Lecture Notes on Deductive Inference

15-816: Substructural Logics
Frank Pfenning

Lecture 1
August 30, 2016

According to Wikipedia[1], the ultimate authority on *everything*:

> **Logic** [. . .] is the formal systematic study of the principles of valid inference and correct reasoning.

We therefore begin the course with the study of deductive inference. This starting point requires surprisingly little machinery and is sufficient to understand the central idea behind substructural logics, including linear logic. We aim to develop all other concepts and properties of linear logic systematically from this seed.

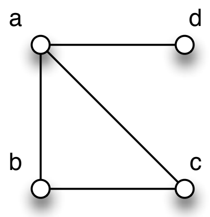
Our approach is quite different from that of Girard [Gir87], whose discovery of linear logic originated from semantic considerations in the theory of programming languages. We arrive at almost the same spot. The convergence of multiple explanations of the same phenomena is further evidence for the fundamental importance of linear logic. At some point in the course we will explicitly talk about the relationship between Girard's linear logic and our reconstruction of it.

1 Example: Reasoning about Graphs

As a first example we consider graphs. Mathematically, (undirected) graphs are often defined as consisting of a set of vertices V and a set of edges E, where E is a set of unordered pairs of vertices.

[1]in January 2012

In the language of logic, we represent the nodes (vertices) as *constants* ($\mathsf{a}, \mathsf{b}, \ldots$) and a unary predicate $\mathsf{node}(x)$ that holds for all vertices. The edges are represented with a binary *predicate* $\mathsf{edge}(x, y)$ relating connected nodes.



The sample graph above could be represented by the *propositions*

$$\mathsf{node}(\mathsf{a}), \mathsf{node}(\mathsf{b}), \mathsf{node}(\mathsf{c}), \mathsf{node}(\mathsf{d}),$$
$$\mathsf{edge}(\mathsf{a}, \mathsf{b}), \mathsf{edge}(\mathsf{b}, \mathsf{c}), \mathsf{edge}(\mathsf{a}, \mathsf{c}), \mathsf{edge}(\mathsf{a}, \mathsf{d})$$

One mismatch one may notice immediately is that the edges in the picture seem to be undirected, while the representation of the edges is not symmetric (for example, $\mathsf{edge}(\mathsf{b}, \mathsf{a})$ is not there). We can repair this inadequacy by providing a *rule of inference* postulating that the edge relation is symmetric.

$$\frac{\mathsf{edge}(x, y)}{\mathsf{edge}(y, x)}\ \mathsf{sym}$$

We can apply this rule of inference to the fact $\mathsf{edge}(\mathsf{a}, \mathsf{b})$ to deduce $\mathsf{edge}(\mathsf{b}, \mathsf{a})$. In this application we instantiated the *schematic variables* x and y with a and b. We will typeset schematic variables in italics to distinguish them from constants. The propositions above the horizontal line are called the *premises* of the rule, the propositions below the line are called *conclusions*. This example rule has only one premise and one conclusion. sym is the *name* or *label* of the rule. We often omit rule names if there is no specific need to refer to the rules.

From this single rule and the facts describing the initial graph, we can now deduce the following additional facts:

$$\mathsf{edge}(\mathsf{b}, \mathsf{a}), \mathsf{edge}(\mathsf{c}, \mathsf{b}), \mathsf{edge}(\mathsf{c}, \mathsf{a}), \mathsf{edge}(\mathsf{d}, \mathsf{a})$$

Having devised a logical representation for graphs, we now define a relation over graphs. We write $\mathsf{path}(x, y)$ if there is a path through the graph from x to y. A first path has length zero and goes from a node to itself:

$$\frac{\mathsf{node}(x)}{\mathsf{path}(x, x)}\ \mathsf{refl}$$

The following rule says is that we can extend an existing path by following an edge.

$$\frac{\mathsf{path}(x, y) \quad \mathsf{edge}(y, z)}{\mathsf{path}(x, z)}\ \mathsf{step}$$

From the representation of our example graph, when can then supply the following proof that there is a path from c to d:

$$\cfrac{\cfrac{\cfrac{\mathsf{node}(\mathsf{c})}{\mathsf{path}(\mathsf{c},\mathsf{c})}\ \mathsf{refl} \quad \cfrac{\mathsf{edge}(\mathsf{a},\mathsf{c})}{\mathsf{edge}(\mathsf{c},\mathsf{a})}\ \mathsf{sym}}{\mathsf{path}(\mathsf{c},\mathsf{a})}\ \mathsf{step} \qquad \mathsf{edge}(\mathsf{a},\mathsf{d})}{\mathsf{path}(\mathsf{c},\mathsf{d})}\ \mathsf{step}$$

We can examine the proof and see that it carries some information. It is not just there to convince us that there is a path from c to d, but it tells us the path. The path goes from c to a and then from a to d. This is an example of *constructive content* in a proof, and we will see many other examples. For the system we have so far it will be true in general that we can read off a path from a proof, and if we have a path in mind we can always construct a proof. But with the rules we chose, some paths do not correspond to a unique proof. Think about why before turning the page. . .

