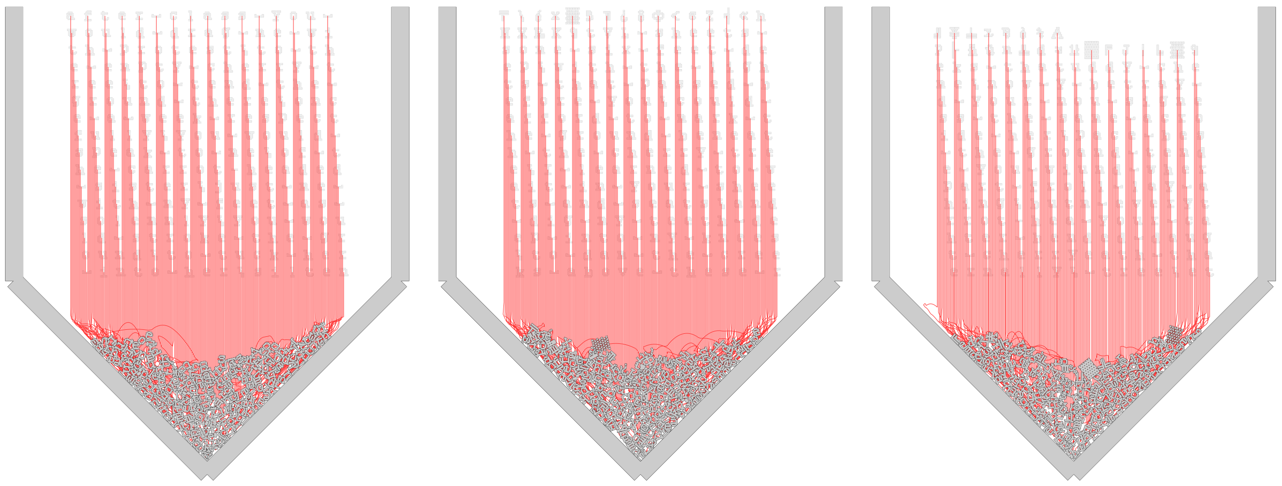


Figure 6: *Hashing Parenthetical Girls'* Survived by her mother, *by chaining together several hash computations. The final hash value of* `0x4606c9903fa4576ab6f38bc87f5260cef9a27a30734e36fb` *is the only thing that differentiates it from Shakespeare, as far as we're concerned.*

The Physics-Based Hashes of Twelve Really Good Ideas

Patent Troll[*]
Corportation, Inc. LLC. Co.

Abstract

I've just had a thought! And another! Wow, there goes another one! This is great – I'm having so many thoughts. Instead of putting in the sweat and tears and blood and potentially other, more objectionable, bodily fluids to actually develop them, though, I think I'll just write brief descriptions and then publish hashes of them.

Then I'll hide over here in the metaphorical bushes, waiting for someone else to actually make something of the idea.

Then I'll strike, like a spring-loaded snake.

Yessssss.

Everyday Health and Medicine
`0xda675f8fa9c2b22b9f2a2a1bdd603a4438305fcde63840ad`

Improving Personal Habits
`0x59ea76bd12f401f08982388737677b439aaab50c4b4d97d4`

Antarctic Salmon Fishery Management
`0xa05308567b06ed79b0cbd8ba073370ea731dfa11a62d4003`

Infinite Forward Temporal Redundancy Planning
`0x5c5b7ac171fec54505aaf1b6bb4ddd3efcd65510d39c7c22`

Government Regulation of Vice
`0x4624f69784ae359ed5ac7eac18479d3110a0520757a5c121`

Early Diversity Education Methodologies
`0x75f0874f58549d8eb1ae0cc1bc3279592a6e989e15a241f1`

Consumeristism Enhancement Protocols
`0x83b674c9aaf7d4d5bcde5572051412de51273a3f02bcc118`

Getting Those Darn Noisy Kids Off My Lawn
`0x5c6587ac9e4f696711e597fecd90eba43eefb0fd1d570b6a`

On the Personal Psychology of Pricing
`0x447f9c404c83d49bd98f0e17a3d5de61e7ae1a1dcea1af5b`

Exploiting the Workforce
`0x2d52d030a0c5358862e1bbe11c1bf70e663aeaeb21882a60`

Institutional Design Pattern Abstractions
`0xccb55678453278ef9ce51e57a06d6c79dd08d9d33c983a54`

Self-referential Regulation
`0xf01514216cb822fab06cafe3a9af91648a2c85deeb6b7eb1`

[*]e-mail: ix@tchow.com

A modest proposal for the purity of programming

Robert J. Simmons
and
Tom Murphy VII

Terrible ideas permeate the world of programming languages, and the harm that these ideas do is lasting. Academic research is intended to ameloriate some of this harm, but the connection between academic PL research and industry grows ever more tenuous. This harms both realms. Terrible ideas continue to hold back the benefits that computer science and software enginnering seek to bring to the world. On the other side, we should note the demoralizing effect of this tendency on academic PL research: why work on "practical" research if you will be universally ignored?

The more principled among us might suggest that we continue to generate the best ideas and see what happens, that the right ideas will win in time. The time for such velvet-glove approaches is long past. This is a war, and it is time to use all the resources at our disposal. We propose a method and apparatus to save the world from itself.

1. INTRODUCTION

Programming languages are fundamentally structured expressions of human thought; they allow mere mortal humans like you and I to wield the power of the fantastically complex and powerful computing devices that live in our phones, coffee machines, and MacBooks. Like most other human enterprises, programming languages are subject to fad and whim and fashion. *Unlike* most other human enterprises, the damage done by ill-considered ideas can be exceptionally lasting if the languages incorporating these ideas are used to build big important systems. If you dislike covariant generics, you should mourn that we will seriously be stuck with Java's covariant arrays until the heat death of the universe. If you *do* like covariant generics, you are wrong, but nevertheless you should mourn that we will seriously be stuck with Java's invariant generics until the heat death of the universe.

