Finally Fifth?
Searching for answers in an uncertain world

Anthony Di Franco
San Francisco Types, Theorems, and Programming Languages
26 October 2015

Rough Outline

Emphasis on rough

Work in progress, seeking early feedback

Still figuring out how to present these ideas best

- Motivation
  - History of software in broader historical / cultural contexts
- Design
  - Compress joint relationships recursively, creating enlarged alphabet of joint relationships at higher scales
  - Track sequences of information propagation steps through joint relationship space that most inform query and plan evaluation around them
- Implementation strategy
- Notable possible applications

“Still Waiting for the Revolution”

(Alan Kay, 2002)

(Me, 2015)

“While considered progress in the eyes of the consumer, what has happened in the computing industry has been more like a slow evolution than the revolution that Alan Kay dreamed of.”

Meanwhile, Carl Hewitt did planner, then actor model, now looking at “inconsistency robustness” and logics capable of expressing uncertainty, what is going on here?

Algorithms ¿=? Software

Dr. Wayne Brown, with didactic intentions:

> "An algorithm is a precise, step-by-step set of instructions for solving a task... We live in the "information age," where many of the tasks people need (or want) to do can be accomplished by computers. However, computers currently have no real understanding or cognition – they can only perform tasks that people have developed algorithms to solve. Therefore, to create algorithms that computers can execute, you must understand what a computer is capable of executing and be able to write a series of unambiguous instructions that a computer can execute successfully. For the remainder of this document we will restrict our discussion to the development of algorithms that can be executed on computers."

The Revolution we deserved

Algorithm = Logic + Control (Kowalski, 1979)

Logic = Algorithm – Control (Me, 2015)

- Logic → specification / relationship structure
- Let's to the whiteboard.

The Strangest Thing About Software: Paths through relationship graphs

(Menzies, 2006)

Slightly reinterpreted, a large number of paths go through a few nodes (“back doors,” “collar variables”)

- Useful for optimization / software testing
- Assume mature software approximates all paths through entire desired relationship graph modulo diminishing returns on effort:
  - Executing specifications should not be too much more complex to run than mature software we already have, despite being much less complex to write.

The Revolutions we got instead

Object-orientation: tools for restricting the topology of the relationship graph with hopes of a priori limiting algorithmic complexity.

Objects, modules, APIs: restricting communication topologies

Originally intended for agent-based simulation, where there is reason to believe this kind of structure is there.

"For an analogy, take a big heavy beef, chop it into small morsels, wrap those morsels within hygienic plastic bags, and link those bags with strings; whereas each morsel is much smaller than the original beef, the end-result will be heavier than the beef by the weight of the plastic and string, in a ratio inversely proportional to the small size of chops (i.e. the more someone boasts about the local simplicity achieved by his µK, the more global complexity he has actually added w.r.t. similar design w/o µK)." - "Microkernel debate", Faré, tunes.org

Better: query as universal API, request to extract information from another context.

Functional programming with types: tools for making sure you're on the path in the relationship graph you should be on. (cf. Constructionism and stratification)

Let's to the whiteboard.

Certain kinds of paths through a relationship graph, without the graph itself

"All programmers are forcing their brains to do things brains were never meant to do in a situation they can never make better, ten to fifteen hours a day, five to seven days a week, and every one of them is slowly going mad… the opportunity to trim Satan's pubic hair while he dines out of my open skull so a few bits of the internet will continue to work for a few more days." - "Programming Sucks," stilldrinking.org

Small digression:
The historical perspective

Philosophical frame of early computing, cybernetics vs. mechanisms

- Different emphasis on the usual frame: Nondeterminism vs. determinism
- Information feedback & Adaptive behavior vs. a priorism and determinism and the bow-tie and requisite variety problem
  - ca. 250 BCE: Legalism vs. Hundred Schools of Thought
  - ca. 1900 CE: substantive vs. procedural justice, and artisans vs. Taylorism
    - James C. Scott: Legibility; Episteme vs. Phronesis, Techne vs. Metis,
    - IBM was making census tabulating machines
  - 1930s – present: auftragstaktik vs. command & control hierarchy
- Rise of the logicians and decline of complexity
  - Type theory vs. the feedback loop
- Fifth generation roads not taken, early neural networks, and the PC meteor wiping out the Cambrian explosion of language / hardware research
  - Planner, schemer, screamer, scheme
  - Prolog, Icon, Python
- Deep learning?

Before Fifth: Forth

Forth provides one tool: dictionary

Permits programmer to arrange for dictionary-based (universal) compression of sequences of executed instructions.

(paths through relationship graph)

cf. Lempel-Ziv(-Welch)

Chuck Moore's personal philosophy: limit scope

(Keep paths few, relationships manageable)

What if we can't or don't want to limit scope?

