Lecture Notes on Deductive Inference

Course on *Linear Logic*
Oregon Programming Languages Summer School 2013
Frank Pfenning

Lecture 1
July 23, 2013

According to Wikipedia, the ultimate authority on *everything*:

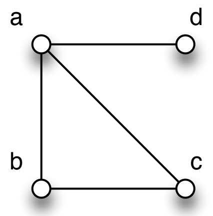
> **Logic** [. . .] is the formal systematic study of the principles of valid inference and correct reasoning.

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 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. We represent the nodes (vertices) as *constants* $(\mathsf{a}, \mathsf{b}, \ldots)$ and the edges with a binary *predicate* edge relating connected nodes.



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

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

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:

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

At this point we cannot quite go back-and-forth between a graph and its logical representation, because a disconnected node will not show up in the edge relation. Therefore, we should have a second predicate $\mathsf{node}(x)$, which holds for every node in the graph.

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

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. As is common with graphs, we do not want to consider the trivial, zero-length path from a node to itself. If we did, it would be the following rule (written in brackets as an indicator that it is only hypothetical):

$$\left[\ \dfrac{\mathsf{node}(x)}{\mathsf{path}(x,x)}\ \mathsf{refl}\ \right]$$

If we had omitted the premise, then the rule would have been problematic because it could be used for objects x which are not even nodes in the graph, leading to nonsensical conclusions.

The following two rules now define the notion of path. The first (e) says that an edge represents a valid path, the second (trans) that paths can be composed, making path a transitive relation.

$$\dfrac{\mathsf{edge}(x,y)}{\mathsf{path}(x,y)}\ \mathsf{e} \qquad\qquad \dfrac{\mathsf{path}(x,y) \quad \mathsf{path}(y,z)}{\mathsf{path}(x,z)}\ \mathsf{trans}$$

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

$$\dfrac{\dfrac{\dfrac{\mathsf{edge}(\mathsf{a},\mathsf{c})}{\mathsf{edge}(\mathsf{c},\mathsf{a})}\ \mathsf{sym}}{\mathsf{path}(\mathsf{c},\mathsf{a})}\ \mathsf{e} \qquad \dfrac{\mathsf{edge}(\mathsf{a},\mathsf{d})}{\mathsf{path}(\mathsf{a},\mathsf{d})}\ \mathsf{e}}{\mathsf{path}(\mathsf{c},\mathsf{d})}\ \mathsf{trans}$$

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. . .

Actually, there is more than source of ambiguity. One is that 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 $\mathsf{edge}(\mathsf{c},\mathsf{a})$. Another is that paths with more than three nodes can be broken down into different subpaths, using transitivity in different ways. The fact that different proofs have the same constructive content does not invalidate their interpretation, but we should be aware of it.

Since we ruled out reflexivity, under which circumstances can we still *prove* $\mathsf{path}(x,x)$? Because we consider undirected graphs, there is a path from x to x exactly if there is at least one neighbor of x, as the following proof shows:

$$\dfrac{\dfrac{\mathsf{edge}(x,y)}{\mathsf{path}(x,y)}\,\mathsf{e} \qquad \dfrac{\dfrac{\mathsf{edge}(x,y)}{\mathsf{edge}(y,x)}\,\mathsf{sym}}{\mathsf{path}(y,x)}\,\mathsf{e}}{\mathsf{path}(x,x)}\,\mathsf{trans}$$

There are several interesting aspects of this proof. For example, it doesn't depend on what x and y are. In other words, it is *schematic* in x and y. Another notable aspect is that it uses a premise $\mathsf{edge}(x,y)$ twice, which intuitively makes sense: a general way to leave x and return to it is to go to some arbitrary adjacent y and immediately return to x, reusing the same edge. We can summarize the deduction above into a single *derived rule of inference*:

$$\dfrac{\mathsf{edge}(x,y)}{\mathsf{path}(x,x)}$$

This inference rule is justified, because we can replace any particular instance of it by an instance of the schematic proof we gave above. As we will see in a later lecture, derived rules of inference play a very important role in logic.

