Vector semantics: from partial inconsistency and bitopology to recurrent neural networks and self-referential dataflow matrix machines

Michael Bukatin

HERE North America LLC, Burlington, MA

Joint work with Steve Matthews and Andrey Radul

- - -

31st Summer Conference on Topology and its Applications, Special Session on Asymmetric Topology, August 2, 2016

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

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 $C^{Op} \times D$

Negative probability

Probabilistic powerdomain is embedded into the vector space of signed measures [Kozen].

(Phase space formulation of quantum mechanics is based on Wigner quasiprobability distribution.

It's enough to have good cancellation properties, complex numbers are not strictly necessary for this.)

Warmus numbers

Start with interval numbers, represented by ordinary segments.

Add pseudosegments $[a, b]$, such that $b < a$.

This corresponds to contradictory constraints, $x \leq b \& a \leq x$.

The new set consists of segments and pseudosegments.

Addition: $[a_1, b_1] + [a_2, b_2] = [a_1 + a_2, b_1 + b_2]$.

True minus: $-[a, b] = [-a, -b]$.

$-[a, b] + [a, b] = [0, 0]$.

This gets us a group and a 2D vector space.

True minus is antimonotonic

$x \sqsubseteq y \Rightarrow -y \sqsubseteq -x$.

True minus maps precisely defined numbers, $[a, a]$, to precisely defined numbers, $[-a, -a]$.

Other than that, true minus maps segments to pseudosegments and maps pseudosegments to segments.

In the bicontinuous setup, true minus is a bicontinuous function from $[R]$ to $[R]^{Op}$ (or from $[R]^{Op}$ to $[R]$).

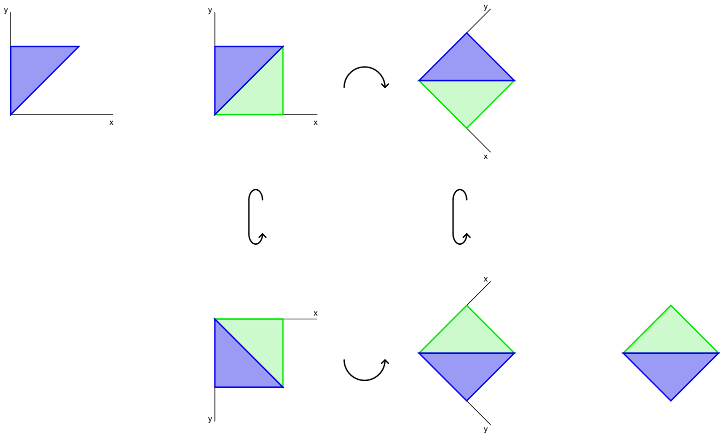
Multiple rediscoveries

Known under various names: Kaucher interval arithmetic, directed interval arithmetic, generalized interval arithmetic, modal interval arithmetic, interval algebraic extensions, etc.

First mention we know: M. Warmus, Calculus of Approximations. Bull. Acad. Pol. Sci., Cl. III, 4(5): 253-259, 1956,
`http://www.cs.utep.edu/interval-comp/warmus.pdf`

A comprehensive repository of literature on the subject is maintained by Evgenija Popova: The Arithmetic on Proper & Improper Intervals (a Repository of Literature on Interval Algebraic Extensions),
`http://www.math.bas.bg/~epopova/directed.html`

From Cartesian to Hasse representation



Partially inconsistent interval numbers as a domain of arrows

$[R] = \mathbb{R} \times \mathbb{R}^{Op}$

(There is a tension between the group structure on $\mathbb{R}$ and $[R]$ and the axioms of domains requiring $\perp$ and $\top$ elements which can be satisfied by restricting to a segment of reals, or by adding $-\infty$ and $+\infty$. I am mostly being ambiguous about this in this slide deck, but this is something to keep in mind.)

Bitopology and *d*-frames

Achim Jung, M. Andrew Moshier. On the bitopological nature of Stone duality. Technical Report CSR-06-13. School of Computer Science, University of Birmingham, December 2006, 110 pages.

d-frame for the (lower, upper) bitopology on $\mathbb{R}$

d-frame elements are pairs $\langle L, U \rangle$ of open rays, $\langle (-\infty, a), (b, +\infty) \rangle$ (a and b are allowed to take $-\infty$ and $+\infty$ as values).

Non-overlapping pairs of open rays are consistent ($a \leq b$), overlapping pairs of open rays ($b < a$) are total.

Correspondence with partially inconsistent interval numbers

The bilattice isomorphism between d-frame elements and partially inconsistent interval numbers with "infinity crust": $\langle(-\infty, a), (b, +\infty)\rangle$ corresponds to a partially inconsistent interval number $[a, b]$.

Consistent, i.e. non-overlapping, pairs of open rays ($a \leq b$) correspond to segments. Total, i.e. covering the whole space, pairs of open rays ($b < a$) correspond to pseudosegments.

