Training Data Extraction for Identifying Useful Lemmas

Michael Rawson [1] Christoph Wernhard [2] Zsolt Zombori [3]

[1]TU Wien [2]University of Potsdam [3]Alfréd Rényi Institute of Mathematics

PAMLTP/DG4D^3

Prague, Czech Republic, April 18–20, 2023

Training Data Extraction for Identifying Useful Lemmas

1. Iterative Improvement via Learned Lemma Selection
2. Learning from Successful as well as Failed Proof Attempts
3. Proofs as Terms: Condensed Detachment
4. Learning Subtree/Unit Lemmas
5. Towards more Powerful Lemmas
6. Conclusion

Training Data Extraction for Identifying Useful Lemmas

Lemmas to Aid Proof Search

- Lemmas can make the proof shorter
- Lemmas can make selecting the next inference harder
- Ideally, we would like to identify just a few relevant lemmas
- Similar to premise selection, but we assume no given premise set
  1. Generate
  2. Filter
  3. Apply

Lemma Generation via Structure Enumeration

- Focus on the structural representation of proofs (tree, DAG etc.)
- Enumerate proof structures
- Avoid duplicates due to different derivations
- Use some proof structure measure to limit the enumeration (tree size, tree height etc.)
- Different measures result in very different lemma sets

Lemma Selection

- Use a trained neural model to filter candidate lemmas
- Model Interface
  - Input: Problem (Conjecture + Axioms), Lemma
  - Output: Utility score $u \in [0, 1]$
- Input Features
  - Expert engineered features (e.g. tree size, compacted tree size)
  - Graph neural network processing formulas and proof terms directly
- Utility score
  - Inference step reduction when lemma is added
  - Whether lemma is present in the proof found
  - ...

Lemma Application

- Can be added as axioms
  - Suitable for any prover, regardless of the calculus
  - If a full proof is needed, the lemma proofs need to be inserted into the prover’s result
- Can have a special treatment
  - Lemmas as macros: replace with proof term
  - Replace inner lemma search/enumeration with accessing input lemmas

Iterative Improvement

- Start from a set of problems
- Search from proofs
- Learn from proof attempts
- Fit a model
- Start search again, using the learned model

Improving other Provers

- Lemma generation and selection produces “promising” lemmas
- Can be used by any other prover, regardless of the calculus
- Cannot be iterated

Learning from Successful Proof Attempts

- Utility measure calculation requires a prover that can produce a proof tree structure
- Given a proof, any substructure can be considered as a lemma that we can learn from
- Lots of training signal from a single proof
- Different proofs of the same problem provide more signal, without (too much) inconsistency

Learning from Failed Proof Attempts

- Any proof attempt constructs a sequence of incomplete proof structures
- Most of these have complete substructures
- These are proof terms of formulas proven as a byproduct of proof search
- We can use any such substructures as a proof to learn from
- Similar to Hindsight Experience Replay [Andrychowicz et al., 2017]
  - Pretend that we wanted to prove what we accidentally proved
- Provides huge amounts of training data from failed proofs

Condensed Detachment (CD) – Background: Substitution and Detachment

- Investigation of axiomatizations of propositional calculi with **substitution and detachment** (modus ponens)
- Jan Łukasiewicz (1878 Lviv – 1956 Dublin)
- For example: *Łukasiewicz* $\vdash$ *Simp*, *Peirce*, *Syll*, where

Łukasiewicz = $CCCpqrCCrpCsp$
$\forall pqrs\, \mathsf{P}(((p \to q) \to r) \to ((r \to p) \to (s \to p)))$
$\mathsf{P}(\mathsf{i}(\mathsf{i}(\mathsf{i}(x, y), z), \mathsf{i}(\mathsf{i}(z, x), \mathsf{i}(u, x))))$

Simp = $CpCqp$
$\forall pq\, \mathsf{P}(p \to (q \to p))$
$\mathsf{P}(\mathsf{i}(x, \mathsf{i}(y, x)))$

Peirce = $CCCpqpp$
$\forall pq\, \mathsf{P}(((p \to q) \to p) \to p)$
$\mathsf{P}(\mathsf{i}(\mathsf{i}(\mathsf{i}(x, y), x), x))$

Syll = $CCpqCCqrCpr$
$\forall pqr\, \mathsf{P}((p \to q) \to ((q \to r) \to (p \to r)))$
$\mathsf{P}(\mathsf{i}(\mathsf{i}(x, y), \mathsf{i}(\mathsf{i}(y, z), \mathsf{i}(x, z))))$

THE SHORTEST AXIOM OF THE IMPLICATIONAL CALCULUS OF PROPOSITIONS.

