Typed λ-calculus: Concepts and Syntax

P. B. Levy

University of Birmingham

1 Introduction

λ-calculus is a small language based on some common mathematical idioms. It was invented by Alonzo Church in 1936, but his version was *untyped*, making the connection with mathematics rather problematic. In this course we'll be looking at a *typed* version.

λ-calculus has had an impact throughout computer science and logic. For example

- it is the basis of functional programming languages such as Haskell, Standard ML, OCaml, Lisp, Scheme, Erlang, Scala, F♯.
- it is often used to give semantics for programming languages. This was initiated by Peter Landin, who in 1965 described the semantics of Algol-60 by translating it into λ-calculus.
- it closely corresponds to a kind of logic called *intuitionistic* logic, via the *Curry-Howard isomorphism*. That isn't in this course, but you may notice that a lot of notation (e.g. $\vdash$) and terminology ("introduction/elimination rule") has been imported from logic into λ-calculus. And the influence in the opposite direction has been much greater.

2 Notations for Sets and Elements

or **Sums your primary school never taught you**

In this section, we're going to learn some notations and abbreviations for describing *sets* and *elements of sets*.

Recall that $x \in A$ means "x is an element of the set A".

2.1 Sets

First, the notations for describing sets.

integers We define $\mathbb{Z}$ to be the set of integers.
booleans We define $\mathbb{B}$ to be the set of booleans $\{\mathsf{true}, \mathsf{false}\}$.
cartesian product Suppose A and B are sets. Then we write $A \times B$ for the set of ordered pairs

$$\{\langle x, y\rangle | x \in A, y \in B\}$$

disjoint union Suppose A and B are sets. Then we write $A + B$ for the set of ordered pairs

$$\{\mathsf{inl}\ x | x \in A\} \cup \{\mathsf{inr}\ x | x \in B\}$$

Here we use inl and inr as "tags". If you like, you could define

$$\mathsf{inl}\ x \stackrel{\text{def}}{=} \langle 0, x\rangle$$
$$\mathsf{inr}\ x \stackrel{\text{def}}{=} \langle 1, x\rangle$$

function space Suppose A and B are sets. Then we write $A \to B$ for the set of functions from A to B. (You will also see this written as B^A.)

These operations on sets correspond to familiar operations on natural numbers. If A is finite with m elements, and B is finite with n elements, then

- $A \times B$ has mn elements
- $A + B$ has $m + n$ elements
- $A \to B$ has n^m elements.

2.2 Integers and Booleans

Recall that $\mathbb{Z}$ is the set of integers, and $\mathbb{B}$ is the set of booleans.
Some ways of describing integers.

Arithmetic Here is an integer:

$$3 + (7 \times 2)$$

Conditionals Here is another integer:

$$\mathsf{case}\ (7 > 5)\ \mathsf{of}\ \{\mathsf{true}.\ 20 + 3, \mathsf{false}.\ 53\}$$

This is an "if... then ... else" construction.

Local definitions Here is another integer:

$$\mathsf{let}\ (2 \times 18) + (3 \times 102)\ \mathsf{be}\ y.\ (y + 17 \times y)$$

This is a shorthand for

$y + 17 \times y$, where we define y to be $(2 \times 18) + (3 \times 102)$

It's rather like a constant declaration in programming.

Exercise 1. What integer is

1. $(2 + 5) \times 8$
2. $\mathsf{case}\ (\mathsf{case}\ 1 > 8\ \mathsf{of}\ \{\mathsf{true}.\ 5 > 2 + 4, \mathsf{false}.\ 3 > 2\})\ \mathsf{of}\ \{\mathsf{true}.\ 3 \times 7, \mathsf{false}.\ 100\}$
3. $\mathsf{let}\ (\mathsf{let}\ 3 + 2\ \mathsf{be}\ x.\ x \times (x + 3))\ \mathsf{be}\ y.\ y + 15$
4. $\mathsf{let}\ (5 + 7)\ \mathsf{be}\ x.\ \mathsf{case}\ x > 3\ \mathsf{of}\ \{\mathsf{true}.\ 12, \mathsf{false}.\ 3 + 3\}$

?

