Relating ordered hypergraph rewriting and proof theory: Work in progress *

Dale Miller
Inria Saclay and LIX, Institut Polytechnique de Paris

This paper explores how concepts from hypergraphs and term graphs can be productively applied to structural proof theory. By treating the sequents that encode proof search states as ordered hypergraphs—where eigenvariables represent nodes and relations represent directed edges—inference rules can directly capture rewriting dynamics. We demonstrate how modern proof-theoretic advances, including linear logic, focusing, synthetic inference rules, and binder mobility, provide a formal foundation for exploiting these graph structures. Ultimately, this work suggests that integrating hypergraph representations into proof theory can enhance the design and implementation of automated and interactive theorem provers.

1 Introduction

The Abella theorem prover [3] is an interactive theorem prover that has been designed on principles taken from structural proof theory and relational specification, instead of the more common foundations of dependently typed λ-calculi, natural deduction, or Frege-style proofs. When an Abella user is searching for a proof, the state of the search is a collection of *sequents* (à la Gentzen) involving relational (not functional) statements. For example, when attempting to prove the theorem that for all natural numbers d, n, and m, if $d \mid n$ and $d \mid m$ then $d \mid (n+m)$, we first must write this a first-order formula based on relations (instead of functions):

```
Theorem div_plus : forall D N M P,
  divides D N -> divides D M -> plus N M P -> divides D P.
```

A typical approach to proving this formula reduces to attempting to prove the sequent on the left of Figure 1. This sequent encodes the fact that given a number of variables (eigenvariables using Gentzen's terminology), and a number of hypotheses (labeled with indexes starting with `H`), one needs only to prove that there exists a number which, when multiplied by `D`, yields `P`. If we pause to examine that sequent, it is clear that these assumed variables and hypotheses can be used to specify three ordered hypergraphs, where variables denote nodes. For example, the hypergraph[1] for `times` contains the triples

```
(D, P1, N), (D, P2, N), (D, Q, P6), (D, P1, XtY), (D, P2, XtZ)
```

which, in turn, denote the facts that

```
D * P1 = N,  D * P2 = N,  D * Q = P6,  D * P1 = XtY,  D * P2 = XtZ
```

When moving from using relations to encode functions (or, equivalently, using hypergraphs to encode term graphs), one must also add facts about these relations that say that they are determinate, as with the following two (previously proved) theorems.

*Accepted for presentation at TermGraph 2026.

[1] Since this encoding is based on atomic formulas, I will assume that all hypergraphs in this paper are *ordered* hypergraphs.

```
Variables: D N M P P1 P2 P6 Q XtY
    XtZ
H3: plus N M P
H4: times D P1 N
H5: times D P2 M
H6: nat N
H7: nat M
H8: nat P
H9: nat D
H10: nat P1
H11: nat P2
H12: plus P1 P2 Q
H13: times D Q P6
H14: times D P1 XtY
H15: times D P2 XtZ
H16: plus XtY XtZ P6
================================
 exists P1, times D P1 P
```

```
Variables: D P1 P2 P6 Q XtY XtZ
H6: nat XtY
H7: nat XtZ
H8: nat P6
H9: nat D
H10: nat P1
H11: nat P2
H12: plus P1 P2 Q
H13: times D Q P6
H14: times D P1 XtY
H15: times D P2 XtZ
H16: plus XtY XtZ P6
================================
 exists P1, times D P1 P6
```

Figure 1: Two sequents from Abella

```
Theorem times_det : forall A B C D, times A B C -> times A B D -> C = D.
Theorem plus_det  : forall A B C D, plus  A B C -> plus  A B D -> C = D.
```

These theorems can be applied in a forward-chaining style to the sequent on the left of Figure 1: the result of that application is that the variables `N`, `M`, `P` are unified with and replaced by the variables `XtY`, `XtZ`, `P6`, respectively, and the assumptions `H3`, `H4`, `H5` become redundant and removed. The result is then the sequent on the right of Figure 1. At this point, the proof can be completed since the existential quantifier can be instantiated with the variable `Q`.