By JAN ŁUKASIEWICZ.

[Read 23 June, 1947. Published 6 April, 1948.]

1 CCCpqrCCrpCsp.
1 p/Cpq, q/r, r/CCrpCsp, s/r + C 1 – 2.
2 CCCCrpCspCpqCrCpq.
1 p/CCrpCsp, q/Cpq, r/CrCpq, s/t + C 2 – 3.
3 CCCrCpqCCrpCspCtCCrpCsp.
3 r/Cpq, t/1 + C 1 r/Cpq – C 1 – 4.
4 CCCpqpCsp.
1 p/Cpq, q/p, r/Csp, s/r + C 4 – 5.
5 CCCspCpqCrCpq.
1 p/Csp, q/Cpq, r/CrCpq, s/t + C 5 – 6.
6 CCCrCpqCspCtCsp.
1 p/CrCpq, q/Csp, r/CtCsp, s/u + C 6 – 7.
7 CCCtCspCrCpqCuCrCpq.
7 t/Cpq, p/q, r/CCsqp, q/p, u/1 + C 1 r/Csq, s/q – C 1 – 8.
8 CCCsqpCqp.
8 s/Cpq, q/r, p/CCrpCsp + C 1 – 9.
9 CrCCrpCsp.
1 p/r, r/CCCrqpCsp, s/t + C 9 r/Crq – 10.
10 CCCCCrqpCsprCtr.
1 p/CCCrqpCsp, q/r, r/Ctr, s/u + C 10 – 11.

27 CpCqp.
25 q/p, r/q + C 22 – 28.
28 CCCpqpp.
21 p/Cpq, r/CCqrCpr, q/CCCprqq + C 26 – C 21 – 29.
29 CCpqCCqrCpr.

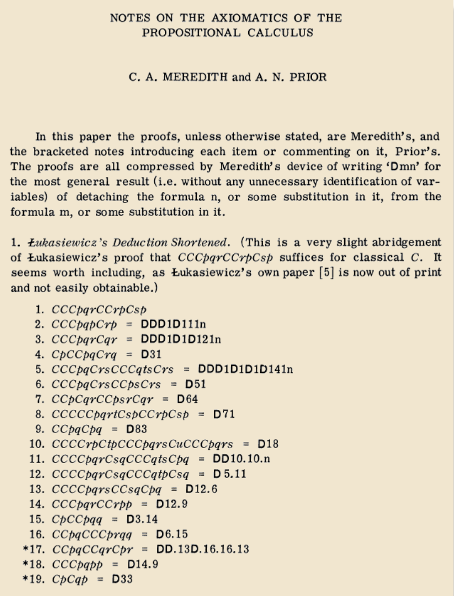
Condensed Detachment (CD)

- Carew Arthur Meredith (1904 Dublin – 1976 Dublin)
- Condensed detachment (mid 1950s)
  - **Unification**
  - **Proof terms** (D-terms, full binary trees)

$$\frac{d_1 : \mathsf{P}(x \rightarrow y) \qquad d_2 : \mathsf{P}(x')}{\mathsf{D}(d_1, d_2) : \mathsf{P}(y)\mathit{mgu}(x, x')}$$

$$\frac{}{1 : \mathsf{P}(t)\mathit{fresh\text{-}copy}} \quad \text{for the axiom } \mathsf{P}(t)$$

- A D-term may have (wrt given axioms) a unique **most general theorem (MGT)**, the formula proven at its root
  - Not all D-terms may have an MGT (unification may fail)
  - Different D-terms may have subsuming MGTs



NOTES ON THE AXIOMATICS OF THE PROPOSITIONAL CALCULUS

C. A. MEREDITH and A. N. PRIOR

In this paper the proofs, unless otherwise stated, are Meredith's, and the bracketed notes introducing each item or commenting on it, Prior's. The proofs are all compressed by Meredith's device of writing 'Dmn' for the most general result (i.e. without any unnecessary identification of variables) of detaching the formula n, or some substitution in it, from the formula m, or some substitution in it.