The research community on programming languages, imperfect as it is, has usually seen the worst of these disasters coming in advance. However, in the diffuse and surreally contentions community of programming languages research, attempts to understand errors and contain their worst damage of bad ideas seem tend only to solemnize their status as "well-understood features of modern programming languages," ensuring their inclusion in the next great disaster.

We propose a radically different strategy: we will save the future from our follies of today by setting our countenances towards discovering tomorrow's bad ideas and tasteless fads *first.* From a pure research perspective, this is the ultimate folly, not that that's ever stopped us before [6; 9; 5; 8; 1]. But the stakes are too high to keep our hands clean. It's time to bring out the big guns, and put our bad ideas to work in the (regulatory) marketplace.

1.1 Primer on patent law

Patent law in the United States traces back to Article 1, Section 8 of the United States Constitution, which gives Congress the power "to promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries." This is the same portion of the constitution that copyright law stems from; however, in their core functionality patent law and copyright law are quite different. Patents are particularly unusual in the degree of exclusivity they give: a patent gives the holder a *monopoly* over the exercise of their patent for a period of time. Under current U.S. law, this period of time extends 20 years from the date when the patent was officially filed.

One view of the patent system is that it is a deal the public makes with an inventor. Patents must provide enough information so that a skilled person can carry out the claimed invention—this is called the *sufficiency of disclosure* requirement. So, a patent gives you 20 years when you have absolute control over who is allowed to use your invention, but this awesome power comes at a cost—you basically guarantee that, in 20 years, everybody is going to be able to use your invention. Furthermore, because patents are published right away, you give everybody else a head start on innovating further based on your ideas—and if someone else comes with an innovation that's awesome enough, they can get a brand new patent that *you* can't use. And Science advances! If you don't want to give your good ideas to anybody else right away and/or if you think you could keep your awesome secret a secret for more than 20 years, then it's to your advantage as an inventor *not* to make this trade, and have your invention be a trade secret rather than a patented invention.

That, at least, is the civics book lesson for how patents work, but the patent system has had some difficulties coping with the complex nature technological innovation in the modern world. We will briefly describe two phenomena [2] that have arisen around patent law in the context of modern information technology: the standardization of *reasonable and non-discriminatory* licensing terms, and the scourge of *patent trolls*. Both of these phenomena [2] are relevant to our method and apparatus for saving the world, `The /dev/urand Foundation`, described in Section 3.

1.1.1 *Reasonable and non-discriminatory terms.* Standards bodies have to work around the fact that many standards inevitably are covered by patents. Standards-setting organizations allow for patented technologies to be used in standards, but require that the patent holder provide **R**easonable **a**nd **N**on-**D**iscriminatory licensing options, or RAND. This term isn't terribly well defined, but the intent is to ensure that anyone can implement the standard, by paying a fair licensing fee to the patent holder until the patent expires, and is generally good for technology.

To see why this is important, consider a standards body with representatives from 57 companies all working on a new standard for Carrier Pigeon Message Formatting modernizing RFC 1149. BBN Labs has a new patent on avian foot massage technology using cardboard [3]. Without revealing this fact to the consortium, BBN Labs influences the standards body to incorporate the requirement for post-packet-delivery corrugated massage as a quality assurance mechanism. Then, after waiting for the standard to be incorporated into every soda machine in America, BBN Labs can demand arbitrary fees from the users and distributors of their massage-enhanced pigeon packet technology. If people refuse to pay, they can be legally barred from using the (standards-backed) technology they have already bought and paid for.

RAND licensing is aimed at avoiding this scenario. When RAND terms are required, then BBN Labs still has something to gain from the adoption of their patented technology in the standard, but they have less to gain from concealing this from their standards-body partners. RAND is not a solution to patent-encumbered standards, merely a way to make them work in the real world. But, critically, RAND licensing is not itself a fundamental part of patent law—if BBN Labs are *not* a part of the standards committee, then they can still wield their patents despite the standards committee members being bound by RAND terms when it comes to their own patented inventions.

1.1.2 *Patent trolls.* The interrelated nature of technological innovation, and the frequency with which fundamental ideas spring into existence in multiple places at the same time, has led to the prevalence of *patent trolls.* The term is generally defined as people and companies that buy lots of patents, wait for people to start using the ideas contained therein naturally, and then extort the maximum rents possible from the users.

Patent trolls generally wield two types of patent: The specific patent that is nearly certain to get rediscovered in time, and the general, overly-broad patent that basically apply to everything ever, like the people who freakin' patented *linked lists* in freakin' 2004 [11] or the patent trolls currently suing all your cell phone makers because they let you select emoticons from a list [7]. The latter form of troll patent is often cited as evidence for the necessity of patent reform, as, despite the fact that these patents are usually unenforceable, the mere threat of patent litigation can be used to extract rents from other innovators and have a chilling effect on innovation overall.

1.2 The chillaxing effect

Our method and apparatus was inspired by some person on Twitter [4], though we stress that this does not count as prior art. As the random Twitter-person observes, it's a good thing that the fundamental good ideas in computer science aren't covered by patents, because they are useful for helping droves of other computer scientists do their work. But what if the *bad* ideas in computer science were covered by patents? It might prevent droves of other computer scientists from doing their work. This *chillaxing effect*, effectively wielded, could simultaneously get the attention of the largest corporate players in programming languages and software engineering and refocus the efforts of academic research in directions that have not been meticulously chillaxed.