2.3 Cartesian Product

Recall that $A \times B$ is the set of ordered pairs $\langle x, y\rangle$ such that $x \in A$ and $y \in B$.

projections If x is an ordered pair, we write πx for its first component, and $\pi' x$ for its second component. For example, here is another integer

$$\mathsf{let}\ \langle 3, 7 + 2\rangle\ \mathsf{be}\ x.\ (\pi x) \times (\pi' x) + (\pi' x)$$

pattern-match We can also pattern-match an ordered pair. For example:

$$\mathsf{let}\ \langle 3, 7 + 2\rangle\ \mathsf{be}\ x.\ \mathsf{case}\ x\ \mathsf{of}\ \langle y, z\rangle.\ y \times z + z$$

Here, you don't need to select the appropriate case, because there's only one. Since x is the pair $\langle 3, 9\rangle$, it matches the pattern $\langle y, z\rangle$, and y and z are thereby defined to be 3 and 9 respectively.

Pattern-matching is often a more convenient notation than projections.

Exercise 2. What integer is

1. $\mathsf{let}\ \langle 7, \mathsf{let}\ 3\ \mathsf{be}\ x.\ x + 7\rangle\ \mathsf{be}\ y.\ \pi y + (\mathsf{case}\ y\ \mathsf{of}\ \langle u, v\rangle.\ u + v)$
2. $\mathsf{case}\ (\pi\langle 7, 357 \times 128\rangle > 2)\ \mathsf{of}\ \{\mathsf{true}.\ 13, \mathsf{false}.\ 2\}$
3. $\mathsf{let}\ \langle 5, \langle 2, \mathsf{true}\rangle\rangle\ \mathsf{be}\ x.\ \pi x + \pi(\mathsf{case}\ x\ \mathsf{of}\ \langle y, z\rangle.\ z)$

?

2.4 Disjoint Union

Recall that $A + B$ is the set of all ordered pairs $\mathsf{inl}\ x$, where $x \in A$, and all ordered pairs $\mathsf{inr}\ x$ where $x \in B$.

We can pattern-match an element of $A + B$. For example, here is an integer:

$$\begin{array}{l}\mathsf{let}\ \mathsf{inl}\ 3\ \mathsf{be}\ x.\ \mathsf{let}\ 7\ \mathsf{be}\ y. \\ \mathsf{case}\ x\ \mathsf{of}\ \{\mathsf{inl}\ z.\ z + y, \mathsf{inr}\ z.\ z \times y\}\end{array}$$

Since x is defined here to be $\mathsf{inl}\ 3$, it matches the pattern $\mathsf{inl}\ z$, and z is thereby defined to be 3.

Exercise 3. What integer is

1. $\mathsf{case}\ (\mathsf{case}\ (3 < 7)\ \mathsf{of}\ \{\mathsf{true}.\ \mathsf{inr}\ (8+1), \mathsf{false}.\ \mathsf{inl}\ 2\})$
 $\mathsf{of}\ \{\mathsf{inl}\ u.\ u + 8, \mathsf{inr}\ u.\ u + 3\}$
2. $\mathsf{let}\ \langle 3, \mathsf{inr}\ \langle 7, \mathsf{true}\rangle\rangle\ \mathsf{be}\ z.\pi z + \mathsf{case}\ \pi' z$
 $\mathsf{of}\ \{\mathsf{inl}\ y.\ y + 2, \mathsf{inr}\ y.\ \mathsf{let}\ 4\ \mathsf{be}\ x.\ ((x + \pi y) + \pi z)\}$

?

2.5 Function Space

Recall that $A \to B$ is the set of all functions from A to B.

λ-abstraction Suppose A is a set. We write $\lambda x_A.$ to mean "the function that takes each $x \in A$ to ". For example, $\lambda x_{\mathbb{Z}}.(2 \times x + 1)$ is the function taking each integer x to $2 \times x + 1$.

application If f is a function from A to B, and $x \in A$, then we write fx to mean f applied to x. For example, here is another integer:

$$(\lambda x_{\mathbb{Z}}.\ (2 \times x + 1))7$$

And that completes our notation.