1. *Łukasiewicz's Deduction Shortened.* (This is a very slight abridgement of Łukasiewicz's proof that *CCCpqrCCrpCsp* suffices for classical *C*. It seems worth including, as Łukasiewicz's own paper [5] is now out of print and not easily obtainable.)

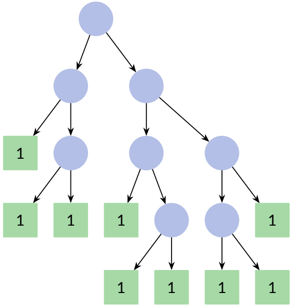
1. *CCCpqrCCrpCsp*
2. *CCCpqpCrp* = DDD1D111n
3. *CCCpqrCqr* = DDD1D1D121n
4. *CpCCpqCrq* = D31
5. *CCCpqCrsCCCqtsCrs* = DDD1D1D1D141n
6. *CCCpqCrsCCpsCrs* = D51
7. *CCpCqrCCpsrCqr* = D64
8. *CCCCCpqrtCspCCrpCsp* = D71
9. *CCpqCpq* = D83
10. *CCCCrpCtpCCCpqrsCuCCCpqrs* = D18
11. *CCCCpqrCsqCCCqtsCpq* = DD10.10.n
12. *CCCCpqrCsqCCCqtpCsq* = D 5.11
13. *CCCCpqrsCCsqCpq* = D12.6
14. *CCCpqrCCrpp* = D12.9
15. *CpCCpqq* = D3.14
16. *CCpqCCCprqq* = D6.15

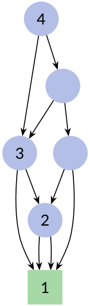
*17. *CCpqCCqrCpr* = DD.13D.16.16.13

*18. *CCCpqpp* = D14.9

*19. *CpCqp* = D33

- **Tree size**: 8
- **Height**: 4
- **Compacted size**: 5 – size of minimal DAG; number of distinct compound subterms





Term representation

$\mathsf{D}(\mathsf{D}(1, \mathsf{D}(1, 1)), \mathsf{D}(1, \mathsf{D}(\mathsf{D}(1, 1)), \mathsf{D}(\mathsf{D}(1, 1), 1)))$

Term representation by factor equations

$$\begin{aligned} 2 &= \mathsf{D}(1, 1) \\ 3 &= \mathsf{D}(1, 2) \\ 4 &= \mathsf{D}(3, \mathsf{D}(3, \mathsf{D}(2, 1))) \end{aligned}$$

n		0	1	2	3	4	5	6
Tree size	OEIS:A000108	1	1	2	5	14	42	132
Height	OEIS:A001699	1	1	3	21	651	457,653	210,065,930,571
Compacted size	OEIS:A254789	1	1	3	15	111	1,119	14,487

