Distributive Logic

J.R.B. Cockett
Department of Computer Science
University of Tennessee
Knoxville
TN 37996-1301

October 28, 1992

Abstract

Equational logics are very good at handling algebraic theories. However, in programming languages which have branch instructions this sort of logic is ultimately inadequate. A branch instruction is a map to a coproduct and the problem with equational logics is that they cannot handle coproduct types.

This paper introduces an equational style logic which incorporates the coproduct as a fundamental component. It is proven that the logic corresponds to distributive categories, that is categories with a final object, finite products, and coproducts, in which the product distributes over the coproduct.

1 Introduction

Ode To a Lost Coproduct

In times of yore when **else**s dangled
when **case** statements were new fangled
and **goto**s little jumps did no harm
the coproduct governed every branching arm.

But coproduct, coproduct where did you go
when you failed to fit logics fine flow?
And how did programs still flow about
when lambda calculus missed you out?

The coproduct is a much underrated construction missing from most standard logics and yet fundamental to the flow of control in programs. It is surprising that even with the increased interest in appling logic to computer science that the lack of this construct has not been more sorely felt.

In almost all settings of computational interest we are interested in a coproduct which distributes over the product. In this paper I will describe "distributive logic" which is an equational style logic which has the distributing coproduct as one of its basic constructs. Mechanisms are described for translating this logic into categorical combinators (categorical notation) and for translating in the reverse direction. We shall show that these translations are consistent and give an equivalence of representations between distributive logic and distributive categories.

2 Distributive Categories

A *distributive category* is a category with a final object, finite products and finite coproducts such that the product distributes over the coproduct. It is well-known that the initial distributive category is (equivalent to) finite sets, $Sets_{fin}$. More generally, distributive categories are of some computational interest as they provide a model for acyclic code.

One way of setting up a distributive category is by introducing the final object and products together with a *parameterized* coproduct. This essentially says that whenever we have maps:

$$f_1 : A \times C \longrightarrow G, f_2 : B \times C \longrightarrow G$$

there is a unique map

$$dst_0(f_1, f_2) : (A + B) \times C \longrightarrow G$$

such that $\langle p_0.b_0, p_1\rangle.dst_0(f_1, f_2) = f_1$ and $\langle p_0.b_1, p_1\rangle.dst_0(f_1, f_2) = f_2$.

It suffices, however, to add $d_0 = dst_0(b_0, b_1) : (A + B) \times C \longrightarrow (A \times C) + (B \times C)$ as inverse to the map $\langle\langle p_0.b_0, p_1\rangle; \langle p_0.b_1, p_1\rangle\rangle$ to obtain the same result. This is the usual approach and the one we shall take in this paper in order to give a simple axiomatization of distributive categories. However, the flavor of what we intend is a parameterization.

It is worth noting in the above development that we have actually *right* parameterized the coproduct and, of course, we could have *left* parameterized the coproduct with dst_1. In this case the distributor $d_1 = dst_1(b_0, b_1) : C \times (A + B) \longrightarrow (C \times A) + (C \times B)$ is enough to characterize the parameterization.

Of course, *left* parameterizing a construct implies the category is *right* parameterized in that construct as well. Thus, theoretically it suffices to consider a one-sided parameterization. Practically it may be advantageous to have both sides.

A distributive category can be axiomatized as follows (where the usual type constraint on composition and product and coproduct pairs is assumed):

Associativity: $x.(y.z) = (x.y).z$

Identity: $x.i = i.x = x$

Final: $x.! = !$

Projection: $\langle x, y\rangle.p_0 = x, \langle x, y\rangle.p_1 = y$

Product pairing: $\langle x.p_0, x.p_1\rangle = x$

Embedding: $b_0.\langle x; y\rangle = x, b_1.\langle x; y\rangle = y$

Coproduct pairing: $\langle b_0.x; b_1.x\rangle = x$

Distribution:

$$d_0.\langle\langle p_0.b_0, p_1\rangle; \langle p_0.b_1, p_1\rangle\rangle = i$$
$$\langle\langle p_0.b_0, p_1\rangle; \langle p_0.b_1, p_1\rangle\rangle.d_0 = i$$

The notation used for the maps above is what shall be called *categorical combinator* notation. An interesting aspect of this notation is that, as the type information is suppressed, it is polymorphic. When specialized to the distributive category setting we shall call them *distributive category combinators*.

The main purpose of this paper is to obtain an equivalent *equational calculus* together with translation mechanisms both ways. Equational notation has a considerable advantage as it eliminates the projection combinators. However, it is also impossible to express common subexpressions equationally and this makes it clumsy as a programming language if left unadorned. On the other hand, the combinator language is very expressive in this respect; but it is almost unreadable! A hybrid language would therefore be a useful compromise. The translation mechanisms which are described below make this possible.

The following are elementary results for distributive categories:

Lemma 2.1

(i) $\langle x.b_0, y\rangle.d_0 = \langle x, y\rangle.b_0$

(ii) $\langle x.b_1, y\rangle.d_0 = \langle x, y\rangle.b_1$

(iii) $\langle x, y\rangle.d_0 = \langle x, i\rangle.d_0.\langle\langle p_0, p_1.y\rangle.b_0; \langle p_0, p_1.y\rangle.b_1\rangle.$

Free distributive categories have a terminating confluent rewriting system for their elements. Also the equality of maps in such categories is decidable.

3 Distributive Logic

A distributive theory, $\mathcal{D} = (S, F, E)$, consists of a set of *primitive types*, S, a set of *function symbols*, F, and a set of *equations*. The *types* of a distributive theory are generated from the primitive types. Each function symbol has a *domain* and a *codomain* in the types. The *terms* of a distributive theory are generated from the function symbols, F, some basic functions and the variables.

3.1 Types

To construct the theory it is first necessary to describe the *types*. The types, $\mathcal{F}_{+,\times}(S)$, of a distributive theory are given by the following recursive definition which builds on the primitive types:

(i) 1 is a type,

(ii) a primitive type, s, is a type,

(iii) if s_0 and s_1 are types then $(s_0 \times s_1)$ is a type,

(iv) if s_0 and s_1 are types then $(s_0 + s_1)$ is a type.

