Lectures on [Co]Induction and [Co]Algebras

Roy L. Crole

April 21, 2006

Abstract

These notes give an introduction to the ideas of Induction and Coinduction at the level of partially ordered sets, and at a categorical level. Knowledge of partial orders and basic category theory will be assumed. We illustrate the use of Induction and Coinduction for partially ordered sets in programming language semantics.

These notes were originally written for a short course of lectures in 1998. This version is a reprint, but with a few minor corrections.

Contents

1 Induction and Coinduction

1.1 Partially Ordered Sets

We assume that readers know the definition of a *partially ordered set* or *poset* $(P, \leq)$ where P is a set and $\leq$ is a *partial order* on P. We shall informally refer to the poset P. A function $f\colon P \to P$ between posets is **monotone** just in case it preserves the order. If $S \subseteq P$ is a subset of P, then we write $\bigwedge S$ for the **meet** or **infimum** of S, defined to be the *greatest lower bound* in P for the set S. Dually, we write $\bigvee S$ for the **join** or **supremum** of S, defined to be the *least upper bound* in P for the set S. A poset P is called a **complete lattice** if the joins of all subsets S exist or (equivalently) the meets of all subsets exist. We note this equivalence as a proposition.

Proposition 1.1 A poset P has all meets just in case it has all joins.

Proof Suppose that S is any subset of P. Note that one has

$$\bigwedge S \overset{\text{def}}{=} \bigvee \{x \mid x \in P \text{ and } \forall s \in S . x \leq s\} \quad \text{and} \quad \bigvee S \overset{\text{def}}{=} \bigwedge \{x \mid x \in P \text{ and } \forall s \in S . s \leq x\}.$$

and that these definitions make sense even when S is empty (exercise: check this). □

If $f\colon P \to P$ is an endofunction on a poset P, we call $x \in P$ a **fixed point** for f if $f(x) = x$; a **pre-fixed point** of f if $f(x) \leq x$; and a **post-fixed point** of f if $x \leq f(x)$. In fact, if P is a complete lattice, and f is monotone, then we have the following theorem.

Theorem 1.2 Let $f\colon P \to P$ be a monotone function between complete lattices. Then the least pre-fixed point of f, denoted by $\mu\, f$, and the greatest post-fixed point, denoted by $\nu\, f$, both exist, and are given by the following formulas:

$$\mu\, f \overset{\text{def}}{=} \bigwedge \{\ x \in P \ \mid\ f(x) \leq x\ \} \quad \text{and} \quad \nu\, f \overset{\text{def}}{=} \bigvee \{\ x \in P \ \mid\ x \leq f(x)\ \}$$

Proof From the definition of $\mu\, f$, it follows that

$$\forall x \in P. \quad f(x) \leq x \Longrightarrow \mu\, f \leq x$$

Hence for any $x \in P$, if $f(x) \leq x$ then $f(\mu\, f) \leq f(x)$ and so $f(\mu\, f) \leq x$. Thus $f(\mu\, f) \leq \mu\, f$ follows immediately from the definition of meet. □

1.2 Inductively and Coinductively Defined Sets

If X is any set, let $\mathbb{P}(X)$ be the *powerset* of X. The powerset of X is in fact a complete lattice when partially ordered by subset inclusion, with meets given by set-theoretic intersection, and joins by union (exercise: check this).

We define a set of **rules** on X to be any subset $\mathcal{R}$ of the form

$$\mathcal{R} \subseteq \mathbb{P}(X) \times X$$

Given such a set of rules, we define the **name** of $\mathcal{R}$ to be the function $\Phi_{\mathcal{R}}\colon \mathbb{P}(X) \to \mathbb{P}(X)$ given by setting

$$\Phi_{\mathcal{R}}(S) \overset{\text{def}}{=} \{\ x \in X \ \mid\ \exists (S', x) \in \mathcal{R} \text{ and } S' \subseteq S\ \}$$

One can check that $\Phi_{\mathcal{R}}$ is a monotone endofunction on the complete lattice $\mathbb{P}(X)$. In view of Theorem 1.2, we can make the following definitions. Given a set X, and a set of rules $\mathcal{R}$ on X, the **subset of X inductively defined** by $\mathcal{R}$ is $\mu\, \Phi_{\mathcal{R}}$, and the **subset of X coinductively defined** by $\mathcal{R}$ is $\nu\, \Phi_{\mathcal{R}}$.

We say that a subset $S \subseteq X$ is **closed under the set of rules** $\mathcal{R}$ if it is a pre-fixed point of $\Phi_{\mathcal{R}}$. Spelling this out, S is closed if

$$\{\, x \in X \mid \exists (S', x) \in \mathcal{R} \text{ and } S' \subseteq S \,\} \subseteq S$$

Note that S is closed just in case for each rule $(H, c) \in \mathcal{R}$,

$$H \subseteq S \Longrightarrow c \in S \qquad (*)$$

We sometimes say that S is **closed under the rule** $R = (H, c)$ if $*$ holds for R. For each element $h \in H$, the assumption that $h \in S$ is called an **inductive hypothesis**.

We say that a subset $S \subseteq X$ is **dense** under the set of rules $\mathcal{R}$ if it is a post-fixed point of $\Phi_{\mathcal{R}}$. Spelling this out, S is dense if

$$S \subseteq \{\, x \in X \mid \exists (S', x) \in \mathcal{R} \text{ and } S' \subseteq S \,\}$$

Bearing in mind Theorem 1.2, we see that the subset of X inductively defined by $\mathcal{R}$ always exists and is the least subset which is closed under $\mathcal{R}$; and that the subset of X coinductively defined by $\mathcal{R}$ always exists and is the greatest subset which is dense under $\mathcal{R}$.

