Linear models of computation and program learning

Michael Bukatin

Nokia Corporation, Burlington, MA

Joint work with Steve Matthews (University of Warwick)

- - -

Global Conference on Artificial Intelligence (GCAI 2015),
Tbilisi State University, October 18, 2015

Electronic coordinates

These slides are linked from my page on partial inconsistency and vector semantics of programming languages:

http://www.cs.brandeis.edu/~bukatin/partial_inconsistency.html

E-mail:

bukatin@cs.brandeis.edu

Outline

Regulation of gene expression

Digital software: **too brittle, too sensitive to minor variation**.

Biological systems: flexible and adaptive with respect to variation.

Biological cells function at wide ranges of the level of expression of various proteins, which are machines working in parallel.

Regulation of the level of expression of specific proteins is a key element of flexibility of biological systems.

Regulation of gene expression

Evolutionary developmental biology: the flexible architecture together with conservation of core mechanisms is crucial for the **observed rate of biological evolution** [Gerhart, Kirschner].

High robustness with respect to varying levels of expression $\Rightarrow$ higher robustness with respect to mutations.

Morphology evolves largely by altering the expression of functionally conserved proteins [Sean Carrol].

Regulation of gene expression

Evolutionary developmental biology: the flexible architecture together with conservation of core mechanisms is crucial for the **observed rate of biological evolution** [Gerhart, Kirschner].

High robustness with respect to varying levels of expression $\Rightarrow$ higher robustness with respect to mutations.

Morphology evolves largely by altering the expression of functionally conserved proteins [Sean Carrol].

Regulation of gene expression and evolutionary programming

To incorporate regulation of expression into a system of genetic programming one might evolve programs describing systems of **parallel computational processes**.

Then one might take the CPU allocation and other computational resources given to a particular computational process as computational equivalent of the level of expression of a particular protein.

Of course, many of the architectures for parallel computations are brittle as well, with delicate mechanisms of writing to shared memory and locks. To achieve flexibility one should use parallel architectures which minimize those delicate interdependencies.

Linear combination of execution runs

Computational architectures which admit the notion of linear combination of execution runs are particularly attractive.

Then one can regulate the system simply by controlling coefficients in a linear combination of its components.

Today's talk: two computational architectures which admit linear combinations of execution runs.

Linear combination of execution runs

Computational architectures which admit the notion of linear combination of execution runs are particularly attractive.

Then one can regulate the system simply by controlling coefficients in a linear combination of its components.

Today's talk: two computational architectures which admit linear combinations of execution runs.

Linear combination of execution runs

Computational architectures which admit the notion of linear combination of execution runs are particularly attractive.

Then one can regulate the system simply by controlling coefficients in a linear combination of its components.

Today's talk: two computational architectures which admit linear combinations of execution runs.

Linear combination of execution runs

Today's talk: two computational architectures which admit linear combinations of execution runs.

- Probabilistic sampling
- Generalized animation

Sampling semantics

The inputs, outputs, and intermediate variables are **streams of elements sampled from appropriate probability distributions**.

Assume samplers generating points of their distributions with a uniform speed, so that the notion of "a number of points generated per unit of time" is well defined.

To implement **linear combinations of probabilistic programs with positive coefficients** run those programs in parallel, merging appropriate output streams. Control the values of coefficients by changing the relative execution speed of those programs.

(Negative coefficients: the last part of the talk.)

Probabilistic sampling and evolutionary programming

The connections between probabilistic programming and evolutionary/genetic programming are much tighter than it is usually acknowledged.

MCMC is essentially an evolutionary method:

- acceptance/rejection of the samples corresponds to selection
- production of new samples via modifications of the accepted ones corresponds to mutations to produce offspring from the survivors.

Probabilistic sampling and evolutionary programming

Bayesian Optimization Algorithm changes the procedure of producing the next generation in genetic algorithms from pairwise crossover to the resampling from the estimated distribution of the individuums selected for fitness.