Other facts about addition are expressed in Abella using the following statements (each of which has a simple inductive proof).

```
Theorem plus_total : forall M N,
  nat M -> nat N -> exists K, nat K /\ plus M N K.
Theorem plus_comm  : forall M N K, plus M N K -> plus N M K.
Theorem plus_assoc : forall A B C AB ABC,
  plus A B AB -> plus AB C ABC -> exists BC, plus B C BC /\ plus A BC ABC.
```

Clearly, these theorems can be seen as rules for rewriting hypergraphs with the possible addition of new variables (using `exists`) and identification of two nodes as the same (using `=`).

In this short note, I explore some applications of hypergraphs, term graphs, and their rewriting dynamics to structural proof theory. I will conclude by suggesting that the sequent calculus might offer useful insights into how term graphs might capture binders.

2 Elements of proof theory

In this section, I briefly describe some aspects of the proof theory of sequent calculus. In general, a sequent will be written using the notation $\Sigma :: \Gamma \vdash C$, where C is a formula (the goal), Γ is a multiset of formulas (the assumptions), and Σ is a declaration (a binding site) for all the eigenvariables that might appear free in Γ and C. For example, the left column of Figure 1 is encoded so that Σ is the set of 10 variables listed at the top of that column, Γ is the multiset of 14 formulas that are assumptions (the name

of the hypotheses are not part of this presentation of sequent calculus), and the goal is the formula at the bottom of that column.

The introduction rules for the logical connectives come in a variety of forms: the following are three examples of such rules.

$$\frac{\Sigma :: \Gamma \vdash B \quad \Sigma :: \Gamma, C \vdash E}{\Sigma :: \Gamma, B \supset C \vdash E} \supset_l \qquad \frac{\Sigma :: \Gamma, B \vdash C}{\Sigma :: \Gamma \vdash B \supset C} \supset_r \qquad \frac{\Sigma :: \Gamma \vdash B \quad \Sigma :: \Gamma \vdash C}{\Sigma :: \Gamma \vdash B \wedge C} \wedge_r$$

2.1 Focusing and synthetic inference rules

The logical connectives of a proof system can be given a *polarity* as follows: it is *negative* if its right-introduction rule is invertible and it is *positive* if its left-introduction rule is invertible. In general, this classification can be ambiguous (both left and right introduction rules are invertible) and partial (neither left and right introduction rules is invertible). One surprising and deep fact about linear logic is that this classification is total and unambiguous, and, furthermore, the De Morgan dual of a connective flips this polarity.

Andreoli [1] used this classification polarities to define a *focused proof system* for linear logic. This result was later extended to provide for focused proof systems, LKF and LJF, for both classical and intuitionistic logics [17]. In these focused proof systems, proofs are organized into two, alternating phases: the negative phase introduces only negative connectives and the positive phase introduces only positive connectives. One combination of positive and negative phases (a *bipole*) turns out to be an elegant definition of *synthetic inference rule*. For example, focusing can turn the formula

$$\forall x \forall y \forall z \, (\mathit{path}(x,y) \supset \mathit{path}(y,z) \supset \mathit{path}(x,z))$$

(which specifies that the predicate *path* is transitive) into one of the following *synthetic rules*: if atomic formulas of the form $\mathit{path}(\cdot,\cdot)$ are assigned to the negative phase, the back-chaining rule

$$\frac{\Sigma :: \Gamma \vdash \mathit{path}(x,y) \quad \Sigma :: \Gamma \vdash \mathit{path}(y,z)}{\Sigma :: \Gamma \vdash \mathit{path}(x,z)}$$

is derived while if atoms are assigned to the positive phase, the forward-chaining rule

$$\frac{\Sigma :: \mathit{path}(x,y),\ \mathit{path}(y,z),\ \mathit{path}(x,z),\ \Gamma \vdash E}{\Sigma :: \mathit{path}(x,y),\ \mathit{path}(y,z), \Gamma \vdash E}$$

is derived. Thus, it is possible to replace this transitivity assumption with either of these inference rules and be assured that the cut-elimination theorem will hold for this extended proof system [19].

