EasyChair Preprint

№ 9423

Linear Refutation and Clause Splitting

Michael Rawson

December 5, 2022

Linear Refutation and Clause Splitting

Michael Rawson[0000−0001−7834−1567]

TU Wien, Austria
michael@rawsons.uk

Abstract. Linear resolution is one of the ancient methods for first-order theorem proving. We extend linear resolution with clause splitting, producing subgoals dispatched independently. An incremental SAT solver keeps track of refutations and thus provides a "lemma" mechanism. We describe some implementation considerations, present some initial experimental results, and discuss future directions for this approach.

Keywords: clause splitting · linear resolution · boolean satisfiability

1 Introduction

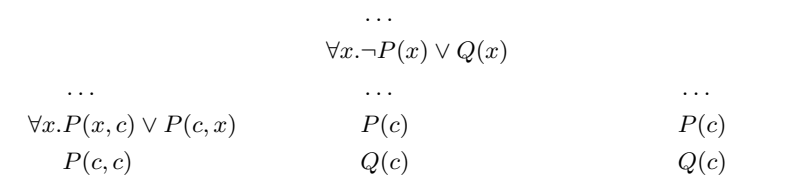
Goal-sensitive, backtracking calculi such as connection tableaux have some advantages over the dominant saturation paradigm. They are relatively simple to implement, do not struggle with large axiom sets, and use little memory. They also accommodate learned guidance very naturally [16,19,50,32], a growing consideration for theorem provers. Linear resolution is an ancestor of many of these backtracking calculi, and it is arguably even simpler. We investigate applying clause splitting to linear resolution, achieving surprisingly strong performance with a number of advantages including a lemma mechanism, universal variables, and the ability to generate useful training data even when no proof is found.

2 Preliminaries

We assume the fully-automatic, clausal, refutational setting for proof search in classical first-order logic. Given a set of first-order axioms and a conjecture, we first negate the conjecture and then refute the resulting set of facts. This is done by converting to clause normal form (CNF) and deriving the empty clause $\bot$. Clauses are the usual disjunctions of literals, written $\forall \bar{x}.\ C$, where $\bar{x}$ is a sequence of variables given in order of occurrence in the sequence of literals C, left-to-right.

2.1 Linear Resolution

Linear resolution [27] is an early (1968) refinement of Robinson's resolution procedure [37] for refuting a set of first-order clauses. It is a sound and complete proof calculus, simple and yet very powerful. Many other techniques can be seen



(a) factoring (b) ancestor resolution (c) input resolution

Fig. 1: Adding centre clauses to a linear derivation. Derivations are written top-down. In the case of input resolution, $\forall x.\neg P(x) \vee Q(x)$ is an input clause.

as instances or refinements [26] of linear resolution, including model elimination [25] and SLD-resolution [2].

Linear resolution operates on a sequence of *centre clauses* C_i, called a *linear derivation*. The first clause, C_0, is a member of the input set of clauses. Any subsequent clause C_{i+1} is derived from its parent C_i by one of (i) *factoring* C_i; (ii) *ancestor resolution* between C_i and a previous centre clause C_j; or (iii) *input resolution* between C_i and an input clause. Figure 1 shows an example for each of these cases. Note that any unification is applied only to the newly-added clause, not the whole derivation (cf. free-variable tableaux, model elimination). The last centre clause C_i is called the *goal*. A linear derivation that ends in $\bot$ is a *linear refutation* and shows the unsatisfiability of the input set.

Linear resolution is not proof-confluent, so alternative centre clauses must be backtracked over to retain completeness. To improve efficiency, it is possible to demand that C_0 be a member of the *set of support*[1]; that no centre clause occurs twice in the derivation; and that only some literal of the goal need take part in inferences. We implement only these refinements, but there are many others, such as the no-tautologies condition, literal selection and various forms of subsumption [27,22].

