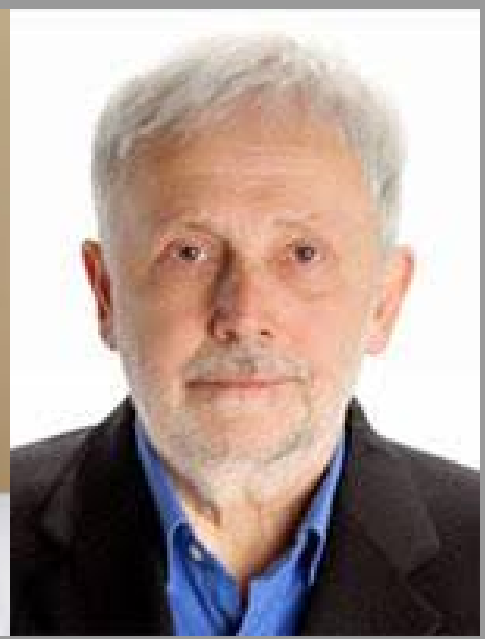
Benchmarking Linear Logic Proofs

Valeria de Paiva
Topos Institute, Berkeley, CA
Visiting DI, PUC-RJ, RJ

November, 2020

Thanks Tom and Anton for the invitation!

Thank you friends for our **continued** collaboration!

Based on

The ILLTP Library for Intuitionistic Linear Logic, Linearity/TLLA 2018 (with Carlos Olarte, Elaine Pimentel and Gisele Reis)

and

Deriving Theorems in Implicational Linear Logic, Declaratively, ICLP 2020 and **Training Neural Networks as Theorem Provers via the Curry-Howard Isomorphism**, Computational Logic and Applications 2020 (with Paul Tarau)

Motivations

- *benchmark*: "To measure the performance of (an item) relative to another similar item in an impartial scientific manner."
- Benchmarks for theorem provers well-developed area of AI
- Since Logic Theorist (LT) 1956 Newell and Simon

"the first artificial intelligence program", proved 38 of the first 52 theorems of Principia Mathematica

Motivation

- Many theorem provers: interactive and automatic
- Since 1993 TPTP: Thousands of Problems for Theorem Provers (http://www.tptp.org/)
- also CASC competition (CADE ATP System Competition) http://www.tptp.org/CASC/
- not much for **non-classical** logics
- intuitionistic logic ILTP (http://www.iltp.de/) and some collections of modal problems including QMLTP (http://www.iltp.de/qmltp/)
- where is Linear Logic?

Goals

- Collect problems/theorems in (a fragment of) Linear Logic
- Investigate variants of the logic
- Investigate variants of translations between logics
- Use provers/benchmarks/ML as tools for experiments in logic

Logic as a Lab Science!

Linear Logic: a tool for semantics

- A proof theoretic logic described by Girard in 1986.
- Basic idea: assumptions cannot be discarded or duplicated. They must be used exactly once – just like dollar bills (except when they're marked by a modality "!")
- Other approaches to accounting for logical resources before. **Relevance Logic**
- Great win for Linear Logic:
  Account for resources when you want to, otherwise fall back to traditional logic via translation $A \to B$ iff $!A \multimap B$

Linear Logic

In Linear Logic formulas denote resources. Resources are premises, assumptions and conclusions, as they are used in logical proofs. For example:

- \$1 $\multimap$ latte

  If I have a dollar, I can get a Latte

- \$1 $\multimap$ cappuccino

  If I have a dollar, I can get a Cappuccino

- \$1 I have a dollar

Can conclude either latte or cappuccino

— But using my dollar and one of the premisses above, say \$1 $\multimap$ latte gives me a latte but the dollar is gone

— Usual logic doesn't pay attention to uses of premisses, A implies B and A gives me B but I still have A...