Exercise 4. What integer is

1. $((\lambda f_{\mathbb{Z} \to \mathbb{Z}}.\ \lambda x_{\mathbb{Z}}.\ (f\ (f\ x)))\ \lambda x_{\mathbb{Z}}.\ (x + 3))\ 2$
2. $\mathsf{let}\ \lambda x_{\mathbb{Z}+\mathbb{B}}.\ \mathsf{case}\ x\ \mathsf{of}\ \{\mathsf{inl}\ y.\ y + 3, \mathsf{inr}\ y.7\}\ \mathsf{be}\ f.\ (f\ \mathsf{inl}\ 5) + (f\ \mathsf{inr}\ \mathsf{false})$
3. $\mathsf{let}\ \lambda x_{\mathbb{Z} \times \mathbb{Z}}.\ (\mathsf{case}\ x\ \mathsf{of}\ \langle y, z\rangle.\ (2 \times y + z))\ \mathsf{be}\ f.\ f\ \langle \mathsf{let}\ 4\ \mathsf{be}\ u.\ u + 1, 8\rangle$

?

3 A Calculus For Integers and Booleans

3.1 Calculus of Integers

We want to turn all of the above notations into a calculus. Typically, calculi are defined inductively. As an example, here is a little calculus of integer expressions:

- $\underline{n}$ is an integer expression for every $n \in \mathbb{Z}$.
- If M is an integer expression, and N is an integer expression, then $M + N$ is an integer expression.
- If M is an integer expression, and N is an integer expression, then $M \times N$ is an integer expression.

Thus an integer expression is a finite string of symbols. Don't get confused between the integer *expression* $\underline{3}+\underline{4}$, and the *integer* $3+4$, which is 7. (I normally won't bother with the underlining, but in principle it's necessary.)

Actually, I lied: an integer expression isn't really a finite string of symbols, it's a finite *tree* of symbols. So $(\underline{3} + \underline{4}) \times \underline{2}$ and $\underline{3} + \underline{4} \times \underline{2}$ represent different expressions. But $\underline{3} + \underline{4} \times \underline{2}$ and $\underline{3} + ((\underline{4} \times \underline{2}))$ are the same expression i.e. the same tree.

Remark 1. Since this isn't a course on induction, I'm not delving into this in any more detail. But here is something for your notes, anticipating what you'll learn in the categories course.

The above inductive definition can be understood as describing a *category*. An object of this category is an *algebra* consisting of a set X, equipped with an element $\underline{n} \in X$, for each $n \in \mathbb{Z}$, and two binary operations $+$ and $\times$. A morphism is an *algebra homomorphism* i.e. a function between sets that preserves all this structure. Then the set of integer expressions (trees of symbols) is an *initial algebra*, i.e. an initial object in this category of algebras.

Let us write $\vdash M : \texttt{int}$ to mean "M is an integer expression". Then the above inductive definition can be abbreviated as follows.

$$\frac{}{\vdash \underline{n} : \texttt{int}}\ n \in \mathbb{Z}$$

$$\frac{\vdash M : \texttt{int} \quad \vdash N : \texttt{int}}{\vdash M + N : \texttt{int}} \qquad \frac{\vdash M : \texttt{int} \quad \vdash N : \texttt{int}}{\vdash M \times N : \texttt{int}}$$

The two expressions shown above can be written as “proof trees”, this time with the root at the bottom (like in botany).

$$\dfrac{\dfrac{\dfrac{}{\vdash 3 : \texttt{int}} \qquad \dfrac{}{\vdash 4 : \texttt{int}}}{\vdash 3+4 : \texttt{int}} \qquad \dfrac{}{\vdash 2 : \texttt{int}}}{\vdash (3+4)\times 2 : \texttt{int}}$$

and

$$\dfrac{\dfrac{}{\vdash 3 : \texttt{int}} \qquad \dfrac{\dfrac{}{\vdash 4 : \texttt{int}} \qquad \dfrac{}{\vdash 2 : \texttt{int}}}{\vdash 4\times 2 : \texttt{int}}}{\vdash 3+4\times 2 : \texttt{int}}$$