2.2 Clause Splitting

During proof search in saturation-style systems [46], clauses may grow very long, which is generally considered to be undesirable. One effective approach to solving this problem is to *split* clauses into variable-disjoint subclauses. For example, if we obtain a clause $\forall \bar{x}\bar{y}.\ C[\bar{x}] \vee D[\bar{y}]$, we can deduce $(\forall \bar{x}.\ C[\bar{x}]) \vee (\forall \bar{y}.\ D[\bar{y}])$ by manipulating quantifiers. Splitting can proceed as far as possible until no more sub-clauses can be further split: the so-called *maximal* splitting. Clauses that cannot be split any further are called *non-splittable* clauses, or *components* [45].

The question for saturation systems is then how to partition and subsequently organise the resulting search. This has produced several approaches [47,35], of which arguably the most successful is AVATAR [45].

[1] a profitable example of the support set is "clauses derived from the conjecture" [49,33]

2.3 Boolean Satisfiability

Boolean satisfiability (SAT) is a well-studied problem [5], which we gloss as "given a propositional formula, is there an assignment of propositional variables such that the formula is satisfied?". While as-stated the problem statement is only a decision procedure (and an NP-complete one at that), in practice SAT solvers are highly efficient, can solve *incrementally*, and provide auxiliary information such as a satisfying assignment.

SAT solvers may be embedded into other software, which is exploited to great effect in theorem proving. Good examples include SMT solving [14], instantiation-based methods and global subsumption [20], finite model-finding [13], among many others. We especially highlight AVATAR [45], a technique for organising clause splitting via SAT (or SMT [6]) implemented in the saturation-style system Vampire [21].

3 Linear Resolution with Clause Splitting

$\forall \bar{x}_0.\ C_0$	$\forall \bar{x}_0.\ C_0$	$\forall \bar{x}_0.\ C_0$
$\forall \bar{x}_1.\ C_1$	$\forall \bar{x}_1.\ C_1$	$\forall \bar{x}_1.\ C_1$
$\dots$	$\dots$	$\dots$
$\forall \bar{y}_i \bar{z}_i.\ (D_i \vee E_i)$	$\forall \bar{y}_i.\ D_i$	$\forall \bar{z}_i.\ E_i$
(a) no splitting	(b) splitting, 1st goal	(c) splitting, 2nd goal

Fig. 2: Linear derivations showing the effect of clause splitting. The new goal splits into two components, so 2b and 2c can be solved independently.

We propose augmenting linear resolution with clause splitting. When adding a centre clause C_{i+1} to a linear derivation as above, it may be that it can be split into n components. At this point we can produce n linear derivations, each the same up to C_i, but each with a different component as the new goal, in place of C_{i+1}. We call these *sub-derivations* and call them *solved* if we can produce a linear refutation from them. If we can solve each of these n derivations separately, then we have a refutation for the original derivation. Figure 2 shows an abstract example.

In this work we require all goal clauses to be immediately and maximally split, as we see no clear benefit to optional or non-maximal splitting with respect to proof search. Note that this scheme produces sub-derivations in which centre clauses are always components.

3.1 Motivation

This technique provides a kind of subgoal mechanism, in the same way as clause splitting applied in a saturation context. However, there is almost no bookkeeping required due to the backtracking nature of linear resolution. Smaller clauses are also desirable for linear resolution due to the reduced number of possible factors and ancestor resolvents, and in a heuristic sense they are closer to a refutation. Subgoals may recur during backtracking search, a feature we exploit in §4 to reduce the amount of search required. Opportunities for splitting can occur surprisingly frequently, and almost all problems tested trigger some amount of splitting.

3.2 Soundness and Completeness

Clearly this technique is sound, by a similar argument to clause splitting in other contexts. Completeness is slightly more difficult due to a technicality involving factoring which we will deal with in a moment. However, we believe linear resolution with splitting is in fact complete, and we sketch a proof to that effect below.

Proof sketch. If linear resolution with splitting is complete, then we should be able to translate any linear refutation of an input set into another refutation of the same input set, this time with splitting. Suppose that we have such a linear refutation. If at any point i we have a splittable goal C_i, then it should be immediately split into components. Suppose further and without loss of generality that C_i yields two components, D_i and E_i. Then, the linear refutation of C_i at each step $j \geq i$ performs an inference using a literal from either the component D_j or E_j. This step will leave the other component unchanged for the next step because component variables are disjoint by assumption. Therefore, we generate two sub-refutations from the original by partitioning steps into two: those that affect D_j, and those that affect E_j. Inductively, there is a translation from linear refutations to linear refutations with splitting.

