A Gentle Introduction to Deep Inference
2. Lecture
Open deduction: A 21st century proof formalism

Victoria Barrett and Lutz Straßburger

July 2025

1 What do we want from a proof formalism?

1.1 Normalisation

Proofs can be complex objects, and therefore difficult to study from a mathematical perspective. One of the most important activities in structural proof theory is *normalisation*, rewriting proofs into a normal form in order to facilitate proving properties about proof systems and logics. For example, we might like a proof to not contain any propositional atoms that do not appear in its conclusion, or we might like to separate the linear and non-linear parts of a proof.

The formalism can help in normalisation, by allowing for useful normal forms and simple normalisation procedures; we will see many examples of what we mean by this throughout this course.

For now, very abstractly, we can imagine a situation where there is some rewriting step that would make a proof more 'normal', but break some part of the structure so that it is no longer a proof according to the formalism. A proof formalism is good, according to this metric, if this doesn't happen.

1.2 Semantics

One of the foundational questions in proof theory is, 'when are two proofs the same?'. By this, we don't only mean when are they syntactically identical, nor do we only mean when do they prove the same formula. The first would identify too few proofs and the second too many.

Instead, to get a notion of 'the same', we might appeal to semantics. By relating formulae to some objects and proofs to some maps between objects, we might be able to inherit an appropriate idea of identity for proofs.

We have a pretty good idea for intuitionistic propositional logic (see *star-autonomous categories* or their translation into the lambda calculus) and for multiplicative linear logic (see *proof nets*), where we can abstract away the

'bureaucracy' of proofs and retain only the 'essential' information, but it gets difficult for proofs in other logics where we don't have such good models.

The formalism can help here as well: a formalism which already reduces the bureaucracy will make it easier to see from the syntactic side which proofs should be identified. A formalism which admits as proofs better canonical forms will also help in relating the syntax to a semantics.

1.3 Complexity

We would like our proof formalism to contain short proofs. Smaller proofs are easier to check for correctness, which has benefits for both automated and human reasoning.

Proof compression and proof normalisation can work in opposite directions to one another, with the latter usually making proofs much larger. Nevertheless, proof compression can allow for a clearer perception of the essential parts of a proof, by conflating similar pieces of reasoning, whereas a normalised proof may prove each part separately and from a minimal number of hypotheses.

The proof formalism can help in this by allowing for mechanisms by which proofs can be compressed.

1.4 Proof Search

Given a formula, we would like it to be easy to find a proof of it and quick to show when no proof can exist.

All of the previously listed virtues of proof formalisms may help in proof search, for example if we know that we can restrict our search to good normal forms or canonical representatives, or if proofs will either be short or non-existent. The formalism can restrict our choices in building a proof in such a way that proof search is easier.

2 From sequent calculus to open deduction

We have seen that in the sequent calculus for propositional classical logic, formulae can be constructed by the grammar:

$$A, B \ ::= \ a \mid \bar{a} \mid A \vee B \mid A \wedge B \mid \bot \mid \top$$

and derivations are constructed from (a refinement of) the grammar:

$$\Pi \ ::= \ \frac{\Pi_1 \qquad \dots \qquad \Pi_k}{A_1, \ \dots \ , A_m \vdash B_1, \ \dots \ , B_n}$$

where $k, m, n \geq 0$, $\Pi_1, \dots, \Pi_k$ are derivations, and $A_1, \dots, A_m, B_1, \dots, B_n$ are formulae.

The comma means something different on the left and the right of the turnstile $\vdash$: on the left we read it as a conjunction and on the right as a disjunction, so that if all of the antecedents hold then at least one of the consequents holds. There is also an implicit conjunction between the derivations in the premise: every premise must be proven in order to prove the conclusion $A_1, \ldots, A_m \vdash B_1, \ldots, B_n$.

We therefore have three kinds of conjunction and two kinds of disjunction, with a distinction between what operates at the level of formulae and what operates at the level of derivation.

Inference rules have access to the derivation-level connectives, but not to the formula-level connectives. For example, the (two-sided) cut rule is

$$\mathsf{cut}\,\frac{\Gamma \vdash \Theta, A \qquad A, \Delta \vdash \Lambda}{\Gamma, \Delta \vdash \Theta, \Lambda}$$

and not something like

$$\frac{\Gamma \vdash \Theta, A \vee B \qquad A \wedge C, \Delta \vdash \Lambda}{\Gamma, C, \Delta \vdash \Theta, B, \Lambda}$$

In deep inference formalisms, we adopt a more liberal approach to the composition of derivations. We reject the distinction between formulae- and derivation-level connectives, allowing *inferences* to operate *deep* inside formulae.