Linear Implication and (Multiplicative) Conjunction

Traditional implication: $A, A \to B \vdash B$

$A, A \to B \vdash A \wedge B$ Re-use A

Linear implication: $A, A \multimap B \vdash B$

$A, A \multimap B \nvdash A \otimes B$ Cannot re-use A

Traditional conjunction: $A \wedge B \vdash A$ Discard B

Linear conjunction: $A \otimes B \nvdash A$ Cannot discard B

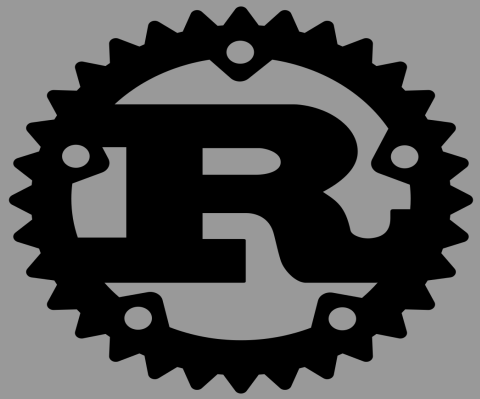
Of course!: $!A \vdash !A \otimes !A$ Re-use

$!(A) \otimes B \vdash B$ Discard

Linear Logic Results

- Soundness and Completeness for coherence spaces and many other interesting models
- "Have-your-cake-and-eat-it" theorem: Intuitionistic logic proves $A \to B$ iff Linear Logic proves $!A \multimap B$.
- A new graphical Natural Deduction system: proof nets
- A new style of proof systems: focused systems
- Some 30 years of limelight, especially in CS

Why Bother with Benchmarks?



- Linear Logic has come of age
- useful in programming languages, game semantics, quantum physics, linguistics
- Several Provers?
  maybe should discuss adequacy or efficiency?
- Nah!
- Because it can help us understand the logic, where it differs or not from traditional systems

(some) Linear Logic Provers

- LLTP (Maude) https://github.com/carlosolarte/Linear-Logic-Prover-in-Maude
- LL prover (http://bach.istc.kobe-u.ac.jp/llprover/)
- linTAP http://www.leancop.de/lintap/ Otten et al
- LL prover explorer, Lolli, etc
- YALLA (Coq) O. Laurent https://perso.ens-lyon.fr/olivier.laurent/yalla/

Choosing Logic Problems

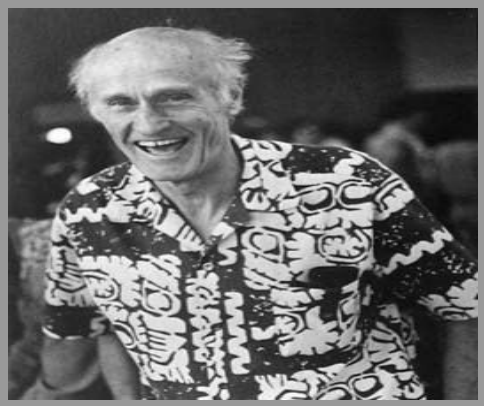
To be used as a standard a collection of problems should satisfy some criteria. at least the following:

- Formulae should able to distinguish different characteristics of the logical systems and provers (design choice points)
- important theorems and paradigmatic formulae should be present (how do we know?)
- Should be large enough so that we can do comparisons between different provers and systems
- (Not taken in consideration here) automatic comparison scripts and efficiency timings should be computed by third parties

Design choices

- Classical or Intuitionistic Linear Logic? FILL?
- differences in provability
- is there a set of "principal" LL theorems?
- Easy place to find LL theorems: Intuitionistic Logic
- Use original theorem, everything provable in IL is provable in LL using Girard's translation
- but hey, there are other (many) translations
- which one to choose and why?
- new use of computational provers and comparisons: to help to clarify the theory

Our choices



- starting: Kleene "Introduction to Metamathematics" 1952
- a basic collection of intuitionistic theorems
- minimal set of intuitionistic theorems that a sound prover should be able to derive
- helpful to uncover bugs and sources of unsoundness
- (historical note: with Sara Kalvala "Linear Logic in Isabelle", 1995. sequent calculus in Isabelle, deprecated now)