Martin Pelikan. Bayesian Optimization Algorithm: from Single Level to Hierarchy, PhD Thesis 2002.
`http://www.medal-lab.org/files/2002023.pdf`

Used by the seminal
Moshe Looks. Competent Program Evolution, PhD Thesis 2006.
`http://metacog.org/doc.html`

Probabilistic sampling and evolutionary programming

These papers seem to indicate that both fields tend to disregard this connection:

Zoubin Ghahramani, Probabilistic machine learning and artificial intelligence, *Nature* **521** (28 May 2015) 452–459.

Agoston Eiben, Jim Smith, From evolutionary computation to the evolution of things, *Nature* **521** (28 May 2015) 476–482.

Probabilistic sampling and evolutionary programming

A weakness of the known genetic programming schemes seems to be that none of them seems to implement the regulation of gene expression.

Variability in the regulation of gene expression, rather than in genes themselves, seems to be an important factor making fast biological evolution feasible.

If we associate a protein with a computational process, then one might want an architecture where proteins correspond to parallel computational processes, and the degree of expression of a given protein corresponds to the share of computational resources the computational process in question gets.

Linear models are especially nice in this sense: the degree of expression of a gene can be simply modelled via the corresponding coefficient in the linear combination of computational processes corresponding to a parallel program (set of genes).

Higher-order evolution/higher-order probabilistic programming

Higher-order evolution, adaptive evolution, evolving the evolutionary mechanisms, acceleration of evolution.

What is the right way to talk about higher-order probabilistic programming?

The tradition is to talk about probabilistic lambda-calculus, or to empasize implementing probabilistic programming within a higher-order functional programming language, but I am not sure it is the angle of view we need here.

Higher-order evolution/higher-order probabilistic programming/**sampling the samplers**

What is "higher-order probabilistic programming"?

Recently we are seeing examples of research implementing higher-order sampling schemas in a more narrow and focused sense of the word: samplers which generate other samplers, probabilistic programs sampling the space of probabilistic programs, generative models which emit other generative models as points.

In this narrow sense, higher-order probabilistic programming is the ability to take **streams of probabilistic programs** as inputs and to produce streams of probabilistic programs as outputs.

This is a particularly important development for program learning.

Sampling the samplers

`http://arxiv.org/abs/1407.2646`

Yura N. Perov, Frank D. Wood.
Learning Probabilistic Programs. July 9, 2014.

- A notion of compilation for probabilistic program (more similar to partial evaluation).
- **Anglican** engine (**PMCMC**, Clojure)
- Maddison-Tarlow paper

Sampling the samplers

http://cims.nyu.edu/∼brenden/LakePhDThesis.pdf

Brenden M. Lake.
Towards more human-like concept learning in machines: Compositionality, causality, and learning-to-learn.
MIT PhD Thesis, September 2014.

- Learning from one or a few examples
- Learning rich conceptual representations

Fuzzy sampling and animations

Fuzzy samplings where points are taken with real coefficients might be even more attractive.

One can think about them as generalized animations, where points might be indexed by a more sophisticated index set than a discretized rectangle.

Here it is easy to allow negative coefficients in linear combinations (speaking in terms of conventional animation this means that 0 is at some grey level, between black and white).

One can leverage existing animations, digital and physical (such as light reflections and refractions in water), as computational oracles.

Expressive power

Music is a fast animation (typically on the index set of 2 points for usual stereo).

Very short programs can express complex dynamics.

A way to incorporate aesthetic criteria into software systems.

Linear combinations of animations

Conventional color images and music are examples of generalized animations.

The use of linear combinations of those is standard in video and audio mixing software.

Evolving animations

Good track record of evolving animations.

Another feature animations seem to share with sampling architecture is that they tend to be non-brittle, and that their mutations and crossover tend to produce meaningful results in the evolutionary setting.

Non-standard secondary structures on the set of points

A lot of expressive power of this architecture comes from the ability to have non-standard secondary structures on the set of points.

Points can be associated with vertices or edges of a graph, grammar rules, positions in a matrix, etc.

We are beginning to formulate mechanisms of higher-order animation programming via variable illumination of elements of such structures.

A bit more about animations

Probabilistic programming is better if the goal is well-defined, animations are better if one wants to explore emergent behavior.

A large and very active **creative coding community**.
Students love this subject.

Hybrid systems

Instead of implementing everything in terms of linear systems one can use a hybrid approach, mixng linear sytems and traditional software.

