Lecture Notes on Structural Logic

15-816: Substructural Logics
Frank Pfenning

Lecture 14
October 13, 2016

At this point we introduce *structural logic*, that is, a logic in which we assume all structural rules for the antecedents: exchange, weakening, and contraction. This is a sequent calculus presentation of what is usually called *intuitionistic logic*, which was developed long before substructural logics. Since all the logics we have considered so far are intuitionistic in nature, however, this is not a distinguishing characteristic.

We begin with a brief excursion to consider quantifiers. Quantified variables may occur arbitrarily often in their scope, and they may occur in any order, which means they are the first fully structural entities we see apart from persistent propositions in the first few lectures.

1 Existential Quantification

From a logical perspective, quantifiers allow us to talk about universal (such as "*all men are mortal*") or existential (such as "*some inference systems do not satisfy cut elimination*)" propositions. From a programming perspective, they allow us to compute over basic data values such as integers or strings. Because of the latter, we will insist on quantifiers being *typed*.

We can prove $\exists x{:}\tau.A$ if we can prove $[t/x]A$ for some term t of type τ. This means we require a new judgment, $\Sigma \vdash t : \tau$, where Σ is a collection of assumptions about the type of variables, $x_1{:}\tau_1, \ldots, x_n{:}\tau_n$. Since there are basic terms with (so far) no particular logic interpretation, we will hedge our bets about which rules this judgment needs to satisfy and collect some requirements from our rules. Typing assumptions for variables now also

become part of main sequent calculus judgment which, to be most general, is still ordered.

$$\frac{\Sigma \vdash t : \tau \quad \Sigma \,;\, \Omega \vdash [t/x]A}{\Sigma \,;\, \Omega \vdash \exists x{:}\tau.\ A}\ \exists R \qquad \frac{\Sigma, a{:}\tau \,;\, \Omega_L\ ([a/x]A)\ \Omega_R \vdash C}{\Sigma \,;\, \Omega_L\ (\exists x{:}\tau.\ A)\ \Omega_R \vdash C}\ \exists L^a$$

In the left rule we introduce a *fresh* parameter a, which does not occur in the whole judgment in the conclusion. We may omit the condition via the general presupposition that the variables declared in a context Σ are all distinct so that $\Sigma, a{:}\tau$ only makes sense if a is not yet declared in Σ.

Because quantified variables are intended to occur multiple times in a proposition, but also possibly not at all, the hypotheses in Σ will be subject to exchange, weakening, and contraction, that is, they can be used arbitrarily often. Just as important is the substitution property, formulated here has two admissible rules of inference

$$\frac{\Sigma \vdash t : \tau \quad \Sigma, a{:}\tau \vdash s : \sigma}{\Sigma \vdash [t/a]s : \sigma}\ \mathsf{subst} \qquad \frac{\Sigma \vdash t : \tau \quad \Sigma, a{:}\tau \,;\, \Omega \vdash C}{\Sigma \,;\, [t/a]\Omega \vdash [t/a]C}\ \mathsf{subst}$$

We obtain the conclusion by systematically going through the proof of the second premise and substituting t for a everywhere.

With this background, how does cut reduction play out? Let's write down the cut where the right rule meets the left rule.

$$\frac{\dfrac{\begin{array}{c}\mathcal{T}\\ \Sigma \vdash t : \tau\end{array} \quad \begin{array}{c}\mathcal{D}'\\ \Sigma \,;\, \Omega \vdash [t/x]A\end{array}}{\Sigma \,;\, \Omega \vdash \exists x{:}\tau.\ A}\ \exists R \quad \dfrac{\begin{array}{c}\mathcal{E}'\\ \Sigma, a{:}\tau \,;\, \Omega_L\ ([a/x]A)\ \Omega_R \vdash C\end{array}}{\Sigma \,;\, \Omega_L\ (\exists x{:}\tau.\ A)\ \Omega_R \vdash C}\ \exists L^a}{\Sigma \,;\, \Omega_L\ \Omega\ \Omega_R \vdash C}\ \mathsf{cut}_{\exists x.A}$$

We have already taken the liberty of writing Σ in both premises of the cut: we can always equalize them by weakening in order to bring them into this form, if necessary.