Automate finding / compressing common paths

The unreasonable ineffectiveness of mathematics

Rigid a priori structure is brittle in the face of a posteriori information:

Physics is the art of recognizing and deploying mutual relationships / constraints to answer questions.

Physics made great early strides understanding simple situations by investigating consequences of guesses of determinism / indistinguishability / uniformity / linearity / convexity / equilibrium / perfect isolation.

Physics matured and birthed engineering by developing the sophistication to relax such assumptions.

Economics failed by applying the same assumptions behind early progress in physics to situations involving much more complex relationships.

Feynman, “cargo cult science,” “physics envy”

Yanis Varoufakis, ***Economic Indeterminism;*** Steve Keen @ Kingston

Software is failing by trying to restrict the relationships it can model instead of trying to deal effectively with the broadest possible range of relationships.

Uncertain information and how to deal with it

	Statistics	Deep Learning	Recursion
Stereotype of pragmatics within paradigm	Count ***events***, interpret	Learn and recognize ***features*** *(patterns of joint variation)*	Define, learn, and generate ***grammars*** *(composition rules)*
Theory of Information	Probability	Ad-hoc, fuzzy-like	Boolean Logic
Applicability	Well-understood mildly uncertain situations of low-dimensional mostly single-scale variation	Multi-scale structure across space (so far)	Process (non-trivial structure in time and across space)
Structure of variation	Discrete / continuous, low dimensional	Continuous, ad-hoc combinatorial	Discrete, combinatorial

Deep Learning

What is it?

No one knows? New word for loose group of old techniques?

Neural networks with more than one hidden layer?

What is special about that?

Nothing: Been around for maybe 40 years?

Everything: Starts to look like it could express general recursion in a framework capable of learning from uncertain / incomplete information?

Deep Learning

Autoassociators

- Nonparametric models of spaces of joint variation
- Approximate, regularized bijection into and from hidden, reduced-dimension-reduced representation space
- Can be recursively composed
  - (see vision experiments informing theoretical neuroscience)
- More than heuristic / special-case regularization rare but possible
  - Information-theoretic priors
  - Search rather than gradient descent possible but not yet attempted to best of my knowledge

Recurrent nets

- LSTM: state maintained with a feedback loop, error gradients can feed "back in time"
  - But seems like only useful if monotonic, i.e. convexish problem?
- Reinforcement Learning Neural Turing Machine?
  - Figuring out how to apply gradient descent to about the least convex thing in existence?

Fifth Generation, Prolog and Lisp

Fifth Generation Computing

was going to conquer the world with Prolog: automated exhaustive enumeration of combinatorially exploding Boolean joint spaces

Worst thing about Prolog is logic (ironically)

Purely discrete combinatorially exploding spaces are hard to search

No one seems to have thought too hard about feeding partial information from a search back in to bias it better because partial information was not part of the system

Lisp

Can think of it as pulling the nondeterminism out of the foundations of Prolog and reintroducing it locally ad-hoc

(opposite of what actually happened historically, (screamer excepted,) but useful)

Fifth Generation, Prolog and Lisp

Nowadays: Probabilistic Programming

- Return of some ambitions to deal with nondeterminism but of a strictly probabilistic kind and in a mostly-deterministic, usually functional context (cf. Kiselyov)

Tomorroways: Monte Carlo Tree Search and efficient POMDP search / global search

- Partial information from a search can feed back to bias it better
- Approximate Planning in Large POMDPs via Reusable Trajectories, Kearns, Mansour, Ng
- Approximate global optimization in high/unbounded dimension NP hard problems is easy enough!
  - Cross entropy method, exhaustive interval search: unums, COSY Infinity, C-XSC
- Only exact is hard in practice, when do we actually need exact?
- This reality has still had zero impact on programming languages so far as I can tell.