Rudimentary Intuitionistic Logic

- Rudimentary fragment of Intuitionistic Logic, ie $(\rightarrow, \wedge, \neg)$-fragment of IL.
- Why this fragment?
- Intuitionistic disjunction poses some problems in LL. Additive or multiplicative disjunction?
- Each way leads to very different systems. Concentrate on easy cases first, hence "rudimentary" IL.
- This gives us some 61 theorems from Kleene's book (next slide)
- problems in Giselle's repo https://github.com/meta-logic/lltp

Kleene Examples

1. $\vdash A \to A$
2. $A \to B, B \to C \vdash A \to C$
3. $A \to (B \to C) \vdash B \to (A \to C)$
4. $A \to (B \to C) \vdash A \wedge B \to C$
5. $A \wedge B \to C \vdash A \to (B \to C)$
6. $A \to B \vdash (B \to C) \to (A \to C)$
7. $A \to B \vdash (C \to A) \to (C \to B)$
8. $A \to B \vdash A \wedge C \to B \wedge C$
9. $A \to B \vdash C \wedge A \to C \wedge B$
10. $\neg A \vdash A \to B$
11. $A \vdash \neg A \to B$
12. $B \vdash A \to B$
13. $A \to B \vdash \neg B \to \neg A$
14. $A \to \neg B \vdash \neg\neg B \to \neg A$
15. $A \to B, B \to A \vdash A \leftrightarrow B$
16. $A \leftrightarrow B \vdash A \to B$
17. $A \leftrightarrow B \vdash B \to A$
18. $A \leftrightarrow B, A \vdash B$
19. $A \leftrightarrow B, B \vdash A$
20. $\vdash A \leftrightarrow A$
21. $A \leftrightarrow B \vdash B \leftrightarrow A$
22. $A \leftrightarrow B, B \leftrightarrow C \vdash A \leftrightarrow C$
23. $A \to (B \to C), \neg\neg A, \neg\neg B \vdash \neg\neg C$
24. $\neg\neg(A \to B) \vdash \neg\neg A \to \neg\neg B$
25. $\neg\neg(A \to B), \neg\neg(B \to C) \vdash \neg\neg(A \to C)$
26. $\vdash \neg\neg(A \wedge B) \leftrightarrow (\neg\neg A \wedge \neg\neg B)$
27. $\vdash \neg\neg(A \leftrightarrow B) \leftrightarrow (\neg\neg(A \to B) \wedge \neg\neg(B \to A))$
28. $A \leftrightarrow B \vdash (A \to C) \leftrightarrow (B \to C)$
29. $A \leftrightarrow B \vdash (C \to A) \leftrightarrow (C \to B)$
30. $A \leftrightarrow B \vdash (A \wedge C) \leftrightarrow (B \wedge C)$
31. $A \leftrightarrow B \vdash (C \wedge A) \leftrightarrow (C \wedge B)$
32. $A \leftrightarrow B \vdash \neg A \leftrightarrow \neg B$
33. $\vdash ((A \wedge B) \wedge C) \leftrightarrow (A \wedge (B \wedge C))$
34. $\vdash (A \wedge B) \leftrightarrow (B \wedge A)$
35. $\vdash (A \wedge A) \leftrightarrow A$
36. $A \vdash (A \to B) \leftrightarrow B$
37. $B \vdash (A \to B) \leftrightarrow B$
38. $\neg A \vdash (A \to B) \leftrightarrow \neg A$
39. $\neg B \vdash (A \to B) \leftrightarrow \neg A$
40. $B \vdash (A \wedge B) \leftrightarrow A$
41. $\neg B \vdash (A \wedge B) \leftrightarrow B$
42. $\vdash A \to \neg\neg A$
43. $\vdash \neg\neg\neg A \leftrightarrow \neg A$
44. $\vdash \neg(A \wedge \neg A)$
45. $\vdash \neg(A \leftrightarrow \neg A)$
46. $\vdash \neg\neg(\neg\neg A \to A)$
47. $\vdash (A \wedge (B \wedge \neg B)) \leftrightarrow (B \wedge \neg B)$
48. $\vdash (A \to B) \to \neg(A \wedge \neg B)$
49. $\vdash (A \to \neg B) \leftrightarrow (\neg(A \wedge B))$
50. $\vdash (\neg(A \wedge B)) \leftrightarrow (\neg\neg A \to \neg B)$
51. $\neg\neg B \to B \vdash (\neg\neg A \to B) \leftrightarrow (A \to B)$
52. $\neg\neg B \to B \vdash (A \to B) \leftrightarrow (\neg(A \wedge \neg B))$
53. $\vdash (\neg\neg A \to B) \to \neg(A \wedge \neg B)$
54. $\vdash (A \wedge B) \to \neg(A \to \neg B)$
55. $\vdash (A \wedge \neg B) \to \neg(A \to B)$
56. $\vdash \neg\neg A \wedge B \to \neg(A \to \neg B)$
57. $\vdash (\neg\neg A \wedge \neg B) \leftrightarrow \neg(A \to B)$
58. $\vdash \neg(A \to B) \leftrightarrow \neg\neg(A \wedge \neg B)$
59. $\vdash \neg\neg(A \to B) \leftrightarrow \neg(A \wedge \neg B)$
60. $\vdash \neg(A \wedge \neg B) \leftrightarrow (A \to \neg\neg B)$
61. $\vdash (A \to \neg\neg B) \leftrightarrow (\neg\neg A \to \neg\neg B)$