Inspiration: hybrid hardware connecting live neural tissue and electronic circuits.

One can decide to use large existing software components and try to automate the process of connecting them together using flexible probabilistic pieces. This is potentially very important.

One can try to use small inflexible components inside the flexible "tissue" of linear models.

Hybrid systems

Instead of implementing everything in terms of linear systems one can use a hybrid approach, mixng linear sytems and traditional software.

Inspiration: hybrid hardware connecting live neural tissue and electronic circuits.

One can decide to use large existing software components and try to automate the process of connecting them together using flexible probabilistic pieces. This is potentially very important.

One can try to use small inflexible components inside the flexible "tissue" of linear models.

Hybrid systems

Instead of implementing everything in terms of linear systems one can use a hybrid approach, mixng linear sytems and traditional software.

Inspiration: hybrid hardware connecting live neural tissue and electronic circuits.

One can decide to use large existing software components and try to automate the process of connecting them together using flexible probabilistic pieces. This is potentially very important.

One can try to use small inflexible components inside the flexible "tissue" of linear models.

Hybrid systems

One can try to use small inflexible components inside the flexible "tissue" of linear models.

This is what we do in Project Fluid.

`https://github.com/anhinga/fluid`

Dataflow programming for linear models of computations

Because probabilistic sampling and generalized animation are both stream-based, dataflow programming is a natural framework for this situation.

Dataflow architecture is convenient for program learning, because syntax of dataflow programs would typically be more closely related to their semantics than the syntax of more conventional programs.

Project Fluid

https://github.com/anhinga/fluid

MIT License

Prototypes for three dataflow architectures.

Implemented in Processing 2.2.1 (https://processing.org/).

Project Fluid

The architectures are good for generalized animations and probabilistic sampling.

Prototypes implemented for conventional animations.

Project Fluid: May 2015 architecture

Bipartite graphs: data nodes and transform nodes.

Now I'd like to show a short demo.