In this course, we are going to focus on the deep inference formalism *open deduction*[3].

We build both formulae and derivations from the following grammar:

$$\Pi, \Delta \;::=\; a \mid \bar{a} \mid \Pi \vee \Delta \mid \Pi \wedge \Delta \mid \bot \mid \top \mid \frac{\Pi}{\Delta}$$

Formulae are just those derivations which can be built without *composition by inference* $\dfrac{\Pi}{\Delta}$.

This has some immediate consequences. Derivations have a single premise and a single conclusion. We can write

$$\begin{array}{c} A \\ \| \\ B \end{array}$$

for a derivation whose premise is A and whose conclusion is B. A ***proof*** is a derivation whose premise is $\top$.

We can compose two derivations horizontally by connective. If

$$\begin{array}{c} A \\ \| \\ B \end{array} \quad \text{and} \quad \begin{array}{c} C \\ \| \\ D \end{array}$$

are derivations then so are

$$\begin{array}{c} A \quad C \\ \| \wedge \| \\ B \quad D \end{array} \qquad \text{and} \qquad \begin{array}{c} A \quad C \\ \| \vee \| \\ B \quad D \end{array}$$

These derivations have premises $A \wedge C$ and $A \vee C$ respectively, and conclusions $B \wedge D$ and $B \vee D$ respectively.

If we have two derivations

$$\begin{array}{c} A \\ \| \\ B \end{array} \qquad \text{and} \qquad \begin{array}{c} B \\ \| \\ C \end{array}$$

then we can glue these directly together to obtain

$$\begin{array}{c} A \\ \| \\ B \\ \| \\ C \end{array}$$

3 From LK to SKSg

In open deduction we have done away with the sequent notation entirely. We translate a sequent

$$\Gamma \vdash \Lambda$$

into a formula

$$\bigvee \bar{\Gamma} \vee \bigvee \Lambda$$

where $\bigvee \bar{\Gamma}$ is a disjunction of the negation of every formula in Γ and $\bigvee \Lambda$ is a disjunction of every formula in Λ. When Γ is empty, we can ignore the empty disjunction and translate the sequent to $\bigvee \Lambda$.

In order to check the correctness of an instance of the cut rule in LK, it is necessary to check that the cut formulae match in both premises, and that the lists of formulae in the conclusion have been correctly split without any duplication or omission. In deep inference proof systems, we are able to decompose the cut rule into smaller steps, isolating these checks.

A deep inference proof system for propositional classical logic, SKSg, is shown in Figure 1, and the translation of the (one-sided) LK cut rule into SKSg is as follows:

$$\mathsf{cut}\,\frac{\vdash \Theta, A \qquad \vdash \bar{A}, \Lambda}{\vdash \Theta, \Lambda} \quad \mapsto \quad = \dfrac{\mathsf{s}\,\dfrac{(\bigvee \Theta \vee A) \wedge (\bar{A} \vee \bigvee \Lambda)}{\mathsf{s}\,\dfrac{(\bigvee \Theta \vee A) \wedge \bar{A}}{\bigvee \Theta \vee \mathsf{i}{\uparrow}\,\dfrac{A \wedge \bar{A}}{\bot}} \vee \bigvee \Lambda}}{\bigvee \Theta \vee \bigvee \Lambda}$$

- The structural rules:

$$\mathsf{i}{\downarrow}\,\frac{\top}{A \vee \bar{A}} \quad \textit{identity} \qquad \mathsf{c}{\downarrow}\,\frac{A \vee A}{A} \quad \textit{contraction} \qquad \mathsf{w}{\downarrow}\,\frac{\bot}{A} \quad \textit{weakening}$$

$$\mathsf{i}{\uparrow}\,\frac{A \wedge \bar{A}}{\bot} \quad \textit{cut} \qquad \mathsf{c}{\uparrow}\,\frac{A}{A \wedge A} \quad \textit{cocontraction} \qquad \mathsf{w}{\uparrow}\,\frac{A}{\top} \quad \textit{coweakening}$$

- The logical rule:

$$\mathsf{s}\,\frac{A \wedge (B \vee C)}{(A \wedge B) \vee C}$$
switch

- An equivalence on formulae, defined to be the minimal equivalence relation generated by the following:

$$A \wedge \top = A \qquad A \wedge B = B \wedge A$$
$$A \vee \bot = A \qquad A \vee B = B \vee A$$
$$\top \vee \top = \top \qquad A \wedge (B \wedge C) = (A \wedge B) \wedge C$$
$$\bot \wedge \bot = \bot \qquad A \vee (B \vee C) = (A \vee B) \vee C$$

  We write $=\frac{A}{B}$ whenever B can be reached from A via this equivalence relation.