I mean, we tried to come up with a bad idea a couple of years ago [8], and then we learned at the ICFP in Baltimore that Milner had actually come up with the *same bad idea* in one of the early ML implementations, before replacing it with the less dumb idea later on. Milner could have *patented* this bad idea when he came up with the better idea, thus giving the forces of sanity with a powerful tool against the next person who tries to implement the bad idea and stop there.

2. EXAMPLES

Coming up with bad ideas for programming languages is very easy. The challenge is to come up with ideas that are broad enough to cover many possible instantiations of the idea, specific enough to be patentable, and likely to be encountered in real up-start languages, where they can be stamped out. As usual in science, our approach is stochastic; we simply generate as many patents as we can. Many patents will never be useful in the fight against bad programming languages, but these cause no harm.[1] As a demonstration, this section contains a list of bad programming language ideas that we came up with, no sweat. Cringe away:

(1) The input programs are written as recipes

(2) You just give examples of what a function should do in certain circumstances, and when it encounters an input that is not specified it. . .
   (a) . . . linearly interpolates between known answers
   (b) . . . uses genetic programming to come up with a short program that satisfies your constraints and also works on this input
   (c) . . . pauses and waits for the programmer to finish the program
   (d) . . . asks the user what the answer should be, adding it to the database
   (e) . . . searches for code on the internet that meets the example-based specs, and prompts the programmer or user as to which one should be used
   (f) . . .

(3) Input programs are written in musical notation

(4) Input programs are graphical diagrams written in UML, XML, flow charts, as maps, circuits, or two-dimensional ASCII

(5) Programs are written in three-dimensional layered text, perhaps in different colors and with alpha channels, to specify interleaved threads

(6) Every program is a substring of the *lorem ipsum* text

(7) Everything in the language is just a. . .
   (a) . . . string literal, including keywords
   (b) . . . capital *i* or lowercase *L*
   (c) . . . continuously differentiable probability density function
   (d) . . . hash table mapping hash tables to hash tables
   (e) . . . n-tuple
   (f) . . . finite permutation
   (g) . . . 7-bit integer
   (h) . . . coercion
   (i) . . . rule
   (j) . . . exception, except exceptions; those are normal

[1]Informal studies in CMU's Principles of Programming group have shown preliminary evidence that bad programming languages can actually cause physical harm among those that have established taste and predisposition to logic. Observed effects include facepalms and grimaces, nausea, fatigue, emotional lability (uncontrolled weeping or laughing), Bobface (first identified by William J. Lovas), and dry mouth. Other harm is more direct, such as lacerations or bruises by being throttled by academic advisors. If this proves to be a problem, simple safety measures such as biohazard suits, coordinate-transform and other reversible mind-encryption systems, or simply employing the inadverse, may be used.

   (k) . . . arbitrary-precision rational number
   (l) . . . priority queue, Fibonacci heap, b-tree, pixel, regular expression, presheaf, commutative diagram, metaphor, monad
   (m) . . . MP3
   (n) . . . SMS
   (o) . . . mutex
   (p) . . . non-uniform rational b-spline

(8) Programming languages for children or the elderly

(9) Programming languages based on telling stories

(10) Programming languages based on architecture, org charts, HTML, CSS, military strategy, or airplanes

(11) Instead of stack-based control-flow, use queue-based, tree-based, dataflow-network-based

(12) 4/3 CPS

(13) Every value is represented as the 256-bit content hash; elimination forms are distributed hashtable lookups; revision control is built into the concrete syntax of the language

(14) Unification always succeeds, forking the program with each of the two expressions to be unified substituted in that position; only if both fail does unification fail

(15) Realize every program you wish to write as actually the test case for a metaprogram that generates the program

(16) Language with only 20 keywords, one for each of the SPEC benchmarks

(17) Language with only one keyword, whose semantics implements a compiler for the language itself

(18) `call-ac`, call with all continuations

(19) Second-class data: All data must be top-level global declarations, and can't change. Functions are first-class.

(20) Gesture-based concrete syntax

(21) Programs are realized as dashboards with knobs, buttons, and cable connections between them

(22) No matter what, the program keeps going, attempting to repair itself and keep trying actions that fail

(23) Programs are abstract geometric shapes

(24) Type has type . . .
   (a) type
   (b) int
   (c) kind
   (d) type $\rightarrow$ type
   (e) object
   (f) null

(25) To protect against the problem where sometimes someone called a function with an empty string, "emptyable" types, which include all values of the type except the "empty" one (`""`, 0, NaN, 0.0, nil, `{}`, false, etc.).

(26) To work around the global `errno` problem, every value of a type includes the possibility of integers standing for an error code

(27) Lazy natural (co-)numbers, where the output of a numeric program is only a lower bound that may get higher as it continues computing

(28) A language where the compiler is integrated into the language as a feature, which takes first class source code to first class compiled binaries, within the language

(29) There's a global registry, in the world, and whenever a function returns, you check to see if any function in the world has registered a hook to process it

You see how easy this is? If you are a programming language expert you might even have thought of some languages that already use these ideas. If so, *this is all the more reason to support our foundation*, because had we started earlier, we could have saved the world some trouble!

It is worthwhile to try to acquire patents that are very broad, since these can be used to attack almost any language, even one with unanticipated bad ideas. For example:

2.1 Method and apparatus for attaching state to an object

This patent describes a method and apparatus for attaching state to objects in computer programs. The invention consists of a symbolic program running in computer memory and an object (which may be a value, hash table, list, function, binary data, array, vector, n-tuple, presheaf, source file, class, run-time exception, finite or infinite tape, or isomorphic representation). The claims are as follows:

1. A system for attaching state to the said object

2. The method of claim 1 where the state is binary data

3. The method of claim 1 where the state is an assertion about the behavior of the object

4. The method of claim 1 where the state is itself an object

5. The method of claim 4 where the state is the same object, or some property therein

6. The methods of claims 1–5 where the apparatus of attachment is reference

6.a. Reference by pointer
6.b. Reference by index
6.c. Reference by symbolic identifier
6.c. Representations isomorphic to those in claims 6.a.–6.c.