3.2 Calculus of Integers and Booleans

Next we want to make a calculus of integers and booleans. We define the set of types (i.e. set expressions) to be $\{\texttt{int}, \texttt{bool}\}$. We write $\vdash M : A$ to mean that M is an expression of type A. To the above rules we add:

$$\dfrac{}{\vdash \texttt{true} : \texttt{bool}} \qquad\qquad \dfrac{}{\vdash \texttt{false} : \texttt{bool}}$$

$$\dfrac{\vdash M : \texttt{int} \quad \vdash N : \texttt{int}}{\vdash M > N : \texttt{bool}} \qquad \dfrac{\vdash M : \texttt{bool} \quad \vdash N : B \quad \vdash N' : B}{\vdash \texttt{case}\ M\ \texttt{of}\ \{\texttt{true}.\ N, \texttt{false}. N'\} : B}$$

3.3 Local Definitions

We next want to add local definitions to our calculus, but this presents a problem. On the one hand, $\texttt{let}\ 3\ \texttt{be}\ \texttt{x}.\ \texttt{x} + 4$ should definitely be an integer expression. If we type it into the computer, we get

```
Answer: 7
```

So we want $\vdash \texttt{let}\ 3\ \texttt{be}\ \texttt{x}.\ \texttt{x} + 4 : \texttt{int}$.

But $\texttt{x} + 4$ is not valid as an integer expression. If we type it into the computer, we get

`Error: you haven't defined x.`

So we don't want $\vdash \mathtt{x} + 4 : \mathtt{int}$.

How then can we define the calculus? We have a valid expression with a subterm that is not syntactically valid!

The solution is to write

$$\mathtt{x} : \mathtt{int} \vdash \mathtt{x} + 4 : \mathtt{int}$$

This means: "once $\mathtt{x}$ has been defined to be some integer, $\mathtt{x}+4$ is an integer expression".

Exercise 5. Which of the following would you expect to be correct statements?

1. $\mathtt{x} : \mathtt{int} \vdash \mathtt{x} + \mathtt{y} : \mathtt{int}$
2. $\mathtt{x} : \mathtt{int} \vdash \mathtt{let}\ 3\ \mathtt{be}\ \mathtt{y}.\ \mathtt{x} + \mathtt{y} : \mathtt{int}$
3. $\mathtt{x} : \mathtt{int}, \mathtt{y} : \mathtt{int} \vdash \mathtt{x} + \mathtt{y} : \mathtt{int}$
4. $\mathtt{x} : \mathtt{int}, \mathtt{y} : \mathtt{int} \vdash \mathtt{x} + 3 : \mathtt{int}$

Some terminology.

1. A, B and C range over types.
2. M and N and (if I'm desperate) P range over terms.
3. $\mathtt{x}$, $\mathtt{y}$ and $\mathtt{z}$ are called *identifiers* (not "variables" please, the binding doesn't change over time).
4. A finite set of distinct identifiers with associated types, such as

$$\mathtt{x} : \mathtt{int}, \mathtt{y} : \mathtt{int}, \mathtt{z} : \mathtt{bool} \tag{1}$$

   is called a *typing context*. Note that[1] a typing context is a *set*, so the order doesn't matter: the typing context

$$\mathtt{x} : \mathtt{int}, \mathtt{z} : \mathtt{bool}, \mathtt{y} : \mathtt{int}$$

   is the same as (1).
5. Γ and Δ range over typing contexts.

[1] At least in these notes. Different papers may follow different conventions.