Translations?

- IL theorems are not necessarily theorems in LL. Need translations.
- Which translations?
- Four 'translations'
  1. Girard: $(A \to B)^G = !A \multimap B$
  2. "Lazy" $(A \to B)^K = !(A \multimap B)$
  3. Liang/Miller's 0-1
  4. Forgetful $=$ read $\to$ as $\multimap$
- Last is not provability preserving!
- First experiment: 61 theorems (IL) multiplied by 4 gives us 244 'problems' to check in Linear Logic
- Give me an automated theorem prover, please!

Results

- Olarte implemented a basic prover for IL as well as for ILLF and LLF (all focused systems)
- specified in Rewriting Logic and implemented in Maude (Meseguer)
- proofs of the original IL sequent, together with the derivation trees of the corresponding ILL sequents, when provable
- 22 sequents are not provable in ILL
- Obtained a collection of basic tests for LL
- Extended this basic collection using (translations of) problems from ILTP and from reachability of Petri Nets
- Ended up with 4,494 formulas in our ILLTP library
- Some comparison of translations, but need more!

Results

#	LJ	m	g	p	0/1
1	31	46	43	83	59
2	48	111	168	226	185
3	49	94	128	208	189
4	48	97	326	219	313
5	38	98	108	182	197
6	48	106	157	252	214
7	51	108	156	261	201
8	34	91	114	191	271
9	35	89	111	173	257
10	34	19 (x)	79	98	98
11	34	18 (x)	82	97	97
12	19	20 (x)	42	74	59
13	47	89	137	184	180
14	62	96	186	297	757
15	48	156	181	527	331
16	34	19 (x)	91	139	137
17	35	21(x)	94	132	142
18	34	24 (x)	97	115	114
19	35	20 (x)	92	113	115
20	18	90	43	126	108

#	LJ	m	g	p	0/1
21	54	160	196	515	1825
22	150	228	◷	◷	◷
23	2318	164	◷	◷	◷
24	211	142	4694	5522	◷
25	4063	202	◷	◷	◷
26	140	20 (x)	27482	◷	◷
27	◷	23 (x)	◷	◷	◷
28	86	227	◷	◷	◷
29	86	240	◷	◷	◷
30	48	191	292	3424	◷
31	54	209	353	3752	◷
32	83	202	12683	◷	◷
33	21	166	123	281	609
34	18	131	81	217	276
35	21	18 (x)	50	180	153
36	35	19 (x)	107	209	184
37	18	19 (x)	67	151	139
38	54	21 (x)	157	333	319
39	67	22 (x)	271	595	626
40	21	18 (x)	67	165	178