7. The methods of claims 1–5 where the apparatus of attachment is containment

(etc.)

2.2 Method and apparatus for determining the control flow of programs

This patent describes a method and apparatus for determining the control flow in computer programs. The invention consists of a symbolic program running in computer memory, with a notion of *current* and *next* state (which may be an instruction pointer, index, expression to evaluate, continuation, value, covalue, stack, queue, execution context, thread or thread pool, process image, cursor, tape head, phonograph stylus, or isomorphic representation). The claims are as follows:

1. A system for determining the next state from the previous state

2. The method of claim 1, where the determination includes the contents of the program's memory

3. The method of claim 1, where the determination includes the current state

4. The method of claim 1, where the determination includes external inputs

5. The method of claim 1, where the determination includes nondeterministic factors

6. The method of claim 1, where the determination is fixed ahead of time

(etc.)

It is hard to imagine any programming language that would not be covered by both of these patents. Many perfectly sensible languages would be impacted as well, but this is not a problem: We can choose to license the patent to languages that we judge to be tasteful, perhaps imposing additional contractual restrictions on the licensees, even regarding things not covered by our patent pool, such as our personal preferences about indentation style and capitalization of identifiers. We can easily develop hundreds of such applications and again use the stochastic method to ensure a high likelihood of having one granted.

3. THE /DEV/URAND FOUNDATION

The patents shall be administered by a new non-profit foundation, known as `The /dev/urand Foundation`. The organization is named for the RAND concept of patent licensing described in Section 1.1.1. The /dev/urand Foundation differs in that its licensing is **U**n**r**easonable **a**nd **N**ot-not **D**iscriminatory: We do not offer licenses for any amount of money or other consideration. Our patents on bad ideas will simply never be licensed, enjoining anyone from using those ideas for the duration of the patents. Our over-broad patents will be licensed in a blatantly discriminatory fashion, only to languages that we think are tasteful. We might even withhold licenses when an individual has done something that we just don't like, like has a name that's hard to spell, or didn't accept one of our papers to a prestigious conference. This is totally legal.

We are going up against some of the biggest companies in the world, such as Larry Wall and Guido van Rossum, however, and so we anticipate that specific patents on bad ideas will be more powerful than potentially indefensible over-broad patents. However, there is certainly a role for both kinds of patent trolling. The point is to strike fear into the hearts of would-be hobbyists and academics, with the expectation that this would be generally *bad* for the programming language ecosystem, which we can all agree is pretty much up shit's creek without a paddle.

Lawsuits: coming to a workshop on reinventing the wheel near you.

4. CONCLUSION

We have presented a plan for saving programming from the scourge of clumsy innovation. Surely the plan is distasteful, and perhaps you think the world would be a better place if the negative effects of out-of-control patent law—whether they be chilling (bad) or chillaxing (awesome)—were curtailed with the limitation or elimination of software patents. Maybe so! But the problem with refusing to let the ends justify the means is that when you do that *the other team ends up with more means*. And we refuse to settle for average.

We will fight fire with whatever firefighting means we can get our fricking hands on. The future of programming depends on it!

REFERENCES

[1] Blum, Benjamin (ed). “Proceedings of SIGBOVIK 2011.” ACH Press. Pittsburgh, Pennsylvania. 2011.

[2] Do do dododo. “Phenomena.” Do do do do. “Phenomena.” Do do dododo dododo dododo dodmuch dododododo do do do do do.

[3] Ebert, Michael A. “Corrugated recreational device for pets.” Patent Application 11/757,456. Filed June 4, 2007.

[4] Gorman, Jason. (jasongorman). “We should think ourselves very lucky that Alan Turing didn't patent “a single machine which can be used to compute any computable sequence”” March 13, 2012. Tweet.

[5] Jones, Laurie A (ed). “Proceedings of SIGBOVIK 2009.” ACH Press. Pittsburgh, Pennsylvania. 2009.

[6] Leffert, Akiva and Jason Reed (eds). “Proceedings of SIGBOVIK 2007.” ACH Press. Pittsburgh, Pennsylvania. 2007.

[7] Nelson, Jonathan O. “Emoticon input method and apparatus.” Patent 7,167,731 B2. Granted January 23, 2007.

[8] Martens, Chris (ed). “Proceedings of SIGBOVIK 2010.” ACH Press. Pittsburgh, Pennsylvania. 2010.

[9] Simmons, Robert J. (ed). “Proceedings of SIGBOVIK 2008.” ACH Press. Pittsburgh, Pennsylvania. 2008.

[10] Simmons, Robert J., Nels E. Beckman, and Dr. Tom Murphy VII, Ph.D. “Functional Perl: Programming with Recursion Schemes in Python.” In *SIGBOVIK 2010*.

[11] Wang, Ming-Jen. “Linked list.” Patent 7,028,023. Granted April 11, 2006.

Preparation-Hom as the ideal completion of a Hemorrhoidal category

Sitsin Cømfort and Minrøv Gørondt
Institute for Advanced Sitting
Denmark

Abstract

Hemorrhoidal categories are a source of considerable pain for many a theorist who has found himself incapacitated by this class of abstract nonsense. In this work we characterize *Preparation-Hom*, which represents the ideal completion of a hemorrhoidal category, and which should therefore provide immediate relief for those classes of problems involving hemorrhoidal categories.