Proofs are not unique because we can go from $\mathsf{edge}(\mathsf{c},\mathsf{a})$ back to $\mathsf{edge}(\mathsf{a},\mathsf{c})$ and back $\mathsf{edge}(\mathsf{c},\mathsf{a})$, and so on, producing infinitely many proofs of $\mathsf{edge}(\mathsf{c},\mathsf{a})$. Here are two techniques to eliminate this ambiguity which are of general interest.

One solution uses a new predicate $\mathsf{pre\_edge}$ and defines all the edges we have used so far as pre-edges. Then we have two rules

$$\frac{\mathsf{pre\_edge}(x,y)}{\mathsf{edge}(x,y)} \qquad \frac{\mathsf{pre\_edge}(x,y)}{\mathsf{edge}(y,x)}$$

To explain the second solution, we begin by making proofs explicit as objects. We write $e : \mathsf{edge}(x,y)$ to name the edge from x to y, and $p : \mathsf{path}(x,y)$ for the path from x to y. Where the proofs are not interesting, we just use an underscore _. We annotate our rules accordingly, which now read:

$$\frac{e : \mathsf{edge}(x,y)}{e^{-1} : \mathsf{edge}(y,x)}\,\mathsf{sym} \qquad \frac{\_ : \mathsf{node}(x)}{\mathsf{r} : \mathsf{path}(x,x)}\,\mathsf{refl} \qquad \frac{p : \mathsf{path}(x,y) \quad e : \mathsf{edge}(y,z)}{p \cdot e : \mathsf{path}(x,z)}\,\mathsf{step}$$

Our initial knowledge base would be annotated

$$\begin{array}{l} \_ : \mathsf{node}(\mathsf{a}), \_ : \mathsf{node}(\mathsf{b}), \_ : \mathsf{node}(\mathsf{c}), \_ : \mathsf{node}(\mathsf{d}), \\ \mathsf{e}_{ab} : \mathsf{edge}(\mathsf{a},\mathsf{b}), \mathsf{e}_{bc} : \mathsf{edge}(\mathsf{b},\mathsf{c}), \mathsf{e}_{ac} : \mathsf{edge}(\mathsf{a},\mathsf{c}), \mathsf{e}_{ad} : \mathsf{edge}(\mathsf{a},\mathsf{d}) \end{array}$$

and then the earlier proof would carry the path information:

$$\frac{\dfrac{\dfrac{\_ : \mathsf{node}(\mathsf{c})}{\mathsf{r} : \mathsf{path}(\mathsf{c},\mathsf{c})}\,\mathsf{refl} \quad \dfrac{\mathsf{e}_{ac} : \mathsf{edge}(\mathsf{a},\mathsf{c})}{\mathsf{e}_{ac}^{-1} : \mathsf{edge}(\mathsf{c},\mathsf{a})}\,\mathsf{sym}}{\mathsf{r} \cdot \mathsf{e}_{ac}^{-1} : \mathsf{path}(\mathsf{c},\mathsf{a})}\,\mathsf{step} \qquad \mathsf{e}_{ad} : \mathsf{edge}(\mathsf{a},\mathsf{d})}{\mathsf{r} \cdot \mathsf{e}_{ac}^{-1} \cdot \mathsf{e}_{ad} : \mathsf{path}(\mathsf{c},\mathsf{d})}\,\mathsf{step}$$

To force uniqueness of proofs of the edge predicate we can assert the equation

$$(e^{-1})^{-1} = e$$

so that we do not distinguish between the original edge and one that has been reversed twice. Note that for any given path there will still be many formal deductions using the inference rules, but they would lead to the same proof object.

Both solutions, restructuring the predicates or identifying proofs, are common solutions to create better correspondences between proofs and the properties of information we would like them to convey.

As a last example here, let's consider what happens when we apply inference indiscriminately, trying to learn as much as possible about the paths in a graph. Unfortunately, even if the edge relation has unique proofs, any cycle in the graph will lead to infinitely many paths and therefore infinitely many proofs. So inference will never stop with complete immediate knowledge of all paths.

One solution[2] is to retreat and say we do not care about the path, we just want to know if there exists a path between any two nodes. In that case inference will reach a point of *saturation*, that is, any further inference would have a conclusion already in our database of facts. The original inference system achieves that, or we can stipulate that any two paths between the same two vertices x and y should be identified:

$$p = q : \mathsf{path}(x, y)$$

Again, inference would reach saturation because after a time new paths would be equal to the one we already have and not recorded as a separate fact.

Another solution would be to allow only non-repetitive paths, by which we mean that edges are not being reused. Then we would have to modify our step rule

$$\frac{p : \mathsf{path}(x,y) \quad e : \mathsf{edge}(y,z) \quad \_ : \mathsf{notin}(e,p)}{p \cdot e : \mathsf{path}(x,z)}\ \mathsf{step}$$

and define a predicate $\mathsf{notin}(e,p)$ which is true if e is not in the path p. As was noted in class, however, this crosses a threshold: propositions (like notin) now refer to proofs. This is the distinction between *logic*, in which proofs entirely separate from propositions, and *type theory* in which propositions can refer to proofs. This particular example shows that type theory can be more expressive in certain helpful ways.

A third solution would be to somehow *consume* the edges during inference so that we cannot reuse them later. Because there is no restriction on how often primitive or derived facts are used in a proof, we will need the expressive power of *substructural logics* to follow this idea.