It is often the case that sets of rules are **finitary**, meaning that for each rule $(H, c) \in \mathcal{R}$ the set H is finite. Note that H might be $\varnothing$, in which case we say that R is a **base** rule. If H is non-empty we say R is a **deductive** rule. In the case that H is non-empty we might write $H = \{\, h_1, \ldots, h_k \,\}$ where $1 \leq k$. We can write down a base rule $R = (\varnothing, c)$ using the following notation

Base

$$\frac{}{c}\, R$$

and a deductive rule $R = (H, c) = (\{\, h_1, \ldots, h_k \,\}, c)$ as

Deductive

$$\frac{h_1 \quad h_2 \quad \ldots \quad h_k}{c}\, R$$

Note that the order of the statements $h_1 \quad h_2 \quad \ldots \quad h_k$ appearing above the line is irrelevant: the h_i are elements of the *set* H.

We give two principles, whose truth can be established from the definitions that we have so far given. We leave the proofs as an exercise.

Principle of Induction

Suppose that $I \subseteq X$ is inductively defined by a set of rules $\mathcal{R}$, and that $S \subseteq I$. Then in order to verify that $S = I$ it is enough to show that S is closed under the rules.

Principle of Coinduction

Suppose that $C \subseteq X$ is coinductively defined by a set of rules $\mathcal{R}$. Then in order to verify that $x \in C$ it is enough to find a set S which is dense under the rules and for which $x \in S$.

Given a finitary set of rules $\mathcal{R}$, a **deduction** for an element $x \in X$ is a finitely branching tree with root x which has the property that for each node $c \in X$, if H is the (possibly empty) finite set of children of c, then (H, c) must be a rule in $\mathcal{R}$. In fact we have the following theorem.

Theorem 1.3 Let $I \subseteq X$ be inductively defined by a finitary set of rules $\mathcal{R}$. Then

$$I = \{\ x \mid \text{there exists a deduction of } x\ \},$$

that is for any element $x \in X$, we have

$$\boxed{x \in I \text{ if and only if there exists a deduction of } x.}$$

Proof Write $S \overset{\text{def}}{=} \{\ x \mid \text{there exists a deduction of } x\ \}$. We show that $S \subseteq I$ as follows: we prove by Mathematical Induction on n that for all x,

for all $n \geq 0$, for all deductions l of height $\leq n$, if l is a deduction of x then $x \in I$.

Check this! Thus if $x \in S$ there must be a deduction of x which has height n for some $n \geq 0$, so that $x \in I$. Hence $S \subseteq I$. One can check that S is closed under $\mathcal{R}$ (do it!) and conclude that $I = S$ from the Principle of Induction. $\square$

2 A Programming Language

2.1 Syntax

We shall define a simple programming language called $\mathbb{UL}$. First, we define its syntax. Let *Var* be a fixed, countably infinite, set of **variables**, for which we assume there is a specified enumeration (list). Thus

$$\mathit{Var} \overset{\text{def}}{=} \{\, v_0, v_1, v_2, \dots \}.$$

We often denote variables by the letters x, y, z, u, v etc, but may on occasion use other letters. Let *Cst* be a set of **constants** where

$$\mathit{Cst} \overset{\text{def}}{=} \{\ \underline{c} \mid c \in \mathbb{Z} \cup \mathbb{B}\ \}$$

and $\mathbb{Z}$ is the set of integers and $\mathbb{B} \overset{\text{def}}{=} \{\ T, F\ \}$ is the set of Booleans. Let *Opr* be a set of **operators** given by

$$\mathit{Opr} \overset{\text{def}}{=} \{ =, \leq, \geq, <, >, +, -, * \}.$$

We shall let the symbol c range over elements of $\mathbb{Z} \cup \mathbb{B}$. Note that we write $\underline{c}$ to indicate that the constant c is "held in memory". We shall require that $\underline{c} = \underline{c'}$ if and only if $c = c'$. Given (for example) $\underline{2}$ and $\underline{3}$ we cannot add these "numbers" until our programming language instructs that the contents of the memory locations be added—thus $\underline{2} + \underline{3} \neq \underline{5}$. However, when $\underline{2}$ is added to $\underline{3}$ by $\mathbb{UL}$, the result is $\underline{5}$, and we *shall* write

$$\underline{2 + 3} = \underline{5}.$$

We now define the set $\mathcal{T}$ of **terms**. *For the time being, you can think of a term informally as a program, but we warned that they are not quite the same thing.* A term is in fact a finite tree. We

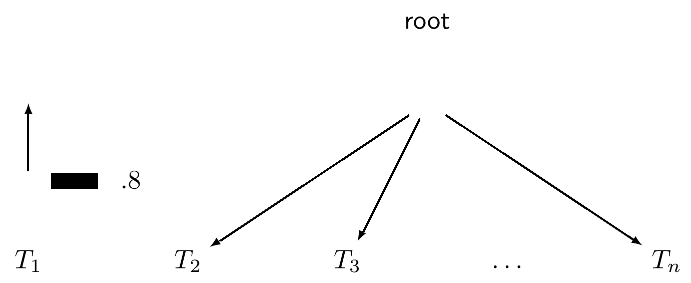
$$\frac{}{x}\,[x \in \Gamma]\ \text{VAR} \qquad \frac{}{\underline{c}}\,[c \in \mathbb{Z} \cup \mathbb{B}]\ \text{CONST} \qquad \frac{M \quad N}{M\ op\ N}\,[op \in \mathit{Opr}]\ \text{OP} \qquad \frac{M \quad N \quad L}{\mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ L}\ \text{COND}$$

$$\frac{M}{\lambda x.M}\ \text{ABS} \qquad \frac{M \quad N}{M\,N}\ \text{AP} \qquad \frac{M}{\mathsf{rec}\,x.M}\ \text{REC}$$

$$\frac{}{\mathsf{nil}}\ \text{NIL} \qquad \frac{M}{\mathsf{hd}(M)}\ \text{HD} \qquad \frac{M}{\mathsf{tl}(M)}\ \text{TL} \qquad \frac{M \quad N}{M : N}\ \text{CONS} \qquad \frac{M}{\mathsf{elist}(M)}\ \text{ELIST}$$

Figure 1: Rules for Inductively Generating the Set $\mathcal{T}$

shall adopt the following notation for finite trees: If T_1, T_2, T_3 and so on to T_n is a (finite) sequence of finite trees, then we shall write $\mathsf{root}(T_1, T_2, T_3, \ldots, T_n)$ for the finite tree which has the form



and whose root is denoted by the symbol root. We refer to root as a **constructor**. The set of **program constructors** is given by

$$\{\, x, \underline{c}, op, \mathsf{cond}, \lambda, \mathsf{ap}, \mathsf{rec}, \mathsf{nil}, \mathsf{hd}, \mathsf{tl}, \mathsf{cons}, \mathsf{elist} \mid x \in \mathit{Var}, c \in \mathbb{Z} \cup \mathbb{B}, op \in \mathit{Opr} \,\}$$

We shall adopt the following notational abbreviations, where M, N and L are finite trees, and x is a finite tree with no branches.