2 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 a predicate nat that holds exactly of the natural numbers, in effect defining a type.

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

We can also define the even and odd numbers through the following three rules.

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

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.

3 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 piece of chalk*" 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

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

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

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

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:

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

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

$$\dfrac{\mathsf{d} \quad \mathsf{d} \quad \dfrac{\mathsf{d}}{\mathsf{n} \quad\quad}}{\mathsf{q}} \!\!\!\! \begin{array}{l} \\ \mathsf{n} \end{array} \qquad\qquad \dfrac{\mathsf{d} \quad \mathsf{d} \quad \mathsf{d}}{\mathsf{q} \quad \mathsf{n}}$$

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:[1]

Linear inference can change the world.

[1] with apologies to Phil Wadler

4 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

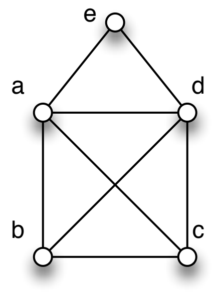
$$\frac{\mathsf{at}(x) \quad \mathsf{edge}(x,y)}{\mathsf{at}(y)}\ \mathsf{step}$$

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[2] and can be drawn in one stroke if we start at b or c, but not if we start at a, d, or

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

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:

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

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.

5 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 $A\ eph$ and $A\ pers$. Here, A stands for a proposition and $A\ eph$ and $A\ pers$ 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:

$A\ ephemeral$	$A\ persistent$
$A\ linear$	$A\ unrestricted$
$A\ true$	$A\ valid$
$A\ contingently\ true$	$A\ necessarily\ true$

Our high-level message is:

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 $pers$.

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

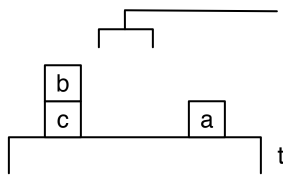
As an even more compact notation, we $\underline{\mathsf{underline}}$ persistent propositions; if they are not underlined, they should be considered ephemeral. For example, there would appear to be little reason for the properties of being even or odd to be ephemeral, since they should be considered 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.

6 Example: Blocks World[3]

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 (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

For example, the state above would be represented by

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

[3]not covered in lecture
[3]not covered in lecture

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 would have created a contradiction!

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* $\mathsf{clear}(\mathsf{c})$. If we assume there are sufficiently many places on the table to place all blocks, we would also have

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

The table is *persistently* clear, that is, always available for putting down a block.

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.

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

Alternatively, we could have a general rule

$$\frac{A\ \mathit{pers}}{A\ \mathit{eph}}\ \mathsf{copy}$$

that allows us to make an ephemeral copy of a persistently known fact. Depending on how we eventually formalize inference, one or the other solution will turn out to be more convenient.

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.

7 Example: Representing Linked Lists

Using the idea of constructors as for natural numbers, it is easy to define lists of natural numbers. We call the constructors nil and cons.

$$\frac{}{\underline{\mathsf{list}}(\mathsf{nil})} \qquad \frac{\underline{\mathsf{nat}}(n) \quad \underline{\mathsf{list}}(l)}{\underline{\mathsf{list}}(\mathsf{cons}(n,l))}$$

In an imperative language such as C, lists are usually represented as *linked lists*. For example, the list

$$\mathsf{cons}(3,\mathsf{cons}(4,\mathsf{cons}(5,\mathsf{nil})))$$

might be layed out in memory as

$$\begin{array}{lcl} a_0 & : & (3,a_1) \\ a_1 & : & (4,a_2) \\ a_2 & : & (5,\mathsf{null}) \\ \mathsf{null} & : & \end{array}$$

where a_0,a_1,a_2 are distict memory addresses, and null is a special address, usually 0, at which nothing can be stored. We have taken the liberty of abbreviating the successor-based representation numbers here by numerals (for example, $\mathsf{s}(\mathsf{s}(\mathsf{s}(0)))$ by 3).