6. If Γ is a typing context, $\mathtt{x}$ an identifier and A a type, we write

$$\Gamma, \mathtt{x} : A$$

to mean Γ *extended* with the declaration $\mathtt{x} : A$. What if $\mathtt{x}$ already appears in Γ? Then that declaration is overwritten by the new one. For example,

$$\mathtt{x} : \mathtt{bool}, \mathtt{y} : \mathtt{int}, \mathtt{z} : \mathtt{bool}, \mathtt{x} : \mathtt{int}$$

describes the typing context (1).
7. Any term that can be proved in the *empty context*, i.e. $\vdash M : A$, is said to be *closed*.

Before I can give you the rules for `let`, I have to go back and change all the rules we've seen so far to incorporate a context. So the rule for + becomes

$$\frac{\Gamma \vdash M : \mathtt{int} \quad \Gamma \vdash N : \mathtt{int}}{\Gamma \vdash M + N : \mathtt{int}}$$

and similarly for $\times$ and $>$.

The rule for 3 becomes

$$\frac{}{\Gamma \vdash 3 : \mathtt{int}}$$

and similarly for all the other integers, and `true` and `false`.

And the rule for conditionals becomes

$$\frac{\Gamma \vdash M : \mathtt{bool} \quad \Gamma \vdash N : B \quad \Gamma \vdash N' : B}{\Gamma \vdash \mathtt{case}\ M\ \mathtt{of}\ \{\mathtt{true}.\ N, \mathtt{false}.\ N'\} : B}$$

We need a rule for identifiers, so that we can prove things like $\mathtt{x} : \mathtt{int}, \mathtt{y} : \mathtt{int} \vdash \mathtt{x} : \mathtt{int}$. Here's the rule:

$$\frac{}{\Gamma \vdash \mathtt{x} : A}\ (\mathtt{x} : A) \in \Gamma$$

And finally we want a rule for `let`. How do we prove that $\Gamma \vdash \mathtt{let}\ M\ \mathtt{be}\ \mathtt{x}.\ N : B$? Certainly we would have to prove something about M and something about N. To be more precise: we have to show that $\Gamma \vdash M : A$, and we have to show $\Gamma, \mathtt{x} : A \vdash N : B$. So the rule is

$$\frac{\Gamma \vdash M : A \quad \Gamma, \mathtt{x} : A \vdash N : B}{\Gamma \vdash \mathtt{let}\ M\ \mathtt{be}\ \mathtt{x}.\ N : B}$$

Exercise 6. Prove $\vdash \mathtt{let}\ 3\ \mathtt{be}\ \mathtt{x}.\ \mathtt{x} + 2 : \mathtt{int}$

4 Bound Identifiers

4.1 Scope and Shadowing

Let's consider the following term:

$$\mathtt{x} : \mathtt{int}, \mathtt{y} : \mathtt{int} \vdash (\mathtt{x} + \mathtt{y}) + \mathtt{let}\ 3\ \mathtt{be}\ \mathtt{y}.\ (\mathtt{x} + \mathtt{y}) : \mathtt{int}$$

There are 4 occurrences of identifiers in this term. The two occurrences of `x` are *free*. The first occurrence of `y` is free, but the second is *bound*. More specifically, it is bound to a particular place.

We can draw a *binding diagram* for any term:

- replace every binding of an identifier by a rectangle
- replace each bound occurrence by a circle, and draw an arrow from the circle to the rectangle where it is bound
- leave the free occurrences

How do we draw this? Every binding has a *scope* which is the term that it is applied to. Any occurrence of `x` that is outside the scope of an `x`-binder is a free occurrence. If it is inside the scope of an `x`-binding, it is bound to that `x`-binding. Sometimes, an `x`-binder sits inside the scope of another `x`-binder:

$$\mathtt{let}\ 3\ \mathtt{be}\ \mathtt{x}.\ \mathtt{let}\ 4\ \mathtt{be}\ \mathtt{x}.\ (\mathtt{x} + 2)$$

This is called *shadowing*, and the scope of the inner binder is subtracted from the scope from the outer binder. So the occurrence of `x` at the end is bound to the second binder. The rule is always

> Given an occurrence of `x`, move up the branch of the tree, and as soon as you hit an `x`-binder, that's the place the occurrence is bound to. If you never hit an `x`-binder, the occurrence is free.