Proof Terms, Formulas and Levels: An Overall Picture

Level T	H	C	F	:	Proof	:	Formula
0	0	0	6	:	1	:	*CCCpqrCCrpCsp*
1	1	1	8	:	D11	:	*CCCCpqCrqCqsCtCqs*
2	2	2	11	:	D1D11	:	*CCCpCqrCCsqCtqCuCCsqCtq*
3	3	3	11	:	D1D1D11	:	*CCCpCCqrCsrCtCruCvCtCru*
3	2	2	8	:	DD11D11	:	*CpCCqrCrCqr*
3	3	3	11	:	DD1D111	:	*CpCCCqrqCsq*
4	4	4	14	:	D1D1D1D11	:	*CCCpCqCrsCtCCurCvrCwCtCCurCvr*
4	3	3	8	:	D1DD11D11	:	*CCCCpqCqCpqrCsr*
4	4	4	11	:	D1DD1D111	:	*CCCCCpqpCrpsCts*
4	3	3	11	:	DD11D1D11	:	*CpCCqqCCrqCqq*
4	3	3	11	:	DD1D11D11	:	*CpCCqrCrr*
4	4	4	11	:	DD1D1D111	:	*CpCCCCqrCsrtCrt*
4	3	3	8	:	DDD11D111	:	*CCpqCqCpq*
4	4	4	11	:	DDD1D1111	:	*CCCpqpCrp*
5	5	5	14	:	D1D1D1D1D11	:	*CCCpCqCCrsCtsCuCvCswCxCuCvCsw*
5	4	4	12	:	D1D1DD11D11	:	*CCCpqCCrsCsCrsCtCCrsCsCrs*
5	5	5	12	:	D1D1DD1D111	:	*CCCpqCCCrsrCtrCuCCCrsrCtr*
5	4	4	11	:	D1DD11D1D11	:	*CCCCppCCqpCpprCsr*
5	4	4	11	:	D1DD1D11D11	:	*CCCCpqCqqrCsr*
5	5	5	11	:	D1DD1D1D111	:	*CCCCCCpqCrqsCqstCut*
5	4	4	8	:	D1DDD11D111	:	*CCCpCqpqCrq*
5	5	5	11	:	D1DDD1D1111	:	*CCCpqCqrCsCqr*
5	3	3	8	:	DD11DD11D11	:	*CpCCqrCrCqr*
5	4	4	11	:	DD11DD1D111	:	*CpCCCqrqCsq*

⋮

T Tree size of proof
H Height of proof
C Compacted size of proof
F Max size of MGT of subproof

CD in ATP, Unit Lemmas

- CD problems as first-order ATP problems

Detachment axiom	$\mathsf{P}(\mathsf{i}(x,y)) \land \mathsf{P}(x) \rightarrow \mathsf{P}(y)$	
Proper axioms	units	e.g. $\mathsf{P}(\mathsf{i}(\mathsf{i}(\mathsf{i}(x,y),z),\mathsf{i}(\mathsf{i}(z,x),\mathsf{i}(u,x))))$
Goal	negative ground unit	e.g. $\neg\mathsf{P}(\mathsf{i}(\mathsf{a},\mathsf{i}(\mathsf{b},\mathsf{a})))$

  - Horn, first-order variables, binary function symbol, cyclic predicate dependency
  - Generalization to arbitrary Horn problems is possible
- CD as inference rule can be translated to **hyperresolution with the detachment axiom**
  - The proving method then involves **unit lemmas** whose proof is a D-term
- Many ideas around OTTER were originally developed for CD problems (hints, hot list, resonance strategy)
  - JAR 2001 special issue on CD (vol 27, no 2)
  - Around 200 CD problems in *TPTP*'s LCL domain, some still very hard
- CD was recently modeled with concepts from the connection method [CW and Bibel 2021]
  - **Proof structure + a global unifying substitution of connected formulas**
  - Allows to propagate a **ground goal** through unification
  - Proof search by **enumerating proof structures** interwoven with unification as in clausal tableaux
  - **Focus on proof structure as a whole**, in contrast to an inference rule
  - Unit lemmas correspond to re-use of subtrees; interplay of D-terms as **trees and their minimal DAGs**

Learning Requirements, Considered Provers

- Lemma generation requires proof structure enumeration (SGCD)
- We require provers that emit proofs as D-terms (SGCD, Prover9, CMProver, CCS)
- Any prover can be used for evaluation

	SGCD	Prover9	CMProver	leanCoP	CCS-Vanilla	Vampire	E
Goal-driven	●/–	–	●	●	●	○	○
CM-CT	○	–	●	●	–	–	–
Proof Structure Enumeration	●	–	●	○	●	–	–
Resolution / Superposition	–	●	–	–	–	●	●
Output proof as D-term	●	●	●	–	●	–	–
Input lemmas that replace search	●	–	–	–	●	–	–

- Comprehensive result table (link)