A type, s, is a *product type* if $s = s_0 \times s_1$. If s is not a product type then it is a *sum type*; explicitly it is either of the form $s_0 + s_1$ or is primitive.

3.2 Variables and variable bases

Every sum type, s, has associated with it a stock of variables:

$$x_1^s, x_2^s, \ldots.$$

A product type does not have any associated variables but has *variable bases*. These are constructed as follows:

(i) () is a variable base for 1,

(ii) a variable x_i^s is a variable base for its sum type s,

(iii) if e_0 and e_1 are variable bases having no common variables for s_0 and s_1, then (e_0, e_1) is a variable base for $s_0 \times s_1$.

3.3 Terms

The function symbols, F, are indexed by pairs of types:

$$\langle dom, codom \rangle : F \longrightarrow \mathcal{F}_{+,\times}(S) \times \mathcal{F}_{+,\times}(S)$$

and are used to build the *terms*. Each term has an associated type and these are given in the following recursive specification:

(i) () is a term of type 1,

(ii) a variable x_i^s is a term of type s,

(iii) if t_0 and t_1 are terms of type s_0 and s_1 respectively, then (t_0, t_1) is a term of type $s_0 \times s_1$,

(iv) if t is a term of type $s_0 \times s_1$, then $p_0(t)$ is a term of type s_0 and $p_1(t)$ is a term of type s_1,

(v) if t is a term of sort s, then $b_0(t)$ and $b_1(t)$ are terms of type $s + s'$ and $s' + s$ respectively, where s' can be any type,

(vi) if $f : s_0 \longrightarrow s_1$ is a function symbol and t is a term of type s_0, then $f(t)$ is a term of type s_1,

(vii) if t_0 and t_1 are terms of type s, t is a term of type $s_0 + s_1$, and e_0 and e_1 are variable bases for s_0 and s_1, then

$$\left[\begin{array}{lcl} b_0(e_0) & \mapsto & t_0 \\ b_1(e_1) & \mapsto & t_1 \end{array} \right] (t)$$

is a term of type s. The term enclosed in square braces is called a *coproduct function*.

Only the last term construction rule is at all unusual. The construct represents a map from a coproduct. In the term the variables in the variable bases e_0 and e_1 are bound with scope the term to which they are assigned. In this respect the logic is significantly different from equational logic.

3.4 Free variables and substitution

As there are bound variables it becomes important to be clear about which variables are *free* and which are bound. This can be determined recursively by:

(i) $vars(()) = \{\}$

(ii) $vars(x_i^s) = \{x_i^s\}$

(iii) $vars((t_0, t_1)) = vars(t_0) \cup vars(t_1)$

(iv) $vars(p_0(t)) = vars(p_1(t)) = vars(t)$

(v) $vars(b_0(t)) = vars(b_1(t)) = vars(t)$

(vi) $vars(f(t)) = vars(t)$

(vii)

$$vars\left(\left[\begin{array}{lcl} b_0(e_0) & \mapsto & t_0 \\ b_1(e_1) & \mapsto & t_1 \end{array}\right](t)\right) = (vars(t_0) - vars(e_0)) \cup (vars(t_1) - vars(e_1)) \cup vars(t).$$

We shall loosely refer to a variable as being *in* a term when that variable occurs as a free variable in the term. The bound variables are invisible and can be renamed without changing the meaning of the term.

The notation $\sigma_{e:=t'}(t)$ where e is a variable base for t and t' is a term of the same type as e is a way of indicating the term obtained by substituting the variables of t by the appropriate component of t'.

While it is obvious what is meant by substitution it will be useful in the sequel to have a systematic way of discussing substitution. We therefore provide a recursive definition:

(i) $\sigma_{e:=t'}(()) = ()$

(ii) $\sigma_{x:=t'}(x) = t'$
$\sigma_{(e_0,e_1):=(t_0',t_1')}(x) = \sigma_{e_0:=t_0'}(x)$, if $x \in e_0$
$\sigma_{(e_0,e_1):=(t_0',t_1')}(x) = \sigma_{e_1:=t_1'}(x)$, if $x \in e_1$

(iii) $\sigma_{e:=t'}((t_0, t_1)) = (\sigma_{e:=t'}(t_0), \sigma_{e:=t'}(t_1))$

(iv) $\sigma_{e:=t'}(f(t)) = f(\sigma_{e:=t'}(t))$ where f is a function symbol or any of the projections or embeddings,

(v)

$$\sigma_{e:=t'}(\left[\begin{array}{lcl} b_0(e_0) & \mapsto & t_0 \\ b_1(e_1) & \mapsto & t_1 \end{array}\right](t)) = \left[\begin{array}{lcl} b_0(e_0) & \mapsto & \sigma_{(e_0,e):=(e_0,t')}(t_0) \\ b_1(e_1) & \mapsto & \sigma_{(e_1,e):=(e_1,t')}(t_1) \end{array}\right](\sigma_{e:=t'}(t)).$$

3.5 Abstract maps and equations

An *abstract map* is a pair $[e \mapsto t]$ where e is a variable base and t is a term such that the variables of e contain the free variables of t.

An *equation*, strictly speaking, is a pair of abstract maps:

$$[e_0 \mapsto t_0] = [e_1 \mapsto t_1]$$

where e_0 and e_1 are variable bases of the same type and t_1 and t_2 have the same type. We may always rename the variables in one of the abstract maps to make $e_0 = e_1$. When this is done it is sensible to write the equation subscripting the equal sign with the common variable base. This gives the following notation:

$$t_0 =_e t_1 .$$

Use of this notation should not, however, be regarded as a commitment to the names of the variables but simply a convenience to avoid the above expanded form.

In proofs using equations the form of the variable base is unimportant. If the base contains no more than the free variables in the terms, then it shall be omitted as any variable base would then suffice.