Exercise 7. Draw a binding diagram for

$$\mathtt{let}\ 3\ \mathtt{be}\ \mathtt{x}.\ \mathtt{let}\ (\mathtt{let}\ \mathtt{x} + 2\ \mathtt{be}\ \mathtt{y}.\ \mathtt{y} + 7)\ \mathtt{be}\ \mathtt{y}.\ \mathtt{x} + \mathtt{y}$$

4.2 α-equivalence

Now here is a variation on the above term:

$$\mathtt{x} : \mathtt{int}, \mathtt{y} : \mathtt{int} \vdash (\mathtt{x} + \mathtt{y}) + \mathtt{let}\ 3\ \mathtt{be}\ \mathtt{z}.\ (\mathtt{x} + \mathtt{z}) : \mathtt{int}$$

The only difference is that we've changed a bound identifier. So the binding diagrams are the same. We say that two terms are *α-equivalent* when the binding diagrams are the same.

α-equivalent terms are, to all intents and purposes, the same. In fact, it would be more accurate to define a term to be a binding diagram. We take this as the definition. Bound identifiers are just a convenient way of writing a term (rather like brackets are), but the term itself is a binding diagram.

Remark 2. An elegant approach to syntax with binders is to describe the set of binding diagrams as an initial algebra. This may be done in several ways, e.g. using a presheaf category as in Fiore, Plotkin and Turi's paper "Abstract syntax and variable binding" in LICS 1999. Since this is beyond the scope of the course, we will make do with the informal description of binding diagrams above.

5 The λ-calculus

5.1 Types

Now that we've learnt the general concepts of a calculus with binding, we're ready to make a calculus out of all the notations that we saw. The *types* of this calculus are given by the inductive definition:

$$A ::= \ \mathtt{int} \mid \mathtt{bool} \mid A \times A \mid A + A \mid A \to A \mid 0 \mid 1$$

where 0 is a type corresponding to the empty set, and 1 is a type corresponding to a singleton set (a set with one element).

Like a term, a type is just a tree of symbols. Don't confuse the *type* $\mathtt{int} \to \mathtt{int}$ with the *set* $\mathbb{Z} \to \mathbb{Z}$.

As we look at the typing rules for $A \times B$ and $A + B$ and $A \to B$, we'll see that there are two kinds.

- The *introduction rules* for a type tell us how to *form* something of that type.

– The *elimination rules* for a type tell us how to *use* something of that type.

In fact, we've already seen these for the type `bool`. The typing rules for `true` and `false` are introduction rules. The typing rule for conditionals is an elimination rule.

(The type `int` is an exception to this neat pattern. Because of problems with infinity, there isn't a simple elimination rule.)

5.2 Cartesian Product

How do we form something of type $A \times B$? We use pairing. So the introduction rule is

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N \rangle : A \times B}$$

How do we use something of type $A \times B$? As we saw before, there's actually a choice here: we can either project or pattern-match. For projections, our elimination rules are

$$\frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi M : A} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi' M : B}$$

For pattern-matching, how do we prove that $\Gamma \vdash \texttt{case}\ M\ \texttt{of}\ \langle \texttt{x}, \texttt{y} \rangle.\ N : C$? Certainly we have to show something about M and something about N. And to be more precise: we have to show that $\Gamma \vdash M : A \times B$, and that $\Gamma, \texttt{x} : A, \texttt{y} : B \vdash N : C$. So the elimination rule is

$$\frac{\Gamma \vdash M : A \times B \quad \Gamma, \texttt{x} : A, \texttt{y} : B \vdash N : C}{\Gamma \vdash \texttt{case}\ M\ \texttt{of}\ \langle \texttt{x}, \texttt{y} \rangle.\ N : C}$$

We also include a type 1, representing a singleton set—the nullary product. The introduction rule is

$$\frac{}{\Gamma \vdash \langle \rangle : 1}$$

If we are using projection syntax, there are no elimination rules. If we are using pattern-match syntax, there is one elimination rule:

$$\frac{\Gamma \vdash M : 1 \quad \Gamma \vdash N : C}{\Gamma \vdash \texttt{case}\ M\ \texttt{of}\ \langle \rangle.\ N : C}$$

5.3 Disjoint Union

The rules for disjoint union are fairly similar to those for `bool`. You might like to think about why this should be so.