Constraint solving and so on

Gecode

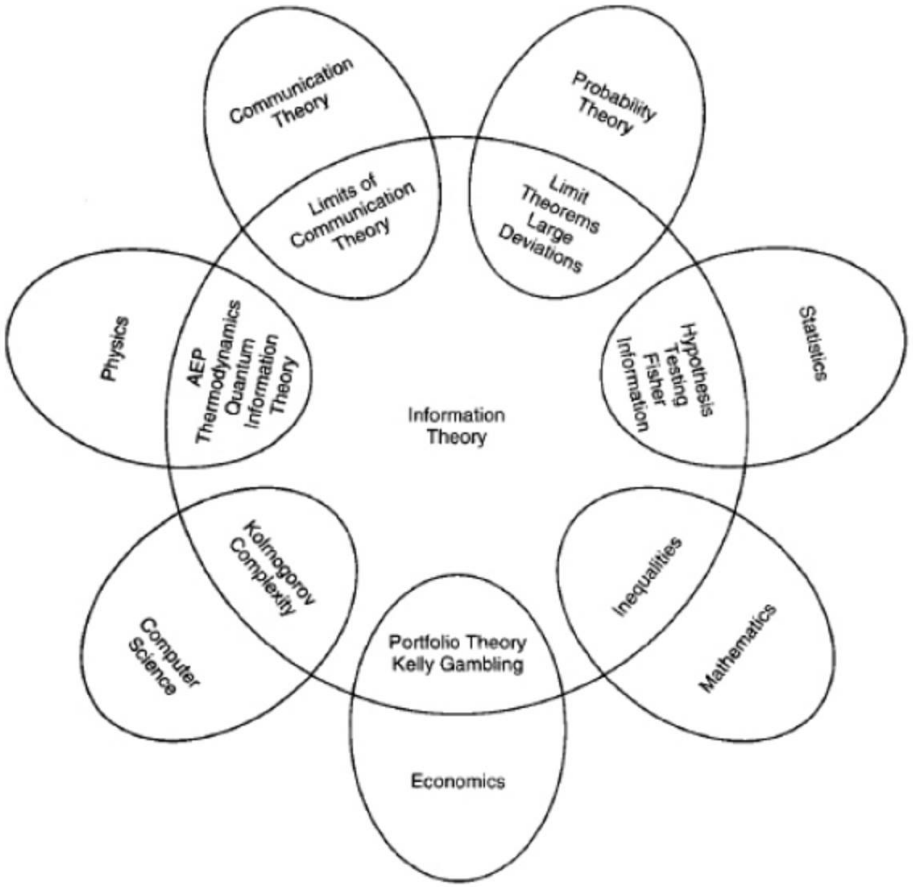
- State of the art performance on benchmarks
- No recursion within the framework
- Custom constraint solvers must be made for composite constraint subproblems ad-hoc outside the language

Sussman & Radul's information propagation

- Moving towards in-system recursion via 'switch' for instantiation deferral of unbounded graph
  - I propose in addition mechanisms similar to function calling / unification instead, generalizations of delimited control operators.
- Represents information explicitly and ties evaluation to passing significant information
  - I propose to add to this an explicit set of query nodes that indicate what information is desired, rather than accumulating information everywhere in the same way.
- See also Zdravko Markov's Net Clause Language

Information theory

Relevant to everything because drawing the best distinctions one can under whatever information is available is relevant to everything!



Cover and Thomas 1991 via N. Taleb (all highly recommended authors)

Generalized information theory (Joslyn)

Nondeterminism is central, how do we faithfully represent it in a context of partial information?

- Probability theory assumes events are a partition of universe under consideration, leading to normalization of distributions over events, but
- Nondeterministic search evaluates options that don't necessarily imply anything about each others' viability. (Kosko)
- Fuzzy logic generalizes the boolean logic version of option validity to partial validity faithfully:
  - Truth values aren't assumed to constrain one another, e.g.
  - More than one option can be completely valid at once, or no option can be at all valid, whereas some event must occur and if one is more likely to occur the rest are less likely.

Generalized Information Theory
Digression: quantum connections?

Constraints as negative truth values

We can follow James & Sabry, "The Two Dualities of Computation: Negative and Fractional Types" and let negative truth values signify constraints in fuzzy logic and probability theory as well.

Can we usefully apply the mechanism of propagating negative as well as positive information?

Wigner quasiprobabilities, which permit negative probabilities for events, provide an alternative formulation of quantum mechanics. Under the constraint interpretation, is this related to pilot wave work? Nondeterminism inherent in fluid dynamics / Navier-Stokes? Does it give better intuition for quantum mechanics? Relate to quantum computing?

Search in terms of partial information

Monte Carlo tree search

- Revolutionizing game AI, especially Go, and robotics
- Gathers approximate information about subtrees via sampling

Heuristic → Metaheuristic → Probabilistically complete search

"A particle swarm algorithm for high dimensional, multi-optima problem spaces"

- Tim Hendtlass, 2005 IEEE Swarm Intelligence Symposium
- Originally due to numerical underflow error in implementation of particle swarm optimization yields fast global optimum finding.
- Effect of error is to add (very) heavy tail to particle jump distributions when many particles converge in same basin. Later implemented using a deliberate mechanism. Without this effect they are Gaussian distributed.
- In 30 dimensional space with 8 local maxima per dimension, i.e. 1.2e+27 maxima (in fact in a nearly waffle-shaped configuration!). Also works in 100-D, 2.0e+90 maxima.
- Similar to Lévy flights, another search emulating animal behavior.

Upshot: Importance sampling + heavy tailed proposal is non-heuristic version?