3.6 Axioms and inference rules

All the equations in E are treated as axioms. In addition there are the following axioms which are always present:

Reflexivity:

$$t =_e t$$

Projection:

$$p_0(t_1, t_2) =_e t_1 , \; p_1(t_1, t_2) =_e t_2$$

Product pairing:

$$(p_0(t), p_1(t)) = t$$

Embedding:

$$\left[\begin{array}{lcl} b_0(e_0) & \mapsto & t_0 \\ b_1(e_1) & \mapsto & t_1 \end{array} \right] (b_0(t)) \quad =_e \quad \sigma_{(e_0,e):=(t,e)}(t_0)$$

$$\left[\begin{array}{lcl} b_0(e_0) & \mapsto & t_0 \\ b_1(e_1) & \mapsto & t_1 \end{array} \right] (b_1(t)) \quad =_e \quad \sigma_{(e_1,e):=(t,e)}(t_1)$$

where $\sigma_{e:=t'}(t)$ is substitution of the variables of e in t with the appropriate component(s) of t'.

Distributive coproduct pairing:

$$\left[\begin{array}{lcl} b_0(e_0) & \mapsto & \sigma_{(z,e):=(b_0(e_0),e)}(t) \\ b_1(e_1) & \mapsto & \sigma_{(z,e):=(b_1(e_1),e)}(t) \end{array} \right] (z) =_{(z,e)} t.$$

The rules of inference are as follows:

Symmetry:

$$\frac{[t_1 =_e t_2]}{t_2 =_e t_1}$$

Transitivity:

$$\frac{[t_1 =_e t_2, t_2 =_e t_3]}{t_1 =_e t_3}$$

Substitution:

$$\frac{[t_1 =_{e_1} t_2, t_1' =_{e_2} t_2']}{\sigma_{e_1:=t_1'}(t_1) =_{e_2} \sigma_{e_1:=t_2'}(t_2)}$$

where e_1, t_1', and t_2' are of the same type and $\sigma_{e:=t'}(t)$ is substitution of the variables of e in t with the appropriate component(s) of t'.

Lemma 3.1

(i)

$$\left(\begin{bmatrix} b_0(e_0) & \mapsto & t_0 \\ b_1(e_1) & \mapsto & t_1 \end{bmatrix}(t_0), t\right) =_e \begin{bmatrix} b_0(e_0) & \mapsto & (t_0, t) \\ b_1(e_1) & \mapsto & (t_1, t) \end{bmatrix}(t_0)$$

(ii)

$$f(\begin{bmatrix} b_0(e_0) & \mapsto & t_0 \\ b_1(e_1) & \mapsto & t_1 \end{bmatrix}(t)) =_e \begin{bmatrix} b_0(e_0) & \mapsto & f(t_0) \\ b_1(e_1) & \mapsto & f(t_1) \end{bmatrix}(t)$$

where f is a projection, embedding, function symbol, or coproduct function.

Proof. There is only one minor difficulty in the proofs below and this arises as f, t, t_0, or t_1 might contain the variable z, which is to be split for the coproduct. However, this can be circumvented by changing all occurrences of z in these maps to z' and later substituting z for z'. Thus, we may assume that z does not occur in any of the above maps.

The first proof follows the pattern set by the second proof:

$$\begin{aligned}
& f(\begin{bmatrix} b_0(z_0) & \mapsto & t_0 \\ b_1(z_1) & \mapsto & t_1 \end{bmatrix}(z)) \\
=_{(z,e)} & \begin{bmatrix} b_0(z_0) & \mapsto & \sigma_{(z,e):=(b_0(z_0),e)}(f(\begin{bmatrix} b_0(z_0) & \mapsto & t_0 \\ b_1(z_1) & \mapsto & t_1 \end{bmatrix}(z))) \\ b_1(z_1) & \mapsto & \sigma_{(z,e):=(b_0(z_0),e)}(f(\begin{bmatrix} b_0(z_0) & \mapsto & t_0 \\ b_1(z_1) & \mapsto & t_1 \end{bmatrix}(z))) \end{bmatrix}(z) \\
=_{(z,e)} & \begin{bmatrix} b_0(z_0) & \mapsto & f(\begin{bmatrix} b_0(z_0) & \mapsto & t_0 \\ b_1(z_1) & \mapsto & t_1 \end{bmatrix}(b_0(z_0))) \\ b_1(z_1) & \mapsto & f(\begin{bmatrix} b_0(z_0) & \mapsto & t_0 \\ b_1(z_1) & \mapsto & t_1 \end{bmatrix}(b_1(z_1))) \end{bmatrix}(z) \\
=_{(z,e)} & \begin{bmatrix} b_0(z_0) & \mapsto & f(t_0) \\ b_1(z_1) & \mapsto & f(t_1) \end{bmatrix}(z).
\end{aligned}$$

□

4 The equivalence of combinators to equational logic

In order to show that this logic is equivalent to the combinator representation of distributive categories, it is necessary to provide translation mechanisms between the two formulations. It is then necessary to show that the translations are consistent and inverse. To show that a translation is consistent it is necessary to show that equivalence expressions of the domain are taken to equivalent expressions of the codomain.

4.1 Translation to combinators

In this section we show that there is a consistent translation from distributive logic into the categorical combinators of a distributive category. To do this we must first describe the translation procedure which we shall denote by $\mathcal{C}$:

(i)
$$\mathcal{C}([e \mapsto ()]) = !$$

(ii)
$$\begin{aligned} \mathcal{C}([x \mapsto x]) &= i \\ \mathcal{C}([(e_0, e_1) \mapsto x]) &= p_0.\mathcal{C}([e_0 \mapsto x]), \text{ if } x \in e_0 \\ \mathcal{C}([(e_0, e_1) \mapsto x]) &= p_1.\mathcal{C}([e_1 \mapsto x]), \text{ if } x \in e_1 \end{aligned}$$

(iii)
$$\mathcal{C}([e \mapsto (t_0, t_1)]) = \langle \mathcal{C}([e \mapsto t_0]), \mathcal{C}([e \mapsto t_1]) \rangle$$