How do we form something of type $A + B$? By pairing with a tag. So we have two introduction rules:

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \texttt{inl}\ M : A + B} \qquad \frac{\Gamma \vdash M : B}{\Gamma \vdash \texttt{inr}\ M : A + B}$$

How do we use something of type $A + B$? By pattern-matching it. To prove that $\Gamma \vdash \texttt{case}\ M\ \texttt{of}\ \{\texttt{inl x}.\ N, \texttt{inr x}.\ N'\} : C$, we have to prove something about M, something about N and something about N'. To be more precise, we have to prove that $\Gamma \vdash M : A + B$, that $\Gamma, \texttt{x} : A \vdash N : C$ and that $\Gamma, \texttt{x} : B \vdash N' : C$. So here's the elimination rule:

$$\frac{\Gamma \vdash M : A + B \quad \Gamma, \texttt{x} : A \vdash N : C \quad \Gamma, \texttt{x} : B \vdash N' : C}{\Gamma \vdash \texttt{case}\ M\ \texttt{of}\ \{\texttt{inl x}.\ N, \texttt{inr x}.\ N'\} : C}$$

We also include a type 0 representing the empty set—the nullary disjoint union. It has no introduction rule and the following elimination rule:

$$\frac{\Gamma \vdash M : 0}{\Gamma \vdash \texttt{case}\ M\ \texttt{of}\ \{\} : A}$$

5.4 Function Space

We're almost done now—we just need the rules for $A \to B$. How do we form something of type $A \to B$? We use λ-abstraction. To show that $\Gamma \vdash M : A \to B$, we need to show that $\Gamma, \texttt{x} : A \vdash M : B$. So the introduction rule is

$$\frac{\Gamma, \texttt{x} : A \vdash M : B}{\Gamma \vdash \lambda \texttt{x}_A.\ M : A \to B}$$

How do we use something of type $A \to B$? By applying it to something of type A. And that gives us something of type B. So the elimination rule is

$$\frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash M\, N : B}$$

6 Substitution

The most important operation on terms (i.e. operation on binding diagrams) is *substitution*. If M and N are terms, we write $M[N/\mathtt{x}]$ for the term in which we substitute N for $\mathtt{x}$ in M. For example, if M is $(\mathtt{x}+\mathtt{y})\times 3$ and N is $(\mathtt{y}\times 2)$ then $M[N/\mathtt{x}]$ is $((\mathtt{y}\times 2)+\mathtt{y})\times 3$. It is most important to remember here that terms are binding diagrams:

1. Suppose M is $\mathtt{x}+\mathtt{let}\ 3\ \mathtt{be}\ \mathtt{x}.\ \mathtt{x}\times 7$, and N is $\mathtt{y}\times 2$, Writing these as binding diagrams ensures that we substitute for only the *free* occurrences. We therefore obtain $(\mathtt{y}\times 2)+\mathtt{let}\ 3\ \mathtt{be}\ \mathtt{x}.\ \mathtt{x}\times 7$.
2. Suppose M is $\mathtt{let}\ 3\ \mathtt{be}\ \mathtt{y}.\ \mathtt{x}+\mathtt{y}$, and N is $\mathtt{y}\times 2$. Writing these as binding diagrams ensures that the free occurrence of $\mathtt{y}$ in N remains free. So we obtain $\mathtt{let}\ 3\ \mathtt{be}\ \mathtt{z}.\ (\mathtt{y}\times 2)+\mathtt{z}$. If we try to substitute naively, we get $\mathtt{let}\ 3\ \mathtt{be}\ \mathtt{y}.\ (\mathtt{y}\times 2)+\mathtt{y}$. That's the wrong answer, because the free occurrence of $\mathtt{y}$ in N has been *captured*. "Substitution" always means *capture-free* substitution.