- We write $M\ op\ N$ for $op(M, N)$;
- $\mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ L$ for $\mathsf{cond}(M, N, L)$;
- $\lambda x.M$ for $\lambda(x, M)$;
- $M\,N$ for $\mathsf{ap}(M, N)$;
- $\mathsf{rec}\,x.M$ for $\mathsf{rec}(x, M)$; and
- $M : N$ for $\mathsf{cons}(M, N)$.

With this, the set of **terms**, $\mathcal{T}$, is inductively defined by the rules in Figure 1.

The intended meanings of most of the terms are just what you would expect from regular programming, except for $\mathsf{rec}\,x.M$. In order to explain its meaning, if P and P' are two programs, we shall write $P \rightsquigarrow P'$ to mean that P "computes in one step" to P' (this notation will be defined properly on page 17). We shall also write $M[N/v]$ to mean "M where v is replaced by N". For example,

$$(\underline{2} + \underline{5}) + \underline{1} \rightsquigarrow \underline{7} + \underline{1} \rightsquigarrow \underline{8} \qquad \text{and} \qquad (x + y)[\underline{4}/y] = x + \underline{4}.$$

The term $\lambda x.M$ is code for the program which is a function whose effect is to map x to M. More carefully, if $F \overset{\text{def}}{=} \lambda x.M$, then $F\,a \rightsquigarrow M[a/x]$. Thus $\lambda x.x + \underline{2}$ is a program whose intended meaning is the function which "adds 2", and we can write (for example) $(\lambda x.x + \underline{2})\,\underline{4} \rightsquigarrow \underline{4} + \underline{2}$.

Now, $\mathsf{rec}\,x.M$ is a recursive program on x, which is specified by the code in M. In fact, $\mathsf{rec}\,x.M$ denotes a solution to the equation $x = M$. We illustrate by example. Write $R \stackrel{\text{def}}{=} \mathsf{rec}\,x.M$. The program R "computes in one step" to $M[R/x]$. Thus if we take $M \stackrel{\text{def}}{=} \underline{0} : x$, then

$$R \rightsquigarrow (\underline{0} : x)[R/x] \equiv \underline{0} : R \rightsquigarrow \underline{0} : (\underline{0} : R) \rightsquigarrow \dots$$

and so R is a *program which recursively evaluates to an infinite list of zeros.* We call each step in the computation of R an **unfolding**.

Remark 2.1 We shall adopt a few conventions to make terms more readable:

• In general, we shall write our "formal" syntax in an informal manner, using brackets "(" and ")" to disambiguate where appropriate. For example, the term $\mathsf{ap}(\lambda(x, M), N)$ (which is unambiguous) will not be written $\lambda x.M\,N$ according to the abbreviation in Remark 2.1 (which is ambiguous) but will be written $(\lambda x.M)\,N$.

• We also *drop* brackets on other occasions. For example, we take the λ-term $\lambda x.M$ to mean $\lambda x.(M)$. Thus we can write $\lambda x.\lambda y.y + \underline{2}$ instead of the more clumsy $\lambda x.(\lambda y.(y + \underline{2}))$. A similar convention applies to $\mathsf{rec}\,x.M$. We call M the **body** of $\lambda x.M$ and $\mathsf{rec}\,x.M$.

• $M_1M_2M_3 \dots M_n$ is shorthand for $(\dots((M_1M_2)M_3)\dots M_n)$. We say that the application constructor (ap) **associates** to the left. For example, $M_1M_2M_3$ is short for $(M_1\,M_2)\,M_3$ (which is in turn a shorthand notation for the tree denoted by $\mathsf{ap}(\mathsf{ap}(M_1, M_2), M_3)$.

• We shall write $M : N : L$ for $M : (N : L)$ and say that the cons constructor (cons) associates to the right.

• The *op* constructors associate to the left. Thus the term $\underline{3}$ *op* $\underline{10}$ *op* $\underline{5}$ is shorthand for $(\underline{3}$ *op* $\underline{10})$ *op* $\underline{5}$.

• We take $\mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ L$ to mean $\mathsf{if}\ (M)\ \mathsf{then}\ (N)\ \mathsf{else}\ (L)$.

There are many terms which do not represent "common sense" programs. By common sense, we mean "well typed" programs. For example, $\mathsf{hd}(\underline{2} + \underline{3})$, $\underline{T} - \underline{3}$ and $\underline{4}(*\underline{T} : \underline{F})$ are all terms.

2.2 Free and Bound Variables

We shall use the symbol $\equiv$ to mean **actual identity**. Thus, for example $2 + 2 \equiv 2 + 2$, but $2 + 2 \not\equiv 4$.

We shall use the notion of a subterm of a term. A **subterm** S of a term M is simply any subtree of the finite tree denoted by the term M. This will be indicated by $S \lhd M$; we omit a formal definition. One can show that all subterms are themselves terms. We say that a variable x **occurs** in a term M if $x \lhd M$. There may be many occurrences. We say that a term M lies in the **scope** of λy or $\mathsf{rec}\,y$ in a term of the form $\lambda y.M$ or $\mathsf{rec}\,y.M$ respectively.

Example 2.2 $u + \underline{2}$ is the scope of λu in $\lambda x.(\lambda u.u + \underline{2})\,z$. Example subterms are

$$z \lhd \lambda x.(\lambda u.u + \underline{2})\,z \text{ and } \lambda u.u + \underline{2} \lhd \lambda x.(\lambda u.u + \underline{2})\,z.$$

If $N \stackrel{\text{def}}{=} \lambda x.xxy\underline{x}zx$ then the underlined x is the *fourth* occurrence of x in N. x occurs in N five times.