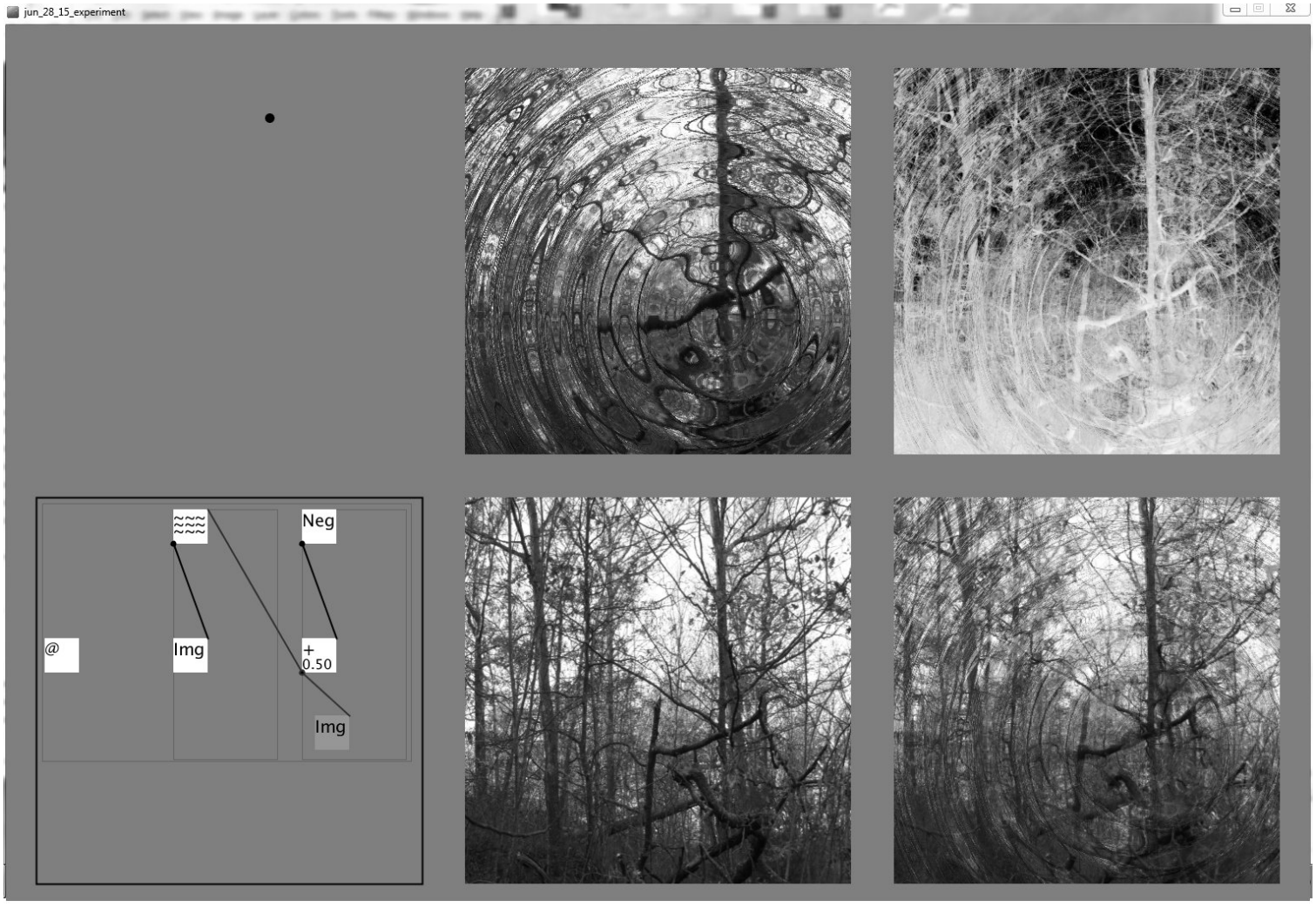
Project Fluid: June 2015 architecture

Almost continuous program transformations.

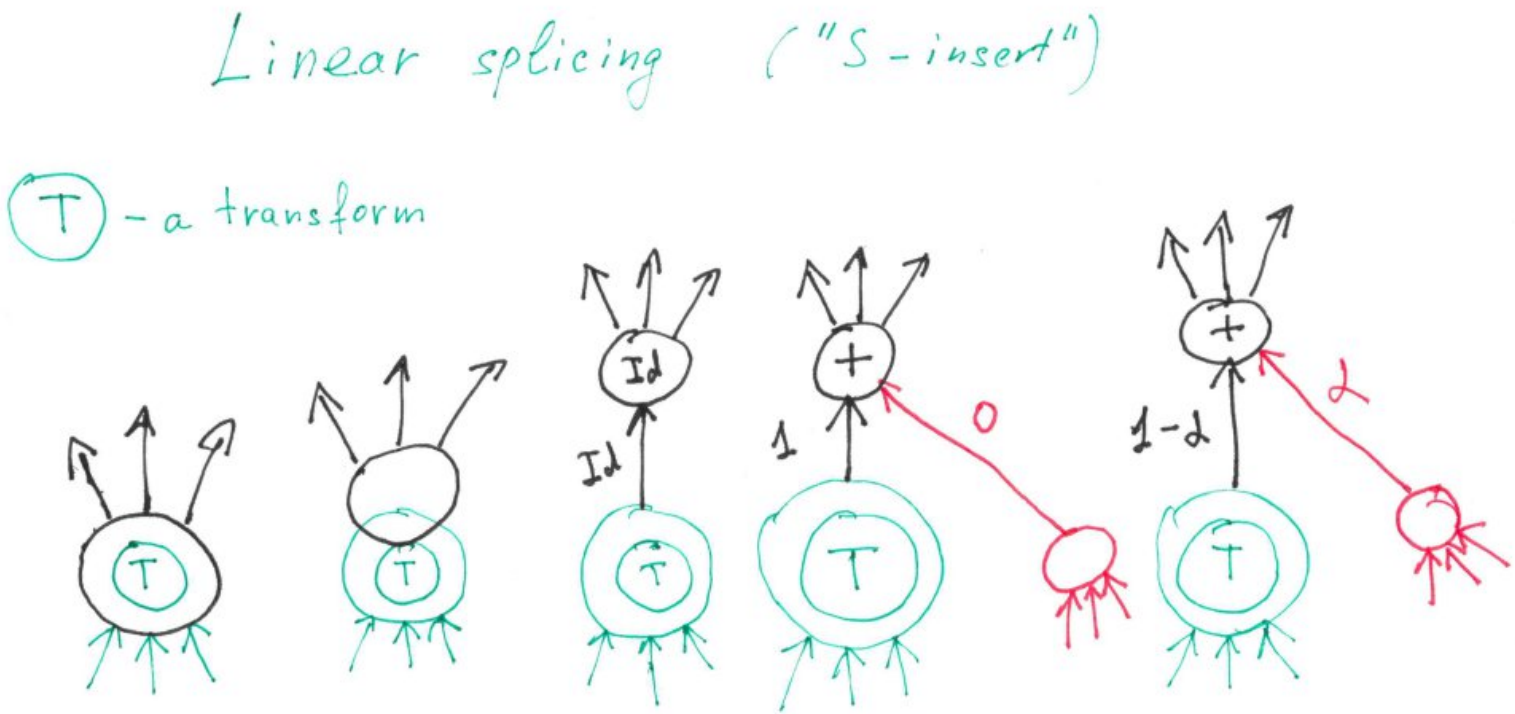
Continuous program transformations punctuated by isolated "benign discontinuities" such as **linear splicing** ("S-insert").

