SWEETPEA: A LANGUAGE FOR EXPERIMENTAL DESIGN

by

Anastasia Cherkaev

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Masters of Science

Department of Computer Science

The University of Utah

October 2018

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Anastasia Cherkaev**
has been approved by the following supervisory committee members:

Vivek Srikumar ,	Chair(s)	**22 Oct 2018** Date Approved
Matthew Flatt ,	Member	**22 Oct 2018** Date Approved
Jonathon Cohen ,	Member	**22 Oct 2018** Date Approved
none ,	Member	**22 Oct 2018** Date Approved
none ,	Member	**22 Oct 2018** Date Approved

by **Ross Whitaker** , Chair/Dean of
the Department/College/School of **Computer Science**
and by **David Kieda** , Dean of The Graduate School.

ABSTRACT

The replicability crisis in experimental science is fueled by a lack of transparent and explicit discussion of experimental design in published work. An experimental design is a description of experimental factors and how to map those factors onto a sequence of trials such that researchers can draw statistically valid conclusions. This thesis introduces SweetPea, a SAT-sampler aided language that facilitates creating understandable, reproducible and statistically robust experiemental designs. SweetPea consists of (1) a high-level language to declaratively describe an experimental design, and (2) a low-level runtime to generate unbiased sequences of trials given satisfiable constraints. The high-level language provides primitives that closely match natural descriptions of experimental designs. To ensure statistically significant results, every possible sequence of trials that satisfies the design must have an equal likelihood of being chosen for the experiment. The low-level runtime samples sequences of trials by compiling experimental designs into Boolean logic, which are then passed to a SAT-sampler. The SAT-sampler provides guarantees that the solutions it finds are statistically robust.

For my parents, Alice and Bob.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOTATION AND SYMBOLS

α	fine-structure (dimensionless) constant, approximately $1/137$
α	radiation of doubly-ionized helium ions, He++
β	radiation of electrons
γ	radiation of very high frequency, beyond that of X rays
γ	Euler's constant, approximately $0.577\,215\ldots$
δ	stepsize in numerical integration
$\delta(x)$	Dirac's famous function
ϵ	a tiny number, usually in the context of a limit to zero
$\zeta(x)$	the famous Riemann zeta function
$\ldots$	$\ldots$
$\psi(x)$	logarithmic derivative of the gamma function
ω	frequency

CHAPTER 1

MOTIVATION

1.1 Reliable Experimental Design and Reproducibility Crisis

The replicability crisis in experimental science is fueled by a lack of transparent and explicit discussion of experimental design in published work. While there are many software tools for modeling and running experiments, there are, to the best of our knowledge, none for designing them. Currently, scientists design experiments by writing complex scripts which manually balance the experimental factors of interests. There are two major issues with this approach: the first is that it may not (and often does not) produce unbiased sequence of trials. In practice, researchers construct these sequences without any statistical guarantees because the brute force solution for constructing unbiased sequences by enumerating all options is intractable; for a typical experiment the size of the search space is 10^{100}. The second issue is that this approach is brittle. It is easy to introduce bugs that go unnoticed but which may have large consequences, and it is difficult to verify and reproduce another researcher's implementation. [1] [2] [4]

1.2 Declarative Programming: Science without the Engineering Burden

blahblahblah

[3]

blahblahblah

[5]

1.3 Requirements Statement

There is a need for a software system that allows domain scientist to design unbiased, replicable experiments. Moreover, this system needs to provide an easy-to-use, declarative

interface so that scientists who are not necessarily trained as software engineers can create and reason about complicated experimental designs, and transparently share their experimental setups and design choices. SweetPea is just such a system; it is a language which provides semantics for describing experiments, a runtime for synthesizing experimental sequences from specifications, and a set of tools for debugging over-constrained designs. While the need for a system to automate experimental design is general to many types of science, we have built a prototype targeted for psychology and neuroscience, where issues of reproducibility and complexity of design have become a focus of attention.

CHAPTER 2

SWEETPEA OVERVIEW

2.1 A Language for Experimental Design

The independent and control variables in an experiment are called *factors*, and a trial is specified by a combination of levels of different factors. As a running example, consider an experiment where subjects are shown shapes of different colors; the factors are "color" and "shape", and each colored shape is a trial. Many experiments have additional constraints on the trials, such as "no more than 4 red shapes in a row".

To ensure statistically significant results, every possible trial sequence that satisfies the constraints must have an equal likelihood of being chosen for the experiment. This guarantees that the method for generating trial sequences is not introducing bias. In practice, however, researchers construct these trial sequences without statistical guarantees. The number of valid sequences is both intractably large and sparse in the space of all sequences, so it is not possible to find a valid sequence by randomly sampling all sequences or by enumerating all valid sequences.