We would like to reduce the cut on $\exists x{:}\tau.\ A$ to a cut on A. But this does not work: $\mathcal{D}'$ is a proof of $[t/x]A$ and the antecedent in $\mathcal{E}'$ is $[a/x]A$. Now we remember that we were asked to choose a to be fresh, is, it doesn't occur in Σ or Ω_L or A or Ω_R or C. So, for example, $[t/a]C = C$. With substitution and these considerations we obtain

$$\begin{array}{c}[t/a]\mathcal{E}'\\ \Sigma \,;\, \Omega_L\ ([t/a][a/x]A)\ \Omega_R \vdash C\end{array}$$

Now we only need to see that $[t/a][a/x]A = [t/x]A$, again because a was chosen fresh and does not occur in A. In summary we then have:

$$\dfrac{\dfrac{\begin{array}{cc}\mathcal{T} & \mathcal{D}' \\ \Sigma \vdash t : \tau & \Sigma\ ;\ \Omega \vdash [t/x]A\end{array}}{\Sigma\ ;\ \Omega \vdash \exists x{:}\tau.\ A}\ \exists R \quad \dfrac{\begin{array}{c}\mathcal{E}' \\ \Sigma, a{:}\tau\ ;\ \Omega_L\ ([a/x]A)\ \Omega_R \vdash C\end{array}}{\Sigma\ ;\ \Omega_L\ (\exists x{:}\tau.\ A)\ \Omega_R \vdash C}\ \exists L^a}{\Sigma\ ;\ \Omega_L\ \Omega\ \Omega_R \vdash C}\ \mathsf{cut}_{\exists x.A}$$

$$\Longrightarrow_R \quad \dfrac{\begin{array}{cc}\mathcal{D}' & [t/a]\mathcal{E}' \\ \Sigma\ ;\ \Omega \vdash [t/x]A & \Sigma\ ;\ \Omega_L\ ([t/x]A)\ \Omega_R \vdash C\end{array}}{\Sigma\ ;\ \Omega_L\ \Omega\ \Omega_R \vdash C}\ \mathsf{cut}_{[t/x]A}$$

If we are counting characters then $[t/x]A$ may be larger than $\exists x.\ A$, but if we consider the *structure* of propositions, then $[t/x]A$ is a subformula of $\exists x.\ A$. We could also resort to just counting the total number of logical connectives and quantifiers, which goes down since t is just a term and may not contain logical connectives or quantifiers.

2 A Computational Interpretation of Existential Quantification

It should be intuitively clear that an application of $\exists R$ carries information, namely the *witness* t. The computational interpretation of $\exists R$ therefore just send the witness, while $\exists L$ receives it.

$$\dfrac{\Sigma \vdash t : \tau \quad \Sigma\ ;\ \Omega \vdash P :: (y : [t/x]A)}{\Sigma\ ;\ \Omega \vdash \mathsf{send}\ y\ t\ ;\ P :: (y : \exists x{:}\tau.\ A)}\ \exists R$$

$$\dfrac{\Sigma, a{:}\tau\ ;\ \Omega_L\ (y : [a/x]A)\ \Omega_R \vdash Q_a :: (z : C)}{\Sigma\ ;\ \Omega_L\ (y : \exists x{:}\tau.\ A)\ \Omega_R \vdash a \leftarrow \mathsf{recv}\ x\ ;\ Q_a :: (z : C)}\ \exists L^a$$

Operationally, all that happens is the term t is transmitted from one process to the other.

$$\dfrac{\mathsf{proc}(y, \mathsf{send}\ y\ t\ ;\ P) \quad \mathsf{proc}(z, a \leftarrow \mathsf{recv}\ x\ ;\ Q_a)}{\mathsf{proc}(y, P) \quad \mathsf{proc}(z, Q_t)}\ \exists C$$

The intention is to only execute processes with no free term variables, and we can see that the computation rule maintains this invariant. This is similar to the restriction that in functional languages we only execute closed

programs. Also important is that there are no linearity or ordering restrictions on term variables $x{:}\tau$ or $a{:}\tau$. Since they stand for basic values such as integers or strings, they can used freely and need not be used.