[2]not discussed in lecture

2 Example: Coin Exchange

So far, deductive inference has always accumulated knowledge since propositions whose truth we are already aware of remain true. Linear logic arises from a simple observation:

Truth is ephemeral.

For example, while giving this lecture "*Frank is holding a whiteboard marker*" was true, and right now it is (most likely) not. So truth changes over time, and this phenomenon is studied with *temporal logic*. In *linear logic* we are instead concerned with the change of truth with a *change of state*. We model this in a very simple way: when an inference rule is applied we *consume* the propositions used as premises and *produce* the propositions in the conclusions, thereby effecting an overall change in state.

As an example, we consider *nickels* (n) worth 5¢, *dimes* (d) worth 10¢, and *quarters* (q) worth 25¢. We have the following rules for exchange between them

$$\frac{\mathsf{d} \quad \mathsf{d} \quad \mathsf{n}}{\mathsf{q}}\,Q \qquad \frac{\mathsf{q}}{\mathsf{d} \quad \mathsf{d} \quad \mathsf{n}}\,Q^{-1} \qquad \frac{\mathsf{n} \quad \mathsf{n}}{\mathsf{d}}\,D \qquad \frac{\mathsf{d}}{\mathsf{n} \quad \mathsf{n}}\,D^{-1}$$

The second and fourth rules are the first rules we have seen with more than one conclusion. Inference now changes state. For example, if we have three dimes and a nickel, the state would be written as

$$\mathsf{d}, \mathsf{d}, \mathsf{d}, \mathsf{n}$$

Applying the first rule, we can turn two dimes and a nickel into a quarter to get the state

$$\mathsf{d}, \mathsf{q}$$

Note that the total value of the coins (35¢) remains unchanged, which is the point of a coin exchange. One way to write down the inference is to cross out the propositions that are consumed and add the ones that are produced. In the above example we would then write something like

$$\mathsf{d}, \mathsf{d}, \mathsf{d}, \mathsf{n} \quad \rightsquigarrow \quad \cancel{\mathsf{d}}, \cancel{\mathsf{d}}, \mathsf{d}, \cancel{\mathsf{n}}, \mathsf{q}$$

In order to understand the meaning of proof, consider how to change three dimes into a quarter and a nickel: first, we change one dime into two nickels, and then the other two dimes and one of the nickels into a quarter. As two state transitions:

$$\mathsf{d}, \mathsf{d}, \mathsf{d} \quad \rightsquigarrow \quad \mathsf{d}, \mathsf{d}, \cancel{\mathsf{d}}, \mathsf{n}, \mathsf{n} \quad \rightsquigarrow \quad \cancel{\mathsf{d}}, \cancel{\mathsf{d}}, \cancel{\mathsf{d}}, \cancel{\mathsf{n}}, \mathsf{q}, \mathsf{n}$$

Using inference rule notation, this deduction is shown on the left, and the corresponding derived rule of inference on the right.

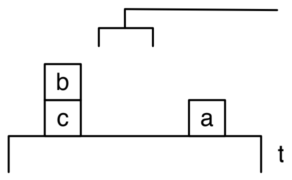
$$\dfrac{\mathsf{d} \quad \mathsf{d} \quad \dfrac{\mathsf{d}}{\mathsf{n} \quad\quad \mathsf{n}}}{\mathsf{q}} \qquad\qquad \dfrac{\mathsf{d} \quad \mathsf{d} \quad \mathsf{d}}{\mathsf{q} \quad \mathsf{n}}$$

To summarize: we can change the very nature of inference if we *consume* the propositions used in the premise to *produce* the propositions in the conclusions. This is the foundation of linear logic and we therefore call it *linear inference*. The requisite pithy saying to remind ourselves of this:[3]

$$\boxed{\textit{Linear inference can change the world.}}$$

3 Example: Blocks World

Next we consider *blocks world*, which is a venerable example in the history of artificial intelligence. We have blocks ($\mathsf{a}, \mathsf{b}, \ldots$) stacked on a table ($\mathsf{t}$). We also have a robot hand which may pick blocks that are not obstructed and put them down on the table or some other block. We assume that the hand can hold just one block.



We represent the state in linear logic using the following predicates

$\mathsf{on}(x, y)$	Block x is on top of y
empty	Hand is empty
$\mathsf{holds}(x)$	Hand holds block x

[3]with a tip of the hat to Phil Wadler

For example, the state above would be represented by

$$\mathsf{empty}, \mathsf{on}(\mathsf{c}, \mathsf{t}), \mathsf{on}(\mathsf{b}, \mathsf{c}), \mathsf{on}(\mathsf{a}, \mathsf{t})$$

The two preconditions for picking up a block x are that the hand is empty, and that nothing is on top x. In (ordinary) logic, we might try to express this last condition as $\neg\exists y.\, \mathsf{on}(y, x)$. However, negation is somewhat problematic in linear logic. For example, it is true in the above state the $\neg\mathsf{on}(\mathsf{b}, \mathsf{t})$. However, after two moves (picking up b and then putting b on the table) it could become true, and we might have created a contradiction! Moreover, proving $\neg\exists y.\mathsf{on}(y, x)$ in linear logic would consume whatever resources we refer to, which is problematic since our intention is just to check a property of the state without changing it. This kind of problem is common in applications of linear logic, so we have developed some techniques of addressing it.