2.2 Saturation via productive proofs

Focused proof search also provides a framework for describing how one can organize forward-chaining until saturation and to provide some general criteria for which saturation is guaranteed to terminate [7]. Using such saturation, it is possible to describe a high-level specification and implementation of congruence closure.

2.3 Equality

There are many treatments for equality in proof-theoretic settings: see, for example, [11] for an early such approach. One particularly elegant approach treats equality as a *logical connective*: that is, equality is given a left and right introduction rule. While this approach was originally developed for first-order terms [14, 35], it has also been extended to treat terms containing bindings [20]. For first-order logic, the inference rules for equality in the single-conclusion setting are the following.

$$\frac{}{\Sigma :: \Gamma \vdash t = t} =_r \qquad \frac{(t \text{ and } s \text{ not unifiable})}{\Sigma :: s = t, \Gamma \vdash C} =_l \qquad \frac{(\theta = \mathrm{mgu}(t,s)) \quad \Sigma\theta :: \Gamma\theta \vdash C\theta}{\Sigma :: s = t, \Gamma \vdash C} =_l$$

In the last inference rule, the notion $\mathrm{mgu}(t,s)$ denotes the *most general unifier* of t and s. In the setting suggested by the example in Section 1, if a hypothesis equates two eigenvariables (which could be encoding the nodes of a hypergraph), say, $u = v$, then the $=_l$ rule requires the replacement of one of these variables by the other variable throughout the sequent. In order to efficiently implement such a rule, one would likely employ e-graphs [27, 41, 45]. The terms involved in equations do not, however, need to be restricted to just variables (nodes) but to other data structures. For example, if $0 = (s\ 0)$ is an assumption, these two terms are not unifiable and, hence, the sequent immediately has a proof (since it has a false hypothesis).

2.4 Linear vs intuitionistic vs classical logics

Since the introduction of linear logic [13], we learned how to treat the proof theory of classical logic, intuitionistic logic, and linear logic as all part of a single framework where the presence or absence of the structural rules for contraction and weakening separates these three logics. In particular, in a proof system based on two-sided sequents, such as $\Sigma :: \Gamma \vdash \Delta$, it is possible to describe classical logic as the result of allowing both structural rules on both sides of the $\vdash$, while intuitionistic logic is the result of allowing both structural rules only on the left side of the $\vdash$, and (multiplicative additive) linear logic is the result of not allowing these structural rules on either side of the $\vdash$. It is natural to consider additionally allowing two *zones* on the left of the sequent and specify that one zone permits these structural rules while the other zone forbids them: the linear logic programming language Lolli makes use of such hybrid sequents [15]. In a linear logic setting, synthetic rules can be used to model multiset (and hypergraph) rewriting, and the concept of *multifocused proofs* also allows for modeling the parallel application of such inference rules [23].

3 Relating proof-theoretic notions with hypergraphs

Since both hypergraphs and term graphs have a long history of supporting automated reasoning [10, 31] and rewriting [4, 16, 18], it is not surprising that there are many ways to connect those topics with computational logic, proof theory, and interactive theorem proving. Since there have been a number of advances in proof theory, especially since the introduction of linear logic, it seems valuable to survey here some of those advances and their possible relationships to the general topic of applying hypergraphs and their rewriting dynamics to interactive theorem proving.

3.1 Sharing, administrative normal form, and positive bias syntax

An important feature of using hypergraphs to encode term structures is the direct structure support for sharing. Proof theory notions make it possible to derive essentially the same structure-sharing aspects of

syntax using the following revealing steps.

1. The *administrative normal form (ANF)* [9, 33] approach to term representation essentially names all subexpressions in term structures. These names can then be shared in multiple places instead of requiring the structures they name to be reproduced.
2. Using focusing within intuitionistic logic [17], one can define *positive-bias syntax* [26, 42] to provide a systematic account for terms in ANF. (The corresponding *negative-bias syntax* corresponds to $\beta\eta$-long normal form, which does not explicitly provide for sharing.)
3. It was shown in [12] that terms in ANF can be represented as sets of atomic, relational expressions (i.e., ordered hypergraphs) by replacing, for example, naming the expression $(x+y)$ as u, with the expression `plus x y u`, where `plus` is addition in relational form.