1 Introduction

Hemorrhoidal categories first emerged from the exploration of alimentary canal structures in the early part of the 20th century [2]. Formed by the enrichment of a *rectal category* [1] with a set of so-called *prolapsed functors* [4], the discovery of which has resulted from years of strenuous labor, and has resulted in considerable difficulty in passing beyond this mathematical obstruction. While controversial, the prevailing view of hemorrhoidal categories is that they were originally developed in categories with too many push outs.

Preparation-Hom represents a novel advance in giving a formal treatment of hemorrhoidal categories, and therefore represents a significant contribution of the present work. In Section 2 we demonstrate the use and potential applications of Preparation-Hom to hemorrhoidal categories, and give a proof of correctness.

2 Correctness

We characterise the correctness of Preparation-Hom in Fig. 1. The diagram will commute for any category that is hemorrhoidal, and the hemorrhoidal endofunctor should be absent after applying the Preparation-Hom transformation.

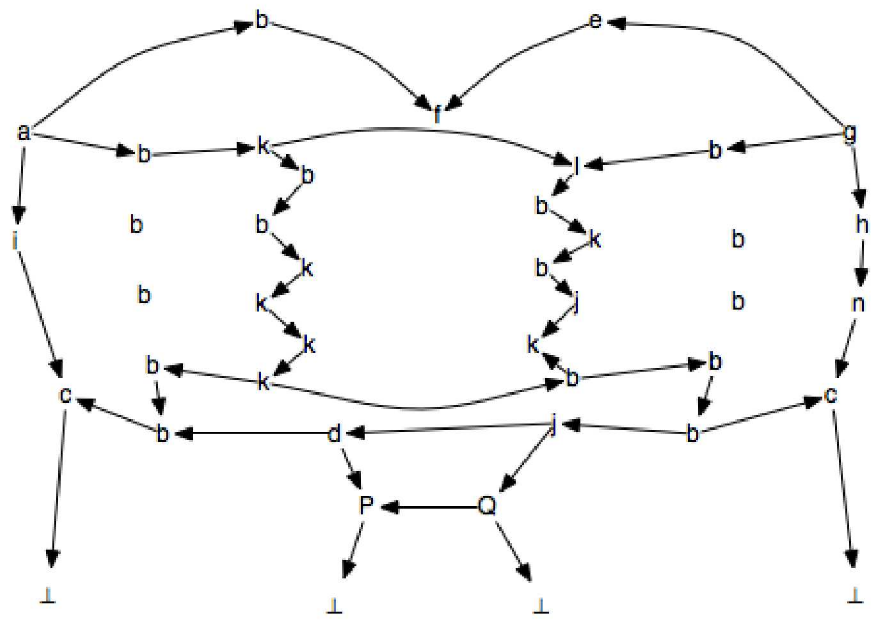


Figure 1: Any category is hemorrhoidal assuming this diagram commutes

3 Future Work

The authors intend to characterize the applicability of Preparation-Hom to both *fistulads* and *co-fistulads*[3].

4 Conclusion

We are so very sorry. Really. Very, very sorry.

References

[1] Seymour Butts. *The Handbook of Applied Procto-logical Mathematics.* Springer, 1976.

[2] W Lovas. Poop Search in Laxative Logic. In *SIGBOVIK 2009*, 2009.

[3] (redacted). *The Children's Illustrated Guide to Fistulads.* (Unpublishable), 1994.

[4] R Santorum. I can prolapse functors and so can you! *Anals of Mathematics*, 2012.

A Randomized Commit Protocol for Adversarial - Nay, Supervillain - Workloads

Ben Blum (bblum@andrew.cmu.edu)

2011.04.01

Abstract

We propose a new algorithm based on binary random number generation for distributed atomic transaction commit, called the Two-Face Commit Protocol. Rather than relying on multiple servers collaboratively voting, as does the current industry standard algorithm, the Two-Face protocol relies on simple coin flips. We show that our algorithm is more robust in the case of adversarial environments, and therefore is more appropriate for supervillain-level workloads.

1 Introduction

Distributed atomic commit is a well-known problem involving multiple participating servers attempting to agree on whether to accept a change. This mandates careful communication among the servers, to ensure that all servers agree at the end on whether the change was accepted or rejected.

The current industry standard algorithm, the Two-Phase Commit Protocol[Raz95], uses two distinct phases in which all servers communicate with a master server, voting on whether to commit or roll back. This is expensive, for it requires excessive multidirectional network traffic during both phases. It is also much more prone to failure, since a single "no" vote from any server will cause the transaction to abort. Furthermore, given potential server crashes, the Two-Phase algorithm may sometimes need manual intervention to restore the collective state machine.

In this paper, we present a commit protocol with none of these downsides. Our algorithm, called the Two-Face Commit Protocol, relies on a much simpler coin-flipping mechanism. In essence, the master server flips a coin to decide whether you get to keep your data or lose it. In contrast with Two-Phase, Two-Face is less prone to state corruption, and also gives more predictable performance in the face of unpredictable, possibly antagonistic, machine failures.

2 Algorithm

(a) Constantius II; Rome, ca. 350   (b) Kronor; Sweden, ca. 1992   (c) Toonie; Canada, ca. 2005

Figure 1: As seen in algorithm 2, the Two-Face Commit Protocol is parameterized over the type of coin, for added flexibility[Ran].

We present the algorithms for both the conventional protocol and our new protocol.

Note that the first line of the new algorithm has no overall effect, and is presented for the sake of contrast. We omit it for performance reasons in our implementation.