A common technique to avoid such paradoxes is to introduce additional predicates that describe properties of the state. Such predicates are maintained by the rules in order to keep the description of the state consistent. Before reading on, you might consider how you can define a new predicate, add appropriate initial facts, and then write rules for picking up and putting down blocks.

Here is one possibility. We use a new predicate $\mathsf{clear}(x)$ to express that there is no other block on top of x. In addition to the propositions above describing the initial state, we would also have

$$\mathsf{clear}(\mathsf{b}), \mathsf{clear}(\mathsf{a})$$

but *not* $\mathsf{clear}(\mathsf{c})$.

Then we only need two rules, one for picking up a block and one for putting one down:

$$\frac{\mathsf{empty} \quad \mathsf{clear}(x) \quad \mathsf{on}(x,y)}{\mathsf{holds}(x) \quad \mathsf{clear}(y)}\ \mathsf{pickup} \qquad \frac{\mathsf{holds}(x) \quad \mathsf{clear}(y)}{\mathsf{empty} \quad \mathsf{clear}(x) \quad \mathsf{on}(x,y)}\ \mathsf{putdown}$$

The two rules are inverses of each other, which makes sense since picking up or putting down a block are reversible actions.

The encoding so far would work under the condition that there is a fixed number of spots on the table where blocks can be deposited, and each is explicitly represented by a $\mathsf{clear}(\mathsf{t})$ proposition. We cannot accidentally pick up the table because it is not *on* anything else, so the last premise of the pickup rule cannot be satisfied when x is t.

If we instead want to follow the usual assumption assume that the table is big enough for arbitrarily many blocks we can revise our encoding in two ways. One would be to introduce a special predicate $\mathsf{ontable}(x)$ and reserve $\mathsf{on}(x,y)$ for blocks x and y. Then we need two additional rules to pick up and put down blocks on the table. Another is to have a *persistent* proposition

$$\underline{\mathsf{clear}}(\mathsf{t})$$

which *can not be consumed by inference*. The intrinsic attribute of the predicate of being persistent is indicated by underlining it.

We view whether propositions are ephemeral (linear, will be consumed by inference) or persistent (not linear, can not be consumed by inference) as intrinsic attributes of the proposition. This allows us to easily mix linear and nonlinear inference, which is a powerful tool in constructing encodings. We can continue to use any encoding in "ordinary"[4] logic but we have some new means of expression.

$$\boxed{\textit{Substructural logics generalize ordinary logics}}$$

One interesting question that arises here is whether we can use persistent facts to instantiate ephemeral premises of rules. This is the intention

[4]by which we mean "not substructural"

here: $\underline{\mathsf{clear}}(t)$ can be used as a premise of the $\mathsf{putdown}$ rule in order to put a block on the table. This is justified since a fact we can use as often as we want should certainly be usable this once. We just have to be careful *not* to consume $\underline{\mathsf{clear}}(t)$ so that it remains available for future inferences.

As before, we should examine the meaning of proofs in this example. Given some initial state, a proof describes a sequence of moves that leads to a final state. Therefore, the process of planning, when given particular goal and final states, becomes a process of proof search.

Another important aspect of problem representations in linear logic are *state invariants* that are necessary so that inference has the desired meaning. For example, there should always be *exactly* one of empty and $\mathsf{holds}(x)$ in w state, otherwise it would not conform to our problem domain and inference is potentially meaningless. Similarly, there shouldn't be cycles such as $\mathsf{on}(\mathsf{a},\mathsf{b}), \mathsf{on}(\mathsf{b},\mathsf{a})$, which does not correspond to any physically possible situation. When showing that our representations in linear logic capture what we intend, we should make such state invariants explicit and verify that they are preserved under the possible inferences. In our example, any rule application replaces empty by $\mathsf{holds}(x)$ for some x, or $\mathsf{holds}(x)$ by empty. So if the state invariant holds initially, it must hold after any inference.

4 Example: King Richard III

As a first example from literature, consider the following quote:

> *My kingdom for a horse!* — King Richard in *Richard III* by William Shakespeare

How do we represent this in linear logic? Let's fix a vocabulary:

$\mathsf{richard}$	King Richard III
$\mathsf{owns}(x,y)$	x owns y
$\mathsf{horse}(x)$	x is a horse
$\mathsf{kingdom}(x)$	x is a kingdom

The offer corresponds to a change of ownership: Richard "owns" a kingdom before (which we know to be England, but which is not part of his utterance), and another person p owns a horse, and after the swap Richard owns the horse while p owns the kingdom.

$$\frac{\mathsf{owns}(\mathsf{richard},k) \quad \underline{\mathsf{kingdom}}(k) \quad \mathsf{owns}(p,h) \quad \underline{\mathsf{horse}}(h)}{\mathsf{owns}(\mathsf{richard},h) \quad \mathsf{owns}(p,k)}$$

As is commonly the case, we model an intrinsic attribute of an object (such as h being a horse or k being a kingdom) with a persistent predicate, while ownership changes and is therefore ephemeral.