A snag. Tacit in the above sketch is that an inference cannot take place between two components in the same clause, and that components remain disjoint once split. Unfortunately, in a linear derivation, factoring may be carried out between two literals belonging to different components, which breaks both of these assumptions. Suppose that we have components within a single clause, $D[\bar{x}] \vee L[\bar{x}]$ and $E[\bar{y}] \vee L'[\bar{y}]$. Without splitting, we might factor L and L' to produce $D'[\bar{z}] \vee E'[\bar{z}] \vee L''[\bar{z}]$. With splitting, this inference would not be possible.

A fix. However, the above case is the only way these assumptions can be broken, and this inference does not seem to be necessary for completeness of linear refutations. Any refutation of $D'[\bar{z}] \vee E'[\bar{z}] \vee L''[\bar{z}]$ could also be applied to either original component as they both subsume it, then the same refutation applied again to the remaining component, unchanged as it is variable-disjoint. Therefore, we may simply forbid factoring across component boundaries in linear refutations and remain complete.

4 SAT and Lemmas

An immediate observation from the above is that sub-derivations may now be repeatedly solved during proof search. Before, if a refutation was found, proof search was over. Now, a moderately difficult goal may recur in several different contexts, wasting system time to dispatch it: ideally, this should be avoided. However, it is quite expensive and fiddly to keep track of dispatched sub-goals and the context necessary to solve them.

4.1 Inferences as SAT

It is possible to maintain a set of components, input clauses containing components, and inferences between components, all encoded as an incremental SAT problem. Components $\forall \bar{x}.\ C$ are mapped injectively to propositional literals $[\forall \bar{x}.\ C]$. The propositional literals representing first-order components are usually positive, but in the case of a component that is a negative ground literal $\neg L$, it is a possible optimisation (but not required) to have $[\neg L] = \neg[L]$ [45].

First, the clauses in the input set are split and encoded directly as propositional clauses. Then, during proof search, inferences can be encoded as they are made at the first-order level. Inferences are implications from one or two components (factoring and resolution respectively) to a clause that may have zero-or-more components after splitting[2]. As it happens, each inference can be treated as a single propositional clause, relating several components. Input resolution hides a detail: it is only necessary to encode an inference involving the component that contains the literal resolved against, ignoring other components in the input clause.

4.2 Unsatisfiability and Lemma Components

Given this encoding, a SAT solver may soundly report unsatisfiability when queried, which indicates a refutation has been found. This happens when sufficient first-order information has been encoded to detect an inconsistency, which may be significantly before the first-order search finds a proof.

However, before unsatisfiability is detected, some SAT solvers can report that a propositional variable is *forced*: that is, it can have no other assignment than it has currently. We can use this to implement a "lemma" mechanism. If the current sub-goal is $\forall \bar{x}.\ C$, and its corresponding propositional literal $[\forall \bar{x}.\ C]$ is forced false, it may be treated as solved.

5 Implementation

We present a prototype system Lickety[3] that implements linear resolution with clause splitting as described above. The system ingests problems written in the

[2] note that the clause with zero components is the empty clause
[3] "lickety-split!"

CNF or FOF dialects of the the TPTP language [41] and attempts to prove them. Conversion of first-order formulae to CNF is achieved with the standard naïve transformation, followed by splitting clauses into components. Equality, if present, is then axiomatised in the usual way. Clauses are split into components using an implementation adapted from AVATAR, which is essentially union-find and very efficient. However, once split, it is practically important to identify component variants up to literal order and renaming, but this can be a little more expensive in our context.

Search begins with clauses derived from the conjecture, and proceeds by iterative deepening on the length of sub-derivations plus the number of literals in the current goal. Input clauses and deductions are inserted as described in §4 into an instance of the PicoSAT [4] API, which is queried after every iterative deepening step for satisfiability, and constantly for forced literals. No attempt is yet made to extract proofs, but this is not impossible. The implementation tested here is available online[4].