SweetPea provides a high-level interface to declaratively describe an experimental design, and a low-level synthesizer to generate unbiased sequences of trials given satisfiable constraints. At the heart of the bias problem is the need to sample from constrained combinatorial spaces with statistical guarantees; SweetPea samples sequences of trials by compiling experimental designs into Boolean logic, which are then passed to a SAT-sampler. The SAT-sampler Unigen provides statistical guarantees that the solutions it finds are approximately uniformly probable in the space of all valid solutions. This means that while producing sequences of trials that are perfectly unbiased is intractable, we do the next best thing– produce sequences that are *approximately* unbiased.

2.1.1 Stroop Experiment

2.2 A Runtime for Uniform Sampling

SweetPea can be viewed as a domain-specific interface to SAT-sampling, and while there are other languages that rely on SAT-solvers , none that we know of leverage the guarantees provided by SAT-samplers. SweetPea provides a rich set of primitives that closely match the terms in which psychologists think about experimental design, allowing them to concisely describe what analysis they wish to perform, rather than how to construct the algorithm to do so.

CHAPTER 3

RELATED WORK

This is a chapter.

3.1 Psychology Toolboxes

- psyScope - psychoPy - OpenSesame

3.1.1 Reproducibility Crisis

Blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah. Blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah. Blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah.

3.2 Domain Specific Languages

- this is a huge area, not sure need to cite anything

3.2.1 Solver Aided Languages

- rosette - sketch: "Domain-Specific Symbolic Compilation" - dafny maybe

- hyperkernel: co-designing a language and the verification

3.3 Combinatorial Search Spaces

- finding solutions in a large search space– no really, very large - how large? - so large - what is the nature of our search constraints? things like 60 red words, 60 blue words (in Stroop, see chapter 2).

3.3.1 Sampling Methods

- sampling is the problem of finding solutions - could try solutions at random: turns out they are sparse (most examples don't have 60 red, 60 blue) - could try to generate all solutions: turns out there are too many (lots of possible arrangements)

- could use MCMC, but doesn't provide guarantees - this project is *really* about providing this guarantee that we're not introducing bias because this is a huge deal

3.3.2 Boolean Satisfiability

- SAT is a classic problem, NP-complete - SAT solvers happen to be really good - To specify something in SAT, you use variables and specify invariants - the SAT solver finds an assignment that satisfies the invariants - we use a SAT sampler which finds multiple assignments, and with guarnatees

- an alternative is using SMT constraints - we compile to SAT because unigen is available; to use a different tool we could pretty easily swap out the backend

3.3.3 Uniform Sampling

- re-read unigen paper - what problem is it solving? want uniform for coverage - who else cares about this problem - how is it solved: universal hash functions - what alternatives exist

CHAPTER 4

SWEETPEA LANGUAGE

The goal of the SweetPea language is to have semantics that match the terms researchers use to describe their experiments, while also being amenable to being translated efficiently to SAT.

TODO: everytime I saw "sometimes" come up with an example.

4.1 Components of an Experiment

To motivate our choice of primitives, we will first look at the components of an experiment.

4.1.1 Descriptions of Stimuli

- Usually factor with discrete levels - experiments typically have 2-7 factors with 2-4 levels each: important because it means a large search space - sometimes they might be nested (light color, dark color) - sometimes want to sample a continous distribution - sometimes don't fully cross all of them because can't keep the person there that long, but really want to try all combinations

4.1.2 Ordering Constraints

- need to specify the ordering to run Experiments, ie if you're testing the effect of the presence of A, you better be able to run it with and without A etc - really these are about relationships (presence or absense) of levels or factors, or arbitrary nestings. (1) congruence and (2) transitions as examples of derived levels - sometimes about relationships of other relationships! like, you can imagine balancing transitions - usually the complexity is limited by experimental limits, ie people actually have to complete these

4.1.3 Experimental Design and Balancing

- often want a full-crossing - sometimes experiment is too large for full crossing, so while experiment has over attributes (factors) they might not appear in all examples - sometimes its impossible to fully-counterweight and it'd be awesome if the tool could tell you if you were trying to do something impossible - sometimes can fix impossible by weighted crossing - sometimes can fix by "near balancing" or toss-away block, ie balancing transitions - ultimate declarative would be just say "these are what I want to analyze, balance for me"

4.1.4 Experimental Structure

- Experimental structure (ie multiple blocks) - sometimes want prologue / epilogue blocks - sometimes need these to balance transitions - sometimes have an experiment that consists of multiple experiments - sometimes want to reason between subjects because space is so large

4.2 Motivating Example (Congruence and/or Transitions)

- first describe the experiment in words - then code listing

4.3 SweetPea Primitives

for all: how does it map onto the pysch and why is it amenable to SAT

4.3.1 Factors and Levels

- possibly nested lists; that is to say trees - an experiment has one option for a level on at a time - we can represent whether a level is on or not as a boolean variable, then have a boolean constraint that says that only one can be on at a time - a nesting isn't represented in the boolean logic, it's just a reference to arbitraty groupings of levels that can be used in the relationships; can build a factor directly or from these groups. Q: what if overlapping level in multiple groupings? probably need to define it as a level then construct multiple references. - don't currently support sampling factors that are cont distributions because it's challenging to translate that to discrete boolean SAT

4.3.2 Derived Levels

- how do we represent these relationships? - because they all have to do with ordering, let's consider a "window". Windows have a width and a stride. ie, transition, ie, congruence. - windows implicitly "select" and group levels. - we can then define functions. they are allowed to depend on the state (on or off) of levels, and use this state to say whether or not *they* are "on". Example, congruence. In this way, they're defining new levels (since a level is just a thing that knows whether it's on or off) – which is why they're derived factors.

