Linear Logic: Linear Inference

What is logic?

When I was a beginning grad student, I thought logic was truth. (My first question on the qualifying exam was "What is truth?" I gave a standard answer which I now think is completely wrong.) Logic is about inference; we're not concerned about what is true but what inferences you may make, and what inferences you are not allowed. So the core of logic is proof theory, because you study the notion of what you can infer from things.

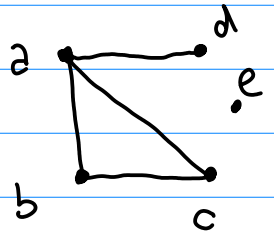
I think it has been held back because mathematicians have not been interested in math. For one thing, truth is ephemeral; but in mathematics it's denied that truth can change. In the 1920s, people thought about how to model discourse, and noticed things were true in one setting and not in another. This lead off to a branch of logic called modal logic.

If you're doing classical logic, you are not necessarily doing any computional content. If you're doing intuitionistic logic, you're doing functional computation. But there are many other models of computation: you may want to mutate state, or have processes communicating with each other. So you might wonder, what other logics govern the principles of other modes of computation? I've made a career of doing this, looking at computation and trying to find a logic which models it, and then building PLs which behave the way the logic dictates.

Linear logic has to do with concurrent/parallel computation. But we want to stay consistent with intuitionistic type theory, because otherwise we would have to throw out everything we know. We'd like the two to work together in a nice way. This is something new that has emerged in the last three years, and I want to talk about why this was missed for a long time. Linear logic was conceived by Girard in 1986, so it's been around a while, but people didn't fully establish the connection to concurrent programming for a long time.

Let's start w/ something simple

open vs closed



predicates

node(a)
node(b)
node(c)
node(d)
node(e)

edge(a,b)
edge(b,c)
edge(c,a)
edge(a,d)

all edges bidirectional

want to assert a bijection

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

schematic variable

$$\frac{edge(x,y)}{path(x,y)}\ ep \qquad \frac{node(x)}{path(x,x)}\ refl \quad \text{(optional)}$$

$$\frac{path(x,y) \quad path(y,z)}{path(x,z)}\ trans$$

not unique!

this is a program: run these rules until you cannot discover anything new

Gentzen/McAllister 2001

finite → terminates (even though there are infinitely many proofs)

bounded size of complete database for execution bound (this db is n^3)

DATALOG (1981)

(forward inferences always terminate)

in contrast to PROLOG (backwards, frequently diverges)

no logical connectives!

the end is called saturation

Linear logic (explained the same way)

truth is ephemeral → using an inference rule uses up its premises

Drawing a graph in one line

at(x) – pen is at node x
edge(x,y) – edge x y has not been written

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

(insert example)

the end is called quiescence

notice: property is an existential one (new!)
no confluence (unlike datalog)

Why not use temporal logic? E.g. A at time t
Problem: now you have to say everything else is the same (the frame problem); in LL this is built in.
↳ hard!

– sometimes, there are facts you need to keep around to reuse.
So divide facts into persistent and ephemeral facts

another note: you would like inference rules to be general, which is why we don't model persistent facts as rules

shows 2 copies diff than 1

$$\frac{node(z)}{node(z) \quad node(z)}$$ ← never quiesces

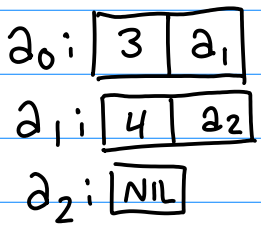
$$\frac{q}{d \quad d \quad n}$$ ← notation trick: can apply both ways

Linear logic is lower level than intuitionistic logic

↳ changing state

$$\frac{}{nat(0)} \qquad \frac{nat(n)}{nat(S(n))}$$

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



how to model addresses?
(numbers too concrete)
only interested in equality

$elem(a_0, 3, a_1)$
$elem(a_1, 4, a_2)$
$seg(a_0, a_2)$
beginning, end

goal: load a list into memory

start: $seg(a_0, a_0)$
$list\ l$

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

new name

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

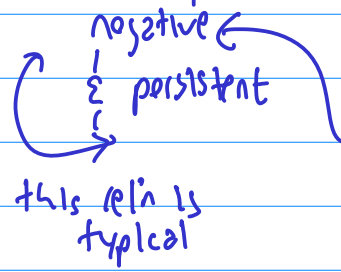
goal: sorting algorithm (next is negative)

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

positive

what do? want this backwards

positive vs. negative



yay bubblesort! one rule

it's also parallel!

Hamiltonian circus — end up at the node you started

Next time: limitations of this view; generalization to sequent calculus

$$\frac{p(b) \quad edge(a,b) \quad edge(b,c)}{edge(a,c)}$$

init: $\forall n : V.\ p(a)$

$\forall e : E.\ edge(e)$

cycle: $\dfrac{edge(a,b) \quad p(a) \quad p(b)}{done}$

circus: $\dfrac{edge(a,a)}{done}$