Towards Co-Inductive Models for Natural Language Semantics *Speculation and Discussion*

Wlodek W Zadrozny

UNC Charlotte (and Duke U.)

wzadrozn@uncc.edu

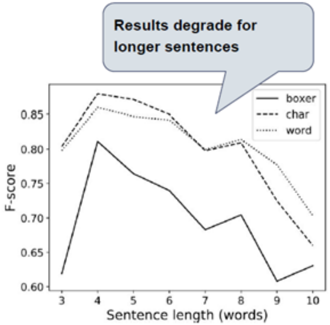
May 13, 2020

POS Tagging	Parsing
Text Mining (e.g. Person)	Computational Semantics
Google Search	Computational Pragmatics
Gist Translation	Good Translation

POS Tagging	Parsing
Text Mining (e.g. Person)	Computational Semantics
Google Search	Computational Pragmatics
Gist Translation	Good Translation

Table: On the left, one intuitive computational model is that of an automaton, with data coming as infinite streams (esp. for training). On the right the models are assembled into structures (often by hand) from previously defined components.

Typical chasm



Compare with Tsai et al. 2018 :

"We achieved an accuracy of 93.29% and 93.40% in the short-phrase validation set and long sentence validation set respectively"

[on a biomedical named entity (NER) task]

10.02.2020, CLASP seminar

1

[1] The average sentence length in the Penn Treebank is 20.54 words; in Genesis 34 words; "Biographia Literaria" 10% of sentences have the average length of 70 words; for patents, Claim 1 averages 150-180, and up to 1400 words

Adding coinduction to semantics,

as a basis for a more realistic, computationally sound, and scalable model of natural language understanding.

To this end we will focus on these questions

1. What is coinduction? (coinductive data, coinductive functions, coinductive proofs, coalgebra)
2. Do we need it?
3. What can be our next steps?

Once we address Questions 1 and 2, and before ending with a few possibilities for Question 3, we'll observe a few cases, where a coinductive method is being used in NLP. (Even though the term itself does not appear in any articles on `aclweb.org`)

Informally, what am I speculating about?

In theoretical computer science we have the concept of *co*-induction, or coinduction

Induction builds structures bottom up and can be viewed as a reductionist process, while coinduction provides top down constraints.

The main idea is to base NLU *partly* on coinduction. E.g. to address the problem of parsing and understanding of long sentences.

Coinduction can be incorporated into NL semantics (helped by the fact induction-coinduction relationships are relatively well investigated in formal logic and theoretical computer science).

Empirical evidence suggests we will be more successful in addressing difficult problems.

Coinductive data,coinductive functions,coinductive proofs

I will use "coinduction" in the most generic meaning, and I should talk about three aspects of coinduction:

- coinductive data
- coinductive functions
- coinductive inference: models and proofs

But I mostly focus on the first bullet.

Coinductive data: two styles of `list`

- **Data:** The 4-element finite list $L4 = [1, 1, 1, 1]$ is built by specifying

$$L4 = cons(1, cons(1, cons(1, cons(1, nil))))$$

where *cons* is a list constructor and *nil*, the empty list, a nullary *constructor*.

- **Codata:** The infinite list

$$L1 = [1, 1, 1, 1, ...]$$

is defined by specifying $hd(L1) = 1$ and $tl(L1) = L1$, where *hd* and *tl* are *destructors*. [2]

[2] This and next slide "Corecursion and coinduction: what they are and how they relate to recursion and induction" Mike Gordon

Coinductive programs: two styles of `addOne`

- Recursion (finite lists, step by step):

```
Add1(nil) = nil
Add1(cons(n, l)) = cons(n+1, Add1(l))
```

- Corecursion (infinite lists with lazy evaluation):

```
null(Add1(l))=(l=nil)
hd(Add1(l))=hd(l)+1
tl(Add1(l))=Add1(tl(l))
```

The recursively defined `Add1` maps finite lists to finite lists; the corecursively defined `Add1` maps finite lists to finite lists and infinite lists to infinite lists.

More generally, recursion defines a function mapping values from a datatype by invoking itself on the components of the *constructors* used to build data values. Corecursion defines a function mapping to a codatatype by specifying the results of applying *destructors* to the results of the function.