The intended meaning of $\lambda x.x + \underline{2}$ is the function which adds 2 to its argument. What about $\lambda y.y + \underline{2}$? Well, it too should be a function which adds 2. The name of the variable used to form such a term is not relevant to the intended meaning of the term—the variables x and y are said to be *bound*. However, the terms $x + \underline{2}$ and $y + \underline{2}$ are certainly different—the value of each term is respectively 2 added to x and 2 added to y, so the values will only be the same if $x = y$. Here, the variables x and y are said to be *free*. Let us give the full definitions:

One reason for defining the notion of a subterm is so that we can give a formal definition of *free* and *bound* variables. Suppose that x is a variable which does occur in a term M—of course x may occur more than once, possibly many times. Each *occurrence* of x (in M) is either free or bound. We say that an occurrence of x is **bound** in M if the occurrence of x in M is in a subterm of the form $\lambda x.N$ or $\mathsf{rec}\,x.N$—this means that whenever $\lambda\,x$ or $\mathsf{rec}\,x$ appear in a term, only those occurrences of x which appear in the *scopes* of $\lambda\,x$ or $\mathsf{rec}\,x$ are bound (as well as the occurrence of x immediately after the λ or rec !!). If there is an occurrence of x in such N then we say that occurrence of x has been **captured** by (the scope of) λx or $\mathsf{rec}\,x$ to mean that the occurrence of x is bound by the respective λx or $\mathsf{rec}\,x$. An occurrence of x in M is **free** iff the occurrence of x is not bound. Before reading on, take a look at Examples 2.3.

We shall write $var(M)$ for the set of all variables which occur in M, that is

$$var(M) \stackrel{\text{def}}{=} \{\ x \ \mid\ x \in \mathit{Var} \text{ and } x \lhd M\ \}.$$

We can give a recursive definition of the set $var(M)$ which is obvious and omitted (cf the definition of $fvar(M)$ which follows). We write $fvar(M)$ for the set of variables which have free occurrences in M. We can define this recursively by the following (obvious!) clauses:

• $fvar(x) \stackrel{\text{def}}{=} \{\,x\,\}$;

• $fvar(\underline{c}) \stackrel{\text{def}}{=} \varnothing$;

• $fvar(M\ op\ N) \stackrel{\text{def}}{=} fvar(M) \cup fvar(N)$;

• $fvar(\mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ L) \stackrel{\text{def}}{=} fvar(M) \cup fvar(N) \cup fvar(L)$;

• $fvar(\lambda x.M) \stackrel{\text{def}}{=} fvar(M) \setminus \{\,x\,\}$; occurrences of x in M are captured by the scope of λx, and hence are not free;

• $fvar(M\,N) \stackrel{\text{def}}{=} fvar(M) \cup fvar(N)$;

• $fvar(\mathsf{rec}\,x.M) \stackrel{\text{def}}{=} fvar(M) \setminus \{\,x\,\}$; occurrences of x in M are captured by the scope of $\mathsf{rec}\ x$, and hence are not free;

• $fvar(\mathsf{nil}) \stackrel{\text{def}}{=} \varnothing$;

• $fvar(\mathsf{hd}(M)) \stackrel{\text{def}}{=} fvar(M)$;

• $fvar(\mathsf{tl}(M)) \stackrel{\text{def}}{=} fvar(M)$;

• $fvar(M : N) \stackrel{\text{def}}{=} fvar(M) \cup fvar(N)$; and

• $fvar(\mathsf{elist}(M)) \stackrel{\text{def}}{=} fvar(M)$

We leave the (easy) recursive definition of the set $bvar(M)$ of the set of variables with bound occurrences in M to the reader.

Examples 2.3 Warning: Note that a variable may occur both free and bound in a term. Here are two examples:

(1)

$$\mathsf{if}\quad \underset{\substack{\uparrow\\ \text{free}}}{(x=\underline{2})}\ \ \mathsf{then}\ \ (\lambda\ \ \underset{\substack{\uparrow\\ \text{bound}}}{y}.\quad \underset{\substack{\uparrow\\ \text{bound}}}{y})\quad \mathsf{else}\ \ (\mathsf{rec}\ \ \underset{\substack{\uparrow\\ \text{bound}}}{z}.\quad \underset{\substack{\uparrow\\ \text{bound}}}{z}\quad \underset{\substack{\uparrow\\ \text{free}}}{x}\quad \underset{\substack{\uparrow\\ \text{free}}}{y})$$

Here, the set of free variables is $\{\,x, y\,\}$ and the set of bound variables is $\{\,y, z\,\}$. We could say that the second occurrence of z in the conditional has been captured by $\mathsf{rec}\,z$.

(2)

$$(\mathsf{rec}\ \underset{\substack{\uparrow\\ \text{bound}}}{x}.\ \underset{\substack{\uparrow\\ \text{bound}}}{x}\ )(\lambda\ \underset{\substack{\uparrow\\ \text{bound}}}{y}.\ \underset{\substack{\uparrow\\ \text{bound}}}{y}\ \underset{\substack{\uparrow\\ \text{free}}}{x}\ )$$

Here, the set of free variables is $\{ x \}$ and the set of bound variables is $\{ x, y \}$.

2.3 Substitution of Terms

Suppose that M and N are terms. If one thinks of M as a functional program, and the free occurrences of a variable x in M as places at which new code could be executed, we might consider replacing the variable x by N. Such a replacement is called a *substitution*. We substitute a term N for free occurrences of x in M simply by replacing each free x with N; this will produce a new term which will be denoted $M[N/x]$. For example, $(\mathsf{if}\ x\ \mathsf{then}\ \underline{4}\ \mathsf{else}\ \underline{5})[\ \underline{1} = \underline{2}\ /x]$ denotes the term $\mathsf{if}\ \underline{1} = \underline{2}\ \mathsf{then}\ \underline{4}\ \mathsf{else}\ \underline{5}$.