6 Experimental Results

Table 1: problems solved in 5 seconds

	FOF	JD bushy		chainy
total problems	500	1240	2078	2078
Vampire	189	481	1162	367
...uniquely	110	112	410	117
Lickety	116	421	774	369
...uniquely	37	52	22	119

We give a very brief, informal experimental evaluation to whet the appetite: Lickety competes with a recent build of Vampire [21], a state-of-the-art superposition system with which we are familiar. Both systems were allowed 1 core and 5 seconds per problem[5] on a desktop machine. Vampire ran in its default mode, with the exception of disabling the limited-resource strategy [36] for reproducibility and not printing proofs for a fair comparison: that is, `vampire -sa discount -p off`. We emphasise that this is far from the "competition strength" of Vampire, which can use extra time, cores and *portfolio modes* [48] very effectively.

We began with the "first-order-formulae" (FOF) division from the previous CASC [42] competition, CASC-28 [43]. This contains 500 problems of mixed difficulty from a variety of domains. Naturally, we do not do well here compared to

[4] https://github.com/MichaelRawson/lickety
revision `f0565b58e5aca47eca142211de96f750080de820`

[5] matching the shortest time limit considered in Sledgehammer: Judgement Day

the mature Vampire. However, of the 37 problems Lickety solves that Vampire does not, most contain large numbers of possibly-redundant axioms to show a relatively-simple conjecture. Such problems often arise when reasoning over a "knowledge base" (e.g. [30]), or when generated from formalised mathematics, such as with Isabelle [28]/Sledgehammer [7] or Mizar [38]/MPTP [44].

Therefore, we also evaluate other benchmarks from this area. *Sledgehammer: Judgement Day* [8] (JD) evaluated the performance of the Sledgehammer system on 7 diverse Isabelle theories, yielding 1240 problems for the automatic systems E [39], Spass [47] and Vampire. The MPTP2078 [1] benchmark provides 2078 problems from Mizar and comes in two flavours: *bushy* (fewer axioms) and *chainy* (more axioms). Lickety does better here — even surpassing Vampire in *chainy* — although we expect that Vampire's implementation of SInE [18] would help enormously if enabled. Table 1 shows the headline figures.

7 Related Work

This approach has common ground with much other work. As a backtracking system, it is in the same family as model elimination/connection tableaux systems [25,23], albeit with fewer refinements and universal variables. Since in this work subgoals are entirely independent and do not need to be backtracked over to find alternative ways to solve them, it provides one possible way of looking at *restricted backtracking* [29,15]. With respect to the treatment of variables, there is also a passing resemblance to hyper tableaux [3] and the disconnection tableaux calculus [24], although both of these are proof confluent rather than backtracking in nature. The use of a SAT solver in this context to detect global inconsistency and provide lemmas suggests instance/model-based reasoning and semantic guidance [20,9,40,11,31] Implementation of variable splitting borrows heavily from AVATAR [45] and the general technique relates to variable splitting in analytic tableaux [17]

8 Perspective

We are cautiously optimistic about clause splitting as a refinement of linear resolution. Initial experimental results seem very promising for what is a very simple implementation, and many of the desirable properties of linear resolution are retained: simplicity, goal-sensitivity, low memory use, tolerance for large axiom sets, susceptibility to learned heuristics, universal variables. However, the ability to produce independently-dispatched subgoals introduces some new possibilities. Subgoals can be closed by lemma mechanisms, possibly driven by SAT or SMT. Additionally, since at least some subgoals are typically solved during a run even if the system ultimately fails to find a proof, we gain the ability to produce training data even without complete proofs.

There are many future directions to explore. Refinements of linear resolution are plentiful, although it is not clear how many of them are compatible with splitting. Techniques such as Brand's modification [10] or paramodulation [12]

are known to be compatible with linear refutation, possibly helping with equality reasoning. Clause splitting is helpful with applied to inductive reasoning in a saturation context [34], which suggests the possibility of a linear system with induction capabilities. Integrating learned guidance into the system is already in progress.