Algorithm 1 The conventional Two-Phase Commit Protocol.

```
function PHASE1(servers)
    x ←TRUE
    for all s ∈ servers do
        x ← x∧RECEIVEVOTE(s)
    end for
    return x
end function

function PHASE2/SUCCESS(servers)
    for all s ∈ servers do
        NOTIFYSUCCESS(s)
    end for
    COMMIT()
end function

function PHASE2/FAIL(servers)
    for all s ∈ servers do
        NOTIFYFAILURE(s)
    end for
    ROLLBACK()
end function
function TWOPHASE(servers)
    success ←PHASE1(s)
    if success then
        PHASE2/SUCCESS(s)
    else
        PHASE2/FAIL(s)
    end if
end function
```

Algorithm 2 The new Two-Face Commit Protocol.

```
function TWOFACE(servers, coin)
    success ←PHASE1(s)
    success ←FLIP(coin)
    if success then
        PHASE2/SUCCESS(s)
    else
        PHASE2/FAIL(s)
    end if
end function
```

3 The Benchmark SIGBOVIK Deserves, But Not the Benchmark SIGBOVIK Needs

We tested the Two-Face Commit Protocol on several different system configurations. First, we tested multiple different architectures for Two-Face effectiveness.

(a) Intel 80486, 1990 (b) Cyrix Cx5X86, 1995 (simulated) (c) AMD Phenom B3, 2008

Figure 2: Implementations of Two-Face on different testing platforms.

We also tested Two-Face by parameterizing it over several different coin implementations, which are depicted in figure 1.

First we measure the performance of the Two-Face Commit Protocol against the more popular Two-Phase Commit Protocol. We perform 50,000 trials of each protocol, and measure for each protocol how many commits succeed, how many fail, and how many required manual intervention to fix a stuck state caused by failure of one or more participating servers. These figures represent the average of the results on all hardware and coin configurations.

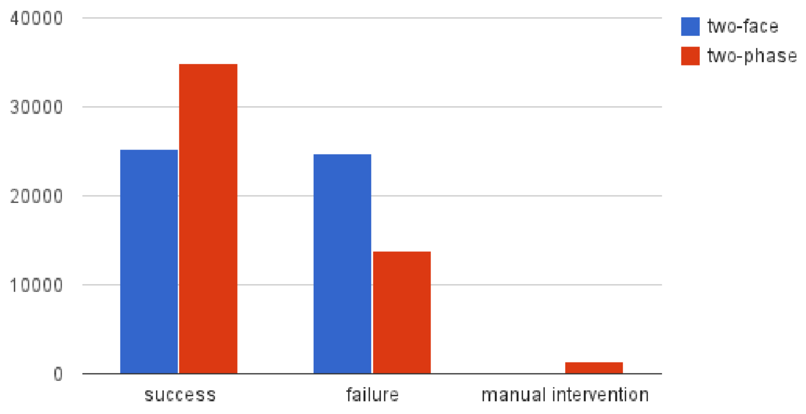


Figure 3: Basic comparison of Two-Face against Two-Phase

We see that in general, the venerable Two-Phase Commit Protocol generates more successes than our

Two-Face algorithm. This is not surprising, considering the law of large numbers.

Next we investigate the success rate of the two algorithms with increasing rates of spontaneous server failures. We configured each of the participating servers in our trial to artificially crash with a certain probability, and produced the following graph.

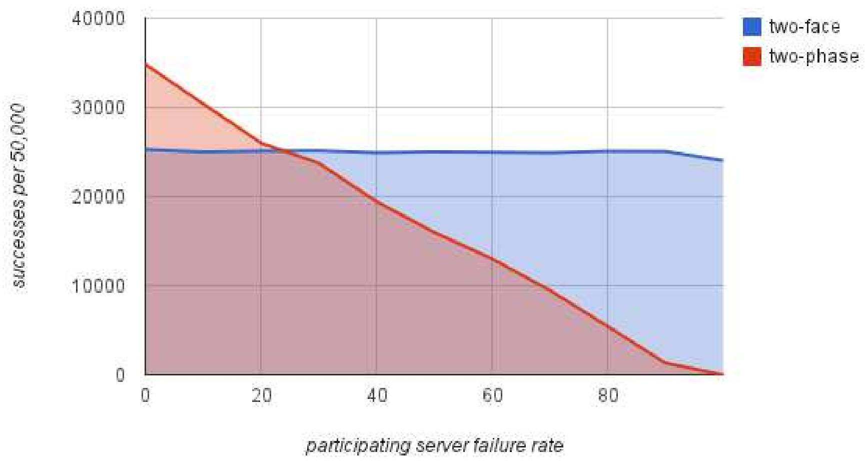


Figure 4: Success rate of Two-{Fac,Phas}e with increasing server failure rates

Here we see that our algorithm is much more resilient against spontaneous participant failure than the industry standard algorithm.

4 Future Work

To address the deficiency presented in figure 3, we plan to investigate the performance of the Two-Face algorithm when using weighted coins, and perhaps even double-headed and double-tailed coins.

We also anticipate that the Two-Face Commit Protocol will be employed in an upcoming work that will publish this summer[NNG12]. We plan to analyze the effectiveness of our algorithm in this real-world environment.

5 Conclusion

We have presented the Two-Face Commit Protocol, a randomized algorithm for atomic committing of distributed transactions using coin-flipping. The algorithm is designed to be useful with highly antagonistic workloads, such as having acid thrown in your face, ACID thrown in your filesystem, or being tied up in a burning building in a room full of oil drums.

References

[NNG12] Christopher Nolan, Jonathan Nolan, and David S. Goyer. The dark knight rises. In *Proceedings of the 3rd Symposium on Superhero Principles*, SOSP, Summer 2012.

[Ran] Random Dot Org. Coin flipper. `http://www.random.org/coins`.