3 Universal Quantification

The universal quantifier is symmetric to the existential.

$$\frac{\Sigma, a{:}\tau\ ;\ \Omega \vdash [a/x]A}{\Sigma\ ;\ \Omega \vdash \forall x{:}\tau.\ A}\ \forall R^a \qquad \frac{\Sigma \vdash t : \tau \quad \Sigma\ ;\ \Omega_L\ ([t/x]A)\ \Omega_R \vdash C}{\Sigma\ ;\ \Omega_L\ (\forall x{:}\tau.\ A)\ \Omega_R \vdash C}\ \forall L$$

This is reflected in the cut reduction (see Exercise 1). The proof term assignment reuses the same terms as for the existential, but the roles of provider and client are reversed.

$$\frac{\Sigma, a{:}\tau\ ;\ \Omega \vdash P_a :: (y : [a/x]A)}{\Sigma\ ;\ \Omega \vdash a \leftarrow \mathsf{recv}\ y\ ;\ P_a :: (y : \forall x{:}\tau.\ A)}\ \forall R^a$$

$$\frac{\Sigma \vdash t : \tau \quad \Sigma\ ;\ \Omega_L\ (y : [t/x]A)\ \Omega_R \vdash Q :: (z : C)}{\Sigma\ ;\ \Omega_L\ (y : \forall x{:}\tau.\ A)\ \Omega_R \vdash \mathsf{send}\ y\ t\ ;\ P :: (z : C)}\ \forall L$$

Computationally:

$$\frac{\mathsf{proc}(y, a \leftarrow \mathsf{recv}\ y\ ;\ P_a) \quad \mathsf{proc}(z, \mathsf{send}\ y\ t\ ;\ Q)}{\mathsf{proc}(y, P_t) \quad \mathsf{proc}(z, Q)}\ \forall C$$

We can see that order is largely independent to the meaning of the quantifiers, but we have to take care to treat variables structurally, that is, allow them to be used arbitrarily often, in arbitrary order.

Identity expansion, as well as cut and identity elimination go through as before. In the proof of admissibility of cut, we only have to remember that $[t/x]A$ is structurally a subformula of $\forall x{:}\tau.\ A$ and $\exists x{:}\tau.\ A$.

4 Two Logical Examples: Quantifier Exchange

We give[1] two short logical examples which illustrate the quantifiers can be exchanged in one direction but not the other. We use here an uninterpreted

[1] Actually, we skipped these during lecture; they are provided here as bonus material.

type ι about which we make no assumptions.

$$\forall x{:}\iota.\, \exists y{:}\iota.\, A(x,y) \not\vdash \exists y{:}\iota.\, \forall x{:}\iota.\, A(x,y)$$
$$\exists y{:}\iota.\, \forall x{:}\iota.\, A(x,y) \vdash \forall x{:}\iota.\, \exists y{:}\iota.\, A(x,y)$$

Note that $A(x,y)$ is an arbitrary proposition, possibly depending on x and y We start with the first example:

$$\vdots$$
$$\forall x{:}\iota.\, \exists y{:}\iota.\, A(x,y) \vdash \exists y{:}\iota.\, \forall x{:}\iota.\, A(x,y)$$

At this point neither $\forall L$ or $\exists R$ apply, unless we make as assumption about the type ι. Let's assume it has at least one element a, which means we rewrite the judgment as

$$\vdots$$
$$a{:}\iota\ ;\ \forall x{:}\iota.\, \exists y{:}\iota.\, A(x,y) \vdash \exists y{:}\iota.\, \forall x{:}\iota.\, A(x,y)$$

Now we have to make a decision whether to start on the left or on the right. Both attempts will eventually fail to produce a proof—we show only one.

$$\dfrac{\begin{array}{c}\vdots\\ a{:}\iota\ ;\ \exists y{:}\iota.\, A(a,y) \vdash \exists y{:}\iota.\, \forall x{:}\iota.\, A(x,y)\end{array}}{a{:}\iota\ ;\ \forall x{:}\iota.\, \exists y{:}\iota.\, A(x,y) \vdash \exists y{:}\iota.\, \forall x{:}\iota.\, A(x,y)}\ \forall L$$

Next we use $\exists L$, which is actually invertible so we don't have to look for other rules.