It is implied in the exclamation that the rule can only be used once, because there is only one k that qualifies as "*my kingdom*". Otherwise, he would have said "*One of my kingdoms for a horse!*". Another way to capture this aspect of the offer would be to consider the inference rule itself to be *ephemeral* and hence can only be used once. We do not have a good informal notation for such rules, but they will play a role again in the next lecture. If the rule above were persistent, it would more properly correspond to the offer "*My kingdoms for horses!*".

5 Example: Grammatical Inference

We have seen how we can add expressive power to logic simply by designating some propositions as being consumable by logical inference. We now explore going even further: we also impose an order on the propositions describing our state of knowledge. This is the fundamental idea behind the *Lambek calculus* [Lam58][5]. Lambek's goal was to describe the syntax of natural (and formal) languages and apply logical tools to problems such as parsing. We only scratch the surface of the work, just enough to appreciate how Lambek employs logical inference to describe grammatical inference. We will also see our first logical connectives, called "over" and "under", although you would be unlikely to recognize them as such at first glance.

We begin by introducing so-called *syntactic types* that describe the role of words and phrases. The primitive types we consider here are only n for *names* and s for *sentences*. We ascribe n for proper names such as *Alice* or *Bob*, but phrases that can stand for names in sentences will also have type n. s is the type of complete sentences such as *Alice works* or *Bob likes Alice*.

There are two constructors two combine types: Given types x and y, we can form $x \mathbin{/} y$ (read: x *over* y) and $x \setminus y$ (read: x *under* y). If a phrase w has type $x \mathbin{/} y$ it means that we obtain an x if we find a y to its right. For example, *poor Bob* can act as a name because we can (syntactically) use it where we use *Bob*. This means we can assign *poor* the type $n \mathbin{/} n$: If we find a name to its right the result forms a name. Conversely, $x \setminus y$ describes a phrase that acts as a y if there is an x to its *left*. For example, an intransitive

[5]I highly recommend this paper, which has stood the test of time almost 50 years after its publication.

verb like *works* will have type $n \setminus s$, because if we find a name to its left we obtain a sentences, as in *Alice works*.

This informal definition can be seen as a form of logical inference. We have the two rules

$$\frac{x \,/\, y \quad y}{x}\;\mathsf{over} \qquad\qquad \frac{y \quad y \setminus x}{x}\;\mathsf{under}$$

For these to make sense from the grammatical point of view, it is critical that the premises of the rules must be adjacent and in the right order.

Under this point of view, proofs correspond to parse trees and the words of the phrase we are trying to parse are labeled by their type. Using p and q to stand for phrases (that is, sequences of words), we would have

$$\frac{p : x \,/\, y \quad q : y}{(p\, q) : x}\;\mathsf{over} \qquad\qquad \frac{q : y \quad p : y \setminus x}{(q\, p) : x}\;\mathsf{under}$$

For example, we might start with the phrase *poor Alice works* as

$$(\mathit{poor} : n \,/\, n)\ (\mathit{Alice} : n)\ (\mathit{works} : n \setminus s)$$

In this situation, there are two possible inferences, either connecting *poor* with *Alice* or connecting *Alice* with *works*. Let's try the first one. We then obtain

$$\frac{\mathit{poor} : n \,/\, n \quad \mathit{Alice} : n}{(\mathit{poor\,Alice}) : n}\;\mathsf{over} \qquad \mathit{works} : n \setminus s$$

After one more step we obtain a complete parse

$$\frac{\dfrac{\mathit{poor} : n \,/\, n \quad \mathit{Alice} : n}{(\mathit{poor\,Alice}) : n}\;\mathsf{over} \qquad \mathit{works} : n \setminus s}{((\mathit{poor\,Alice})\ \mathit{works}) : s}\;\mathsf{under}$$

Trying the other order fails, since *poor* does not properly modify *Alice works*. Let's see how this failure arises during deduction. After one step we have

$$\mathit{poor} : n \,/\, n \qquad \frac{\mathit{Alice} : n \quad \mathit{works} : n \setminus s}{(\mathit{Alice\,works}) : s}\;\mathsf{under}$$

At this point in the deductive process, no further inference is possible.

We now consider a few more words to find appropriate types for them. The word *here* transform a sentence to another sentence when attached to the end. For example, writing the typing judgment vertically and indicating the expected parse tree by parentheses, we have

$$\begin{array}{ccc} (\textit{Alice} & \textit{works}) & \textit{here} \\ : & : & : \\ n & n \setminus s & ? \end{array}$$

We see that the type of *here* should be $s \setminus s$.

$$\begin{array}{ccc} (\textit{Alice} & \textit{works}) & \textit{here} \\ : & : & : \\ n & n \setminus s & s \setminus s \end{array}$$

or, making the inference more explicit

$$\dfrac{\dfrac{\begin{array}{c}\textit{Alice}\\ :\\ n\end{array} \quad \begin{array}{c}\textit{works}\\ :\\ n \setminus s\end{array}}{s} \quad \begin{array}{c}\textit{here}\\ :\\ s \setminus s\end{array}}{s}$$

How about *never*, which modifies an intransitive verb?

$$\begin{array}{ccc} \textit{Bob} & (\textit{never} & \textit{works}) \\ : & : & : \\ n & ? & n \setminus s \end{array}$$

We suggest you work this out before going on to the next page. . .