(iv)
$$\begin{aligned} \mathcal{C}([e \mapsto p_0(t)]) &= \mathcal{C}([e \mapsto t]).p_0 \\ \mathcal{C}([e \mapsto p_1(t)]) &= \mathcal{C}([e \mapsto t]).p_1 \end{aligned}$$

(v)
$$\begin{aligned} \mathcal{C}([e \mapsto b_0(t)]) &= \mathcal{C}([e \mapsto t]).b_0 \\ \mathcal{C}([e \mapsto b_1(t)]) &= \mathcal{C}([e \mapsto t]).b_1 \end{aligned}$$

(vi)
$$\mathcal{C}([e \mapsto f(t)]) = \mathcal{C}([e \mapsto t]).f$$

(vii)
$$\begin{aligned} &\mathcal{C}([e \mapsto \begin{bmatrix} b_0(e_0) & \mapsto & t_0 \\ b_1(e_1) & \mapsto & t_1 \end{bmatrix}(t)]) \\ &= \langle \mathcal{C}([e \mapsto t]), i \rangle.d_0.\langle \mathcal{C}([(e_0, e) \mapsto t_0]); \mathcal{C}([(e_1, e) \mapsto t_1]) \rangle. \end{aligned}$$

We may now state the desired result:

Theorem 4.1 *$\mathcal{C}$ gives a consistent translation from distributive logic into distributive category combinators.*

The proof involves checking all the axioms and inference rules for consistency. In order to simplify notation we shall omit explicit mention of the translation function $\mathcal{C}$, thus by $[e \mapsto t]$ shall be meant the translation of this into combinators.

It is useful to start the proof by proving an essential result which shows that substitution is really just composition:

Lemma 4.2 (Substitution is composition)

$$[e \mapsto t].[e' \mapsto t'] = [e \mapsto \sigma_{e':=t}(t')].$$

Proof. The proof works by a structural induction on the complexity of the terms. The induction works by assuming the result holds for subterms and proving that this means it holds for the whole term. Essentially the proof follows the definition of substitution:

(i) If $t' = ()$ then the result is clear.

(ii) If $t' = x$ then $[e \mapsto \sigma_{e':=t'}(x)]$ breaks into three subcases. The last two subcases, however, are almost identical so that we shall handle the first two:

- $e' = x$ in which case $\sigma_{e':=t}(x) = t$ and we have

  $$[e \mapsto t].[x \mapsto x] = [e \mapsto t]$$

  which is certainly the case.
- $e' = (e_0, e_1)$ and $t' = (t_0, t_1)$ and $x \in e_0$ in which case:

  $$\sigma_{(e_0,e_1):=(t_0,t_1)}(x) = \sigma_{e_0:=t_0}(x)$$

  and

  $$\begin{aligned} & [e \mapsto (t_0, t_1)].[(e_0, e_1) \mapsto x] \\ = \;& \langle [e \mapsto t_0], [e \mapsto t_1] \rangle . p_0 . [e_0 \mapsto x] \\ = \;& [e \mapsto t_0].[e_0 \mapsto x]. \end{aligned}$$

(iii) If $t' = (t_0, t_1)$ then

$$\begin{aligned} & [e \mapsto t].[e' \mapsto (t_0, t_1)] \\ = \;& [e \mapsto t].\langle [e' \mapsto t_0], [e' \mapsto t_1] \rangle \\ = \;& \langle [e \mapsto t].[e' \mapsto t_0], [e \mapsto t].[e' \mapsto t_1] \rangle \\ = \;& \langle [e \mapsto \sigma_{e':=t}(t_0)], [e \mapsto \sigma_{e':=t}(t_1)] \rangle \\ = \;& [e \mapsto \sigma_{e':=t}((t_0, t_1))]. \end{aligned}$$

(iv) If $t' = f(t_0)$ where f is a function symbol, projection, or embedding then:

$$\begin{aligned}
&[e \mapsto t].[e' \mapsto f(t_0)] \\
&\quad = \quad [e \mapsto t].[e' \mapsto t'].f \\
&\quad = \quad [e \mapsto \sigma_{e':=t}(t')].f \\
&\quad = \quad [e \mapsto f(\sigma_{e':=t}(t'))].
\end{aligned}$$

(v) If t' is a coproduct term then we have the following:

$$\begin{aligned}
&[e \mapsto t].[e' \mapsto \left[\begin{array}{lcl} b_0(e_0) & \mapsto & t_0 \\ b_1(e_1) & \mapsto & t_1 \end{array}\right](t_2)] \\
&\quad = \quad [e \mapsto t].\langle [e' \mapsto t_2], i\rangle.d_0.\langle [(e_0,e') \mapsto t_0]; [(e_1,e') \mapsto t_1]\rangle \\
&\quad = \quad \langle [e \mapsto \sigma_{e':=t}(t_2)], [e \mapsto t]\rangle.d_0.\langle [(e_0,e') \mapsto t_0]; [(e_1,e') \mapsto t_1]\rangle \\
&\quad = \quad \langle [e \mapsto \sigma_{e':=t}(t_2)], i\rangle.d_0.\langle \langle p_0,p_1.[e \mapsto t]\rangle.[(e_0,e') \mapsto t_0]; \langle p_0,p_1.[e \mapsto t]\rangle. \\
&\qquad\qquad\qquad [(e_1,e') \mapsto t_1]\rangle \\
&\quad = \quad \langle [e \mapsto \sigma_{e':=t}(t_2)], i\rangle.d_0.\langle [(e_0,e) \mapsto (e_0,t)].[(e_0,e') \mapsto t_0]; [(e_1,e) \mapsto (e_1,t)]. \\
&\qquad\qquad\qquad [(e_1,e') \mapsto t_1]\rangle \\
&\quad = \quad \langle [e \mapsto \sigma_{e':=t}(t_2)], i\rangle.d_0.\langle [(e_0,e) \mapsto \sigma_{(e_0,e'):=(e_0,t)}(t_0)]; [(e_1,e) \mapsto \sigma_{(e_1,e'):=(e_1,t)}(t_1)]\rangle \\
&\quad = \quad [e \mapsto \sigma_{e':=t}(\left[\begin{array}{lcl} b_0(e_0) & \mapsto & t_0 \\ b_1(e_1) & \mapsto & t_1 \end{array}\right](t_2))].
\end{aligned}$$