Dataflow graph inside its own node

Dataflow graph inside its own node as it changes during the execution of the dataflow program.



Linear splicing

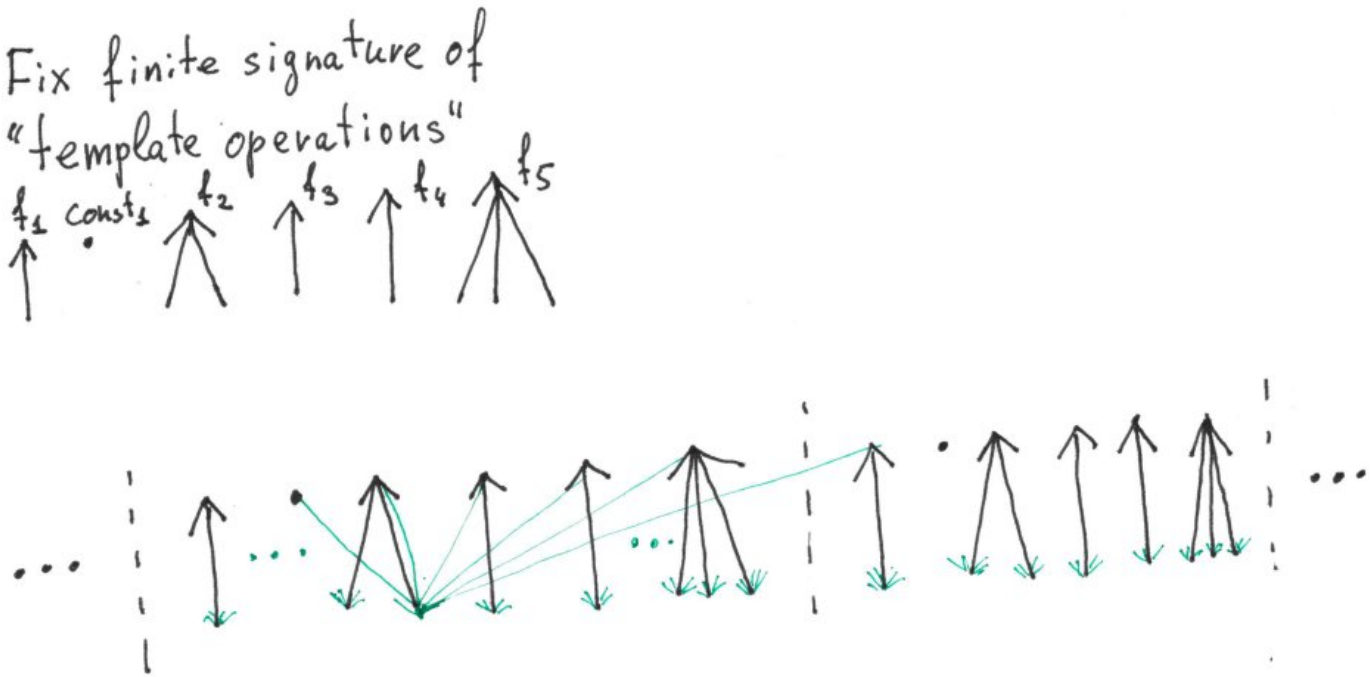


Project Fluid: August 2015 architecture

Dataflow graphs as matrices.