Learning Unit Lemmas

- Any full D-Term, i.e., one whose leaves are axioms represents a unit lemma along with its proof
- When a proof is found, its full D-term contains lots of full subtrees, i.e., unit lemmas
- A failed proof attempt yields partial D-terms, but they likely still have full subtrees
- So far we mostly focused on unit lemmas

- Assume a Prolog predicate that **enumerates proof-MGT pairs for a given level**

$$\texttt{enum\_dterm\_mgt\_pairs(+}\mathit{Level}\texttt{, ?}\mathit{DTerm}\texttt{, ?}\mathit{Formula}\texttt{)}$$

- Level characterizations can be e.g. tree size or height of the D-term
- Depending on the parameter instantiation the predicate serves different purposes

$+Dterm$	$+Formula$	verifying a proof
$+Dterm$	$-Formula$	computing the MGT
$-Dterm$	$+Formula$	proving a formula (goal-driven)
$-Dterm$	$-Formula$	generating lemmas (axiom-driven)

- Its implementation can access a $Cache$ of solutions in lower levels
- SGCD embeds it in **nested loops of goal- and axiom-driven phases**
- The $Cache$ can be **heuristically restricted on the basis of MGTs**

```
Cache := ∅;
for l := 0 to maxLevel do
    for m := l to l + preAddMaxLevel do
        enum_dterm_mgt_pairs(m, d, goal);
        throw proof_found(d)
    N := {⟨l, d, f⟩ | enum_dterm_mgt_pairs(l, d, f)};
    if N = ∅ then throw exhausted;
    Cache := merge_news_into_cache(N, Cache)
```

Performance of different provers over 2 iterations of training a linear model

	SGCD				Prover9				CMProver				CCS-Vanilla			
Time	50s	100s	500s	30m	50s	100s	500s	30m	50s	100s	500s	30m	50s	100s	500s	30m
Base	266	275	285	285	240	252	259	262	82	85	94	103	81	88	99	105
Iter 1	280	282	284	281	250	254	262	257	83	93	105	121	96	101	117	130
Iter 2	281	283	281	283	247	247	267	265	79	98	95	126	96	97	120	128
Total	282	284	286	286	253	258	269	267	91	105	112	141	106	105	133	145

Number of problems solved without and with additional lemmas using various time limits

	Vampire				E				Prover9				leanCoP			
Time	50s	100s	500s	30m	50s	100s	500s	30m	50s	100s	500s	30m	50s	100s	500s	30m
Base	221	224	252	263	253	264	275	281	236	244	257	260	70	71	77	77
Lemmas	249	257	274	283	256	266	275	275	246	250	261	269	100	103	111	113
Total	249	257	276	284	269	276	287	286	248	252	264	269	100	103	111	113

Detailed Result Table (Link)

Further Experimental Results on 312 CD Problems

Problems solved by Vampire and SGCD as we alter the number of extracted lemmas (time limit 100s)

	Vampire						SGCD					
Lemma count	10	25	50	100	200	500	10	25	50	100	200	500
Base	227	227	227	227	227	227	275	275	275	275	275	275
Lemmas	226	242	246	258	257	258	278	285	284	281	283	284
Total	231	243	247	258	257	258	282	285	284	283	284	285

PSP-Level (Proof-SubProof-Level)

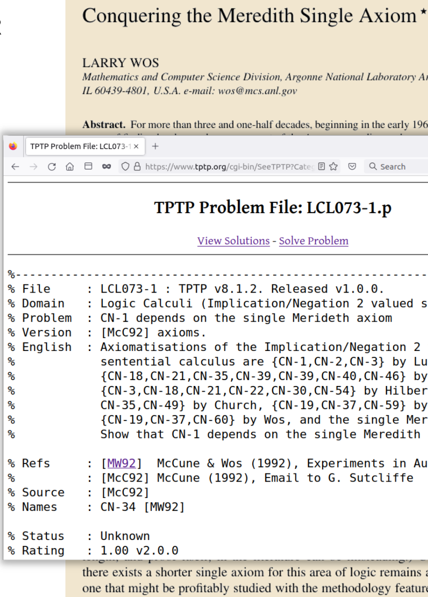
- Recall that SGCD enumerates and caches structures by “level”, e.g. tree size or height
- A principle **observed** in many steps of a proof by Łukasiewicz and a variation by Meredith [CW and Bibel 2021] can be turned into a further level characterization