We see that *never* should have type $(n \setminus s) \,/\, (n \setminus s)$.

$$\begin{array}{ccc} \textit{Bob} & (\textit{never} & \textit{works}) \\ : & : & : \\ n & (n \setminus s) \,/\, (n \setminus s) & n \setminus s \end{array}$$

or, as a complete inference (= parse) tree:

$$\dfrac{\begin{array}{c}\textit{Bob} \\ : \\ n\end{array} \qquad \dfrac{\begin{array}{c}\textit{never} \\ : \\ (n \setminus s) \,/\, (n \setminus s)\end{array} \qquad \begin{array}{c}\textit{works} \\ : \\ (n \setminus s)\end{array}}{n \setminus s}}{s}$$

What about a transitive verb such as *likes*? Setting up a standard sentence

$$\begin{array}{ccc} \textit{Alice} & \textit{likes} & \textit{Bob} \\ : & : & : \\ n & ? & n \end{array}$$

it seems like there are two possibilities, depending on whether we prefer the parse *(Alice likes) Bob* or *Alice (likes Bob)*. The first yields the type $n \setminus (s \,/\, n)$, while the second yields $(n \setminus s) \,/\, n$. Using the tools of logic we will see in the next lecture that these two types are equivalent in the sense that when we can parse a phrase with the first we can always parse it with the second and vice versa. So it should not matter which type we assign, although if we think of parsing as proceeding from left to right one might have an intuitive preference for the first one.

Of course, natural language has many ambiguities and the same work can be used in different ways. For example, the work *and* can conjoin sentences (for example, *Alice works and Bob rests* with type *and* : $s \setminus (s \,/\, s)$) but also names (for example, *Alice and Bob work* with type *and* : $n \setminus (n^* \,/\, n)$ where n^* denotes plural ouns such as *women*).

Here is a little summary table, showing some of the possible types.

Word	Type	Part of Speech
works	$n \setminus s$	intransitive verb
poor	$n \,/\, n$	adjective
here	$s \setminus s$	adverb
never	$(n \setminus s) \,/\, (n \setminus s)$	adverb
likes	$n \setminus (s \,/\, n)$	transitive verb
and	$s \setminus (s \,/\, s)$	conjunction

6 Summary

We have examined the notion of *logical inference* from three different angles. The first was to accumulate knowledge by deducing new facts from facts we already knew, as in describing graphs using propositions $\mathsf{vertex}(x)$ and $\mathsf{edge}(x, y)$ and paths through graphs using $\mathsf{path}(x, y)$. Proofs here contain important information. For example, the proof that there is a path from x to y contains the actual edges traversed.

We then generalized the situation by stipulating that inference *consumes* the propositions it uses unless they are specified as *persistent*. This allows us to represent change of state in a logical manner, which we illustrated using a coin exchange and blocks world, a classic artificial intelligence domain. A proof here corresponds to a (partially ordered) sequence of actions.

Finally, we followed Lambek [Lam58] by requiring our facts not only to be consumed by inference, but remain *ordered* so we could represent grammatical inference. We saw our first two logical connectives, $x \mathbin{/} y$ and $x \setminus y$ in order to represent the syntactic types of various parts of speech. In this example, a proof corresponds to a parse tree.

7 Bonus Materials

We complete the lecture notes here with some bonus materials from previous instance of the course, not covered this year.

7.1 [Example: Natural Numbers]

As a second example, we consider natural numbers $0, 1, 2, \ldots$. A convenient way to construct them is via iterated application of a successor function s to $\mathsf{0}$, written as

$$\mathsf{0}, \mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(\mathsf{0})), \ldots$$

We refer to s as a *constructor*. Now we can define the even and odd numbers through the following three rules.

$$\frac{}{\mathsf{even}(\mathsf{0})} \qquad \frac{\mathsf{even}(x)}{\mathsf{odd}(\mathsf{s}(x))} \qquad \frac{\mathsf{odd}(x)}{\mathsf{even}(\mathsf{s}(x))}$$

As an example of a derived rule of inference, we can summarize the proof on the left with the rule on the right:

$$\frac{\dfrac{\mathsf{even}(x)}{\mathsf{odd}(\mathsf{s}(x))}}{\mathsf{even}(\mathsf{s}(\mathsf{s}(x)))} \qquad \frac{\mathsf{even}(x)}{\mathsf{even}(\mathsf{s}(\mathsf{s}(x)))}$$

The structure of proofs in these examples is not particularly interesting, since proofs that number a number of n is even or odd just follow the structure of the number n.