3.2 Graph rewriting and forward-chaining

Linear logic has been used to specify multiset rewriting from the early days of linear logic to the present [15, 22, 24, 28, 8]. It is a short step to move from the encoding of multiset rewriting to encoding the rewriting of hypergraphs [40, 32] and Petri nets [6]. In fact, the quality of such encoding can be strong enough that *multifocusing* (which corresponds to forward-chaining or back-chaining on multiple formulas at once) can be used to encode *parallel application* of rewriting steps [23].

3.3 Hypergraph rewriting and logical reasoning

In the particular case of Abella in Section 1, forward chaining is not used to rewrite the multiset of hypotheses but rather to augment them with more information. This phenomenon can be explained within linear logic by moving to that subset of linear logic that encodes intuitionistic logic (via the ! exponential). In such a setting, forward chaining supports other aspects of deduction, including the identification of two nodes in a hypergraph (as illustrated by the use of the `times_det`) as well as the creation of new eigenvariables (encoding of nodes) by forward-chaining on formulas such as

```
Theorem times_total : forall M N,
   nat M -> nat N -> exists K, nat K /\ times M N K.

Theorem times_assoc : forall A B C AB ABC,
  times A B AB -> times AB C ABC -> exists BC, times B C BC /\ times A BC ABC.
```

Operationally, applying the associativity theorem via forward-chaining requires matching two atomic assumptions: `times A B AB` and `times AB C ABC`. Rather than deleting these original hypotheses, the step generates a new eigenvariable `BC` and augments the multiset with two new atomic facts: `times B C BC` and `times A BC ABC`. This choice—whether rewrite rules consume or preserve atomic facts—is easily specified and controlled by moving between full linear logic and its intuitionistic (or classical) sublogics.

3.4 Treatment of bindings in expressions

The importance of treating variable binding in syntax has long been recognized in the proof assistant literature [2, 21], as well as in recent research concerning hypergraph and term graph representations of syntax [34, 39, 43, 44].

In order to motivate a distinctly proof-theoretic approach to the treatment of bindings, I like quoting Alan Perlis's epigram #47 for its insight into the nature of variable bindings: "There is no such thing as a

free variable" [29]. Within the sequent calculus, this perspective is realized through the notion of *binder mobility*, where term-level binders (such as λ-binders) can transition into formula-level binders (such as the universal quantifier) and ultimately into proof-level binders (eigenvariables). This style of binder mobility is central to the treatment of bindings in λProlog and is also available in Rocq via the Rocq-Elpi plugin [37, 36]. When the operational semantics of logic specifications support binder mobility, such specifications can handle bindings by either instantiating them or moving them, thereby bypassing the need to address the low-level implementation details of the binder itself since binders always remain binders. Of course, the implementer of the proof assistant needs to be concerned with the details of how to implement binders. However, the person using the proof assistant will not need to see those details. This approach stands in contrast to most modern proof assistants, which typically require the user to incorporate formal treatments of bindings into their specifications—for instance, by employing De Bruijn's nameless dummies [5] or nominal techniques [30].

The introduction of the ∇-quantifier [25, 38] provided an additional mechanism for a formula-level binder (via ∇) to move to the proof level (as a nominal constant). The proof theory of this new quantifier integrates seamlessly with the other aspects of proof theory illustrated here and has been successfully incorporated into the Abella prover. A compelling direction for future work is to explore the extent to which this approach to bindings can be applied to the term graph setting, particularly since term graphs are frequently used to rewrite and optimize programming language expressions involving complex binding structures. A preliminary experiment using Abella's ∇ and nominal constants has already demonstrated a simple specification of bisimulation for λ-graphs [26].

4 Conclusion

This short note is intended to outline how proof theory and the implementation of systems based on proof theory (such as the Abella proof assistant) might benefit from drawing clear connections to the large literature on hypergraphs and term graphs. While some connections between these two subjects have been known for years, the recent advances in proof theory—in particular, linear logic, focusing, and binder mobility—suggest that this connection might be fruitfully revisited.