Coinductive inference: Largest vs Smallest models

When we give an **inductive** definition, we mean the smallest set that satisfies the given constraints; everything that's in the set has some justification. We build *the smallest model (i.e. the smallest fixpoint of the construction).*

Coinductive definitions specify the largest set that is consistent with them. The construction of a model proceeds by finding *the largest fixpoint consistent with the specifications*.

Which construction do you think will be more robust in real applications?
— Usually, irrelevant things are consistent with specs, so with coinductive constructions, we don't have to predict in advance all possible cases.

Interaction is coinductive

Example: Dialogue (CoInductive)

`Conversation` $\rightarrow$ `(Turn, Conversation)`

- I don't need to specify what a `Conversation` is. I only need to have an operational definition of a `Turn`
- `Conversation` can have hidden states. `Turn` is an observable, e.g. I can use it to find out who dominates a conversation, if it provides speaker names
- **Example:** my personalized Google search, where a Turn consists of a `(query, answer)` pair but the Conversation never ends, and is only partly observable (e.g. I don't know how Google modifies my profile for search).

Why we might benefit from a coinductive model of dialogue

Example from a CLASP seminar, R.Kempson: "Speaker/hearer exchange roles across all syntactic/semantic dependencies":

Ruth: *I'm afraid I burned the kitchen ceiling.*
Michael: *Did you burn*
Ruth: *myself? No, fortunately not.*

A: *Have all the students handed in*
B: *their term papers?*
A: *or even any assignments?*

WZ: Can a system be built based on a simple coinductive view of dialogues? Yes.

What is coalgebra?

Example: A simple model of an operating system:

$$\mathit{Stream} \rightarrow A \times \mathit{Stream}$$

Definition(informal):
A *coalgebra* consists of the domain S and a function c from S into some properly defined data structure containing S. (e.g. collection of states and records that depend on S)

Other Examples:

- Coalgebra of orbits of points: $f : X \rightarrow X$

  $$< x >= \{x, f(x), f(f(x)), ...\}$$

  periodicity = fixpoints
- Even simpler: Decimal expansions of reals, infinite series in math, ...

Algebras vs Coalgebras as maps

Algebras with “carrier” X are maps into X, of the form:

$$\boxed{X \quad \cdots \quad X} \xrightarrow[\text{construct}]{} X$$

Coalgebras with “state space” X are maps out of X, of the form:

$$X \xrightarrow[\text{modify/observe}]{} \boxed{X \quad \cdots \quad X}$$

"Introduction to Coalgebra” slides, Bart Jacobs, EWSCS 2011: 16th Estonian Winter School in Computer Science

Automaton as a coalgebra

$$c : S \to \{halt\} \cup (A \times S)$$

S is a set of states, A outputs (or observables).

Examples(with some modification of the transition functions)

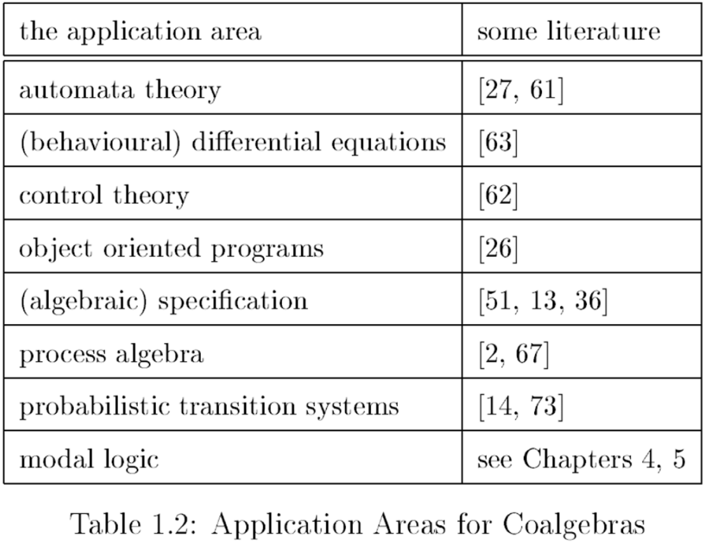
- Any `regex`
- Chunkers that can operate with limited information; e.g. sentence splitters, phrase finders, ","-analyzers etc.
- RNNs (LSTMs, BERT encoders) share the same spirit but would have more complex defitions. [3]
- Actually, in "The differential calculus of causal functions" Sprunger & Jacobs, show that learning in NN is coinductive (at least for some types of RNNs).