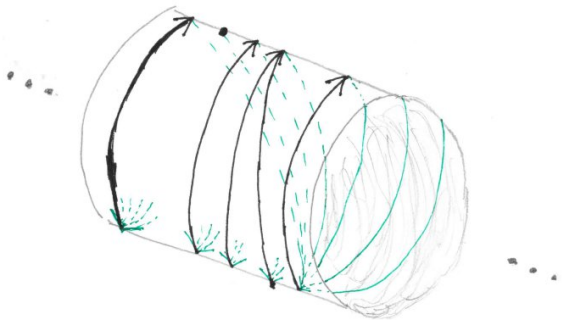
From the discipline of bipartite graphs connecting target nodes of general transformations and target nodes of linear transformations to the **matrix representation of dataflow graphs**.

Matrix machine



Green arrows form countable linear combinations, but only a finite number of green arrows in the graph carry non-zero coefficients a_{ij} at any given moment of time.

Matrix machine as an infinite cylinder



Every input of a template operation is a countable linear combination of all outputs of template operations, but only a finite number of green arrows in the graph carry non-zero coefficients a_{ij} at any given moment of time, all other coefficients are zero.

Preprint: "Dataflow graphs as matrices"

http://www.cs.brandeis.edu/~bukatin/DataFlowGraphsAsMatrices.pdf

Meaningful row and column names (strings instead of numbers).

Programming with higher-order matrix elements: mechanisms for associating a_{ij} with the target nodes of the graphs.

Beyond that preprint

We decided to transpose all matrices going forward (to swap rows and columns).

Higher-order mechanisms in the preprint are not enough. One should add more powerful mechanisms to generate new non-zero elements a_{ij} (e.g. stochastic generation of new non-zero links).

Promising platform

Looking for new methods to synthesize programs as matrices.

Looking for new idioms of higher-order programming expressed in terms of changing the matrix elements.

It is not difficult to have rich sets of template operations.

Continuous program transformations and continuous trajectories in large spaces of programs are therefore available .

One can evolve programs in continuous fashion.

One can sample continuous trajectories in a space of programs.

Partial inconsistency landscape

- **Negative distance/probability/degree of set membership**
- Bilattices
- Partial inconsistency
- **Non-monotonic inference**
- Bitopology
- $x = (x \wedge 0) + (x \vee 0)$ or $x = (x \wedge \bot) \sqcup (x \vee \bot)$
- Scott domains tend to become embedded into vector spaces
- Modal and paraconsistent logic and possible world models
- Bicontinuous domains
- The domain of arrows, $D^{Op} \times D$ or $C^{Op} \times D$

Partially inconsistent interval numbers

Interval numbers don't form a group with respect to $+$.

Add pseudosegments $[a, b]$ with contradictory property that $b < a$ (the length $b - a$ is negative).