References

[1] Jean-Marc Andreoli (1992): *Logic Programming with Focusing Proofs in Linear Logic*. *J. of Logic and Computation* 2(3), pp. 297–347, doi:10.1093/logcom/2.3.297.

[2] Brian E. Aydemir, Aaron Bohannon, Matthew Fairbairn, J. Nathan Foster, Benjamin C. Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich & Steve Zdancewic (2005): *Mechanized Metatheory for the Masses: The POPLmark Challenge*. In: *Theorem Proving in Higher Order Logics: 18th International Conference*, *LNCS* 3603, Springer, pp. 50–65, doi:10.1007/11541868_4.

[3] David Baelde, Kaustuv Chaudhuri, Andrew Gacek, Dale Miller, Gopalan Nadathur, Alwen Tiu & Yuting Wang (2014): *Abella: A System for Reasoning about Relational Specifications*. *J. of Formalized Reasoning* 7(2), pp. 1–89, doi:10.6092/issn.1972-5787/4650.

[4] H. P. Barendregt, M. C. J. D. van Eekelen, J. R. W. Glauert, J. R. Kennaway, M. J. Plasmeijer & M. R. Sleep (1987): *Term graph rewriting*. In: *Parallel Architectures and Languages Europe, Volume II: Parallel Languages*, *Lecture Notes in Computer Science* 259, Springer, Berlin, Heidelberg, pp. 141–158, doi:10.1007/3-540-17945-3_8.

[5] Nicolaas Govert de Bruijn (1972): *Lambda Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with an Application to the Church-Rosser Theorem*. Indagationes Mathematicae 34(5), pp. 381–392, doi:10.1016/1385-7258(72)90034-0.

[6] Iliano Cervesato (1995): *Petri Nets and Linear Logic: a case study for logic programming*. In Maria Alpuente & Maria I. Sessa, editors: Proceedings of the 1995 Joint Conference on Declarative Programming (GULP-PRODE'95), Palladio Press, Marina di Vietri, Italy, pp. 313–318, doi:10.1184/R1/6608369.

[7] Kaustuv Chaudhuri, Arunava Gantait & Dale Miller (2025): *Designing a Safe Forward Chaining Tactic Using Productive Proofs*. In G. L. Pozzato & T. Uustalu, editors: Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX), LNAI 15980, Springer, pp. 299–317, doi:10.1007/978-3-032-06085-3_16.

[8] Flávio Cruz, Ricardo Rocha, Seth Copen Goldstein & Frank Pfenning (2014): *A Linear Logic Programming Language for Concurrent Programming over Graph Structures*. Theory Pract. Log. Program. 14(4-5), pp. 493–507, doi:10.1017/S1471068414000167.

[9] Cormac Flanagan, Amr Sabry, Bruce F. Duba & Matthias Felleisen (1993): *The essence of compiling with continuations*. ACM SIGPLAN Notices 28(6), pp. 237–247, doi:10.1145/155090.155113.

[10] Pascal Fontaine, Stephan Merz & Bruno Woltzenlogel Paleo (2011): *Compression of Propositional Resolution Proofs via Partial Regularization*. In: Automated Deduction – CADE-23, Lecture Notes in Computer Science 6803, Springer, pp. 237–251, doi:10.1007/978-3-642-22438-6_19.

[11] Jean H. Gallier (1986): *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper & Row.

[12] Ulysse Gérard & Dale Miller (2017): *Separating Functional Computation from Relations*. In Valentin Goranko & Mads Dam, editors: 26th EACSL Annual Conference on Computer Science Logic (CSL 2017), LIPIcs 82, pp. 23:1–23:17, doi:10.4230/LIPIcs.CSL.2017.23.

[13] Jean-Yves Girard (1987): *Linear Logic*. Theoretical Computer Science 50(1), pp. 1–102, doi:10.1016/0304-3975(87)90045-4.