□

This allows us to complete the proof that the translation to combinators is consistent:

Proof (4.1 continued). It is immediate that reflexivity, projection, and product pairing are all consistently translated. Similarly, the inference rules of symmetry, transitivity, and substitution (given in the above lemma) are consistently translated. The only difficulties arise with embedding and distributive pairing. The proof for the first embedding is as follows:

$$\begin{aligned}
&[e \mapsto \left[\begin{array}{lcl} b_0(e_0) & \mapsto & t_0 \\ b_1(e_1) & \mapsto & t_1 \end{array}\right](b_0(t))] \\
&\quad = \quad \langle [e \mapsto b_0(t)], i\rangle.d_0.\langle [(e_0,e) \mapsto t_0]; [(e_1,e) \mapsto t_1]\rangle \\
&\quad = \quad \langle [e \mapsto t].b_0, i\rangle.d_0.\langle [(e_0,e) \mapsto t_0]; [(e_1,e) \mapsto t_1]\rangle \\
&\quad = \quad \langle [e \mapsto t], i\rangle.b_0.\langle [(e_0,e) \mapsto t_0]; [(e_1,e) \mapsto t_1]\rangle \\
&\quad = \quad \langle [e \mapsto t], i\rangle.[(e_0,e) \mapsto t_0] \\
&\quad = \quad [e \mapsto (t,e)].[(e_0,e) \mapsto t_0] \\
&\quad = \quad [e \mapsto \sigma_{(e_0,e):=(t,e)}(t_0)].
\end{aligned}$$

The proof for the second embedding is similar.

The proof of the consistency of distributive pairing is as follows:

$$
\begin{aligned}
&[(z,e) \mapsto \begin{bmatrix} b_0(e_0) \mapsto \sigma_{(z,e):=(b_0(e_0),e)}(t) \\ b_1(e_1) \mapsto \sigma_{(z,e):=(b_1(e_1),e)}(t) \end{bmatrix}(z)] \\
&= \langle [(z,e) \mapsto z], i\rangle.d_0.\langle [(e_0,(z,e)) \mapsto \sigma_{(z,e):=(b_0(e_0),e)}(t)]; [(e_1,(z,e)) \mapsto \sigma_{(z,e):=(b_1(e_1),e)}(t)]\rangle \\
&= \langle p_0, i\rangle.d_0.\langle [(e_0,(z,e)) \mapsto (b_0(e_0),e)].[(z,e) \mapsto t]; [(e_1,(z,e)) \mapsto (b_1(e_1),e)].[(z,e) \mapsto t]\rangle \\
&= \langle p_0, i\rangle.d_0.\langle \langle p_0.b_0, p_1.p_1\rangle; \langle p_0.b_1, p_1.p_1\rangle\rangle.[(z,e) \mapsto t] \\
&= \langle p_0, p_1\rangle.[(z,e) \mapsto t] \\
&= [(z,e) \mapsto t].
\end{aligned}
$$

□

4.2 Translation from combinators

It is possible to translate in the opposite direction. We shall denote this translation $\mathcal{L}$ but then, as above, omit it to reduce notation. The translation process is given by following the specification of a category.

In giving the translation it is convenient to use some notational conveniences. They are:

- $\mathcal{L}(x) = [e \mapsto \mathcal{L}_e(x)]$ where e is a variable base for the domain of x,
- $[e \mapsto t].[e' \mapsto t'] = [e \mapsto \sigma_{e':=t}(t')]$
- $\langle [e \mapsto t_0], [e' \mapsto t_1]\rangle = [e \mapsto (t_0, \sigma_{e':=e}(t_1)]$
- $\langle [e_0 \mapsto t_0]; [e_1 \mapsto t_1]\rangle = \begin{bmatrix} b_0(e_0) & \mapsto & t_0 \\ b_1(e_1) & \mapsto & t_1 \end{bmatrix}(z)$.

Using this notation we have:

(i) $\mathcal{L}(x.y) = \mathcal{L}(x).\mathcal{L}(y)$

(ii) $\mathcal{L}(\langle x, y\rangle) = \langle \mathcal{L}(x), \mathcal{L}(y)\rangle$

(iii) $\mathcal{L}(\langle x; y\rangle) = \langle \mathcal{L}(x); \mathcal{L}(y)\rangle$

(iv) $\mathcal{L}(i) = [e \mapsto e]$ where e is a variable base for the object for which i is the identity.

(v) if f is a function symbol, projection, or embedding:

$$\mathcal{L}(f) = [e \mapsto f(e)]$$

where e is a variable base for the domain of f,

(vi)

$$\mathcal{L}(d_0) = [(x,e) \mapsto \begin{bmatrix} b_0(x_0) & \mapsto & (x_0,e) \\ b_1(x_1) & \mapsto & (x_1,e) \end{bmatrix}(x)].$$

Our purpose is now to prove:

Proposition 4.3 *$\mathcal{L}$ is a consistent translation scheme from distributive category combinators to distributive logic abstract maps.*

It is clearly useful to know that substitution is associative:

Lemma 4.4 (Substitution is associative)

$$\sigma_{e_2:=t_1}(\sigma_{e_3:=t_2}(t_3)) = \sigma_{e_3:=\sigma_{e_2:=t_1}(t_2)}(t_3).$$

Proof. The proof is a structural induction on the term t_3 with a simpler structural induction on the structure of the variable base e_2.