[Raz95] Yoav Raz. The dynamic two phase commitment (d2pc) protocol. In *International Conference on Database Theory*, pages 162–176, 1995.

Address space content randomization: exploit mitigation through data randomization

Carlo Angiuli

March 19, 2012

1 Introduction

Address space layout randomization [5] is a popular technique for mitigating buffer overflow vulnerabilities in software. By randomizing the memory layout of software when it is loaded into memory, and additionally prohibiting memory pages from being both writable and executable, an attacker is limited to guessing the address of any already-loaded code his or her exploit executes.

However, these security mechanisms do not affect the layout of data on the stack, permitting an attacker to access return pointers, and therefore, quickly guess the memory offset of `libc` [6].

Notice that this attack is possible only because memory *content* often contains memory *addresses*, thus leaking information about the randomized layout. Furthermore, we observe that *memory* is susceptible to replay attacks, because it maps addresses to content in a deterministic fashion [3].

We propose a novel method for mitigating memory-related program exploits by randomizing the *content* stored at addresses in memory, which we call Address Space Content Randomization (ASCR). This provides many benefits over previous vulnerability mitigations:

1. By randomizing the contents of memory, we supercede the need to randomize memory addresses via ASLR.

2. Our technique can be implemented in a per-application fashion, rather than relying on kernel facilities.

3. Applications which run under ASCR are necessarily more robust, because they cannot exploit memory replay attacks by expecting deterministic memory behavior.

4. We drastically improve the performance of most applications by removing the need to access memory.

2 Implementation

We discuss our implementations of static and dynamic ASCR as LLVM optimization passes, and the security and performance tradeoffs between these implementations.

2.1 Static ASCR

In static ASCR, all memory reads are replaced by random numbers generated at compile time. As a result, attackers who only have access to a program's source code cannot craft successful exploits, as they cannot determine what behavior the program will have when compiled.

As with ASLR, programmers must ensure that their software works correctly in the presence of ASCR. In this case, local variables cannot be assumed to hold any particular value, although they are guaranteed to hold the same value across multiple invocations of a program. These strict guidelines promote a healthy feeling of mistrust toward one's compiler, which in turn, promotes more robust, secure software.

It is worth noting that ASCR requires no kernel support, unlike ASLR, and thus can be adopted immediately—any programmer can immediately begin shipping code which takes advantage of our strong security guarantees.

We have implemented an LLVM optimization pass which performs static ASCR conversion on any LLVM bitcode, through the following transformations:

1. Load instructions are robustly replaced by random numbers.

2. Store instructions are removed, as they leak information.

By following this by standard compiler optimizations, we achieve surprisingly large speedups and code size reductions in a variety of benchmarks.

2.2 Dynamic ASCR

In dynamic ASCR, all memory reads are replaced by random numbers generated at runtime. As a result, even attackers who have access to a program's binary cannot craft successful exploits, as runtime behavior is unpredictable.

We use `libc`'s `rand` function to generate random numbers, reseeding it with the time at the start of every program. This eases debugging, as programmers can reproduce bugs by resetting their system clock each time they run software with dynamic ASCR.

Dynamic ASCR relies only on the presence of a random number generator, and can again be implemented without additional kernel support. Our LLVM optimization pass is similar to the static ASCR pass:

1. Load instructions are robustly replaced by calls to `rand`.

2. Store instructions are removed, as they leak information.

We achieve greater security than static ASCR, but at the cost of performance—the dynamic nature of this feature reduces the optimization opportunities available afterwards.

3 Evaluation

Our present implementations of ASCR randomize only data, but we anticipate extending it to randomize code as well. This generalized form of ASCR entirely supercedes the need for ASLR; while ASLR makes it difficult for attackers to obtain pointers to protected regions of memory, ASCR simply decouples those pointers from access to any program code or data.

Our observation is that all memory-related attacks hinge upon the association of pointers with data in memory, so by stopping this attack vector entirely, we secure software against a very large class of vulnerabilities.

3.1 Performance

As a simple integer benchmark, we implemented the industry-standard recursive algorithm [1, 1] for Fibonacci numbers:

```
int fib(int x) {
  if (x == 0)
    return 1;
  else if (x == 1)
    return 1;
  else
    return fib(x-1) + fib(x-2);
}
```

and computed `fib(50)`. We compiled the same `fib` function three ways: with static ASCR (followed by standard optimizations), dynamic ASCR (plus optimizations), and `clang -O2` without ASCR. We compare the performance and values of these three implementations in Figures 1 and 2.

Under static ASCR, the `fib` function is compiled to the single instruction:

```
ret i32 1957747793
```

as compared to the 13 instructions required in the non-ASCR implementation. As shown in Figure 1, this optimization results in drastically improved performance. By removing the compiler's dangerous assumption that mutations to local state are persistent, we manage to optimize `fib` so that it executes in constant time—a major improvement.

Under dynamic ASCR, each reference to x is replaced by a call to `rand`. The recursive `fib` calls cannot be optimized away, and are not, in general, called with numbers smaller than x. In particular, under ASCR it is faulty to assume that $x - i < x$. Thus, the algorithm very quickly runs out of stack space and terminates, again much more quickly than without ASCR!

While detractors argue that dynamic ASCR makes it difficult to analyze the behavior of software, we believe that this is due simply to a lack of understanding of the behavior of pseudorandom number generators. If a user desires their application to not segfault, or return a particular number, they need only determine to what to set their system clock.

As with all systems performance analyses, we acknowledge a need to adequately address the long tail distribution [4]. The author believes the long tail distribution has favored the colobus monkey, to which Fortune herself has distributed a marvelous tail of approximately 25 inches of length [2].

