Linear Logic and Session Types

Hype for Types

November 10, 2020

What We'll Talk About

- How to make `malloc` and `free` safe
- Resources and state at the type level
- Owls

Linear Logic

Malloc is Scary...

Consider the following C code:

```
int main () {
  char *str;
  str = (char *) malloc(13);
  strcpy(str, "hypefortypes");
  free(str);
  return(0);
}
```

In C, we have to make sure we allocate and deallocate every memory cell exactly once.

Question

Is there a way to make our *types* guarantee correctness?

The Problem With Constructive Logic

In "normal" constructive logic, we have no concept of *state*.

- Want to be able to get rid of assumptions
- Truth should no longer be persistent, but rather ephemeral

This property comes from **structural rules**: weakening and contraction.

Weakening

```
int main() {
  int *x = (int *) malloc(sizeof(int));
  *x = 3;
  return *x;
}
```

Weakening: we can “drop” assumptions

$$\frac{\Gamma \vdash e : \tau}{\Gamma, x : \tau' \vdash e : \tau} \ (\textsc{Weak})$$

Contraction

```
void f(int *x) {
  free(x);
}

int main() {
  int *x = (int *) malloc(sizeof(int));
  *x = 3;
  f(x);
  return *x;
}
```

Contraction: we can "duplicate" assumptions

$$\frac{\Gamma, x_1 : \tau, x_2 : \tau \vdash e : \tau'}{\Gamma, x : \tau \vdash [x, x/x_1, x_2]e : \tau'} \ (\textsc{Cntr})$$

Introduction to Linear Logic

In **linear logic**, we have neither weakening nor contraction.

- Requirement that we use each piece of data exactly once - no duplication, no dropping
- Comes with an inherent idea of "resources" that are used up
- Allows us to write safe, stateful (imperative!) programs

Practical Example

The programming language Rust uses *affine logic*, which has weakening but not contraction (meaning we can use data at most once).

The Linear Rules

Identity

Constructive Logic

$$\frac{}{\Gamma, A \vdash A} \text{ (Hyp)}$$

Linear Logic

$$\frac{}{A \vdash A} \text{ (Hyp)}$$

Intuition

"Given A and nothing else, we can use up A"

Implication

Constructive Logic

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \; (\Rightarrow\mathrm{I})$$

$$\frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} \; (\Rightarrow\mathrm{E})$$

Linear Logic

$$\frac{\Delta, A \vdash B}{\Delta \vdash A \multimap B} \; (\multimap\mathrm{I})$$

$$\frac{\Delta \vdash A \multimap B \quad \Delta' \vdash A}{\Delta, \Delta' \vdash B} \; (\multimap\mathrm{E})$$

Note

In the elimination rule, we *split* the contexts to prove the necessary premises.

Disjunction

Constructive Logic

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \ (\vee I_1)$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \ (\vee I_2)$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C} \ (\vee E)$$

Linear Logic

$$\frac{\Delta \vdash A}{\Delta \vdash A \oplus B} \ (\oplus \mathrm{I1})$$

$$\frac{\Delta \vdash B}{\Delta \vdash A \oplus B} \ (\oplus \mathrm{I2})$$

$$\frac{\Delta, A \vdash C \quad \Delta, B \vdash C}{\Delta, A \oplus B \vdash C} \ (\oplus \mathrm{E})$$

Conjunction

Constructive Logic

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ (\wedge\mathrm{I})$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ (\wedge\mathrm{E1})$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ (\wedge\mathrm{E2})$$

Linear Logic

$$\frac{\Delta \vdash A \quad \Delta' \vdash B}{\Delta, \Delta' \vdash A \otimes B}\ (\otimes\mathrm{I})$$

$$\frac{\Delta \vdash A \otimes B \quad \Delta', A, B \vdash C}{\Delta, \Delta' \vdash C}\ (\otimes\mathrm{E})$$

Problem

In the constructive elimination rules, what happens to B and A, respectively? What do we do for linear?

Choice

So far, we've seen:

- Disjunction: "I could get either A or B, but I don't know which" **(internal choice)**
- Conjunction: "I have both A and B simultaneously"

...But there's another form of choice, when it comes to resources.

- **External choice**: "I can have either A or B, but not both at the same time"

External Choice/Alternative Conjunction

$$\frac{\Delta \vdash A \quad \Delta \vdash B}{\Delta \vdash A \& B}\ (\&\mathrm{I}) \qquad \frac{\Delta \vdash A \& B}{\Delta \vdash A}\ (\&\mathrm{E1}) \qquad \frac{\Delta \vdash A \& B}{\Delta \vdash B}\ (\&\mathrm{E2})$$

Examples:

- Given $7, I can buy *either* a sandwich from ABP *or* pancakes from the Underground, but not both.
- If I have 1 egg, I can make *either* scrambled eggs *or* a fried egg, but not both.

The Linear Rules

$$\frac{}{A \vdash A}\ (\textsc{Hyp}) \qquad \frac{\Delta, A \vdash B}{\Delta \vdash A \multimap B}\ (\multimap\mathrm{I}) \qquad \frac{\Delta \vdash A \multimap B \quad \Delta' \vdash A}{\Delta, \Delta' \vdash B}\ (\multimap\mathrm{E})$$

$$\frac{\Delta \vdash A \quad \Delta' \vdash B}{\Delta, \Delta' \vdash A \otimes B}\ (\otimes\mathrm{I}) \qquad \frac{\Delta \vdash A \otimes B \quad \Delta', A, B \vdash C}{\Delta, \Delta' \vdash C}\ (\otimes\mathrm{E})$$

$$\frac{\Delta \vdash A}{\Delta \vdash A \oplus B}\ (\oplus\mathrm{I1}) \qquad \frac{\Delta \vdash B}{\Delta \vdash A \oplus B}\ (\oplus\mathrm{I2}) \qquad \frac{\Delta, A \vdash C \quad \Delta, B \vdash C}{\Delta, A \oplus B \vdash C}\ (\oplus\mathrm{E})$$

$$\frac{\Delta \vdash A \quad \Delta \vdash B}{\Delta \vdash A \& B}\ (\&\mathrm{I}) \qquad \frac{\Delta \vdash A \& B}{\Delta \vdash A}\ (\&\mathrm{E1}) \qquad \frac{\Delta \vdash A \& B}{\Delta \vdash B}\ (\&\mathrm{E2})$$

Session Types

Back to Curry-Howard

Constructive Logic	**Linear Logic**
Functional programming	Concurrent programming
Functions	Processes
“SML” types	Session types
Evaluation	Owls

Note

In the context of imperative programming: a memory cell can be thought of as a process that “remembers” some data.

Processes

- Processes (wizards) communicate across channels (owls) by sending and receiving messages (letters)

$$\texttt{decl}\ \ P : (x_1 : A_1)(x_2 : A_2)\ldots(x_n : A_n) \vdash (x : A)$$

The Process Judgment

"Process P provides a channel x carrying type A, using channels $x_1, \ldots, x_n$ carrying $A_1, \ldots, A_n$."

Live Coding

Conclusion

Things We Talked About

- Linearity as a way of representing state
- Linear propositions in terms of resources
- The Curry-Howard correspondence between linear logic and session types
- Writing programs with processes[1]

Things We Didn't Cover

- Concurrent programming (spawning processes in parallel)
- Resource tracking (identify the cost of different programs)
  - Amortized analysis
  - Automatic bound derivation
  - Granularity control?

[1] https://bitbucket.org/fpfenning/rast/src/master/