[3](Note that binary-state NNs are equivalent to finite automata).

What are coalgebras, intuitively (B.Jacobs)

- Mathematical models for state-based computation
  - your computer has a complicated internal state
  - you as user can only modify it to some extend
  - also, you can observe only so much
- Basically only two kinds of operations:
  - move to a next state, somehow (deterministically, non-deterministically, probabilistically, . . .)
  - make an observation ("measurement") about the current state
  - These operations can be combined ("observation has side-effect")
- Describing such state-based systems and their dynamics requires a new kind of mathematics

`http://cs.ioc.ee/ewscs/2011/jacobs/jacobs-slides.pdf`, cited almost verbatim



the application area	some literature
automata theory	[27, 61]
(behavioural) differential equations	[63]
control theory	[62]
object oriented programs	[26]
(algebraic) specification	[51, 13, 36]
process algebra	[2, 67]
probabilistic transition systems	[14, 73]
modal logic	see Chapters 4, 5

Table 1.2: Application Areas for Coalgebras

"Coalgebras and Modal Logic" Alexander Kurz 10.1.1.17.8962

Coalgebras applicable to semantics?

Example from a CLASP seminars: Hannes Rieser: *A Process Algebra Account of Speech-Gesture Interaction*.

- **Claim:** *description of speech-gesture coordination cannot be given solely in a naïve compositional way*
- **Caveat:** *composition does play a role finally, when the speech-gesture contact points have been identified*

- My translation: Observation first, analysis later.
- Process algebra $\simeq$ Coalgebra $\simeq$ Coinduction (per previous slide)

Stepping into the mainstream

Lexicography

Syntax

Semantics

Lexicography: Cognitive Aspects of the Lexicon (CogALex 2020)

“Rather than considering the lexicon as a static entity, ..., dictionaries are now viewed dynamically, i.e., as lexical graphs, ... whose weight links may vary over time.

”While lexicographers view words as products (holistic entities), **psychologists and neuroscientists view them as processes (decomposition)**, involving various steps or layers (representations) between an input and an output.” [4]

Note: Coinduction uses **decomposition** (destructors such as `hd` and `tl` shown earlier)

[4] `https://sites.google.com/view/cogalex-2020/home/call-for-papers`

Syntax coinductively?

Example from a CLASP seminar: M. Gotham and D. Haug, *Glue semantics for Universal Dependencies*

- *What the approaches just mentioned* (Minimalism, Montague –wz) *have in common is the view that syntactic structure plus lexical semantics determines interpretation.*
- *From this it follows that if a sentence is ambiguous, (...), then that ambiguity must be either lexical or syntactic.*
  – (WZ: Explains why syntax driven approaches to semantics are brittle?).
- *The Glue approach is that syntax* **constrains** *what can combine with what, and how.* (emph. wz)

Semantics: Extending TTR through coinduction?

The lemma *constrain* appears 36 times in "Modelling Language, Action, and Perception in Type Theory with Records" Dobnik, Cooper, and Larsson.

So perhaps we only need a small step to make it coinductive.

If I am reading it right, the TTR stratified type system is built bottom up to avoid the Russell paradox (Cooper "Type theory and semantics in flux"), but the stratification is already omitted in Larson "Formal semantics for perceptual classification", to ease the exposition. I suggest we replace the stratification with constraints (per Barwise and Moss "Vicious Circles" such models should be consistent).

TTR can be extended with coinductive types

Proof/Argument:

- TTR uses Martin-Löf's type theory (Cooper& Ginzburg, 2015)
- TTR's main operation is 'merge' i.e. unification

- Martin-Löf's type theory can be extended to admit coinductive types ("General Recursion Via Coinductive Types" Capretta 2005; "On the Semantics of Coinductive Types in Martin-Löf Type Theory" F. De Marchi 2005)
- Unification and counification happily coexist in some variants of logic programming (e.g. Gupta et al. "Infinite computation, co-induction and computational logic." 2011.)