Start by supposing that $t_3 = x$, a variable. There are two cases to check: $e_3 = x$, a straight substitution of the variable or, $e_3 = (e_0, e_1)$, a complex substitution. In the latter case there are two subcases depending on whether x occurs in e_0 or e_1. However, these cases are similar so we may assume x occurs in e_0:

$$\begin{aligned}
&\sigma_{e_2:=t_1}(\sigma_{x:=t_2}(x)) \\
&\quad = \quad \sigma_{e_2:=t_1}(t_2) \\
&\quad = \quad \sigma_{x:=\sigma_{e_2:=t_1}(t_2)}(x) \\
&\sigma_{e_2:=t_1}(\sigma_{(e_0,e_1):=(t_{20},t_{21})}(x)), \quad x \in e_0 \\
&\quad = \quad \sigma_{e_2:=t_1}(\sigma_{e_0:=t_{20}}(x)) \\
&\quad = \quad \sigma_{e_0:=\sigma_{e_2:=t_1}(t_{20})}(x) \\
&\quad = \quad \sigma_{(e_0,e_1):=\sigma_{e_2:=t_1}((t_{20},t_{21}))}(x).
\end{aligned}$$

Next we consider product pairing. First notice that for () the result is obvious. Now consider $t_3 = (t_{30}, t_{31})$:

$$\begin{aligned}
&\sigma_{e_2:=t_1}(\sigma_{e_3:=t_2}((t_{30}, t_{31}))) \\
&\quad = \quad (\sigma_{e_2:=t_1}(\sigma_{e_3:=t_2}(t_{30})), \sigma_{e_2:=t_1}(\sigma_{e_3:=t_2}(t_{31}))) \\
&\quad = \quad (\sigma_{e_3:=\sigma_{e_2:=t_1}(t_2)}(t_{30}), \sigma_{e_3:=\sigma_{e_2:=t_1}(t_2)}(t_{31})) \\
&\quad = \quad \sigma_{e_3:=\sigma_{e_2:=t_1}(t_2)}((t_{30}, t_{31})).
\end{aligned}$$

For function symbols, projections, and embeddings we have:

$$\begin{aligned}
&\sigma_{e_2:=t_1}(\sigma_{e_3:=t_2}(f(t))) \\
&\quad = \quad f(\sigma_{e_2:=t_1}(\sigma_{e_3:=t_2}(t)) \\
&\quad = \quad f(\sigma_{e_3:=\sigma_{e_2:=t_1}(t_2)}(t) \\
&\quad = \quad \sigma_{e_3:=\sigma_{e_2:=t_1}(t_2)}(f(t))
\end{aligned}$$

Finally for coproduct functions we have:

$$\begin{aligned}
&\sigma_{e_2:=t_1}(\sigma_{e_3:=t_2}(\left[\begin{array}{ll} b_0(e_0) & \mapsto t'_0 \\ b_1(e_1) & \mapsto t'_1 \end{array}\right](t))) \\
&\quad = \quad \sigma_{e_2:=t_1}(\left[\begin{array}{lll} b_0(e_0) & \mapsto & \sigma_{(e_0,e_3):=(e_0,t_2)}(t'_0) \\ b_1(e_1) & \mapsto & \sigma_{(e_1,e_3):=(e_1,t_2)}(t'_1) \end{array}\right](\sigma_{e_3:=t_2}(t)))
\end{aligned}$$

$$
\begin{array}{rl}
= & \left[\begin{array}{lcl} b_0(e_0) & \mapsto & \sigma_{(e_0,e_2):=(e_0,t_1)}(\sigma_{(e_0,e_3):=(e_0,t_2)}(t'_0)) \\ b_1(e_1) & \mapsto & \sigma_{(e_1,e_2):=(e_1,t_1)}(\sigma_{(e_1,e_3):=(e_1,t_2)}(t'_1)) \end{array}\right](\sigma_{e_2:=t_1}(\sigma_{e_3:=t_2}(t))) \\
= & \left[\begin{array}{lcl} b_0(e_0) & \mapsto & \sigma_{(e_0,e_3):=\sigma_{(e_0,e_2):=(e_0,t_1)}((e_0,t_2))}(t'_0) \\ b_1(e_1) & \mapsto & \sigma_{(e_1,e_3):=\sigma_{(e_1,e_2):=(e_1,t_1)}((e_1,t_2))}(t'_1) \end{array}\right](\sigma_{e_3:=\sigma_{e_2:=t_1}(t_2)}(t)) \\
= & \left[\begin{array}{lcl} b_0(e_0) & \mapsto & \sigma_{(e_0,e_3):=(e_0,\sigma_{e_2:=t_1}(t_2))}(t'_0) \\ b_1(e_1) & \mapsto & \sigma_{(e_1,e_3):=(e_1,\sigma_{e_2:=t_1}(t_2))}(t'_1) \end{array}\right](\sigma_{e_3:=\sigma_{e_2:=t_1}(t_2)}(t)) \\
= & \sigma_{e_3:=\sigma_{e_2:=t_1}(t_2)}(\left[\begin{array}{lcl} b_0(e_0) & \mapsto & t'_0 \\ b_1(e_1) & \mapsto & t'_1 \end{array}\right](t))
\end{array}
$$

where we observed:

$$\sigma_{(e_0,e_3):=\sigma_{(e_0,e_2):=(e_0,t_1)}((e_0,t_2))}(t'_0) = \sigma_{(e_0,e_3):=(e_0,\sigma_{e_2:=t_1}(t_2))}(t'_0)$$

as e_0 contains no variables in common with t_1.

□

Proof (4.3 continued). The only identities which present any difficulties are the distribution identities:

$$
\begin{array}{l}
d_0.\langle\langle p_0.b_0, p_1\rangle; \langle p_0.b_1, p_1\rangle\rangle \\
\quad = \quad [(z,e) \mapsto \left[\begin{array}{lcl} b_0(e_0) & \mapsto & b_0(e_0,e) \\ b_1(e_1) & \mapsto & b_1(e_1,e) \end{array}\right](z)].[w \mapsto \left[\begin{array}{lcl} b_0(e_0,e) & \mapsto & (b_0(e_0),e) \\ b_1(e_1,e) & \mapsto & (b_1(e_1),e) \end{array}\right](w)] \\
\quad = \quad [(z,e) \mapsto \left[\begin{array}{lcl} b_0(e_0,e) & \mapsto & (b_0(e_0),e) \\ b_1(e_1,e) & \mapsto & (b_1(e_1),e) \end{array}\right](\left[\begin{array}{lcl} b_0(e_0) & \mapsto & b_0(e_0,e) \\ b_1(e_1) & \mapsto & b_1(e_1,e) \end{array}\right](z))].
\end{array}
$$