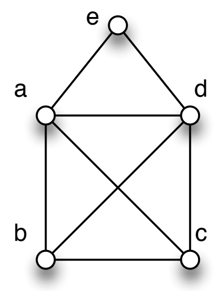
7.2 [Example: Graph Drawing]

We proceed to a slightly more sophisticated example involving linear inference. Before we used ordinary deductive inference to define the notion of path. This time we want to model drawing a graph without lifting the pen. This is the same as traversing the whole graph, going along each edge exactly once. This second formulation suggest the following idea: as we go along an edge we *consume* this edge so that we cannot follow it again. We also have to keep track of where we are, so we introduce another predicate $\mathsf{at}(x)$ which is true if we are at node x. The only rule of *linear* inference then is

$$\frac{\mathsf{at}(x) \quad \mathsf{edge}(x, y)}{\mathsf{at}(y)}\ \mathsf{step}$$

We start with an initial state just as before, with an $\mathsf{edge}(x, y)$ for each edge from x to y, the rule of symmetry (since have an undirected graph), and a starting position $\mathsf{at}(x_0)$. We can see that every time we take a step (by applying the step rule shown above), we consume a fact $\mathsf{at}(x)$ and produce another fact $\mathsf{at}(y)$, so there will always be exactly one fact of the form $\mathsf{at}(-)$ in the state. Also, at every step we consume one $\mathsf{edge}(-)$ fact, so we can take at most as many steps as there are edges in the graph initially. Of course, if we are at a point x there may be many outgoing edges, and if we pick the wrong one we may not be able to complete the drawing, but at least the number of steps we can try is limited at each point. We succeed, that is, we have found a way to draw the graph witout lifting the pen if we reach a state without an $\mathsf{edge}(-)$ fact and some final position $\mathsf{at}(x_n)$.

The following example graph is from a German children's rhyme[6] and can be drawn in one stroke if we start at b or c, but not if we start at a, d, or e.



We leave it to the reader to construct a solution and then translate it to a proof. Also, if we remove node e and its edges to a and d, no solution is possible.

Let's make the meaning of proofs explicit again. Because we have only one inference rule concerned with a move, a proof in general will have the following shape:

$$\dfrac{\dfrac{\dfrac{\mathsf{at}(x_0) \quad \mathsf{edge}(x_0, x_1)}{\ddots}\,\mathsf{step} \qquad \mathsf{edge}(x_{n-2}, x_{n-1})}{\mathsf{at}(x_{n-1})}\,\mathsf{step} \qquad \mathsf{edge}(x_{n-1}, x_n)}{\mathsf{at}(x_n)}\,\mathsf{step}$$

[6]*"Das ist das Haus vom Ni-ko-laus."*

This proof represents the path $x_0, x_1, \ldots, x_{n-1}, x_n$. Of course, some of these nodes may be the same, as can be seen by examining the example, but no *edge* can be used twice. If we need to traverse an edge in the opposite direction from its initial specification, we need to use symmetry, which will consume the edge and produce its inverse. We can then use the inverse to make a step. Being able to continually flip the direction of edges is a potential source of nontermination in the inference process. This does not invalidate the observation that a path along non-repeating edges can be written as a proof, and from a proof we can read off a path of non-repeating edges. This path represents a solution if the initial and final states are as described above.

7.3 [Example: Graph Traversal]

If we just want to traverse the graph rather than draw it, we should not destroy the edges as we move. A standard way to accomplish this is to explicitly recreate edges as we move:

$$\frac{\mathsf{at}(x) \quad \mathsf{edge}(x,y)}{\mathsf{at}(y) \quad \mathsf{edge}(x,y)}\,\mathsf{step}$$

Another way is to distinguish *ephemeral propositions* from *persistent propositions*. Ephemeral propositions are consumed during inference, while persistent propositions are never consumed. This allows us to have a uniform framework encompassing both the ordinary logical inference where we just add the conclusions to our store of knowledge, and linear logical inference.

We indicate the disposition of propositions that are known to be true by writing $A\ \mathit{eph}$ and $A\ \mathit{pers}$. Here, A stands for a proposition and $A\ \mathit{eph}$ and $A\ \mathit{pers}$ are called *judgments*. Distinguishing judgments from propositions is one of the cornerstones of Per Martin-Löf's approach to the foundation of logic and programming languages [ML83]. From now on we will follow the idea that the subjects of inference rules are judgments *about* propositions, not the propositions themselves. Mostly, they express that propositions are true, but in a variety of ways: ephemerally true, persistently true, true at time t, etc. There are a number of different terms that have been used for the particular distinction we are making here:

$A\ \mathit{ephemeral}$	$A\ \mathit{persistent}$
$A\ \mathit{linear}$	$A\ \mathit{unrestricted}$
$A\ \mathit{true}$	$A\ \mathit{valid}$
$A\ \mathit{contingently\ true}$	$A\ \mathit{necessarily\ true}$

Our high-level message is:

$$\boxed{\textit{Truth is ephemeral; validity is forever.}}$$

In the example, we can use this by declaring edges to be persistent, in which case the premise and conclusion of the symmetry rule should also be persistent. We abbreviate *ephemeral* by *eph*, and *persistent* by *pers*.

$$\frac{\mathsf{at}(x)\ \mathit{eph} \quad \mathsf{edge}(x,y)\ \mathit{pers}}{\mathsf{at}(y)\ \mathit{eph}}\ \mathsf{step}$$

As an even more compact notation, we underline persistent propositions; if they are not underlined, they should be considered ephemeral. For example, there would appear to be little reason for the properties of being even or odd to be ephemeral, since they should be considered intrinsic properties of the natural numbers. We therefore write

$$\frac{}{\underline{\mathsf{even}}(0)} \qquad \frac{\underline{\mathsf{even}}(x)}{\underline{\mathsf{odd}}(\mathsf{s}(x))} \qquad \frac{\underline{\mathsf{odd}}(x)}{\underline{\mathsf{even}}(\mathsf{s}(x))}$$

The first rule here is an inference rule with no premise. Persistent conclusions of such rules are sometimes called *axioms* in the sense that they are persistently true.