4 Conclusion

We have shown that address space content randomization is a powerful exploit mitigation technique which can be applied to programs at compile time without additional runtime support. ASCR is able to defeat a large class of exploits which use replay attacks on memory, i.e., by relying on any memory cells to contain non-random information.

We hope to generalize our current work by also randomizing code in memory, instead of limiting ourselves to data. This would entirely remove the need for ASLR, by rendering code pointers—and indeed, code—useless in all situations. The author is confident that full ASCR, as described above, would in fact successfully defeat all software exploits.

We believe that the increased effort needed to develop applications which exhibit desired behavior with ASCR is balanced by the security and performance afforded by the ASCR regime. In our experiments, when applied in tandem with traditional compiler optimizations, ASCR results in massively decreased performance time and runtime values.

References

[1] Angiuli, C. Address space content randomization: exploit mitigation through data randomization. In *Proceedings of the 6th ACH conference on Special Interest Group on Harry Qwerty Bovik* (Pittsburgh, PA, USA, 2012), SIGBOVIK '12, ACH.

[2] Brookfield Zoo. Field guide – colobus monkey. `http://www.brookfieldzoo.org/pagegen/htm/fix/fg/fg_body.asp?sAnimal=colobus+monkey`.

[3] Eckhardt, D. Virtual memory. `http://www.cs.cmu.edu/afs/cs/academic/class/15213-s08/www/lectures/class16.pdf`.

[4] Maurer, M. The effect of the long-tail distribution on system performance evaluations. In *Proceedings of the 6th ACH conference on Special Interest Group on Harry Quechua Bovik* (Pittsburgh, PA, USA, 2012), SIGBOVIK '12, ACH.

[5] PaX Team. Pax address space layout randomization (ASLR). `http://pax.grsecurity.net/docs/aslr.txt`.

[6] Shacham, H., Page, M., Pfaff, B., Goh, E.-J., Modadugu, N., and Boneh, D. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM conference on Computer and communications security* (New York, NY, USA, 2004), CCS '04, ACM, pp. 298–307.

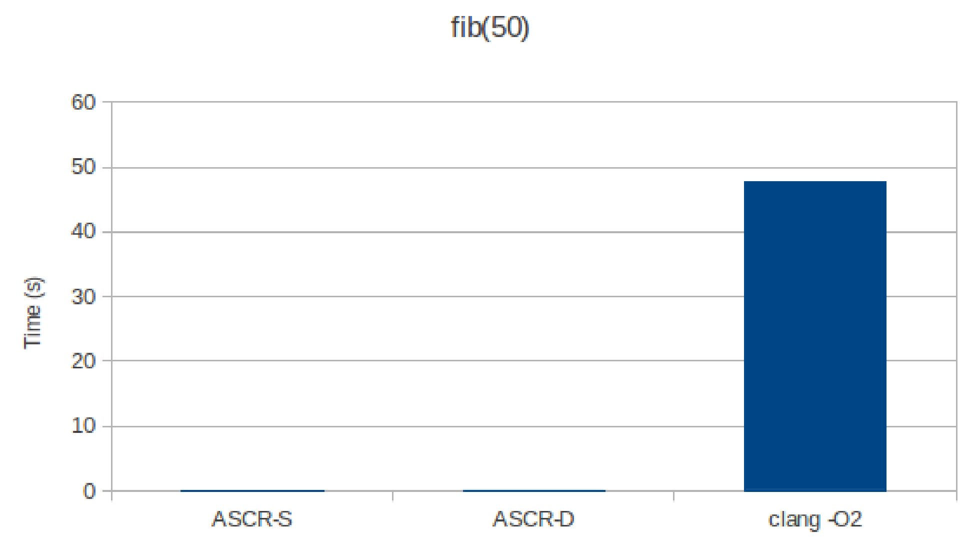


Figure 1: Performance of `fib(50)` using static/dynamic/no ASCR. (Lower is better.)

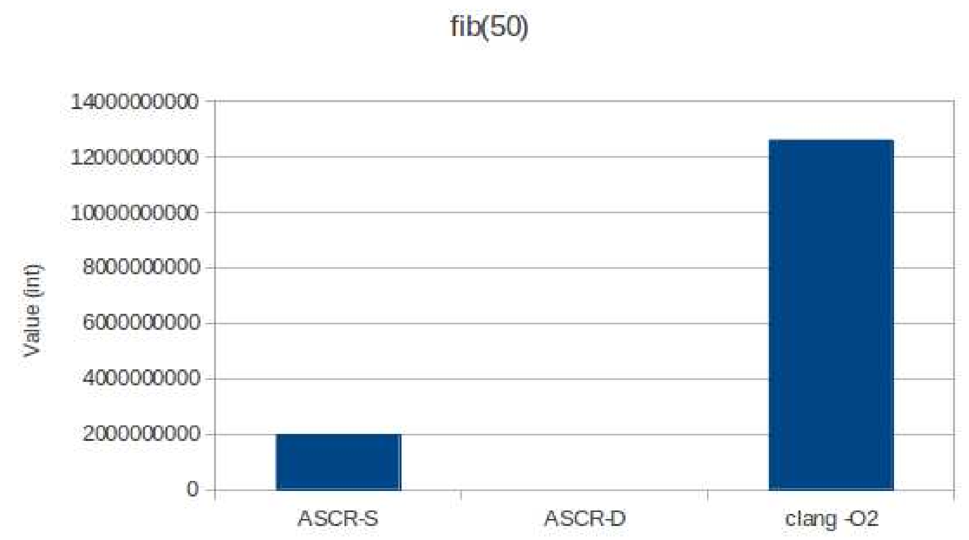


Figure 2: Value of `fib(50)` using static/dynamic/no ASCR. (Lower is better.)