But things are not entirely straightforward! Suppose that $f \stackrel{\text{def}}{=} \lambda x.L$. Given any term N, the intended meaning of $f\,N$ is $L[N/x]$. Thus if L is y, then $f\,N = y[N/x] = y$. So if $M \stackrel{\text{def}}{=} \lambda x.y$, the intended meaning of M is "the function with constant value y". Now, y occurs freely in M, and x is a term, so we can try substituting the term x for the free occurrence of y, giving a new term denoted by $M[x/y]$. Now, $M[x/y]$ ought to be "the function with constant value x". But in fact $M[x/y]$ is clearly the term $\lambda x.x$, which is the identity function! The problem arises because when the variable x is substituted for the *free* variable y in $\lambda x.y$, x becomes captured by the scope of the abstraction λx.

Note that the terms $\lambda x.y$ and $\lambda z.y$ can be regarded as "the same" in the sense that the intended meaning of *each* term is the "function with constant value" y. When we attempted to substitute x for the free y in $\lambda x.y$, we noted that x would become bound. But if the intended *meaning* of $\lambda z.y$ is the same as $\lambda x.y$, what about substituting x for y in $\lambda z.y$ to get $\lambda z.x$? The latter term is indeed what we were after—the function with constant value x. Informally we say that we *re-name* the bound variable x in $\lambda x.y$ as a new variable z so that when x is substituted for y it does not become bound.

Examples 2.4 Informal examples are

$$(\lambda x.x + y)[\underline{2}/y] = \lambda x.x + \underline{2} \qquad \text{and} \qquad (\lambda x.x + y)[x/y] = \lambda u.u + x.$$

In the second example, the substituted x will appear in the scope of λx, so we rename (to u) the bound x's to avoid capture.

We have now introduced a minor problem, which we shall deal with below. In the previous example, should $(\lambda x.x + y)[x/y]$ be $\lambda u.u + x$ or $\lambda z.z + x$ or ...? We can make a unique choice by appealing to the fixed enumeration (list) of the variables in *Var* (recall page 4). This is made clear in the following definition.

We now give a formal definition of **substitution** of terms. Given terms M and N, and a variable x, we shall define a new term denoted by $M[N/x]$, which is the term M with free occurrences of x replaced by N, *by recursion on the finite tree structure of M*:

- $x[N/x] \stackrel{\text{def}}{=} N$ (if $M \equiv x$);
- $y[N/x] \stackrel{\text{def}}{=} y$ where $x \neq y$ (if $M \equiv y$);
- $\underline{c}[N/x] \stackrel{\text{def}}{=} \underline{c}$ (if $M \equiv \underline{c}$);

- $(L\ op\ L')[N/x] \overset{\text{def}}{=} L[N/x]\ op\ L'[N/x]$ (if $M \equiv L\ op\ L'$ for some L and L');
- $(\mathsf{if}\ L\ \mathsf{then}\ L'\ \mathsf{else}\ L'')[N/x] \overset{\text{def}}{=} \mathsf{if}\ L[N/x]\ \mathsf{then}\ L'[N/x]\ \mathsf{else}\ L''[N/x]$ (if $M \equiv \ldots$ etc etc);
- $(L\,L')[N/x] \overset{\text{def}}{=} L[N/x]\,L'[N/x]$;
- $(\lambda x.L)[N/x] \overset{\text{def}}{=} \lambda x.L$; and
- $(\lambda y.L)[N/x] \overset{\text{def}}{=} \lambda y.L[N/x]$ if $x \neq y$ and $x \notin \mathit{fvar}(L)$ or $y \notin \mathit{fvar}(N)$;
- $(\lambda y.L)[N/x] \overset{\text{def}}{=} \lambda z.L[z/y][N/x]$ if $x \neq y$ and $x \in \mathit{fvar}(L)$ and $y \in \mathit{fvar}(N)$, where z is chosen as the first variable in (the fixed enumeration of) *Var* for which $z \notin \mathit{var}(N) \cup \mathit{var}(L)$. So occurrences of y in $\lambda y.L$ will be renamed to the variable z to ensure that occurrences of y in N will not be captured upon substitution;
- $(\mathsf{rec}\,x.L)[N/x] \overset{\text{def}}{=} \mathsf{rec}\,x.L$; and
- $(\mathsf{rec}\,y.L)[N/x] \overset{\text{def}}{=} \mathsf{rec}\,y.L[N/x]$ if $x \neq y$ and $x \notin \mathit{fvar}(L)$ or $y \notin \mathit{fvar}(N)$;
- $(\mathsf{rec}\,y.L)[N/x] \overset{\text{def}}{=} \mathsf{rec}\,z.L[z/y][N/x]$ if $x \neq y$ and $x \in \mathit{fvar}(L)$ and $y \in \mathit{fvar}(N)$, where z is chosen as the first variable in (the fixed enumeration of) *Var* for which $z \notin \mathit{var}(N) \cup \mathit{var}(L)$. So occurrences of y in $\mathsf{rec}\,y.L$ will be renamed to the variable z to ensure that occurrences of y in N will not be captured upon substitution;
- $\mathsf{nil}[N/x] \overset{\text{def}}{=} \mathsf{nil}$;
- $\mathsf{hd}(L)[N/x] \overset{\text{def}}{=} \mathsf{hd}(L[N/x])$;
- $\mathsf{tl}(L)[N/x] \overset{\text{def}}{=} \mathsf{tl}(L[N/x])$;
- $(L : L')[N/x] \overset{\text{def}}{=} L[N/x] : L'[N/x]$;
- $\mathsf{elist}(L)[N/x] \overset{\text{def}}{=} \mathsf{elist}(L[N/x])$.