Figure 1: Proof system SKSg

Switches are used to bring the cut formulae together, and the structural rule $\mathsf{i}{\uparrow}$ annihilates them. Each instance of the switch rule corresponds to splitting the context of the cut rule into the two premises.

In the case where a sequent calculus rule has no premise, we translate this to $\top$, so that the identity rule is translated as follows:

$$\mathsf{id}\,\frac{}{\vdash A, \bar{A}} \quad \mapsto \quad \mathsf{i}{\downarrow}\,\frac{\top}{A \vee \bar{A}}$$

In the case of the rule $\top$, both premise and conclusion are therefore translated to $\top$, and so we translate this to the formula $\top$, not an inference rule.

Exercise 1: Translate the other rules of LK into SKSg.[1]

[1] If you choose to translate the one-sided system, you won't need to use $\mathsf{w}{\uparrow}$ or $\mathsf{c}{\uparrow}$, but you will if you translate the two-sided system.

Exercise 2: Show that there is an SKSg derivation $\begin{array}{c} A \\ \| \\ B \end{array}$ if and only if there is an SKSg proof $\frac{\bar{A} \vee B}{\prod}$.

4 Symmetry

Sequent calculus derivations do not have top-down symmetry: they can have multiple premises but only one conclusion. Furthermore, although we can understand the identity rule and the cut rule as being dual, in the sense that one creates a pair of formulae and the other annihilates, the formalism of the sequent calculus obscures this.

Deep inference restores this: the cut rule and the identity rule can be obtained the one from the other by negating both the premise and conclusion and swapping them. The other rules of SKSg have their duals as well: contraction and cocontraction form a dual pair, as do weakening and coweakening, and the switch rule is its own dual.

This top-down symmetry and duality in the rule shape combine to allow us to dualise entire derivations simply by flipping and negating. A proof of A can immediately be transformed into a *refutation* of $\bar{A}$, that is, a derivation whose premise is $\bar{A}$ and whose conclusion is $\bot$.

This is not to say that there is no symmetry present in the sequent calculus; rather there we see a left-right symmetry in the rules of proof systems, which can also be striking. For this reason, it may be more correct to see sequents $\Gamma \vdash \Lambda$ as corresponding to deep-inference derivations $\begin{array}{c} \bigwedge \Gamma \\ \| \\ \bigvee \Lambda \end{array}$. We take the approach of translating sequent calculus inference rules into deep-inference derivations because it makes it clearer to see which rules we will need in our proof system.

5 Bureaucracy

Open deduction is designed to reduce *syntactic bureaucracy*; specifically, it eliminates what is called *bureaucracy of type A* in [5] (and indeed in earlier notes about deep inference).

This is the syntactic bureaucracy of independent processes being obliged by the formalism to occur in sequence, instead of in parallel. Consider, for example, the sequent calculus derivations

$$\mathsf{cont}\,\dfrac{\mathsf{exch}\,\dfrac{\vdash A, A, B, C}{\vdash A, B, C}}{\vdash A, C, B} \qquad \text{and} \qquad \mathsf{exch}\,\dfrac{\mathsf{cont}\,\dfrac{\vdash A, A, B, C}{\vdash A, A, C, B}}{\vdash A, C, B}$$

The formalism obliges us to distinguish these derivations, but the contraction and exchange applied are independent of each other. In open deduction, both

- The structural rules:

$$atomic\left\{\begin{array}{lll} \mathsf{ai}{\downarrow}\dfrac{\top}{a \vee \bar{a}} & \mathsf{ac}{\downarrow}\dfrac{a \vee a}{a} & \mathsf{aw}{\downarrow}\dfrac{\bot}{a} \\ \textit{identity} & \textit{contraction} & \textit{weakening} \\ \mathsf{ai}{\uparrow}\dfrac{a \wedge \bar{a}}{\bot} & \mathsf{ac}{\uparrow}\dfrac{a}{a \wedge a} & \mathsf{aw}{\uparrow}\dfrac{a}{\top} \\ \textit{cut} & \textit{cocontraction} & \textit{coweakening} \end{array}\right\}$$

- The logical rules:

$$\mathsf{s}\frac{A \wedge (B \vee C)}{(A \wedge B) \vee C} \quad \textit{switch} \qquad \mathsf{m}\frac{(A \wedge B) \vee (C \wedge D)}{(A \vee C) \wedge (B \vee D)} \quad \textit{medial}$$

- An equivalence on formulae, defined to be the minimal equivalence relation generated by the following:

$$\begin{array}{ll} A \wedge \top = A & A \wedge B = B \wedge A \\ A \vee \bot = A & A \vee B = B \vee A \\ \top \vee \top = \top & A \wedge (B \wedge C) = (A \wedge B) \wedge C \\ \bot \wedge \bot = \bot & A \vee (B \vee C) = (A \vee B) \vee C \end{array}$$

  We write $=\dfrac{A}{B}$ whenever B can be reached from A via this equivalence relation.