> Structures in **PSP-level** $n + 1$ are the D-terms where
> - one argument term is at PSP-level n
> - and the other argument is a subterm of that term

- Enumeration by PSP-level
  - is incomplete (some D-terms are omitted)
  - has features of DAG enumeration (D-terms in PSP-level n have compacted size n)
  - is suitable for SGCD’s simple caching
- Applications of enumeration by PSP-level
  - A proof of Łukasiewicz’s problem that is shorter then the original human proofs (and drastically shorter than known ATP proofs)
  - For many CD problems it leads to proofs with small compacted size
  - Very useful for input lemma generation for other provers
  - Key technique to solve a truly hard problem

- Proven in ATP only by Wos in 2000 with several invocations of OTTER
- Proven now with SGCD and replacing lemmas
  - 98,198 lemmas generated by SGCD for PSP-level, cache limit 5,000, termination by exhaustion (60 s)
  - Ordered heuristically according to 5 general features (190 s)
  - The best 2,900 are supplied as replacing input lemmas to SGCD
  - SGCD called for proving: axiom-driven by PSP-level, goal-driven by height (preAddMaxLevel=0), cache limit 1,500, general heuristic restrictions (20 s)
  - The structure of the proof reflects PSP-level plus one height step

	Here	Wos	Meredith
Compacted size	46	74	40
Tree size	3,276	9,207	6,172
Height	40	48	30
Double negation	•	–	•
Max size of MGT of subproof	19	18	18



Training Data Extraction for Identifying Useful Lemmas

- **D-term with variables**
  $\mathsf{D}(1, \mathsf{D}(1, x))$

- **D-term with combinators**
  Obtained from a D-term with variables with standard techniques
  $\lambda x.\mathsf{D}(1, \mathsf{D}(1, x)) = \mathsf{D}(\mathsf{D}(\mathbf{B}, 1), 1)$
  (Recall that $\mathbf{B} \stackrel{\text{def}}{=} \lambda xyz.\mathsf{D}(x, \mathsf{D}(y, z))$)

- **Horn clause**
  Obtained from the D-term with variables like the MGT, but
  each variable becomes a body atom, with the formula substitution of its position applied
  Equivalently obtained by resolution from the Detachment clause and the proper axioms
  For $\mathsf{D}(1, \mathsf{D}(1, x))$ and axiom $1 : \mathsf{P}(\mathsf{i}(\mathsf{i}(x, y), \mathsf{i}(\mathsf{i}(y, z), \mathsf{i}(x, z))))$ we obtain
  $\mathsf{P}(\mathsf{i}(\mathsf{i}(\mathsf{i}(x, y), z), \mathsf{i}(\mathsf{i}(u, y), z))) \leftarrow \mathsf{P}(\mathsf{i}(x, u))$

- Tree grammars with variables in nonterminals applied to proof structures

- Connection structures [Eder 1989] for Horn problems

The Combinator View Maps the More Powerful Sharing to Shared Subtrees

- Recall that $\mathbf{B} \overset{\text{def}}{=} \lambda xyz.\mathsf{D}(x, \mathsf{D}(y, z))$ and $\lambda x.\mathsf{D}(1, \mathsf{D}(1, x)) = \mathsf{D}(\mathsf{D}(\mathbf{B}, 1), 1)$
- Consider

$$\mathsf{D}(\mathsf{D}(\mathsf{D}(\mathbf{B}, 1), 1), \mathsf{D}(\mathsf{D}(\mathsf{D}(\mathbf{B}, 1), 1), 1))$$

- As list of factor equations it is

$$\begin{array}{rcl} 2 & = & \mathsf{D}(\mathsf{D}(\mathbf{B}, 1), 1) \\ 3 & = & \mathsf{D}(2, \mathsf{D}(2, 1)) \end{array}$$