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ a{:}\iota, b{:}\iota\ ;\ A(a,b) \vdash \exists y{:}\iota.\, \forall x{:}\iota.\, A(x,y)\end{array}}{a{:}\iota\ ;\ \exists y{:}\iota.\, A(a,y) \vdash \exists y{:}\iota.\, \forall x{:}\iota.\, A(x,y)}\ \exists L^b}{a{:}\iota\ ;\ \forall x{:}\iota.\, \exists y{:}\iota.\, A(x,y) \vdash \exists y{:}\iota.\, \forall x{:}\iota.\, A(x,y)}\ \forall L$$

Looking ahead, we can see the only promising witness to pick for y on the right is a

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ a{:}\iota, b{:}\iota\ ;\ A(a,b) \vdash \forall x{:}\iota.\, A(a,y)\end{array}}{a{:}\iota, b{:}\iota\ ;\ A(a,b) \vdash \exists y{:}\iota.\, \forall x{:}\iota.\, A(x,y)}\ \exists R}{a{:}\iota\ ;\ \exists y{:}\iota.\, A(a,y) \vdash \exists y{:}\iota.\, \forall x{:}\iota.\, A(x,y)}\ \exists L^b}{a{:}\iota\ ;\ \forall x{:}\iota.\, \exists y{:}\iota.\, A(x,y) \vdash \exists y{:}\iota.\, \forall x{:}\iota.\, A(x,y)}\ \forall L$$

Using the $\forall R$ rule however must pick a *new* parameter c. We cannot reuse B. The resulting sequent therefore has no proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{no rule applies}}{a{:}\iota, b{:}\iota, c{:}\iota \;;\; A(a,b) \vdash A(a,c)}\,\forall R^c}{a{:}\iota, b{:}\iota \;;\; A(a,b) \vdash \forall x{:}\iota.\, A(a,y)}\,\exists R}{a{:}\iota, b{:}\iota \;;\; A(a,b) \vdash \exists y{:}\iota.\, \forall x{:}\iota.\, A(x,y)}\,\exists L^b}{a{:}\iota \;;\; \exists y{:}\iota.\, A(a,y) \vdash \exists y{:}\iota.\, \forall x{:}\iota.\, A(x,y)}\,\forall L}{a{:}\iota \;;\; \forall x{:}\iota.\, \exists y{:}\iota.\, A(x,y) \vdash \exists y{:}\iota.\, \forall x{:}\iota.\, A(x,y)}$$

This does not constitute a formal proof that the proposed entailment does not hold, but with cut elimination (which does continue to hold) and refuting all possible attempt, it would turn into one.

The provable example is more straightforward and we show only the completed proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a{:}\iota, b{:}\iota \;;\; A(a,b) \vdash A(a,b)}\,\mathsf{id}}{a{:}\iota, b{:}\iota \;;\; \forall x{:}\iota.\, A(x,b) \vdash A(a,b)}\,\forall L}{a{:}\iota, b{:}\iota \;;\; \forall x{:}\iota.\, A(x,b) \vdash \exists y{:}\iota\, A(a,y)}\,\exists R}{a{:}\iota \;;\; \exists y{:}\iota.\, \forall x{:}\iota.\, A(x,y) \vdash \exists y{:}\iota\, A(a,y)}\,\exists L^b}{\exists y{:}\iota.\, \forall x{:}\iota.\, A(x,y) \vdash \forall x{:}\iota.\, \exists y{:}\iota.\, A(x,y)}\,\forall R^a$$

5 A Computational Example: Parallel Insertion Sort

As a computational example we will use a parallel form of insertion sort. We add the elements of a list, one by one, into a priority queue and then remove them in order.

$$\begin{aligned}\mathsf{pqueue} = \&\{\; & \mathsf{ins} : \forall x{:}\mathsf{int}.\, \mathsf{pqueue}, \\ & \mathsf{del} : \oplus\{\mathsf{none} : \mathbf{1}, \mathsf{some} : \exists x{:}\mathsf{int}.\, \mathsf{pqueue}\}\}\end{aligned}$$