Figure 2: Proof system SKS

are translated to

$$\mathsf{c}{\downarrow}\frac{A \vee A}{A} \vee = \frac{B \vee C}{C \vee B}$$

Other forms of bureaucracy still remain in open deduction (for example, what is called *bureaucracy of type B* in [5]); we aren't yet able to identify in the syntax all of the proofs that we would like.

6 Locality

Suppose that we have an object that was built from the grammar

$$\Pi, \Delta \ ::= \ a \mid \bar{a} \mid \Pi \vee \Delta \mid \Pi \wedge \Delta \mid \bot \mid \top \mid \frac{\Pi}{\Delta}$$

and we want to check whether it is a proof in the proof system SKSg. In order to verify this, we need to check that every inference step in this object is an instance of an inference rule in SKSg.

This means that we need to compare formulae of unbounded size: to check whether $\dfrac{A \vee B}{C}$ really is a contraction, we need to compare A, B, and C and verify that they are identical formulae.

In order to do this, we need to have access to a global view of the formulae, but we can imagine a situation where they are too large to be held in local memory on a single processor, and so our procedure to check them needs to traverse between the locations where they're stored. This problem can be solved at the level of the implementation, but we are interested in using the formalism to solve it at the level of the proof system.

With deep inference, we are able to decompose the general contraction rule $\mathsf{c}{\downarrow}\dfrac{A \vee A}{A}$ into an atomic contraction rule and a linear rule called the ***medial***:

$$\mathsf{ac}{\downarrow}\frac{a \vee a}{a} \qquad\qquad \mathsf{m}\frac{(A \wedge B) \vee (C \wedge D)}{(A \vee C) \wedge (B \vee D)}$$

Then contractions can be checked locally, replacing one global procedure with many local procedures.

The decomposition of the cut rule $\mathsf{i}{\uparrow}$ into the switch rule and the atomic cut is shown in the lecture slides, and the local proof system SKS is shown in Figure 2.

Exercise 3: Decompose the other structural rules into their atomic forms. More precisely:

- Derive $\mathsf{i}{\downarrow}$ from $\mathsf{ai}{\downarrow}$, s, and $=$
- Derive $\mathsf{c}{\downarrow}$ from $\mathsf{ac}{\downarrow}$, m, and $=$
- Derive $\mathsf{c}{\uparrow}$ from $\mathsf{ac}{\uparrow}$, m, and $=$
- Derive $\mathsf{w}{\downarrow}$ from $\mathsf{aw}{\downarrow}$ and $=$
- Derive $\mathsf{w}{\uparrow}$ from $\mathsf{aw}{\uparrow}$ and $=$

7 Conclusion

Open deduction is a generalisation of the sequent calculus. By admitting a larger collection of objects as proofs, we hope that within those proofs we will find better canonical representatives, better normalisation procedures, and smaller proofs. However, we have also disrupted existing results from the sequent calculus, and we need to put them back together.

Tomorrow, we will start to talk about normalisation, and see how the atomicity of the structural rules helps in giving good normalisation procedures. The day after, we will apply atomicity to the problem of reducing syntactic bureaucracy in the representation of proofs.

8 Further Reading

Introductions to open deduction can be found in the paper [3] and the thesis [4]; deep inference papers before this used a formalism called *the calculus of structures*, which doesn't allow for the same parallelism as described in Section 5. These papers also introduce *atomic flows*, which we will see on Wednesday.

The paper [2] gives a full translation between SKS (in the calculus of structures) and a sequent calculus presentation of propositional classical logic and the thesis [1] explores SKS in detail.

References

[1] Kai Brünnler. *Deep Inference and Symmetry for Classical Proofs*. PhD thesis, Technische Universität Dresden, 2003.

[2] Kai Brünnler and Alwen Fernanto Tiu. A Local System for Classical Logic. Technical Report WV-01-02, Technische Universität Dresden, 2001.

[3] Alessio Guglielmi, Tom Gundersen, and Michel Parigot. A Proof Calculus Which Reduces Syntactic Bureaucracy. In Christopher Lynch, editor, *21st International Conference on Rewriting Techniques and Applications (RTA)*, volume 6 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 135–150. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2010.

[4] Tom Gundersen. *A General View of Normalisation Through Atomic Flows*. PhD thesis, University of Bath, 2009.

[5] Lutz Straßburger. From Deep Inference to Proof Nets Via Cut Elimination. *Journal of Logic and Computation*, 21(4):589–624, 2011.