- **Therefore:** TTR with inductive and coinductive (ML) types, and with merge and co-merge operations should have a well defined computational model.
- And it would be nice to make it explicit, and perhaps even implement it.
- Oh wait, hasn't this already been done? ("Probabilistic Type Theory and Natural Language Semantics" Cooper et al. 2015; https://github.com/robincooper/pyttr/blob/master/README) – partially it seems.

TTR can be extended with coinductive types

Proof/Argument:

- TTR uses Martin-Löf's type theory (Cooper& Ginzburg, 2015)
- TTR's main operation is 'merge' i.e. unification

- Martin-Löf's type theory can be extended to admit coinductive types ("General Recursion Via Coinductive Types" Capretta 2005; "On the Semantics of Coinductive Types in Martin-Löf Type Theory" F. De Marchi 2005)
- Unification and counification happily coexist in some variants of logic programming (e.g. Gupta et al. "Infinite computation, co-induction and computational logic." 2011.)

- **Therefore:** TTR with inductive and coinductive (ML) types, and with merge and co-merge operations has a well defined computational model.
- *So, yes, it would be nice to make it explicit, and implement it.* [5]

[5] Ditto for construction grammars, cf. e.g. Distributional Semantics Meets Construction Grammar. Towards a Unified Usage-Based Model of Grammar and Meaning. Rambelli et al. 2019

One more example: Intentionality (Fox & Lappin)

Remember two definitions of `List` – on finite domains they are equivalent.

F&L write:
Given the distinction between denotational and operational meaning we can now interpret the non-identity of terms in the representation language as an operational difference in the functions that these terms express.
... we are taking the semantic content of terms in a natural language to be the functions that we use to compute the denotations of these expressions.

- Operational semantics might work for dialogue – if we have a procedure for generating the next utterance.
- It feels like a "process algebra", except that parallel processes do not necessarily behave like functions (Beaten 2005 p.136).

This talk. Where are we?

- Introduced coinduction (and most of related terms, except for bisimulation, and coinductive proofs). Informally, but it's ok for the purpose of this talk.
- Showed we are already using coinduction in practice and we need more coinduction.
  — Coinduction – in principle – addresses several problems discussed in earlier CLASP seminars
- Argued for extending TTR with coinductive records.

- My old 1994 L & P paper shows that *any* semantics can be encoded coalgebraically
- IBM Watson was coinductive [6]

Are we done?

[6] Obviously it wasn't designed with coinduction in mind, but forced by the need to provide accurate answers using texts with long sentences.

Not everything should be done via coinduction

Example from CLASP: M. Baroni "Artificial neural networks and the challenge of compositional generalization"

RNNs "generalization skills do not display systematic compositionality"

(Recurrent) neural networks are remarkably powerful and general

- *Agnostic "end-to-end" learners from input-output pairs*
- *They can generalize to new inputs that are different from those they were trained on...*
- *... but their generalization skills do not display systematic compositionality*
- *Thus, they cannot adapt fast to continuous stream of new inputs in domains such as language, math, and more generally reasoning*

Not everything should be done via coinduction

M. Baroni @ CLASP:

RNNs "generalization skills do not display systematic compositionality"

My hypothesis: perhaps this is not their function!

- We are now at a stage where some researchers create NN architectures for specific data sets, and others add new layers to type systems to extend compositionality to new constructions.
- What if both groups are pushing too hard, trying to use their tools for tasks which are better done with the other set?

Limitations of coinductive view of human cognition

Coinduction (i.e. DNN) doesn't work for:

- Algebra and arithmetic – even with special NN architectures the performance is limited, and for general purpose NN (even transformers) it is really bad.(starting point: arXiv:2001.05016v1)
- Any task requiring common sense, causality, and depth. (Lake et al. BBS arXiv:1604.00289v3)
  — Including game playing with slightly modified objectives
  - Get the lowest possible score.
  - Get closest to 100, or any level, without going over.
  - Beat your friend, but just barely, not by too much,
  - Go as long as you can without dying.
  - Die as quickly as you can.
  - Pass each level at the last possible minute
- General dialogue

Coinduction and Induction Together (Example)