Acknowledgements We gratefully acknowledge Geoff Sutcliffe's help in finding material on linear methods.

References

1. Alama, J., Heskes, T., Kühlwein, D., Tsivtsivadze, E., Urban, J.: Premise selection for mathematics by corpus analysis and kernel methods. Journal of Automated Reasoning **52**(2), 191–213 (2014)
2. Apt, K.R., Van Emden, M.H.: Contributions to the theory of logic programming. Journal of the ACM (JACM) **29**(3), 841–862 (1982)
3. Baumgartner, P., Furbach, U., Niemelä, I.: Hyper tableaux. In: European Workshop on Logics in Artificial Intelligence. pp. 1–17. Springer (1996)
4. Biere, A.: PicoSAT essentials. Journal on Satisfiability, Boolean Modeling and Computation **4**(2-4), 75–97 (2008)
5. Biere, A., Heule, M., van Maaren, H.: Handbook of satisfiability, vol. 185. IOS press (2009)
6. Bjøner, N., Reger, G., Suda, M., Voronkov, A.: AVATAR modulo theories. In: 2nd Global Conference on Artificial Intelligence. pp. 39–52 (2016)
7. Blanchette, J.C., Bulwahn, L., Nipkow, T.: Automatic proof and disproof in Isabelle/HOL. In: International Symposium on Frontiers of Combining Systems. pp. 12–27. Springer (2011)
8. Böhme, S., Nipkow, T.: Sledgehammer: judgement day. In: International Joint Conference on Automated Reasoning. pp. 107–121. Springer (2010)
9. Bonacina, M.P., Plaisted, D.A.: SGGS theorem proving: an exposition. In: PAAR@ IJCAR. pp. 25–38 (2014)
10. Brand, D.: Proving theorems with the modification method. SIAM Journal on Computing **4**(4), 412–430 (1975)
11. Brown, M., Sutcliffe, G.: System description: PTTP+GLiDeS semantically guided pttp. In: International Conference on Automated Deduction. pp. 411–416. Springer (2000)
12. Chang, C.L., Slagle, J.R.: Completeness of linear refutation for theories with equality. Journal of the ACM (JACM) **18**(1), 126–136 (1971)
13. Claessen, K., Sorensson, N.: New techniques that improve MACE-style model finding. In: Model Computation (2003)
14. De Moura, L., Bjørner, N.: Satisfiability modulo theories: introduction and applications. Communications of the ACM **54**(9), 69–77 (2011)
15. Färber, M.: A curiously effective backtracking strategy for connection tableaux. CoRR **abs/2106.13722** (2021), https://arxiv.org/abs/2106.13722
16. Färber, M., Kaliszyk, C., Urban, J.: Machine learning guidance for connection tableaux. Journal of Automated Reasoning **65**(2), 287–320 (2021)
17. Hansen, C.M., Antonsen, R., Giese, M., Waaler, A.: Incremental variable splitting. Journal of Symbolic Computation **47**(9), 1046–1065 (2012)