#	LJ	m	g	p	0/1
41	33	21 (x)	130	172	209
42	41	61	83	120	122
43	96	183	271	528	7431
44	34	59	88	102	141
45	96	19 (x)	◷	5241	◷
46	66	25 (x)	185	250	427
47	61	19 (x)	234	186	875
48	48	94	146	181	370
49	66	161	204	535	46292
50	94	245	2618	18580	◷
51	882	295	◷	◷	◷
52	112	257	◷	◷	◷
53	67	126	255	335	14764
54	49	74	115	170	268
55	49	92	136	187	345
56	64	97	181	253	3946
57	385	20 (x)	◷	◷	◷
58	118	20 (x)	◷	◷	◷
59	168	20 (x)	◷	◷	◷
60	96	214	4004	8427	◷
61	9785	288	◷	◷	◷

Figure 1: Comparison of translations: x indicates that the formula is not provable; ◷ indicates timeout (over 60 seconds). Times are measured in **miliseconds**.

What have we got?

- proposed a set of Linear Logic sequent tests, and have implemented provers for logics LJ, ILL and CLL.
- All three provers are focused-based
- As a first experiment we tried 4 translations and we can empirically compare times for them
- Can we see patterns on these timeouts and failures?
- Does this help when comparing translations?
- Would like to compare with other translations, e.g. LK_{tq}
- Would like to have parallel/ensemble provers, perhaps
- how can we push this work forward?

Changing Gears

- Lukewarm reception at FLoC2018: not enough interest in the benchmarks from theorem provers (mostly they're interested in specific applications, like automating proofs in logic itself – several embeddings of LL into Coq)
- Co-authors wanted to do different things
- Tarau's papers "Formula Transformers and Combinatorial Test Generators for Propositional Intuitionistic Theorem Provers" and "Combinatorial Testing Framework for Intuitionistic Propositional Theorem Provers" (2019)
- Promised test generators and formula readers automatically converting the 'human-made' tests of the ILTP library to Prolog and Python form. Mentioned a testing framework focusing on the implicational fragment of intuitionistic logic
- I asked if we could do the same for **Linear Logic**

Building theorems

- This is the work presented on ICLP2020
- Main insight of work: instead of building formulas of a logic and trying to prove whether they are theorems or not, with given rules, build only the provable linear ones
- True by construction and provably so, using the Curry-Howard isomorphism
- how?
- generate all theorems of a given size in LL
- use a low polynomial type inference algorithm associating a type (when it exists) to a lambda term
- rely on the Curry-Howard isomorphism, to generate simply typed lambda terms in normal form

What do we get?

- an implicational intuitionistic logic prover specialized for linear formulas
- a big dataset for training neural networks that prove intuitionistic theorems
- preliminary results: very high success rate with seq2seq encoded LSTM neural networks (not in the paper https://arxiv.org/pdf/2009.10241.pdf)
- intuition: use combinatorial generation of lambda terms + type inference (easy) to solve some type inhabitation problems (hard).

How do we go about it?

- implicational linear formulas (Prolog code), built as:
  - binary trees of size N, counted by Catalan numbers $Catalan(N)$
  - labeled with variables derived from set partitions counted by $Bell(N+1)$ (see **A289679** in OEIS)
- linear lambda terms (proof terms for the implicational formulas)
- linear skeleton Motzkin trees (binary-unary trees with constraints enforcing one-to-one mapping from variables to their lambda binders)
- *closed* linear lambda terms
- after a **chain of refinements**, we derive a compact and efficient generator for *pairs of linear lambda terms in normal form* and their types (which always exist as they are all typable!)
- Voilá! almost 8 billion theorems in a few hours

Want to machine learn tautologies?