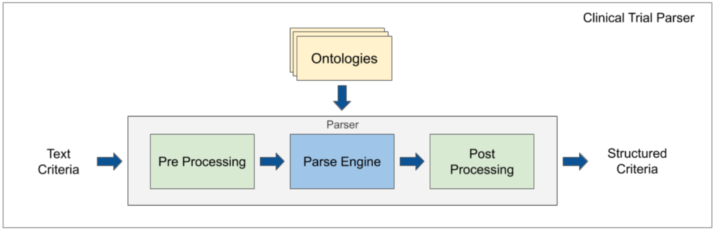
`*The younger woman might have been tall and, and the older one definitely was, blond.` (From COLA)

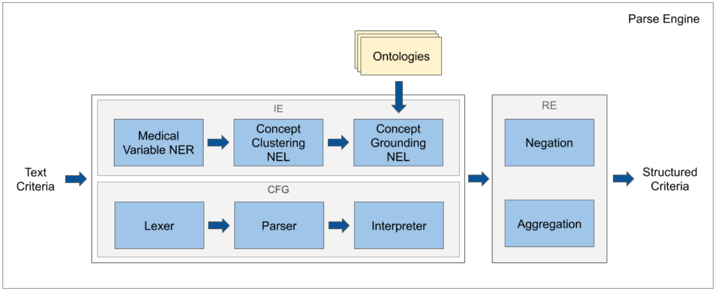
If this sentence appears in a dialogue transcript:

1. Coinductively (sentence recognition) we find the boundaries of the sentence, and phrases (parts) — pattern matching works well here.
2. Inductively (CFG) or Coinductively (data trained dependency parser), structures are assigned to the parts and/or the whole.

   Some `predicate(argument)` semantics is assigned to the structures — pattern matching or CFG[A]. This produces constraints on a DRS.
3. Coinductively (in written text), we find the "`, phrase,`" pattern (or construction).
4. Inductively, we try to apply the *ellipsis* as a possible meaning frame.
5. Inductively, we see that `younger_woman:tall and blond` is a possible interpretation. We add it to the DRS (and check its consistency).

Just in (May 7): A new FB parser: CFG + IE





https://ai.facebook.com/blog/open-sourcing-a-new-parser-to-improve-clinical-trial-participant-recruitment/

Summary, Open Problems, and Acknowledgments

Summary:

Coinduction is a mathematical and computational tool for specifying constraints on program behavior and data.

It is a natural complement to standard (i.e. bottom up) ways of defining compositional semantics or syntactic correctness.

It provides a principled (formal) view of the current practice in human-computer interaction, parsing, machine translation and others.

It might extend the process/program based view of semantic interpretation in (Fox & Lappin)

It covers possible worlds semantics

It has practical limitations (partial observations do not resolve all questions about interpretations), but is less brittle for longer text.

It has been used before, and shown (in principle) to be able encode any semantics

The obvious question:

Can we create a model combining inductive and coinductive representations, and using one formalism for

- lexicon
- syntax
- syntax-semantic interface
- semantics
- pragmatics

and which would outperform alternative models based solely on either inductive or coinductive representations.

Use TTR as a starting point? Construction grammars?

Other questions

- Lexicon (Cogalex inspired): Could we do something useful in this space using coalgebraic specifications?
- Make TTR explicitly coinductive and apply it somewhere?
- Is there a greedy left-to-right algorithm for inductive/coinductive semantic parsing? Or should several processes of interpretation operate in parallel in the style of a process algebra. [7]
- Build a game-based view of dialogue (games and coinduction are related)[9]

[7] If the latter, what is the NL interpretation of various process algebra postulates, e.g. the right associativity. [8]

[9] J. Barwise and L Moss. Vicious circles: on the mathematics of non-wellfounded phenomena. Center for the Study of Language and Information, 1996.

Acknowledgments

- Discussion with the CLASP members in 2019
- L.Moss, personal communication[10], two weeks ago
- N.Ruozzi, personal communication,[11] this week.

- (Possibly, I missed to list some sources, but this will be corrected in the next version. This revised set of slides will also list resources I used, with short commentaries on their relevance to this endeavor).

10 mentioning games, among other things

11 coinductive principles could be applicable in deep reinforcement learning for NL.

Questions?

Thank you!