In order to avoid dealing with the special address null, we use *lists segments* identified by their beginning and ending address. For example, the list above would be represented as the segment (a_0,a_3)

$$\begin{array}{lcl} a_0 & : & (3,a_1) \\ a_1 & : & (4,a_2) \\ a_2 & : & (5,a_3) \end{array}$$

where all the addresses a_i are different. What's actually stored at a_3 is irrelevant.

We represent this in logic with two forms of proposition, $\mathsf{seg}(b, e)$ (we have a list segment starting with address b and ending with address e) and $\mathsf{elem}(a, n, a')$ (we have an element n stored at address a with the next element of the list at a'). Returning to the example, we obtain the representation

$$\begin{array}{l}\mathsf{seg}(a_0, a_3),\\ \mathsf{elem}(a_0, 3, a_1),\\ \mathsf{elem}(a_1, 4, a_2),\\ \mathsf{elem}(a_2, 3, a_3)\end{array}$$

Next let's write some inference rules that tell us how to load a list into memory, starting at a given address a_0. We start with propositions[4]

$$\mathsf{load}(l), \mathsf{seg}(b, b)$$

where nothing is stored at b and would like to write some inference rule that can transform this to the linked list representation

$$\mathsf{seg}(b, e), \ldots$$

for some e and "..." contains the appropriately linked elem propositions.

The rules are based on the analysis of l in $\mathsf{load}(l)$. If l is nil, we are done and all the elements of the initial list have been loaded.

$$\frac{\mathsf{load}(\mathsf{nil}) \quad \mathsf{seg}(b, e)}{\mathsf{seg}(b, e)}$$

If the initial list is empty, the $\mathsf{seg}(b, b)$ is its representation, which is the correct segment representing the empty linked list.

Second, we have to treat the case that the list contains some element n. In that case, we need to "allocate" new memory. At the level of abstraction of our representation here, this just means that we need to obtain a new address. But a new address is just a new name, in logic sometimes called a *parameter*. In general, we annotate the inference rules itself with $[a_1, \ldots, a_n]$ to indicate that $a_1, \ldots, a_n$ in the conclusions of the rule are freshly chosen distinct names. By "freshly chosen" we mean that the names are distinct from all the names we have been using so far.

$$\frac{\mathsf{load}(\mathsf{cons}(n, l)) \quad \mathsf{seg}(b, e)}{\mathsf{load}(l) \quad \mathsf{elem}(e, n, e') \quad \mathsf{seg}(b, e')}\,[e']$$

[4]During lecture, we wrote $\mathsf{list}(l)$ here instead of $\mathsf{load}(l)$, which is an unfortunate overloading of the name "list".

Here, the freshly chosen name is e'. Because it is new, nothing is stored there yet, that is, there is no $\mathsf{elem}(e', -, -)$.

We suggest that you simulate this algorithm on the example list to see that the result comes out correctly.

As one more example of working with linked lists, we can write a very short program (a single rule!) to sort a linked list "in place" (without allocating any new elements). For this we need a predicate $\mathsf{gt}(x, y)$ which holds when x is greater than y. This is easy to define by inference rules:

$$\frac{}{\underline{\mathsf{gt}}(\mathsf{s}(x), 0)}\ \mathsf{gt\_0} \qquad \frac{\underline{\mathsf{gt}}(x, y)}{\underline{\mathsf{gt}}(\mathsf{s}(x), \mathsf{s}(y))}\ \mathsf{gt\_s}$$

These rules, like $\underline{\mathsf{nat}}(n)$ or $\underline{\mathsf{list}}(l)$ are not suitable for inference from the premises to the conclusion because they will never saturate. Therefore we think of these propositions as being proved on demand, by constructing (or failing to construct) a proof of $\underline{\mathsf{gt}}(x, y)$ when we know what x and y are.[5]

Now a kind of bubble sort just switches adjacent element when they are out of order.

$$\frac{\mathsf{elem}(a, x, b) \quad \mathsf{elem}(b, y, c) \quad \underline{\mathsf{gt}}(x, y)}{\mathsf{elem}(a, y, b) \quad \mathsf{elem}(b, x, c)}\ \mathsf{exch}$$

When this rule reaches quiescence, we know that the linked list must be sorted. Moreover, it doesn't matter in which order we apply various instances of the rule that might fire: we always end up with the same sorted lists. In this way, this rule represents a kind of opportunistically parallel bubble sort. It is easy to see here that firing of a rule must be *atomic*: we must match the premises and then is a single step replace the elements in the premise with the elements in the conclusion.