We see that the quantified variables do not occur in our language of types. This is because we have not yet introduced *type families*, which correspond to logical predicates. When there is no such dependency, we simplify the notation and write

$$\begin{array}{rcll} \tau \to A & = & \forall x{:}\tau.\, A & \text{for } x \text{ not in } A \\ \tau \wedge A & = & \exists x{:}\tau.\, A & \text{for } x \text{ not in } A \end{array}$$

The type of priority queues then looks slightly simpler.

$$\begin{array}{l}\mathsf{pqueue} = \&\{\ \mathsf{ins} : \mathsf{int} \to \mathsf{pqueue},\\ \qquad\qquad \mathsf{del} : \oplus\{\mathsf{none} : \mathbf{1}, \mathsf{some} : \mathsf{int} \wedge \mathsf{pqueue}\}\}\end{array}$$

We have two process definitions: one to create an empty priority queue and one holding an element n and a channel q referencing the remainder of the queue.

$$\begin{array}{rcl} \cdot\ ;\ \cdot & \vdash & \mathit{empty} :: (p : \mathsf{pqueue}) \\ n{:}\mathsf{int}\ ;\ q{:}\mathsf{pqueue} & \vdash & \mathit{elem} :: (p : \mathsf{pqueue}) \end{array}$$

First, the definition of *empty*, which very much follows our previous implementation of stacks, queues, and stores.

$$\cdot\ ;\ \cdot \vdash \mathit{empty} :: (p : \mathsf{pqueue})$$

$$\begin{array}{l} p \leftarrow \mathit{empty} = \\ \quad \mathsf{case}\ p\ (\mathsf{ins} \Rightarrow n \leftarrow \mathsf{recv}\ p\ ; \\ \qquad\qquad\quad e \leftarrow \mathit{empty}\ ; \\ \qquad\qquad\quad p \leftarrow \mathit{elem}\ n \leftarrow e \\ \qquad\ \mid \mathsf{del} \Rightarrow p.\mathsf{none}\ ;\ \mathsf{close}\ p) \end{array}$$

For the definition of *elem* we indicate the dependence on a parameter n:int by writing elem n on the left-hand side of the definition. We assume functions *max* and *min* on integers.

$$n{:}\mathsf{int}\ ;\ q{:}\mathsf{pqueue} \vdash \mathit{elem} :: (p : \mathsf{pqueue})$$

$$\begin{array}{l} p \leftarrow \mathit{elem}\ n \leftarrow q = \\ \quad \mathsf{case}\ p\ (\mathsf{ins} \Rightarrow k \leftarrow \mathsf{recv}\ p\ ; \\ \qquad\qquad\quad q.\mathsf{ins}\ ;\ \mathsf{send}\ q\ (\mathit{max}\ n\ k)\ ; \\ \qquad\qquad\quad p \leftarrow \mathit{elem}\ (\mathit{min}\ n\ k) \leftarrow q \\ \qquad\ \mid \mathsf{del} \Rightarrow p.\mathsf{some}\ ;\ \mathsf{send}\ p\ n\ ; \\ \qquad\qquad\quad p \leftarrow q) \end{array}$$

Even though we reuse the syntax for sending and receiving of channels, sending and receiving of basic data values has a different semantics. In particular, they are not subject to linearity or order even though this example lies squarely within ordered logic. The fact that both n and k are used twice in the ins branch is perfectly legal. The type checker, of course, will know through the assigned type whether variables stand for data values or channels. Current implementations track this more obviously by distinguishing channels syntactically from ordinary variables.

To sort a list in ascending order, we add all the elements from the list into the priority queue, and then remove them all. The lists are slightly different from the lists before because they hold integers, not channels.

$$\mathsf{list} = \oplus\{\mathsf{cons} : \mathsf{int} \wedge \mathsf{list}, \mathsf{nil} : \mathbf{1}\}$$

We have three process definitions: *sort* tieing everything together *enqueue* which transfers the elements from the input list to a priority queue, and *dequeue* which transfers the elements from the priority to the output list.

$$\begin{array}{rcl} k{:}\mathsf{list} & \vdash & \mathit{sort} :: (l : \mathsf{list}) \\ (k{:}\mathsf{list})\ (p{:}\mathsf{pqueue}) & \vdash & \mathit{enqueue} :: (l : \mathsf{list}) \\ p{:}\mathsf{pqueue} & \vdash & \mathit{dequeue} :: (l : \mathsf{list}) \end{array}$$