Examples 2.5

(1)

$$\begin{aligned}((v_1 + \underline{2}) : (v_3 v_2))[\underline{10}/v_2] &= (v_1 + \underline{2})[\underline{10}/v_2] : (v_3 v_2)[\underline{10}/v_2] \\ &= (v_1[\underline{10}/v_2] + \underline{2}[\underline{10}/v_2]) : (v_3[\underline{10}/v_2] v_2[\underline{10}/v_2]) \\ &= (v_1 + \underline{2}) : (v_3\ \underline{10})\end{aligned}$$

Note that in the first example, we wrote down each of the recursive steps. It's not too difficult (!) to write the result of the substitution straight down, or at least miss out some of the steps, as in the next example:

(2)

$$\begin{aligned}(\mathsf{rec}\,v_3.v_6\,v_3 : \mathsf{nil})[v_3\,v_1/v_6] &=_* \mathsf{rec}\,v_2.(v_6\,v_3 : \mathsf{nil})[v_2/v_3][v_3\,v_1/v_6] \\ &= \mathsf{rec}\,v_2.(v_6\,v_2 : \mathsf{nil})[v_3\,v_1/v_6] \\ &= \mathsf{rec}\,v_2.((v_3\,v_1)\,v_2 : \mathsf{nil})\end{aligned}$$

where at * note that $v_6 \in \mathit{fvar}(v_6\,v_3 : \mathsf{nil})$ and $v_3 \in \mathit{fvar}(v_3\,v_1)$, so we have to rename v_3 to avoid capture. We rename v_3 to be the first variable in *Var* not appearing in

$$\mathit{fvar}(v_6\,v_3 : \mathsf{nil}) \cup \mathit{fvar}(v_3\,v_1) = \{\, v_1, v_3, v_6 \,\}$$

which is v_2.

We have claimed that for any two terms M and N, and variable x, there is a term $M[N/x]$ which is specified by the previous definition. We should, of course, prove that $M[N/x]$ *is* a term. While this can be done, the proof is a subtle induction, and we omit it.

2.4 α-Equivalence

We have seen that the two terms $\lambda x.y$ and $\lambda z.y$ have the same intended meaning, namely that they both represent the function with constant value y. You will also note the the definition of substitution is a little unwieldy due to the clauses which involve a renaming of bound variables. Whenever a renaming takes place we have to choose "the first variable in the enumeration $v_0, v_1, \ldots$ which does not appear in the terms involved in the substitution". Now, if we were to implement substitution, we would have to be explicit about *what* we renamed variables to, when avoiding capture. But, in fact, as regards the overall meaning of $\mathbb{UL}$ terms, it does not really matter what we rename variables to, *provided we choose a fresh variable.* Thus the computational meaning of both $\lambda u.u + x$ and $\lambda z.z + x$ in Remark 2.3 is the same—they are both functions which add x.

For these reasons, we shall *regard terms which differ only in the names of their bound variables as equivalent.* We have to give a proper definition of what it means for two terms to be "equal" if they "differ only in the names of their bound variables".

To do this we shall define an equivalence relation, denoted by $\sim_\alpha$, on the set $\mathcal{T}$ of terms. So formally $\sim_\alpha$ is a set (of pairs), and in particular a subset of $\mathcal{T} \times \mathcal{T}$. We define it inductively by the rules in Figure 2. We comment about the notation. Instead of writing the rules R in the form

$$\frac{(M_1, M_1') \quad (M_2, M_2') \quad (M_3, M_3') \quad \ldots \quad (M_k, M_k')}{(M, M')}\ R$$

(where $R \subseteq \mathbb{P}(\mathcal{T} \times \mathcal{T}) \times (\mathcal{T} \times \mathcal{T})$) we write them in the more suggestive form

$$\frac{M_1 \sim_\alpha M_1' \quad M_2 \sim_\alpha M_2' \quad M_3 \sim_\alpha M_3' \quad \ldots \quad M_k \sim_\alpha M_k'}{M \sim_\alpha M'}\ R$$

The formal definition of two terms differing only in their bound variables is of course that the terms are α-equivalent. We wish to consider a term as being "equal" to all other α-equivalent terms, and we can do this by considering α-equivalence classes.

We define the set $\mathcal{E}$ of **expressions** to be the set of α-equivalence classes of terms:

$$\mathcal{E} \quad \stackrel{\text{def}}{=} \quad \mathcal{T}/\sim_\alpha = \{\, \overline{M} \mid M \in \mathcal{T} \,\}.$$

Example 2.6 We have

$$\overline{\lambda u.u + x} = \{\, M \mid \lambda u.u + x \sim_\alpha M \,\} = \{\, \lambda u.u + x, \lambda z.z + x, \ldots \,\} = \overline{\lambda z.z + x} = \ldots$$

Check this!! Rule (1) gives us (for example) $\lambda u.u + x \sim_\alpha \lambda z.z + x$ taking M to be $u + x$, v to be u and v' to be z.

The formal definition of α-equivalence amounts to saying that two terms are α-equivalent if one can be transformed to the other by a sequence of changes of bound variables. The definition in Figure 2 makes this intuitive idea watertight. Instead of writing $\overline{M}$ for an expression, we adopt the convention that we simply write M, that is we shall denote an α-equivalence class by a representative. We shall "treat" expressions as though they are terms, but whenever we give a definition involving expressions, we must not forget that expressions are in fact α-equivalence classes and that we have to check that the definition is well-defined.

If $M, N \in \mathcal{T}$ and $M \sim_\alpha N$, then of course $\overline{M} = \overline{N}$. For example $\lambda x.x \sim_\alpha \lambda z.z$ and so $\overline{\lambda x.x} = \overline{\lambda z.z}$. Following the above convention, we can simply write $\lambda x.x = \lambda z.z$. And magically, the convention also allows us to write

$$(\lambda x.x + y)[x/y] = \lambda u.u + x = \lambda z.z + x.$$

Finally, what about substitution of expressions? What expression is $M[N/x]$ when M and N are expressions, rather than terms? In practice, we can just "forget" that M and N are α-equivalence

$$\frac{}{M \sim_\alpha M}\,\text{REF} \qquad \frac{M \sim_\alpha M'}{M' \sim_\alpha M}\,\text{SYM} \qquad \frac{M \sim_\alpha M' \quad M' \sim_\alpha M''}{M \sim_\alpha M''}\,\text{TRAN}$$

$$\frac{M \sim_\alpha M' \quad N \sim_\alpha N'}{M \; op \; N \sim_\alpha M' \; op \; N'} \qquad \frac{M \sim_\alpha M' \quad N \sim_\alpha N' \quad L \sim_\alpha L'}{\mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ L \sim_\alpha \mathsf{if}\ M'\ \mathsf{then}\ N'\ \mathsf{else}\ L'}$$