- these should allow us to define things that skip elements, like the "bait" example

- why is this amenable to SAT? Internally, we pick out the levels, create new levels whose value depends on a boolean relationship of those levels. Q: wait why does this work? Does this work? Make sure this works.

TODO ASAP

here's an idea: I can do it if I have a literal truth table. I can generate a truth table for the defined function by assume each input is a boolean level (this should be enforced) then take all boolean combos and literally run it that generates a truth table I can encode that truth table in the logic by translating to CNF

4.3.3 Experimental Design, Balacing and Experimental Structure

- really straightforward. An experimental sequence is a list of experimental blocks. An experimental block is at the high level these derivations and at the low level linearized into a list of SAT that the runtime can execute. A design is literally which factors are visible and for balancing we currently only support full-crossings.

CHAPTER 5

SWEETPEA RUNTIME

- Runtime works by translating to SAT - One hot encoding - Efficient encoding w/ Tsietin transform

Blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah. Blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah. Blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah.

5.1 Representing SweetPea Primitives in Boolean Formulas

5.1.1 Representing Levels and Factors

5.1.2 Representing Derived Levels and Derivation Functions

Blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah. Blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah. Blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah.

5.2 Efficient Encoding with the Tseitin Transform

Blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah. Blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah. Blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah.

5.3 Communicating with the SAT-Sampler

Blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah. Blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah. Blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah.

5.4 Correctness Guarantees

Blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah. Blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah. Blah

blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah.

CHAPTER 6

IMPLEMENTING SWEETPEA

This is a chapter. yes it is

6.1 Language Implementation: Decisions and Alternatives

6.1.1 Internal Representations

6.1.2 Embedding Language

6.1.3 Level Encodings

- trade-off between more variables and more clauses

6.2 Tsietin Transform Implementation

6.2.1 Binding Variables: Iff

6.2.2 Adders

6.2.3 Ripple Carry Adders

6.2.4 Pop Count Circuit

6.2.5 Exhaustive Testing

6.3 Runtime Implementation: Interfacing with Unigen

6.3.1 CNF

6.3.1.1 Representing Derived Levels in CNF

6.3.2 modified CNF: xor Constraints

6.4 Output Formats

6.4.1 Human Readable

6.4.2 Interfacing with PsyNeuLink

CHAPTER 7

EXAMPLE PROGRAMS AND DISCUSSION

7.1 small Stroop + look at sampled distributions

7.2 transitions and congruence

7.3 more complicated derivation functions

CHAPTER 8

FUTURE WORK

8.1 Beyond Psychology

8.2 Future Language

8.2.1 Weighted Crossings

8.2.2 Sampling Continuous Factors

8.2.3 Automated Experimental Design

8.2.4 Syntactic Sugar

8.3 Future Runtime

8.3.1 Verified Core

8.3.2 Debugging unSAT experiments

8.3.3 Iterative experimental design and Partial Satisfiability

CHAPTER 9

CONCLUSION

- Problem we tried to solve - Why it's important - What we did - What are the consequences + forward looking

REFERENCES

[1] J. Cohen, B. MacWhinney, M. Flatt, and J. Provost, *Psyscope: An interactive graphic system for designing and controlling experiments in the psychology laboratory using macintosh computers*, Behavior Research Methods, Instruments, & Computers, 25 (1993), pp. 257–271.

[2] S. Mathôt, D. Schreij, and J. Theeuwes, *Opensesame: An open-source, graphical experiment builder for the social sciences*, Behavior research methods, 44 (2012), pp. 314–324.

[3] K. S. Meel, M. Y. Vardi, S. Chakraborty, D. J. Fremont, S. A. Seshia, D. Fried, A. Ivrii, and S. Malik, *Constrained sampling and counting: Universal hashing meets sat solving.*, 2016.

[4] J. W. Peirce, *Generating stimuli for neuroscience using psychopy*, Frontiers in neuroinformatics, 2 (2009), p. 10.

[5] E. Torlak and R. Bodik, *A lightweight symbolic virtual machine for solver-aided host languages*, in ACM SIGPLAN Notices, vol. 49, ACM, 2014, pp. 530–541.