[14] Jean-Yves Girard (1992): *A Fixpoint Theorem in Linear Logic*. An email posting to linear@cs.stanford.edu archived at `https://www.seas.upenn.edu/~sweirich/types/archive/1992/msg00030.html`.

[15] Joshua Hodas & Dale Miller (1994): *Logic Programming in a Fragment of Intuitionistic Linear Logic*. Information and Computation 110(2), pp. 327–365, doi:10.1006/inco.1994.1036.

[16] Yves Lafont (1995): *From Proof Nets to Interaction Nets*. In J.-Y. Girard, Y. Lafont & L. Regnier, editors: Advances in Linear Logic, London Mathematical Society Lecture Notes 222, Cambridge University Press, pp. 225–247.

[17] Chuck Liang & Dale Miller (2009): *Focusing and Polarization in Linear, Intuitionistic, and Classical Logics*. Theoretical Computer Science 410(46), pp. 4747–4768, doi:10.1016/j.tcs.2009.07.041. Abstract Interpretation and Logic Programming: A Special Issue in Honor of Professor Giorgio Levi.

[18] Ian Mackie (1995): *The geometry of interaction machine*. In: Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of programming languages, Association for Computing Machinery, New York, NY, USA, pp. 198–208, doi:10.1145/199448.199483.

[19] Sonia Marin, Dale Miller, Elaine Pimentel & Marco Volpe (2022): *From axioms to synthetic inference rules via focusing*. Annals of Pure and Applied Logic 173(5), pp. 1–32, doi:10.1016/j.apal.2022.103091.

[20] Raymond McDowell & Dale Miller (2000): *Cut-elimination for a logic with definitions and induction*. Theoretical Computer Science 232, pp. 91–119, doi:10.1016/S0304-3975(99)00171-1.

[21] Raymond McDowell & Dale Miller (2002): *Reasoning with Higher-Order Abstract Syntax in a Logical Framework*. ACM Trans. on Computational Logic 3(1), pp. 80–136, doi:10.1145/504077.504080.

[22] Dale Miller (1996): *Forum: A Multiple-Conclusion Specification Logic*. Theoretical Computer Science 165(1), pp. 201–232, doi:10.1016/0304-3975(96)00045-X.

[23] Dale Miller (2025): *Linear logic using negative connectives*. In Maribel Fernández, editor: *10th International Conference on Formal Structures for Computation and Deduction (FSCD 2025)*, LIPIcs, pp. 29:1–29:22, doi:10.4230/LIPIcs.FSCD.2025.29.

[24] Dale Miller (2025): *Proof Theory and Logic Programming: Computation as Proof Search*. Cambridge University Press, doi:10.1017/9781009561280. Preprint available at `https://www.lix.polytechnique.fr/Labo/Dale.Miller/ptlp/`.

[25] Dale Miller & Alwen Tiu (2005): *A proof theory for generic judgments*. *ACM Trans. on Computational Logic* 6(4), pp. 749–783, doi:10.1145/1094622.1094628.

[26] Dale Miller & Jui-Hsuan Wu (2023): *A positive perspective on term representations*. In Bartek Klin & Elaine Pimentel, editors: *31st EACSL Annual Conference on Computer Science Logic (CSL 2023)*, *LIPIcs* 252, Dagstuhl, Germany, pp. 3:1–3:21, doi:10.4230/LIPIcs.CSL.2023.3.

[27] Charles Gregory Nelson (1980): *Techniques for Program Verification*. Ph.D. thesis, Stanford University, Stanford, CA, USA. AAI8011683.

[28] Vivek Nigam & Dale Miller (2009): *Algorithmic specifications in linear logic with subexponentials*. In António Porto & Francisco Javier López-Fraguas, editors: *ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP)*, ACM, pp. 129–140, doi:10.1145/1599410.1599427.

[29] Alan J. Perlis (1982): *Epigrams on Programming*. *ACM SIGPLAN Notices* 17(9), pp. 7–13, doi:10.1145/947955.1083808.

[30] Andrew M. Pitts (2003): *Nominal Logic, A First Order Theory of Names and Binding*. *Information and Computation* 186(2), pp. 165–193.