Substituting $b_0(e_0)$ and $b_1(e_1)$ for z in the term and using distributive coproduct pairing we obtain:

$$
\begin{array}{rl}
= & [(z,e) \mapsto \left[\begin{array}{lcl} b_0(e_0) & \mapsto & (b_0(e_0),e) \\ b_1(e_1) & \mapsto & (b_1(e_1),e) \end{array}\right](z)] \\
= & [(z,e) \mapsto (\left[\begin{array}{lcl} b_0(e_0) & \mapsto & b_0(e_0) \\ b_1(e_1) & \mapsto & b_1(e_1) \end{array}\right](z),e)] \\
= & [(z,e) \mapsto (z,e)].
\end{array}
$$

For the second identity:

$$
\begin{array}{l}
\langle\langle p_0.b_0, p_1\rangle; \langle p_0.b_1, p_1\rangle\rangle \\
\quad = \quad [w \mapsto \left[\begin{array}{lcl} b_0(e_0,e) & \mapsto & (b_0(e_0),e) \\ b_1(e_1,e) & \mapsto & (b_1(e_1),e) \end{array}\right](w)].[(z,e) \mapsto \left[\begin{array}{lcl} b_0(e_0) & \mapsto & b_0(e_0,e) \\ b_1(e_1) & \mapsto & b_1(e_1,e) \end{array}\right](z)] \\
\quad = \quad [w \mapsto \left[\begin{array}{lcl} b_0(e_0) & \mapsto & b_0(e_0,p_1(\left[\begin{array}{lcl} b_0(e_0,e) & \mapsto & (b_0(e_0),e) \\ b_1(e_1,e) & \mapsto & (b_1(e_1),e) \end{array}\right](w))) \\ b_1(e_1) & \mapsto & b_1(e_1,p_1(\left[\begin{array}{lcl} b_0(e_0,e) & \mapsto & (b_0(e_0),e) \\ b_1(e_1,e) & \mapsto & (b_1(e_1),e) \end{array}\right](w))) \end{array}\right]
\end{array}
$$

$$(p_0(\begin{bmatrix} b_0(e_0,e) & \mapsto & (b_0(e_0),e) \\ b_1(e_1,e) & \mapsto & (b_1(e_1),e) \end{bmatrix}(w)))]$$

$$= [w \mapsto \begin{bmatrix} b_0(e_0) & \mapsto & b_0(e_0, \begin{bmatrix} b_0(e_0,e) & \mapsto & e \\ b_1(e_1,e) & \mapsto & e \end{bmatrix}(w))) \\ b_1(e_1) & \mapsto & b_1(e_1, \begin{bmatrix} b_0(e_0,e) & \mapsto & e \\ b_1(e_1,e) & \mapsto & e \end{bmatrix}(w))) \end{bmatrix}$$

$$(\begin{bmatrix} b_0(e_0,e) & \mapsto & b_0(e_0) \\ b_1(e_1,e) & \mapsto & b_1(e_1) \end{bmatrix}(w))]$$

$$= [w \mapsto \begin{bmatrix} b_0(e_0) & \mapsto & b_0(e_0,e) \\ b_1(e_1) & \mapsto & b_1(e_1,e) \end{bmatrix}(\begin{bmatrix} b_0(e_0,e) & \mapsto & b_0(e_0) \\ b_1(e_1,e) & \mapsto & b_1(e_1) \end{bmatrix}(w)))]$$

$$= [w \mapsto \begin{bmatrix} b_0(e_0,e) & \mapsto & \begin{bmatrix} b_0(e_0) & \mapsto & b_0(e_0,e) \\ b_1(e_1) & \mapsto & b_1(e_1,e) \end{bmatrix}(b_0(e_0)) \\ b_1(e_1,e) & \mapsto & \begin{bmatrix} b_0(e_0) & \mapsto & b_0(e_0,e) \\ b_1(e_1) & \mapsto & b_1(e_1,e) \end{bmatrix}(b_1(e_1)) \end{bmatrix}(w)))]$$

$$= [w \mapsto \begin{bmatrix} b_0(e_0,e) & \mapsto & b_0(e_0,e) \\ b_1(e_1,e) & \mapsto & b_1(e_1,e) \end{bmatrix}(w)))]$$

$$= [w \mapsto w].$$

□

Finally we can prove the equivalence:

Theorem 4.5 (Distributive combinator and logic equivalence) *$\mathcal{C}$ and $\mathcal{L}$ gives an equivalence between distributive category combinators and distributive logic.*

Proof. Let us consider $\mathcal{C}(\mathcal{L}(x))$. It suffices to show that the primitive terms and constructions are preserved over the construction:

$$\mathcal{C}(\mathcal{L}(i)) = \mathcal{C}([e \mapsto e]) = i$$

$$\mathcal{C}(\mathcal{L}(!)) = \mathcal{C}([e \mapsto ()]) =!$$

$$\mathcal{C}(\mathcal{L}(x.y)) = \mathcal{C}(\mathcal{L}(x).\mathcal{L}(y)) = \mathcal{C}(\mathcal{L}(x)).\mathcal{C}(\mathcal{L}(y))$$

$$\mathcal{C}(\mathcal{L}(\langle x,y\rangle)) = \mathcal{C}(\langle \mathcal{L}(x), \mathcal{L}(y)\rangle) = \langle \mathcal{C}(\mathcal{L}(x)), \mathcal{C}(\mathcal{L}(y))\rangle$$

$$\mathcal{C}(\mathcal{L}(f)) = \mathcal{C}([e \mapsto f(e)]) = f$$

for f a function symbol projection, or embedding.