18. Hoder, K., Voronkov, A.: Sine qua non for large theory reasoning. In: International Conference on Automated Deduction. pp. 299–314. Springer (2011)
19. Kaliszyk, C., Urban, J., Michalewski, H., Olšák, M.: Reinforcement learning of theorem proving. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. pp. 8836–8847 (2018)
20. Korovin, K.: Instantiation-based automated reasoning: From theory to practice. In: International Conference on Automated Deduction. pp. 163–166. Springer (2009)
21. Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: International Conference on Computer Aided Verification. pp. 1–35. Springer (2013)
22. Kowalski, R., Kuehner, D.: Linear resolution with selection function. Artificial Intelligence **2**(3-4), 227–260 (1971)
23. Letz, R., Stenz, G.: Model elimination and connection tableau procedures. In: Handbook of Automated Reasoning, pp. 2015–2114. Elsevier (2001)
24. Letz, R., Stenz, G.: Proof and model generation with disconnection tableaux. In: International Conference on Logic for Programming Artificial Intelligence and Reasoning. pp. 142–156. Springer (2001)
25. Loveland, D.W.: Mechanical theorem-proving by model elimination. In: Automation of Reasoning, pp. 117–134. Springer (1968)
26. Loveland, D.W.: A unifying view of some linear Herbrand procedures. Journal of the ACM (JACM) **19**(2), 366–384 (1972)
27. Loveland, D.W.: A linear format for resolution. In: Automation of Reasoning, pp. 399–416. Springer (1983)
28. Nipkow, T., Wenzel, M., Paulson, L.C.: Isabelle/HOL: a proof assistant for higher-order logic. Springer (2002)
29. Otten, J.: Restricting backtracking in connection calculi. AI Communications **23**(2-3), 159–182 (2010)
30. Pease, A., Sutcliffe, G., Siegel, N., Trac, S.: Large theory reasoning with SUMO at CASC. AI communications **23**(2-3), 137–144 (2010)
31. Rawson, M., Reger, G.: Eliminating models during model elimination. In: International Conference on Automated Reasoning with Analytic Tableaux and Related Methods. pp. 250–265. Springer (2021)
32. Rawson, M., Reger, G.: lazyCoP: Lazy paramodulation meets neurally guided search. In: International Conference on Automated Reasoning with Analytic Tableaux and Related Methods. pp. 187–199. Springer (2021)
33. Reger, G., Suda, M.: Set of support for theory reasoning. In: IWIL/LPAR (2017)
34. Reger, G., Voronkov, A.: Induction in saturation-based proof search. In: International Conference on Automated Deduction. pp. 477–494. Springer (2019)
35. Riazanov, A., Voronkov, A.: Splitting without backtracking. In: IJCAI. pp. 611–617 (2001)
36. Riazanov, A., Voronkov, A.: Limited resource strategy in resolution theorem proving. Journal of Symbolic Computation **36**(1-2), 101–115 (2003)
37. Robinson, J.A.: A machine-oriented logic based on the resolution principle. Journal of the ACM (JACM) **12**(1), 23–41 (1965)
38. Rudnicki, P.: An overview of the Mizar project. In: Proceedings of the 1992 Workshop on Types for Proofs and Programs. pp. 311–330 (1992)
39. Schulz, S.: E — a brainiac theorem prover. AI Communications **15**(2-3), 111–126 (2002)
40. Sutcliffe, G.: The semantically guided linear deduction system. In: International Conference on Automated Deduction. pp. 677–680. Springer (1992)
41. Sutcliffe, G.: The TPTP problem library and associated infrastructure. Journal of Automated Reasoning **43**(4), 337–362 (2009)

42. Sutcliffe, G.: The CADE ATP system competition — CASC. AI Magazine **37**(2), 99–101 (2016)
43. Sutcliffe, G., Desharnais, M., Baumgartner, P., Fontaine, P., Weidenbach, C.: CASC-28 (2021), http://www.tptp.org/CASC/28/
44. Urban, J.: MPTP 0.2: Design, implementation, and initial experiments. Journal of Automated Reasoning **37**(1), 21–43 (2006)
45. Voronkov, A.: AVATAR: The architecture for first-order theorem provers. In: International Conference on Computer Aided Verification. pp. 696–710. Springer (2014)
46. Waldmann, U., Tourret, S., Robillard, S., Blanchette, J.: A comprehensive framework for saturation theorem proving. In: International Joint Conference on Automated Reasoning. pp. 316–334. Springer (2020)
47. Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: SPASS version 3.5. In: International Conference on Automated Deduction. pp. 140–145. Springer (2009)
48. Wolf, A., Letz, R.: Strategy parallelism in automated theorem proving. International journal of pattern recognition and artificial intelligence **13**(02), 219–245 (1999)
49. Wos, L., Robinson, G.A., Carson, D.F.: Efficiency and completeness of the set of support strategy in theorem proving. Journal of the ACM (JACM) **12**(4), 536–541 (1965)
50. Zombori, Z., Csiszárik, A., Michalewski, H., Kaliszyk, C., Urban, J.: Towards finding longer proofs. In: International Conference on Automated Reasoning with Analytic Tableaux and Related Methods. pp. 167–186. Springer (2021)