- It normalizes to the following plain D-term, which has no multiply occurring subterm

$$\mathsf{D}(1, \mathsf{D}(1, \mathsf{D}(1, \mathsf{D}(1, 1))))$$

- This is utilized in CCS for **proof search**
  - Enumerated proof terms may involve combinators
  - Enumeration is by increasing DAG size, to benefit from the compression through the combinators
  - Can be operated goal-driven, i.e. with goal instantiation like SGCD and CM-CT provers
  - Refinement: only allow structures built from specified proof structure templates whose semantics is given by D-terms with combinators (or D-terms with λ-bound variables)
  - Structure templates can simulate resolution variants, optionally in goal-driven refinements

Possibilities of Applying The Lemmas

- Converting to CCS proof structure templates
  - Variants of resolution with restrictions (e.g. by clause length)
  - Optionally goal-driven
- Adding as Horn input lemmas
  - For CM-CT provers: adding “a bit of resolution”
  - For resolution/superposition provers: adding specific resolvents

Re-using Proofs to Generate Training Data

- Difficulty: **Translation of proofs to D-terms, or D-terms with combinators**, which represent plain D-terms in a standardized compressed form. This should work for binary resolution proofs of Horn problems

Possibilities of Generating Training Data and Input Lemmas

- **Note: There is no unique “maximally compressed” structure like the minimal DAG**
- Enumerating lemmas up to a given size
- Selecting from a table of a few thousand lemmas detected previously
- Applying grammar-based tree compression: *TreeRePair* [Lohrey, Maneth, Mennicke 2013]

More Powerful Lemmas: Estimated Effect in Practice

- Much can be done already with just unit lemmas, but there might be some problems where more powerful lemmas are necessary
- In our experiments with 312 problems 2 of 296 solved problems could be proven quickly with Vampire and E but not with any of the provers that yield D-terms
- Moderate success is suggested by experimental data on proof search with CCS for *TPTP* Horn problems and compression of given proofs for CD problems

Training Data Extraction for Identifying Useful Lemmas

1. Iterative Improvement via Learned Lemma Selection
2. Learning from Successful as well as Failed Proof Attempts
3. Proofs as Terms: Condensed Detachment
4. Learning Subtree/Unit Lemmas
5. Towards more Powerful Lemmas
6. **Conclusion**

Conclusion

- Lemmas are helpful to find a proof
- Generate, filter, apply lemmas
- Lemma generation brings a bit of resolution into non-resolution based provers
- Blurs the distinction between forward and backward reasoning

References I

[Andrychowicz et al., 2017] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., and Zaremba, W. (2017).
Hindsight experience replay.
In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[Lohrey et al., 2013] Lohrey, M., Maneth, S., and Mennicke, R. (2013).
XML tree structure compression using RePair.
Inf. Syst., 38(8):1150–1167.
System available from `https://github.com/dc0d32/TreeRePair`, accessed Jun 30, 2022.

[Łukasiewicz, 1948] Łukasiewicz, J. (1948).
The shortest axiom of the implicational calculus of propositions.
In *Proc. of the Royal Irish Academy*, volume 52, Sect. A, No. 3, pages 25–33.
Republished in [Łukasiewicz, 1970], p. 295–305.

[Łukasiewicz, 1970] Łukasiewicz, J. (1970).
Selected Works.
North Holland.
Edited by L. Borkowski.

[Meredith and Prior, 1963] Meredith, C. A. and Prior, A. N. (1963).
Notes on the axiomatics of the propositional calculus.
Notre Dame J. of Formal Logic, 4(3):171–187.

[Rawson et al., 2023] Rawson, M., Wernhard, C., Zombori, Z., and Bibel, W. (2023).
Lemmas: Generation, selection, application.
CoRR, abs/2303.05854.
Submitted, preprint: `https://arxiv.org/abs/2303.05854`.

[Ulrich, 2001] Ulrich, D. (2001).
A legacy recalled and a tradition continued.
J. Autom. Reasoning, 27(2):97–122.

[Wernhard, 2022a] Wernhard, C. (2022a).
CD Tools – Condensed detachment and structure generating theorem proving (system description).
`https://arxiv.org/abs/2207.08453`.