$k{:}\mathsf{list} \vdash \mathit{sort} :: (l : \mathsf{list})$
$l \leftarrow \mathit{sort} \leftarrow k =$
  $p \leftarrow \mathit{empty}$;
  $l \leftarrow \mathit{enqueue} \leftarrow k\ p$

$(k{:}\mathsf{list})\ (p{:}\mathsf{pqueue}) \vdash \mathit{enqueue} :: (l : \mathsf{list})$

$l \leftarrow \mathit{enqueue} \leftarrow k\ p =$
  $\mathsf{case}\ k\ (\mathsf{cons} \Rightarrow n \leftarrow \mathsf{recv}\ k$;
    $p.\mathsf{ins}$; $\mathsf{send}\ p\ k$;
    $l \leftarrow \mathit{enqueue} \leftarrow k\ p$
  $\mid \mathsf{nil} \Rightarrow\ \mathsf{wait}\ k$;
    $l \leftarrow \mathit{dequeue} \leftarrow p)$

$p{:}\mathsf{pqueue} \vdash \mathit{dequeue} :: (l : \mathsf{list})$

$l \leftarrow \mathit{dequeue} \leftarrow p =$
  $p.\mathsf{del}$;
  $\mathsf{case}\ p\ (\mathsf{none} \Rightarrow\ \mathsf{wait}\ p$;
    $l.\mathsf{nil}$; $\mathsf{close}\ l$
  $\mid \mathsf{some} \Rightarrow n \leftarrow \mathsf{recv}\ p$;
    $l.\mathsf{cons}$; $\mathsf{send}\ l\ n$;
    $l \leftarrow \mathit{dequeue} \leftarrow p)$

Let's do a quick parallel complexity analysis. Our implementation of priority queues is structurally the same as the previous implementation of queues, stacks, and stores. Therefore, the reaction time for both insertions and deletions is constant. To sort a list of length n we perform n insertions

followed by n deletions, which gives $O(n)$ throughput and latency for sorting. The total amount of work is $O(n^2)$ since each insertion will take $O(k)$ work, where k is the length of the list so far, while each deletion takes a constant amount of work.

6 Structural Intuitionistic Logic

We now introduce plaing intuitionistic logic, with the structural rules of exchange, weakening, and contraction. First, we build in exchange with the same technique as for linear logic: The collection of antecedents is considered as a multiset where the order of the antecedents does not matter. We write $\Gamma \vdash A$ for such a sequent in structural intuitionistic logic.

For weaking and contraction, there are two standard techniques. One is to add the following explicit rules:

$$\frac{\Gamma \vdash C}{\Gamma, A \vdash C}\ \mathsf{weaken} \qquad \frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}\ \mathsf{contract}$$

These are structural rules and independent of what any particular proposition A might be.

There are two downsides to this simple and elegant formulation of structural logic. One is that it introduces a lot of nondeterminism into proof construction, since at any point we could decide to apply either weakening and contraction. Contraction in particular is prolific, since it could be immediately applied again, and again, etc. The other downside it that the rule of contraction significantly complicates the proof of admissibility of cut.

An alternative approach is to build weakening and contraction into the rules themselves. *Weakening* is incorporated by allowing unused antecedent in the identity rule (and also the $\top R$ and $\bot L$ rules later; see Section 6). *Contraction* is incorporated by propagating all antecedents to all premises of multi-premise rules. We illustrate with id, cut and the rules for implication $A \to B$.

$$\frac{}{\Gamma, A \vdash A}\ \mathsf{id}_A \qquad \frac{\Gamma \vdash A \quad \Gamma, A \vdash C}{\Gamma \vdash C}\ \mathsf{cut}_A$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \to R \qquad \frac{\Gamma, A \to B \vdash A \quad \Gamma, A \to B, B \vdash C}{\Gamma, A \to B \vdash C} \to L$$

Notice in particular how the antecedent $A \to B$ in the $\to L$ rule persists in both premises. This is in fact a good intuition for this formulation of

the logic: antecedents are *persistent*, just as we had persistent propositions when discussing logical inference. From the perspective of provability, some of these antecedents are redundant. For example, in the second premise of $\to L$, the persistent antecedent $A \to B$ is no longer needed since we already have the strong assumption B. These kinds of optimizations often interfere with our operational interpretation of the logic. Therefore, they should be carefully considered in each situation rather than being built into the very definition of the logic.