- cf. cross entropy method, Bickson's stable approximators (NIPS 2010)

Putting the pieces back together

What if we reintroduced the explicit query of Prolog / SQL

(cf. Berkeley BlinkDB)

In the information propagation framework,

(tastefully modified)

With compression of joint relationship spaces,

(cf. Jocelyn Ireson-Paine, Binary holographic reduced representations for SWI-Prolog)

And search-driven evaluation.

(with optimization of most common evaluation patterns *a posteriori*)

Then…

Evaluation could be driven by a query planner…

Rewarded by information accumulation…

Searching a state space of size roughly equal to the information content of the problem instead of a naive enumeration of the state space.

- (facts and relationships viewed through bottleneck of queries)

Evaluation is inherently fine-grained parallel

- adaptively optimizing it and mapping it onto parallel hardware conceptually straightforward

Last idea I'll ever need to have? I hope so!

Implementation Strategy

Symbolic interpreter at core

Generalized Information Theory model of information about every variable

Explicit query

Planner tries to generate recursive relationship expansions that lead to least costly / most informative information propagation paths across facts they begin from

Joint relationship spaces are recursively compressed

Put information propagation messages on work queues

Sample work queues as a JIT samples stacks

Cache and optimize most common propagation sequences as e.g. OpenCL kernels

- Hardware to accelerate this is very close to pure actor model

Keep sequences in compressed / succinct sequence data structure

- FM-index, Edward Kmett's succinct

Keep facts / compressed representations of joint relationships in something like cache-oblivious B-tree with high-dimension-appropriate indexing of joint spaces to identify those close to given facts

Refactoring mental categories

Software's defining trait is mutability

To the extent software isn't adapting to the environment it runs in, it isn't really software.

What currently passes for software (mostly fixed, mostly deterministic algorithms) is a large amount of virtual hardware running multiplexed on a smaller amount of physical hardware.

Systems programming vs. Application programming

Currently distinction is fairly arbitrary, no qualitative difference in the two areas, rough sense of high algorithmic complexity in systems and lower algorithmic complexity in applications. Not often borne out by counting sizes of teams / lines of code / total economic activity in practice. Perhaps density of code in some sense? Number of equivalent ways to get any given thing right enough?

Another possibility: systems have more stringent performance / correctness requirements, provide more broadly applicable tools. Closer.

Clear distinction should be: systems programming implements algorithms to support general declarative problem solving, application programming applies them.

Systems programming is where you want strong a priori guarantees and the best of current tools (e.g. Rust) will do. Application programming is where you want maximum ability to adapt to carry out the task and we've got a lot of work to do to build the systems for that.

Potential Benefits

Compilation → In-image synthesis, partial evaluation

Everything adapts to the input data

Garbage collection / memory management possibly simplified by guarantees of recomputability, fact database data structure, …

Run legacy program on parallel hardware by copying logic, ignoring control, querying for observable behavior

With some sort of tabling / caching, everything incrementally re-computes in response to changes

- Subsuming (Functional) Reactive Programming given a temporal logic

Get bounds / uncertainty estimates on everything

Identify, rigorously characterize, generate, search abstractly very large spaces in terms of specific data / queries

Combinatorial reduction in *a priori* system complexity

- Kowalski: Software = logic + control, control is combinatorial elaboration of logic, only write the logic and discover the control adaptively
- A lot more leverage in the bottleneck in the center of the bow-tie

Application areas that could benefit most

More rigorous foundation for tools for coming to conclusions based on data

As developed throughout in contrast to limitations of statistical approaches

Economics / Econometrics

Statistical physics enjoyed great success by making physically plausible assumptions of uniformity / equilibrium to get to a useful probabilistic framing. Economics uncritically copied many of these assumptions and the resulting techniques, leading to disaster. Even physics has moved beyond these assumptions where they don't apply. Better tools could make modeling without unrealistic assumptions more practical. Cf. Yanis Varoufakis' Economic Indeterminacy, Steve Keen's Minsky, Nassim Taleb.

Computer security doesn't have to be an oxymoron

Instead of writing an algorithm and hoping it implicitly enforces an invariant necessary to security, write the invariant. Only sane approach in an adversarial environment where you can be pretty sure your opponents are using nondeterministic search

Meredith Patterson, Hammer: use an actual parser to parse your inputs.

Computer Graphics

Camera is query, lights and scene geometry are facts, light transport dynamics are relationships;

Importance sampling comes for free as part of how evaluation works by default.

User interfaces

Constraint-based layouts without having to worry about linearity / convexity

Relationship-based connections between information and representation in diverse media

Games

Declarative AI: search technique sufficient for general game playing is already how evaluation works.

Robotics / control

Recursive relationships can be learned empirically, then used for control / planning.

Thanks!

Thanks to Adrian and Mixrank for organizing / hosting.

Thanks to all the researchers mentioned or otherwise involved in developing these ideas.

If this sounds like something you could use, let's talk!