$$\frac{}{\lambda v.M \sim_\alpha \lambda v'.M[v'/v]}\ (1) \qquad \frac{M \sim_\alpha M'}{\lambda x.M \sim_\alpha \lambda x.M'} \qquad \frac{M \sim_\alpha M' \quad N \sim_\alpha N'}{M\,N \sim_\alpha M'\,N'}$$

$$\frac{}{\mathsf{rec}\,v.M \sim_\alpha \mathsf{rec}\,v'.M[v'/v]}\ (2) \qquad \frac{M \sim_\alpha M'}{\mathsf{rec}\,x.M \sim_\alpha \mathsf{rec}\,x.M'}$$

$$\frac{M \sim_\alpha M'}{\mathsf{hd}(M) \sim_\alpha \mathsf{hd}(M')} \qquad \frac{M \sim_\alpha M'}{\mathsf{tl}(M) \sim_\alpha \mathsf{tl}(M')} \qquad \frac{M \sim_\alpha M' \quad N \sim_\alpha N'}{M : N \sim_\alpha M' : N'}$$

$$\frac{M \sim_\alpha M'}{\mathsf{elist}(M) \sim_\alpha \mathsf{elist}(M')}$$

In (1) *and* (2), v' *may be any variable different from* v *and which does not occur in* M

Figure 2: Rules for Inductively Generating the α-Equivalence Relation $M \sim_\alpha M'$

classes, and take the "expression" $M[N/x]$ to be the equivalence class of the term $M'[N'/x]$ where M' and N' are any representatives (ie $M \sim_\alpha M'$ and $N \sim_\alpha N'$). Thus, it turns out that because we are dealing with α-equivalence classes, when we rename variables to avoid capture, we can choose any new name we like. And this avoids the hassle of a specific choice of variable, as we had on page 8. It is actually quite tricky to prove that this all works out, and we omit to do this. We look at one example:

Example 2.7 Dealing with α-equivalence classes we have

$$(\lambda x.(x+y))\,[\mathsf{rec}\,z.xz/y] = \lambda u.(u + \mathsf{rec}\,z.xz).$$

But (for example)

$$\lambda x.x + y = \lambda w.w + y \quad \text{and} \quad \mathsf{rec}\,z.xz = \mathsf{rec}\,v.xv$$

and so we ought to have

$$\lambda u.u + \mathsf{rec}\,z.xz = \lambda w.w + \mathsf{rec}\,v.xv. \qquad (*)$$

It is "easy to see" that $(*)$ holds via a renaming of bound variables. Here is how we could give a formal deduction:

$$\cfrac{\cfrac{\cfrac{\cfrac{}{u \sim_\alpha u}\,\text{REF} \qquad \cfrac{}{\mathsf{rec}\,z.xz \sim_\alpha \mathsf{rec}\,v.xv}\,(2)}{u + \mathsf{rec}\,z.xz \sim_\alpha u + \mathsf{rec}\,v.xv}}{\lambda u.(u + \mathsf{rec}\,z.xz) \sim_\alpha \lambda u.(u + \mathsf{rec}\,v.xv)} \qquad \cfrac{}{\lambda u.(u + \mathsf{rec}\,v.xv) \sim_\alpha \lambda w.(w + \mathsf{rec}\,v.xv)}\,(1)}{\lambda u.(u + \mathsf{rec}\,z.xz) \sim_\alpha \lambda w.(w + \mathsf{rec}\,v.xv)}\,\text{TRAN}$$

2.5 Terms with Contexts

We will shortly use the concept of expressions to give an abstract definition of a (functional) program. Before we do this, we need one further technical device. It is very convenient, when dealing with expressions, to keep track of the free variables appearing in an expression. We will do this by defining judgements of the form $\Gamma \vdash M$ where Γ is a set of variables, M is a term, and the free variables of M all appear in Γ. An example is

$$\underbrace{\{\, x, y, z \,\}}_{\text{set of variables}} \quad \vdash \quad \underbrace{x + y}_{\text{term}}$$

For clarity, we usually drop the curly braces from the set of variables, writing this example as $x, y, z \vdash x + y$.

We shall define a relation $\vdash$ between finite sets of variables and terms. More formally, $\vdash$ is a relation between $\mathcal{P}_{fin}(\mathit{Var})$ and $\mathcal{T}$. We often write Γ for a typical element of $\mathcal{P}_{fin}(\mathit{Var})$, and it will be convenient to write Γ, x for $\Gamma \cup \{\, x \,\}$ and Γ, Γ' for $\Gamma \cup \Gamma'$. We define $\vdash$ inductively by the rules in Figure 3, where instead of writing (Γ, M), we use the more readable $\Gamma \vdash M$.

We define the set of **terms whose free variables appear in a context** Γ, denoted by $\mathcal{T}(\Gamma)$, and the set of **expressions whose free variables appear in** Γ, denoted by $\mathcal{E}(\Gamma)$, by

$$\mathcal{T}(\Gamma) \stackrel{\text{def}}{=} \{\, M \mid \Gamma \vdash M \,\} \qquad \text{and} \qquad \mathcal{E}(\Gamma) \stackrel{\text{def}}{=} \mathcal{T}(\Gamma)/\sim_\alpha$$

where you should note that the equivalence relation $\sim_\alpha$ on $\mathcal{T}$ induces an equivalence relation (also written $\sim_\alpha$) on $\mathcal{T}(\Gamma)$. Note that we write $\vdash M$ when Γ is empty, that is, $\varnothing \vdash M$. If $\vdash M$ we say that M is **closed**.