[31] Detlef Plump (1999): *Term Graph Rewriting*. In H. Ehrig, G. Engels, H.-J. Kreowski & G. Rozenberg, editors: *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools*, chapter 1, World Scientific, pp. 3–61, doi:10.1142/9789812815149_0001.

[32] Tikhon Pshenitsyn (2025): *First-Order Intuitionistic Linear Logic and Hypergraph Languages*. In Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine & Gabriele Puppis, editors: *52nd International Colloquium on Automata, Languages, and Programming (ICALP 2025)*, *LIPIcs* 334, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 170:1–170:19, doi:10.4230/LIPIcs.ICALP.2025.170.

[33] Amr Sabry & Matthias Felleisen (1993): *Reasoning about Programs in Continuation-Passing Style*. *Lisp and Symbolic Computation* 6(3-4), pp. 289–360, doi:10.1007/BF01019462.

[34] Rudi Schneider, Thomas Kœhler & Michel Steuwer (2025): *Slotted E-Graphs: First-Class Support for (Bound) Variables in E-Graphs*. *Proceedings of the ACM on Programming Languages* 9(PLDI), p. 223, doi:10.1145/3729326.

[35] Peter Schroeder-Heister (1993): *Rules of Definitional Reflection*. In M. Vardi, editor: *8th Symp. on Logic in Computer Science*, IEEE Computer Society Press, IEEE, pp. 222–232, doi:10.1109/LICS.1993.287585.

[36] Enrico Tassi (2025): *Elpi: rule-based meta-languge for Rocq*. In: *Proceedings of CoqPL 2025*, ACM.

[37] Enrico Tassi (2026): *Elpi: rule-based extension language*. Habilitation à diriger des recherches, Université Côte d'Azur, Valbonne, France. Available at `http://www-sop.inria.fr/members/Enrico.Tassi/hdr/hdr.pdf`.

[38] Alwen Tiu (2004): *A Logical Framework for Reasoning about Logical Specifications*. Ph.D. thesis, Pennsylvania State University. Available at `https://etda.libraries.psu.edu/files/final_submissions/119`.

[39] Aleksei Tiurin, Dan R. Ghica & Nick Hu (2025): *E-Graphs With Bindings*. *arXiv*, doi:10.48550/arXiv.2505.00807.

[40] Paolo Torrini & Reiko Heckel (2009): *Towards an embedding of Graph Transformation in Intuitionistic Linear Logic*. In Filippo Bonchi, Davide Grohmann, Paola Spoletini & Emilio Tuosto, editors: *Proceedings 2nd Interaction and Concurrency Experience: Structured Interactions, ICE 2009*, *EPTCS* 12, pp. 99–115, doi:10.4204/EPTCS.12.7.

[41] Max Willsey, Remy Wang, Chandrakana Nandi, Zachary Tatlock & Pavel Panchekha (2021): *Egg: Fast and Extensible E-graphs*. Proc. ACM Program. Lang. 5(POPL), doi:10.1145/3434304.

[42] Jui-Hsuan Wu (2024): *Proof theory, syntactic representations, logic, and sharing*. Ph.D. thesis, Institut Polytechnique de Paris, doi:10.70675/c7bca972zc1d0z48ccz9c93z4dbb1e82b169.

[43] A. Yasen & K. Ueda (2017): *Unification of Hypergraph λ-Terms*, pp. 136–152. 10608, Springer, Cham, doi:10.1007/978-3-319-68953-1_9.

[44] Alimujiang Yasen & Kazunori Ueda (2018): *Name Binding is Easy with Hypergraphs*. IEICE Transactions on Information and Systems E101-D(4), pp. 1126–1140, doi:10.1587/transinf.2017EDP7257.

[45] Yihong Zhang, Yisu Remy Wang, Oliver Flatt, David Cao, Philip Zucker, Eli Rosenthal, Zachary Tatlock & Max Willsey (2023): *Better Together: Unifying Datalog and Equality Saturation*. Proc. ACM Program. Lang. Volume 7, Issue PLDI, pp. 468–492, doi:10.1145/3591239.