Monotonic evolution of Warmus numbers by additions

Consider $x \sqsubseteq (x + x_1) \sqsubseteq (x + x_1 + x_2) \sqsubseteq \ldots$

Then every $x_i = [a_i, b_i]$ must be a pseudo-segment anti-approximating zero:

$[0, 0] \sqsubseteq [a_i, b_i]$, that is $b_i \leq 0 \leq a_i$.

Logic for fuzzy paraconsistent mathematics

Traditional fuzzy math: logic based on $[0, 1]$.

Paraconsistent math: logic based on the 4-valued bilattice.

Fuzzy paraconsistent math: logic based on the bilattice of Warmus numbers (probably within [0,1] or within [-1,1], or all reals with added "infinity crust").

References

Section 4 of Bukatin and Matthews, *Linear models of computation and program learning,* GCAI 2015, EasyChair Proceedings in Computing, **36**, pages 66–78, 2015, `http://easychair.org/publications/download/Linear_Models_of_Computation_and_Program_Learning`

Slides of my November 2014 talk at Kent State University: `http://www.cs.brandeis.edu/~bukatin/PartialInconsistencyProgressNov2014.pdf`

Kozen denotational semantics

One can think about probabilistic programs as transformers from the probability distributions on the space of inputs to the probability distributions on the space of outputs.

It is fruitful to replace the space of probability distributions by the space of signed measures (containing the probabilistic powerdomain).

D. Kozen, Semantics of Probabilistic Programs, Journal of Computer and System Sciences **22** (3), 328–350 (1981)

Open problem: domain equations

First steps towards higher-order theory are made in [Kozen, Semantics of Probabilistic Programs] and also in [Keimel, Bicontinuous Domains and Some Old Problems in Domain Theory].

However, in this context I have not seen anything coming close to the solution of domain equations, such as $D \cong [D \to D]$.

If one follows the approach by Kozen, where programs denote linear operators, and if one focuses on reversible programs, what seems to be called for here is **applicative representation theory**.

Denotational semantics vs operational semantics

Denotational semantics: it is not clear how to pull the constructions made in the spaces of meanings back to the realm of programs in the ways which would be computationally effective.

To address this problem let's focus on **operational semantics**, namely on software architectures which allow to take linear combinations of actual computational processes, rather than just of program meanings.

Sampling semantics and generalized animations

Linear streams: streams admitting the notion of linear combination of several streams.

Two years ago we considered two classes of linear streams: *streams of samples* from probability distributions (actually, from *signed measures*, so that we can have negative coefficients in our linear combinations) and *generalized animations*.

Since that time our group achieved the following progress.

Programming with linear streams: progress in 2015-2016

Dataflow matrix machines (DMMs) are a powerful generalization of **recurrent neural networks**.

- Large classes of dataflow programs were parametrized by matrices of numbers. `http://arxiv.org/abs/1601.01050`
- Recurrent neural networks are an important partial case of that. `http://arxiv.org/abs/1603.09002`
- Streams of matrices defining the network allow for self-referential facilities and self-modifying networks. `http://arxiv.org/abs/1605.05296`
- DMMs seem to be a powerful programming platform, unlike recurrent neural networks. `http://arxiv.org/abs/1606.09470`

Recurrent neural networks - the core part

A finite set of neurons indexed by $i \in I$.

"Two-stroke engine":

"Up movement": for all i, $y_i := f(x_i)$.

"Down movement": for all i, $x_i := \sum_{j \in I} w_{ij} * y_j$.

(w_{ij}) is the matrix of weights.

"The unreasonable effectiveness of recurrent neural networks"

Andrej Karpathy (PhD student at Stanford):

`http://karpathy.github.io/2015/05/21/rnn-effectiveness/`

Rectifier

It was typical to use sigmoid non-linearities as f, but a few years ago people discovered that **ReLU** (rectified linear units) tend to work much better: $f(x) = max(0, x)$.

This is an integral of the Heaviside step function. Lack of smoothness at 0 does not seem to interfere with gradient methods, and otherwise it's nice when the derivatives are so simple.

Strangely enough, our standard quasi-metrics on reals are closely related to ReLU:

$q_1(x, y) = f(x - y) = q_2(y, x)$.

Problem of vanishing gradients

The training only started to work well after people figured out how to fight the problem of vanishing gradients.

LSTM (original flavor): 1997.

A lot of options now, including a variety of LSTM flavors and other schemas. For a nice compact overview see this paper from Nanjing University: Guo-Bing Zhou, Jianxin Wu, Chen-Lin Zhang, Zhi-Hua Zhou, *Minimal Gated Unit for Recurrent Neural Networks*. http://arxiv.org/abs/1603.09420

Convergence problems

Convergence of methods for solving optimization problems during training of recurrent neural nets is tricky, and is not so much science, but feels more like a black art.

To get a flavor of it, see this nicely written paper: Matthew Zeiler, *ADADELTA: An Adaptive Learning Rate Method*.
`https://arxiv.org/abs/1212.5701`

The available selection of optimization methods is huge.

Boom in various flavors of neural nets

Explosion of research in neural nets and of industrial adoption.

NIPS conference: 1600 submitted papers last year, 2600 submitted papers this year.