A first, important observation is that weakening is an admissible rule. Moreover, the proof resulting from weakening has *exactly the same structure* as the proof before weakening, because we only adjoin an extra unused antecedent to every sequent. This means if we perform structural induction over proofs, we can freely apply weakening and still apply the induction hypothesis.

Theorem 1 (Admissibility of Weakening and Contraction)

$$\frac{\Gamma \vdash C}{\Gamma, A \vdash C}\ \mathsf{weaken} \qquad \frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}\ \mathsf{contract}$$

Moreover, in both cases the proof of the conclusion is structurally identical to the proof of the premise.

Proof: Formally, by structural induction on the given proofs. For weakening, we adjoin an extra, unused antecedent A to every sequent in the given proof to obtain the proof of the conclusion. For contraction, we replace every use of either one of the two copies of A in every sequent of the given proof by a use of the same, single A to obtain the proof of the conclusion. □

There are some new and interesting cases in the proof of admissibility of cut. We show only two: a new case for the identity, and the principal case for implication. As before, we write $\Gamma \Vdash A$ for a cut-free proof of A from Γ.

Theorem 2 (Admissibility of Cut in the Cut-Free Structural Sequent Calculus)

$$\frac{\Gamma \Vdash A \quad \Gamma, A \Vdash C}{\Gamma \Vdash C}\ \mathsf{cut}$$

Proof: By a nested induction, first on the structure of the cut formula A, and second on the proofs $\mathcal{D}$ and $\mathcal{E}$ of the two premises. The proof breaks down into the same classes of cases as before: identity, principal, and commutative cases.

Case:

$$\overset{\mathcal{D}}{\Gamma', B \Vdash A} \quad \text{arbitrary, and} \quad \mathcal{E} = \frac{}{\Gamma', B, A \Vdash B}\ \mathsf{id}_B$$

where $\Gamma = (\Gamma', B)$. We need to show $\Gamma', B \Vdash B$ which follows by id_B. What is interesting about this case is that $\mathcal{D}$ is dropped entirely because it proves an unused proposition A.

Case:

$$\mathcal{D} = \frac{\overset{\mathcal{D}_1}{\Gamma, A_2 \Vdash A_1}}{\Gamma \Vdash A_2 \to A_1}\ {\to}R \quad \text{and} \quad \mathcal{E} = \frac{\overset{\mathcal{E}_2}{\Gamma, A_2 \to A_1 \Vdash A_2} \quad \overset{\mathcal{E}_1}{\Gamma, A_2 \to A_1, A_1 \Vdash C}}{\Gamma, A_2 \to A_1 \Vdash C}\ {\to}L$$

We need to show $\Gamma \Vdash C$. We would like to cut $\mathcal{E}_2$ with $\mathcal{D}_1$ and $\mathcal{D}_1$ with $\mathcal{E}_1$, but there will be an extra copy of $A_2 \to A_1$ left over. Se we first apply what is called a *cross-cut*, applying the induction hypothesis to *all of* $\mathcal{D}$ and $\mathcal{E}_2$ and also $\mathcal{E}_1$ to eliminate the persistent copies of $A_2 \to A_1$.

$\mathcal{E}_2' = \Gamma \Vdash A_2$	By i.h. on $A_2 \to A_1$, $\mathcal{D}$, and $\mathcal{E}_2$
$\mathcal{D}' = \Gamma, A_1 \Vdash A_2 \to A_1$	By weakening on $\mathcal{D}'$
$\mathcal{E}_1' = \Gamma, A_1 \Vdash C$	By i.h. on $A_2 \to A_2$, $\mathcal{D}'$, and $\mathcal{E}_1$
$\mathcal{D}_1' = \Gamma \Vdash A_1$	By i.h. on A_2, $\mathcal{E}_2'$, and $\mathcal{D}_1$
$\mathcal{F} = \Gamma \Vdash C$	By i.h. on A_1, $\mathcal{D}_1'$, and $\mathcal{E}_1'$

□

Please make sure you understand why each appeal to the induction hypothesis in this proof is justified.