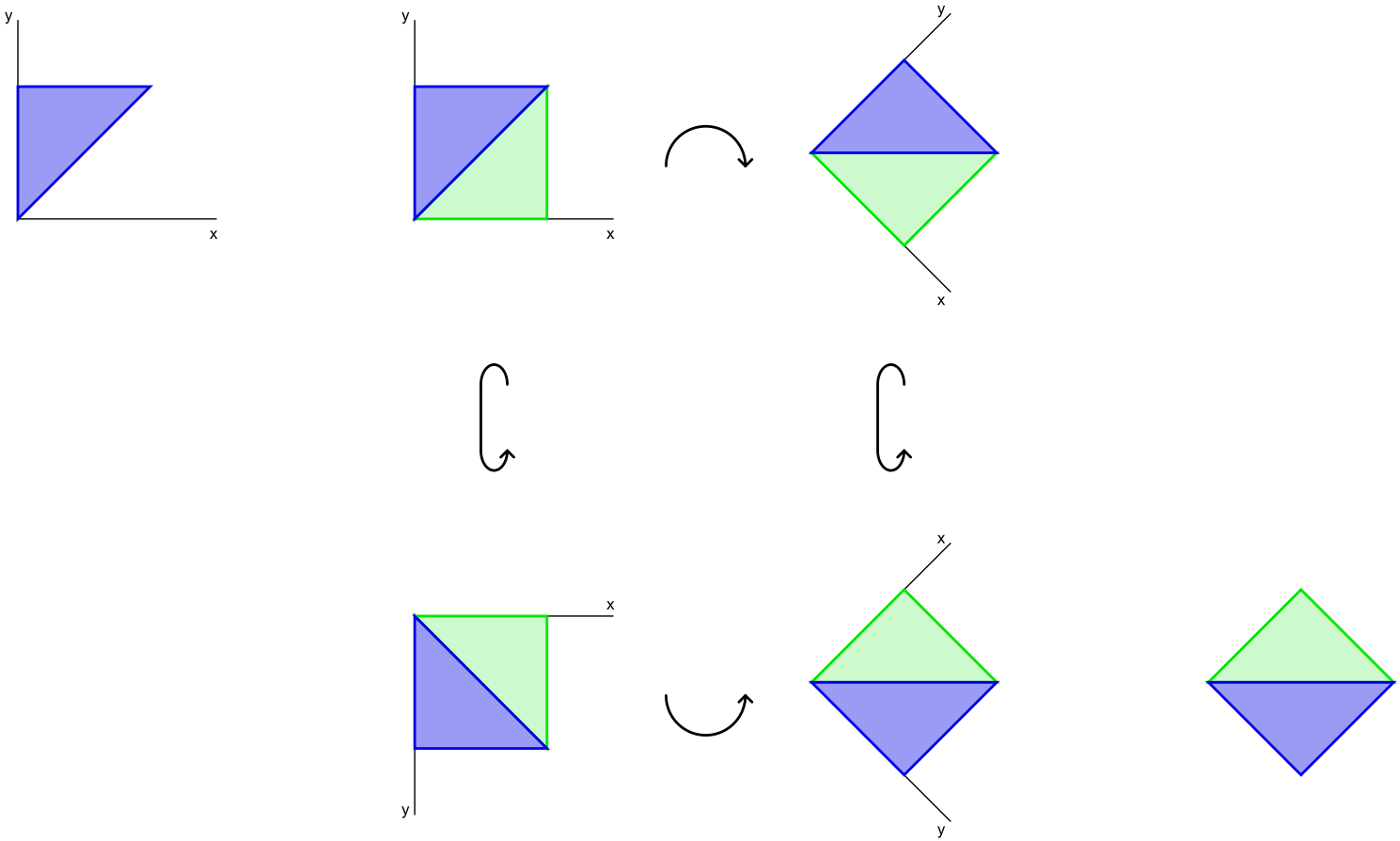
The resulting structure is a group and a vector space
(this was rediscovered many times in the last 60 years).

Two partial orders (a bilattice):

Informational partial order: $[a, d] \sqsubseteq [b, c]$ iff $a \leq b$ & $c \leq d$.

Material partial order: $[a, d] \leq [b, c]$ iff $a \leq b$ & $d \leq c$.

From Cartesian to Hasse representation



Negative probability

The space of signed measures ("charges") is a vector space.

The most well-known example:
Wigner quasiprobability distribution.

Intuition: see [Richard Feynman, "Negative probability"].

Uses of negative probability

Phase space formulation of quantum mechanics.

Denotational semantics of probabilistic programs [Kozen].

Occasional use in machine leaning.

Occasional use in quantum algorithms.

Should be used in probabilistic models of neural systems.

Outline

1. Introduction
   - Regulation of gene expression and evolutionary programming
   - Linear models of computation: probabilistic programming
   - Linear models of computation: generalized animations
2. Dataflow programs and linear models of computation
   - **Project Fluid:** `https://github.com/anhinga/fluid`
   - Almost continuous program transformations
   - **Dataflow graphs as matrices**
3. Partial inconsistency landscape

Electronic coordinates

These slides are linked from my page on partial inconsistency and vector semantics of programming languages:

http://www.cs.brandeis.edu/~bukatin/partial_inconsistency.html

E-mail:

bukatin@cs.brandeis.edu