Exercise 8. Substitute

$$\mathtt{let}\ \mathtt{x}+1\ \mathtt{be}\ \mathtt{x}.\ \mathtt{x}+\mathtt{y}$$

for $\mathtt{x}$ in

$$\mathtt{x}+(\mathtt{let}\ \mathtt{x}+2\ \mathtt{be}\ \mathtt{y}.\ \mathtt{let}\ \mathtt{x}+\mathtt{y}\ \mathtt{be}\ \mathtt{x}.\ \mathtt{x}+\mathtt{y})$$

7 Exercises

1. Turn some of the descriptions of integers from the notes into expressions. Write out binding diagrams and proof trees for these examples (hint: use a large piece of paper in landscape orientation).
2. What integer is

$$\begin{array}{l}\mathsf{let}\ 3\ \mathsf{be}\ x.\\ \mathsf{let}\ \mathsf{inl}\ \lambda y_{\mathbb{Z}}.\,(x+y)\ \mathsf{be}\ u.\\ \mathsf{let}\ 4\ \mathsf{be}\ x.\\ x+(\mathsf{case}\ u\ \mathsf{of}\ \{\mathsf{inl}\ f.\ f\,2, \mathsf{inr}\ f.\ 0\})\end{array}$$

?

3. What integer is

$$\begin{array}{l}\mathsf{let}\ \lambda x_{\mathbb{Z}}.\ \mathsf{inl}\ \lambda y_{\mathbb{Z}}.\ (x+y)\ \mathsf{be}\ f.\\ \mathsf{let}\ f\,0\ \mathsf{be}\ u.\\ \mathsf{case}\ u\ \mathsf{of}\ \{\\ \quad \mathsf{inl}\ g.\ \mathsf{let}\ f\,1\ \mathsf{be}\ v.\ \mathsf{case}\ v\ \mathsf{of}\ \{\mathsf{inl}\ h.\ g\,3, \mathsf{inr}\ h.\ 0\},\\ \quad \mathsf{inr}\ g.\ 0\\ \}\end{array}$$

?

4. (variant record type) For sets A, B, C, D, E, we define $\alpha(A, B, C, D, E)$ to be the set of tuples

$$\{\langle \#\mathsf{left}, x, y\rangle | x \in A, y \in B\} \cup \{\langle \#\mathsf{right}, x, y, z\rangle | x \in C, y \in D, z \in E\}$$

Now think of α as an operation on types. Give typing rules for
   - $\langle \#\mathsf{left}, M, N\rangle$
   - $\langle \#\mathsf{right}, M, N, P\rangle$
   - $\mathtt{case}\ M\ \mathtt{of}\ \{\langle \#\mathsf{left}, \mathtt{x}, \mathtt{y}\rangle.\ N,\ \langle \#\mathsf{right}, \mathtt{x}, \mathtt{y}, \mathtt{z}\rangle.\ N'\}$

   i.e. two introduction rules and one elimination rule for α.
5. (variant function type) For sets A, B, C, D, E, F, G, we define $\beta(A, B, C, D, E, F, G)$ to be the set of functions that take
   - a sequence of arguments $(\#\mathsf{left}, x, y)$, where $x \in A$ and $y \in B$, to an element of C
   - a sequence of arguments $(\#\mathsf{right}, x, y, z)$, where $x \in D$ and $y \in E$ and $z \in F$, to an element of G.

   Thus the first argument is always a tag, indicating how many other arguments there are, what their type is, and what the type of the result should be.
   Now think of β as an operation on types. Give typing rules for
   - $M(\#\mathsf{left}, N, N')$
   - $M(\#\mathsf{right}, N, N', N'')$
   - $\lambda\{(\#\mathsf{left}, \mathtt{x}, \mathtt{y}).M, (\#\mathsf{right}, \mathtt{x}, \mathtt{y}, \mathtt{z}).M'\}$

   i.e. two elimination rules and one introduction rule for β.