7.4 [Example: Opportunity]

A common proverb states:

Opportunity doesn't knock twice. — Anonymous

Again, let's fix the vocabulary:

opportunity	opportunity
knocks(x)	x knocks

Then the preceding saying is just

$$\mathsf{knocks}(\mathsf{opportunity})\ \mathit{eph}$$

where we have written out the judgment *ephemeral* for emphasis.

Clearly, when this fact is used it cannot be used again. If it is never used, we do not consider opportunity to having ever knocked, so the judgment above captures the idea that opportunity knocks at most once (and therefore not twice).

Exercises

Exercise 1 As an alternative way to defining paths through graph, we can say that paths represent the reflexive and transitive closure of the (symmetric) edge relation. Write out the inference rules as in Section 1, first without proof terms. Now add proof terms and an appropriate equality relation on paths. Explain your choice of equational theory.

Exercise 2 Write out proof for the example graph from subsection 7.2 that demonstrates that the figure can be drawn in one stroke.

Exercise 3 Consider the representation of undirected graphs from Section 1, with predicates $\mathsf{node}(x)$ and $\mathsf{edge}(x, y)$.

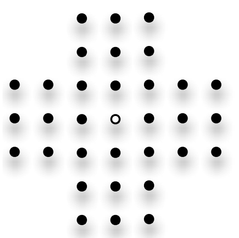
A *Hamiltonian cycle* is a path through a graph that visits each vertex exactly once and also returns to the starting vertex. Define any additional predicates you need and give inference rules such that inference corresponds to traversing the graph, and there is a simple condition that can be checked at the end to see if the traversal constituted a Hamiltonian cycle.

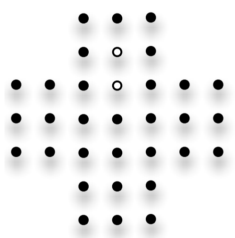
Exercise 4 Consider the representation of undirected graph from Section 1, with predicates $\mathsf{node}(x)$ and $\mathsf{edge}(x, y)$. We add a new predicate $\mathsf{color}(x)$ to express the color of node x. A *valid coloring* is one where no two adjacent nodes (that is, two nodes connected by an edge) have the same color.

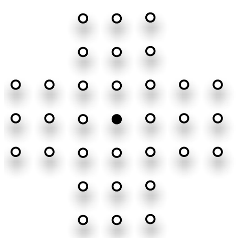
(i) Give inference rules that can deduce $\mathsf{invalid}(x, y)$ if and only if there are adjacent nodes x and y with the same color.

(ii) Give inference rules where proofs from an initial state to a final state satisfying an easily checkable condition correspond to valid colorings. Carefully describe your initial state and the property of the final state.

Exercise 5 Consider the game *Peg Solitaire*. We have a layout of holes (shown as hollow circles), all but one of which are filled with pegs (shown as filled circles). We move by taking one peg, jumping over an adjacent one into an empty hole behind it, removing the peg in the process. The situation after one of the four possible initial moves is shown in the second diagram. The third diagram shows the desired target position.


initial position


after one move


target position

Define a vocabulary and give inference rules in a representation of peg solitaire so that linear inference corresponds to making legal jumps. Explain how you represent the initial position, and how to test if you have reached the target position and thereby won the solitaire game.

Exercise 6 We consider the blocks world example from Section 3. As for graphs where we had the node predicate, it is convenient to add a new ephemeral predicate $\mathsf{block}(x)$ which is true for every block in the configuration.

Write a set of rules such that they can consume all ephemeral facts in the state (leaving only the persistent $\underline{\mathsf{clear}}(\mathsf{t})$) if and only if all of the following conditions are satisfied:

(i) the configuration of blocks is a collection of simple stacks,

(ii) the top of each stack is known to be clear, and

(iii) either the hand is empty or holds a block x, but not both.

If you believe it cannot be done, solve as many of the conditions as you can, explain why not all of them can be checked, and explore alternatives. Note linear inference allows rules with no premises or no conclusions.

Exercise 7 Render the following statement by an American president as an inference rule in linear logic:

If you can't stand the heat, get out of the kitchen. — Harry S. Truman

Use the following vocabulary:

$\mathsf{toohot}(x)$	x cannot stand the heat
$\mathsf{in}(x,y)$	x is in y
$\mathsf{kitchen}$	the kitchen

Exercise 8 Render the following statement in linear logic (without the use of any logical connectives):

> *Truth is ephemeral; validity is forever.* — Frank Pfenning, page 19

Use the vocabulary

truth	truth
validity	validity

References

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Lam58] Joachim Lambek. The mathematics of sentence structure. *The American Mathematical Monthly*, 65(3):154–170, 1958.

[ML83] Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. Notes for three lectures given in Siena, Italy. Published in *Nordic Journal of Philosophical Logic*, 1(1):11-60, 1996, April 1983.