The two remaining constructions require some non-trivial manipulation. Notice that the translation symbols are suppressed in line with our earlier convention.

$$\mathcal{C}(\mathcal{L}(\langle x;y\rangle))$$

$$= \mathcal{C}(\langle \mathcal{L}(x); \mathcal{L}(y)\rangle)$$

$$= \mathcal{C}([z \mapsto \begin{bmatrix} b_0(e_0) & \mapsto & \mathcal{L}_{e_0}(x) \\ b_1(e_1) & \mapsto & \mathcal{L}_{e_1}(y) \end{bmatrix}(z)])$$

$$
\begin{aligned}
&= \langle i,i\rangle.d_0.\langle[(e_0,z)\mapsto\mathcal{L}_{e_0}(x)];[(e_1,z)\mapsto\mathcal{L}_{e_1}(y)]\rangle\\
&= \langle i,i\rangle.d_0.\langle p_0.[e_0\mapsto\mathcal{L}_{e_0}(x)];p_0.[e_1\mapsto\mathcal{L}_{e_1}(y)]\rangle\\
&= \langle i,i\rangle.d_0.\langle\langle p_0.b_0,p_1\rangle;\langle p_0.b_0,p_1\rangle\rangle.p-0.\langle[e_0\mapsto\mathcal{L}_{e_0}(x)];[e_1\mapsto\mathcal{L}_{e_1}(y)]\rangle\\
&= \langle[e_0\mapsto\mathcal{L}_{e_0}(x)];[e_1\mapsto\mathcal{L}_{e_1}(y)]\rangle\\
&= \langle x;y\rangle.
\end{aligned}
$$

The last thing we need to check is the effect on d_0:

$$
\begin{aligned}
\mathcal{C}(\mathcal{L}(d_0)) &= \mathcal{C}([(z,e)\mapsto\begin{bmatrix} b_0(e_0) & \mapsto & b_0(e_0,e)\\ b_0(e_1) & \mapsto & b_1(e_1,e)\end{bmatrix}(z)])\\
&= \langle p_0,i\rangle.d_0.\langle[(e_0,(z,e))\mapsto b_0(e_0,e)];[(e_1,(z,e))\mapsto b_1(e_1,e)]\rangle\\
&= \langle p_0,i\rangle.d_0.\langle\langle p0,p_1.p_1\rangle.b_0;\langle p_0,p_1.p_1\rangle.b_1\rangle\\
&= \langle p_0,p_1\rangle.d_0.\langle b_0;b_1\rangle\\
&= d_0.
\end{aligned}
$$

Now consider $\mathcal{L}(\mathcal{C}([e\mapsto t]))$. We shall do a structural induction on t and e. For $t=x$ and $e=x$ we have:

$$\mathcal{L}(\mathcal{C}([x\mapsto x]))=[y\mapsto y]$$

which, up to renaming of variables, is an equivalence. For

$$
\begin{aligned}
&\mathcal{L}(\mathcal{C}([(e_0,e_1)\mapsto x])),\quad x\in e_0\\
&= \mathcal{L}(p_0.\mathcal{C}([e_0\mapsto x]))\\
&= \mathcal{L}(p_0).\mathcal{L}(\mathcal{C}([e_0\mapsto x]))\\
&= [(e_0',e_1')\mapsto e_0'].[e_0\mapsto x]\\
&= [(e_0',e_1')\mapsto x'].
\end{aligned}
$$

It is clear that substitution is preserved. This fact can be used to replace the recursive term construction using:

$$\mathcal{L}(\mathcal{C}([e\mapsto f(e)]))=\mathcal{L}(f)=[e'\mapsto f(e')]$$

and substitution.

Obviously () is preserved. For product pairing we have:

$$
\begin{aligned}
\mathcal{L}(\mathcal{C}([e\mapsto(t_0,t_1)])) &= \mathcal{L}(\langle\mathcal{C}([e\mapsto t_0]),\mathcal{C}([e\mapsto t_1])\rangle)\\
&= \mathcal{L}(\langle\mathcal{C}([e\mapsto t_0]),\mathcal{C}([e\mapsto t_1])\rangle)\\
&= [e'\mapsto\mathcal{L}_{e'}(\mathcal{C}([e\mapsto t_0]),\mathcal{C}([e\mapsto t_1]))]\\
&= [e'\mapsto\sigma_{e:=e'}(t_0,t_1)].
\end{aligned}
$$

Finally for coproduct functions we have:

$$
\begin{aligned}
&\mathcal{L}(\mathcal{C}([e\mapsto\begin{bmatrix} b_0(e_0) & \mapsto & t_0\\ b_0(e_1) & \mapsto & t_1\end{bmatrix}(z)]))\\
&= \mathcal{L}(\langle[e\mapsto z],i\rangle.d_0.\langle[(e_0,e)\mapsto t_0];[(e_1,e)\mapsto t_1]\rangle)
\end{aligned}
$$

$$
\begin{aligned}
&= [e \mapsto (z,e)].[(w,e) \mapsto \begin{bmatrix} b_0(e_0) & \mapsto & b_0(e_0,e) \\ b_1(e_1) & \mapsto & b_1(e_1,e) \end{bmatrix}(w)].[k \mapsto \begin{bmatrix} b_0(e_0,e) & \mapsto & t_0 \\ b_1(e_1,e) & \mapsto & t_1 \end{bmatrix}(k)] \\
&= [e \mapsto (z,e)].[(w,e) \mapsto \begin{bmatrix} b_0(e_0,e) & \mapsto & t_0 \\ b_1(e_1,e) & \mapsto & t_1 \end{bmatrix}(\begin{bmatrix} b_0(e_0) & \mapsto & b_0(e_0,e) \\ b_1(e_1) & \mapsto & b_1(e_1,e) \end{bmatrix}(w))] \\
&= [e \mapsto (z,e)].[(w,e) \mapsto \begin{bmatrix} b_0(e_0) & \mapsto & t_0 \\ b_1(e_1) & \mapsto & t_1 \end{bmatrix}(w)] \\
&= [e \mapsto \begin{bmatrix} b_0(e_0) & \mapsto & t_0 \\ b_1(e_1) & \mapsto & t_1 \end{bmatrix}(z)].
\end{aligned}
$$

□