There is not much more to say about the rules for the other connectives. What may be interesting is that, logically speaking, the two forms of conjunction $A \otimes B$ and $A \mathbin{\&} B$ "collapse" in that they are now logically equivalent. However, computationally they can still be different, so from an intuitionistic perspective we should be careful before eliminating one or the other. We therefore avoid writing $A \wedge B$ for conjunction, prefering $A \mathbin{\&} B$ for the conjunction with the same notation in linear logic, and $A \times B$

for that corresponding to $A \otimes B$ in linear logic.

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}\,\&R \qquad \frac{\Gamma, A \,\&\, B, A \vdash C}{\Gamma, A \,\&\, B \vdash C}\,\&L_1 \quad \frac{\Gamma, A \,\&\, B, B \vdash C}{\Gamma, A \,\&\, B \vdash C}\,\&L_2$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \times B}\,\times R \qquad \frac{\Gamma, A \times B, A, B \vdash C}{\Gamma, A \times B \vdash C}\,\times L$$

Disjunction $A \vee B$ is unsurprising and has the rules completely analogous to the linear case for $A \oplus B$, except for the persistence of the disjunction in the $\vee L$ case.

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\,\vee R_1 \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\,\vee R_2 \qquad \frac{\Gamma, A \vee B, A \vdash C \quad \Gamma, A \vee B, B \vdash C}{\Gamma, A \vee B \vdash C}\,\vee L$$

Finally, the cases for $\top$ and $\bot$, which are the units $\&$ and $\vee$, respectively.

$$\frac{}{\Gamma \vdash \top}\,\top R \qquad \text{no } \top L \text{ rule}$$

$$\text{no } \bot R \text{ rule} \qquad \frac{}{\Gamma, \bot \vdash C}\,\bot L$$

Exercises

Exercise 1 Show the cut reduction for universal quantification.

Exercise 2 Show the identity expansion for existential quantification.

Exercise 3 Show the identity expansion for universal quantification.

Exercise 4 *Show two cases in the proof of admissibility of cut for quantifiers in ordered logic.*

1. *The principal case for the cut formula $\exists x{:}\tau.\ A$.*
2. *A commutative case for an $\exists L$ rule on a side formula of the cut.*

Exercise 5 The parallel insertion sort in Section 5 uses the priority queue in a restricted manner: first, it only inserts integers and then it only deletes integers.

Create more precise types for *enqueue* and *dequeue* which enforce this protocol. Under which conditions or rules do *empty*, *elem*, *enqueue*, *dequeue* and *sort* type-check against these more precise types?

Exercise 6 Implement set of integers as binary search trees. Use the interface

$$
\begin{array}{l}
\mathsf{list} = \oplus\{\mathsf{cons} : \mathsf{int} \wedge \mathsf{list}, \mathsf{nil} : \mathbf{1}\} \\
\mathsf{bool} = \oplus\{\mathsf{true} : \mathbf{1}, \mathsf{false} : \mathbf{1}\} \\
\mathsf{tree} = \&\{\ \mathsf{ins} : \mathsf{int} \to \mathsf{tree}, \\
\qquad\qquad \mathsf{member} : \mathsf{int} \to \mathsf{bool} \bullet \mathsf{tree}, \\
\qquad\qquad \mathsf{to\_list} : \mathsf{list}\}
\end{array}
$$

where to_list should create a sorted list in ascending order. The tree does not need to be balanced. Also implement one additional binary operation on trees such as union or intersection.

Exercise 7 We reconsider the principal case for $A_2 \to A_1$ in the proof of admissibility of cut in structural intutionistic logic. Try to apply the induction hypothesis first to $\mathcal{E}_2$ and (a weakened form of) $\mathcal{D}_1$ and then to (a weakened form of the result) and $\mathcal{E}_1$. Then eliminate the extra copy of $A_2 \to A_1$ from the result with another appeal to the induction hypothesis with $\mathcal{D}$. Poinpoint exactly where this argumentation goes wrong.

Exercise 8 Prove the principal case for $A_1 \times A_2$ in the proof of admissibility of cut for structural intuitionistic logic.