DeepMind (London) publishes a lot of very interesting papers in this field: `https://deepmind.com/publications`

Etc, etc...

Dataflow matrix machines as a generalization of RNNs

Arbitrary linear streams.

A finite or countable collection of available **kinds of linear streams**.

A finite or countable collection of **neuron types**.

Each neuron type:

- a nonnegative input arity,
- a nonnegative output arity,
- a particular kind of linear streams is associated with each input and each output,
- a particular *built-in stream transformation*.

Dataflow matrix machines as a generalization of RNNs

Countable collection of neurons of each type.

Hence countable number of inputs x_i and outputs y_j.

Take countable matrix of weights with finite number of non-zero elements, and in particular make sure that w_{ij} can be non-zero only if the same kind of linear streams is associated with x_i and y_j.

Only neurons with at least one nonzero input or output weight are **active**, otherwise we keep them silent and treat their outputs as zeros. Hence only a finite number of neurons are active.

"Two-stroke engine"

"Up movement": for all active neurons C,
$y_{1,C}, ..., y_{n,C} := f_C(x_{1,C}, ..., x_{m,C})$.

n, m, and f_C correspond to the type of the neuron C.

"Down movement": for all inputs i having non-zero weights associated with them, $x_i := \sum_{\{j \ | \ w_{ij} \neq 0\}} w_{ij} * y_j$.

Self-referential facilities

Allow the kind of linear streams of countably-sized matrices (w_{ij}) with finite number of non-zero elements.

Introduce neuron `Self` having a stream of matrices (w_{ij}) on its output and use the current last value of that stream as the network matrix (w_{ij}) during the computations on each "down movement":

$x_i := \sum_{\{j \mid w_{ij} \neq 0\}} w_{ij} * y_j$.

Self-referential facilities

In the preprints above, `Self` has a single input taking the same kind of stream of matrices and the identity transformation of streams, so it just passes its input through.

Its output is connected with weight 1 to its input, hence it is functioning as an accumulator of additive contributions of other neurons connected to its input with non-zero weights.

Self-referential facilities

Now we think, it is more convenient to have two separate inputs for `Self`, x_W and $x_{\Delta W}$, connect the output of `Self` y_W to x_W with weight 1, take additive contribution from other neurons at the $x_{\Delta W}$ input, and compute $x_W + x_{\Delta W}$ on the "up movement".

This is the mechanism we propose as a replacement of untyped lambda-calculus for dataflow matrix machines.

If we limit ourselves to one kind of linear streams, namely streams of matrices (w_{ji}), we obtain the "moral equivalent" of programming within pure untyped lambda-calculus.

RNNs as a general-purpose programming platform

RNNs are Turing-universal.

However they are not a convenient general-purpose programming language, but belong to the class of

`https://en.wikipedia.org/wiki/Esoteric_programming_language`

and

`https://en.wikipedia.org/wiki/Turing_tarpit`

together with many other elegant and useful Turing-universal systems such as Conway's Game of Life and LaTeX.

DMMs as a general-purpose programming platform

DMMs are much more powerful than RNNs:

- arbitrary linear streams
- neurons with multiple input arity
- selection of convenient built-in transformations
- friendliness of DMMs towards sparse vectors and matrices
- self-referential facilities
- approximate representations of infinite-dimensional vectors (samples from probability distributions and signed measures)

DMMs as a general-purpose programming platform

Representing characters as vectors: “1-in-N” representation is standard in RNNs.

Take the alphabet as the basis, represent characters as vectors with 1 at the corresponding coordinate, and zeros at all others.

Sparse arrays are extremely important here, especially for large alphabets like Unicode (100,000+ characters).

DMMs as a general-purpose programming platform

Multiple inputs allow us to program via **multiplicative masks**, which can dynamically turn on and off, attenuate and amplify parts of the network.

By turning parts of the network on and off one can implement conditionals, redirect flows of data within the network, and precisely orchestrate coordination.

DMMs as a general-purpose programming platform

The lack of neurons with identity transform in usual RNNs is incredibly inconvenient, because one needs them for accumulators, leaky accumulators, and more.

We don't have this problem in DMMs.

DMMs as a general-purpose programming platform

We did some experimental work on sketching a programming language to interactively update a DMM while its running.

We try to follow the following informal principle:

Principle of **self-referential completeness** of the DMM signature relative to the language available to describe and edit the DMMs.

(DMM signature = available kinds of linear streams and types of neurons.)

The principle states that for all updates one can do in the language, one should be able to accomplish those updates by triggering appropriate neurons producing additive changes to `Self`.

DMMs as a general-purpose programming platform

More details in preprints:

Michael Bukatin, Steve Matthews, Andrey Radul, *Dataflow matrix machines as programmable, dynamically expandable, self-referential generalized recurrent neural networks*, `http://arxiv.org/abs/1605.05296`

Michael Bukatin, Steve Matthews, Andrey Radul, *Programming patterns in dataflow matrix machines and generalized recurrent neural nets*, `http://arxiv.org/abs/1606.09470`

This is just a start, a lot of further work is required...

Initial open-source prototypes

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

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`