The exercises will give you additional opportunities to write inference rules manipulating linked lists in interesting ways.

[5]From the proof-theoretic perspective we declare $\underline{\mathsf{gt}}$ to be *negative*, while other propositions involved in inference are declared to be *positive*. We might explain this in detail in a future lecture.

8 Example: King Richard III[6]

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!*".

9 Example: Opportunity

A common proverb states:

Opportunity doesn't knock twice. — Anonymous

[6]not covered in lecture

Again, let's fix the vocabulary:

$\mathsf{opportunity}$	opportunity
$\mathsf{knocks}(x)$	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 Write out proof for the example graph from Section 4 that demonstrates that the figure can be drawn in one stroke.

Exercise 2 Consider the representation of undirected graphs from Section 1, with predicates node(x) and edge(x, y).

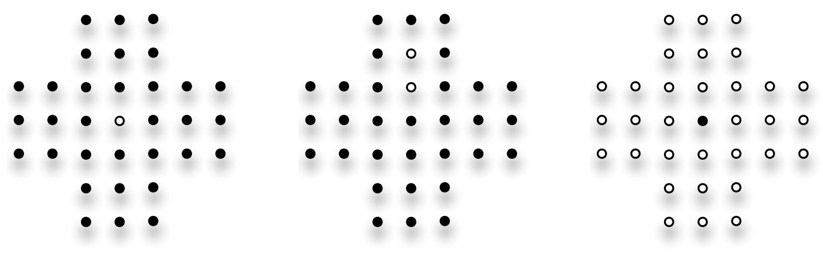
A *Hamiltonian cycle* is a path through a graph that visits each vertex exactly once and also returns to the starting vertex. Define any additional predicates you need and give inference rules such that inference corresponds to traversing the graph, and there is a simple condition that can be checked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

Exercise 3 Consider the representation of undirected graph from Section 1, with predicates node(x) and edge(x, y). We add a new predicate 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 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 4 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 5 We consider the blocks world example from Section 6. 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 6 Assume the representation of lists of natural numbers used in lecture with constructors nil and cons. Further assume two new predicates, $\mathsf{orig}(l)$ and $\mathsf{perm}(l)$ for lists l. Write a simple program such that $\mathsf{orig}(l) \quad \leadsto \quad \mathsf{perm}(l')$ if and only if l' is a permutation of l and the final state (which consists just of $\mathsf{perm}(l')$) is quiescent. You may introduce auxiliary predicates as needed. Your program should work correctly assuming don't-care nondeterminism.

Exercise 7 Assume the representation of lists of natural numbers used in lecture with predicates $\mathsf{seg}(b, e)$ and $\mathsf{elem}(a, x, a')$. Write the following programs:

(i) Write a program that reaches quiescence at $\mathsf{collected}(l)$, where l is the list represented by the segment.

(ii) Assuming the list is sorted in ascending order and a new initial proposition $\mathsf{insert}(x)$, create a new list where x has been inserted so as to maintain sortedness.

(iii) Reverse the list without allocating new addresses.

(iv) Append two segments without allocating new addresses.

Exercise 8 Now we use lists segments of bits 0 and 1 instead of segments of arbitrary natural numbers. A segment of bits naturally represents a number in binary notation, using predicates $\mathsf{seg}(h, l)$ (where h is the address of the highest bit and l the address of the lowest bit) and $\mathsf{elem}(a', x, a)$ where xx is a bit (0 or 1). Write a program to increment the number represented by a bit segment.

Exercise 9 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 10 Render the following statement in linear logic (without the use of any logical connectives):

Truth is ephemeral; validity is forever. — Frank Pfenning, page 9

Use the vocabulary

truth	truth
$\mathsf{validity}$	validity

References

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

[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.