[Wernhard, 2022b] Wernhard, C. (2022b).
Generating compressed combinatory proof structures – an approach to automated first-order theorem proving.
In Konev, B., Schon, C., and Steen, A., editors, *PAAR 2022*, volume 3201 of *CEUR Workshop Proc.* CEUR-WS.org.
Preprint: `https://arxiv.org/abs/2209.12592`.

[Wernhard and Bibel, 2021] Wernhard, C. and Bibel, W. (2021).
Learning from Łukasiewicz and Meredith: Investigations into proof structures.
In Platzer, A. and Sutcliffe, G., editors, *CADE 28*, volume 12699 of *LNCS (LNAI)*, pages 58–75. Springer.

[Wernhard and Bibel, 2023] Wernhard, C. and Bibel, W. (2023).
Investigations into proof structures.
Preprint, `http://cs.christophwernhard.com/papers/investigations/`.

[Wos, 2001] Wos, L. (2001).
Conquering the Meredith single axiom.
J. Autom. Reasoning, 27(2):175–199.

Problem

goal : *FormulaTerm*
axiom(*NatNum*) : *FormulaTerm*
number_of_axioms : *NatNum*

Proof

problem : problem
source : *Term*
meta_info : *KeyValueList*
dcterm : *DCTerm*
d_csize : *NatNum*
d_tsize : *NatNum*
d_height : *NatNum*

Lemma

problem : problem
lf_proof : proof
lf_is_in_proof : NatNum
formula : lemma(*Head*,*Body*)
dcterm : *DCTerm*
method : *Term*
lf_d_csize : *NatNum*
lf_d_tsize : *NatNum*
lf_d_height : *NatNum*
lf_d_grd_csize : *NatNum*
lf_d_major_minor_relation : *NatNum*
lf_d_number_of_terminals : *NatNum*
lfp_containing_proof : proof
lfp_d_occs : *NatNum*
lfp_d_incoming : *NatNum*
lfp_d_occs_innermost_matches : *NatNum*
lfp_d_occs_outermost_matches : *NatNum*
lfp_d_min_goal_dist : *NatNum* lf_b_length : *NatNum*
lf_hb_distinct_hb_shared_vars : *NatNum*
lf_hb_distinct_h_only_vars : *NatNum*
lf_hb_distinct_b_only_vars : *NatNum*
lf_hb_singletons : *NatNum*
lf_hb_double_negation_occs : *NatNum*
lf_hb_nongoal_symbol_occs : *NatNum*
lf_h_excluded_goal_subterms : *NatNum*
lf_h_subterms_not_in_goal : *NatNum*
lf_hb_compression_ratio_raw_deflate : *NormalizedValue*
lf_hb_compression_ratio_treerepair : *NormalizedValue*
lf_hb_compression_ratio_dag : *NormalizedValue*
lf_hb_organic : *NatNum*
lf_hb_name : *Atom*
lf_hb_name_status : *NatNum*
lf_*COMP*_csize : *NatNum*
lf_*COMP*_tsize : *NatNum*
lf_*COMP*_height : *NatNum*
lf_*COMP*_distinct_vars : *NatNum*
lf_*COMP*_*ITEM*_occs : *NatNum*
lf_*COMP*_occs_of_most_frequent_*ITEM* : *NatNum*

Considered Utility Values

`u_tsize_reduction` : *NormalizedValue*
`u_height_reduction` : *NormalizedValue*
`u_csize_reduction` : *NormalizedValue*
`u_tsize_reduction_subst1` : *NormalizedValue*
`u_height_reduction_subst1` : *NormalizedValue*
`u_csize_reduction_subst1` : *NormalizedValue*
`u_tsize_reduction_subst2` : *NormalizedValue*
`u_height_reduction_subst2` : *NormalizedValue*
`u_csize_reduction_subst2` : *NormalizedValue*
`u_occs` : *NormalizedValue*
`u_incoming` : *NormalizedValue*
`u_close_to_goal_path` : *NormalizedValue*
`u_close_to_axioms_height` : *NormalizedValue*
`u_close_to_axioms_tsize` : *NormalizedValue*
`u_reproof` : *Float*