(Tarau's *Training Neural Networks as Theorem Provers via the Curry-Howard Isomorphism* at CLA 2020
`https://www.youtube.com/channel/UCk0-_Hgr_o3KbRQWdeoYIHA`)

- Can we train neural networks to learn inference on an interesting logic?
- Yes! our logic is implicational intuitionistic linear logic.
- We need to derive an **efficient** algorithm requiring a low polynomial effort per generated theorem and its proof term.
- Outcomes:
  - implicational intuitionistic linear logic prover
  - a dataset for training neural networks
  - high success rate **seq2seq** encoded **LSTM** neural nets
  - open: can techniques extend to harder (e.g., PSPACE-complete) logics?

Machine learning tautologies

Several versions for Boolean logic, e.g. *Can Neural Networks Understand Logical Entailment?* Evans, Saxton, Amos, Kohli, and Grefenstette, 2018. arXiv:1802.08535 (*Automated proof synthesis for propositional logic with deep neural networks*, arxiv 1805.11799 for intuitionistic logic) not the same ML-problem

- *implicational fragment* of LL is decidable
- Curry-Howard holds: linear lambda-calculus corresponds to implicational only linear logic
- polynomial algorithms for generating its theorems are useful:
  - when turned into *test sets*, combining tautologies and their proof terms can be useful for testing correctness and scalability of linear logic theorem provers
  - when turned into *datasets*, they can be used for training deep learning networks focusing on *neuro-symbolic* computations

Machine learning tautologies

- can use combinatorial generation of lambda terms + type inference (easy) to solve some type inhabitation problems (hard)
- GOOD NEWS: there's a *size-preserving bijection between linear lambda terms in normal form and their principal types*!
- a proof follows immediately from a paper by Noam Zeilberger who attributes this observation to Grigori Mints
- the bijection is proven by exhibiting a *reversible transformation* of oriented edges in the tree describing the linear lambda term in normal form, into corresponding oriented edges in the tree describing the linear implicational formula, acting as its principal type
- obtained a generator for all theorems of implicational linear intuitionistic logic of a given size, as measured by the number of lollipops, without having to prove a single theorem!

The datasets

- the dataset containing generated theorems and their proof-terms in prefix form (as well as their LaTeX tree representations marked as Prolog "%" comments) is available at `http://www.cse.unt.edu/~tarau/datasets/lltaut/`
- it can be used for correctness, performance and scalability testing of linear logic theorem provers
- the `<formula, proof-term>` pairs in the dataset are usable to test deep-learning systems on theorem proving tasks
- also, formulas with non-theorems added for **IPILL**

Can Neural Nets help Theorem Proving?

- We search for good frameworks for neuro-symbolic computing
- theorem provers are computation-intensive, sometimes Turing-complete search algorithms
- two ways neural networks can help:
  - fine-tuning the search, by helping with the right choice at choice points
  - used via an interface to solve low-level 'perception'-intensive tasks
- can we simply replace the symbolic theorem prover given a large enough training dataset? do we want to?

Evaluating the Performance of our Neural Nets as Theorem Provers

- in fact, our `seq2seq` LSTM recurrent neural network trained on encodings of theorems and their proof-terms performs unusually well
- the experiments with training the neural networks using the IPILL and IIPC theorem dataset are available at: `https://github.com/ptarau/neuralgs`
- the $< formula, \quad proof \ \ term >$ generators are available at: `https://github.com/ptarau/TypesAndProofs`
- the generated datasets are available at: `http://www.cse.unt.edu/~tarau/datasets/`

Conclusions

Yeah, this was too fast for me too!

But it works and it's based on Mints' and Zeiberger' results for Intuitionistic Logic, which are simplified in the Linear implicational case.

Future work is to see if we can extend it to Linear-non-Linear Logic. To see if we can learn many more theorems.

THANKS!