$$\frac{}{\Gamma \vdash x}\,[x \in \Gamma] \qquad \frac{}{\Gamma \vdash \underline{c}}\,[c \in \mathbb{Z} \cup \mathbb{B}] \qquad \frac{\Gamma \vdash M \quad \Gamma \vdash N}{\Gamma \vdash M\ op\ N}\,[op \in Opr]$$

$$\frac{\Gamma \vdash M \quad \Gamma \vdash N \quad \Gamma \vdash L}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ L} \qquad \frac{\Gamma, x \vdash M}{\Gamma \vdash \lambda x.M} \qquad \frac{\Gamma \vdash M \quad \Gamma \vdash N}{\Gamma \vdash M\, N} \qquad \frac{\Gamma, x \vdash M}{\Gamma \vdash \mathsf{rec}\, x.M}$$

$$\frac{}{\Gamma \vdash \mathsf{nil}} \qquad \frac{\Gamma \vdash M}{\Gamma \vdash \mathsf{hd}(M)} \qquad \frac{\Gamma \vdash M}{\Gamma \vdash \mathsf{tl}(M)} \qquad \frac{\Gamma \vdash M \quad \Gamma \vdash N}{\Gamma \vdash M : N} \qquad \frac{\Gamma \vdash M}{\Gamma \vdash \mathsf{elist}(M)}$$

Figure 3: Rules for Inductively Generating the Relation $\Gamma \vdash M$

Proposition 2.8 If $\Gamma \vdash M$, then $\mathit{fvar}(M) \subseteq \Gamma$.

Proof We use the Principle of Induction for the inductively defined set $\vdash$. We prove that the set

$$S \stackrel{\text{def}}{=} \{\, (\Gamma, M) \in \vdash \;\mid\; \mathit{fvar}(M) \subseteq \Gamma \,\}$$

is closed under each of the rules in Figure 3. It follows that $S = \vdash$, and the result follows. Let us give one example.

(*Closure under the rule*):

$$\frac{\Gamma, x \vdash M}{\Gamma \vdash \lambda x.M}$$

The inductive hypothesis amounts to $\mathit{fvar}(M) \subseteq \Gamma \cup \{\, x \,\}$. We have to prove that $\mathit{fvar}(\lambda x.M) \subseteq \Gamma$. We calculate

$$\begin{array}{rcll} \mathit{fvar}(\lambda x.M) & \stackrel{\text{def}}{=} & \mathit{fvar}(M) \setminus \{\, x \,\} & \\ & \subseteq & (\Gamma \cup \{\, x \,\}) \setminus \{\, x \,\} & \text{using the inductive hypothesis} \\ & = & \Gamma. & \end{array}$$

The rest of the proof is left as an exercise. □

2.6 Programs, Values and Evaluation

A *program* will be a closed expression. A program is closed so that it is a "self contained" expression, into which no further data need be input. A program is required to be an expression, so that programs which differ only in their bound variables are equal.

We shall soon give rules which tell us how a program can be "evaluated" or "computed" to a value. A value will be a program that is as "fully evaluated as possible" according to a particular kind of evaluation or computation strategy. For example, $(\lambda x.x + \underline{2})\underline{3}$ is a program which computes to the value $\underline{5}$, and we can write this as

$$(\lambda x.x + \underline{2})\underline{3} \quad \Downarrow \quad \underline{5}$$

reading $\Downarrow$ as "evaluates to". Note that $\underline{5}$ is a value, but it is also a very trivial program—it *is* an expression with no free variables!! Functions, that is programs of the form $\lambda x.M$, will also be regarded as values. The idea is that the body M of the function will not be evaluated until an argument has been passed to the function. Finally, lists of the form $P : Q$, where P and Q are programs, are also values. This may seem odd at first sight—think of some examples. As we

$$\frac{}{V \Downarrow V}\ \text{VAL} \qquad \frac{P \Downarrow \underline{m} \quad Q \Downarrow \underline{n}}{P\ op\ Q \Downarrow \underline{m\ op\ n}}\ \text{OP}$$

$$\frac{P \Downarrow \underline{T} \quad Q \Downarrow V}{\mathsf{if}\ P\ \mathsf{then}\ Q\ \mathsf{else}\ Q' \Downarrow V}\ \text{COND}_1 \qquad \frac{P \Downarrow \underline{F} \quad Q' \Downarrow V}{\mathsf{if}\ P\ \mathsf{then}\ Q\ \mathsf{else}\ Q' \Downarrow V}\ \text{COND}_2$$

$$\frac{P \Downarrow \lambda x.M \quad M[Q/x] \Downarrow V}{P\,Q \Downarrow V}\ \text{AP} \qquad \frac{M[\mathsf{rec}\,x.M/x] \Downarrow V}{\mathsf{rec}\,x.M \Downarrow V}\ \text{REC}$$

$$\frac{P \Downarrow P' : Q \quad P' \Downarrow V}{\mathsf{hd}(P) \Downarrow V}\ \text{HD} \qquad \frac{P \Downarrow P' : Q \quad Q \Downarrow V}{\mathsf{tl}(P) \Downarrow V}\ \text{TL}$$

$$\frac{P \Downarrow \mathsf{nil}}{\mathsf{elist}(P) \Downarrow \underline{T}}\ \text{ELIST}_1 \qquad \frac{P \Downarrow P' : Q}{\mathsf{elist}(P) \Downarrow \underline{F}}\ \text{ELIST}_2$$

Figure 4: Rules for Inductively Generating the Evaluation Relation $P \Downarrow V$ of $\mathbb{UL}$

shall soon see in more detail, $\mathbb{UL}$ is a *lazy* language, meaning that "program fragments are only evaluated if they are used". Thus the head or tail of a list will only be evaluated *if* "extracted" by a hd or tl function. So $(\underline{3} + \underline{4}) : \mathsf{nil}$ is a value; it does not evaluate to $\underline{7} : \mathsf{nil}$.

We shall define a **program** P to be a closed expression. A **value** V is any program given by the grammar

$$V ::= \underline{c} \mid \lambda x.M \mid \mathsf{nil} \mid M : M$$

where M ranges over expressions. The set of programs is denoted by $\mathcal{P}$, and values by $\mathcal{V}$.

We can define a binary relation between programs and values, with relationships denoted by $P \Downarrow V$, as an inductively defined set given by the rules in Figure 4. We shall also use the notations $(P, V) \in \Downarrow$ and $P \Downarrow V$ interchangeably when we feel that no confusion can arise.

We refer to the definition of $\Downarrow$ as a **structured operational semantics** for $\mathbb{UL}$. The word *semantics* refers to the fact that the rules defining $\Downarrow$ give a "meaning" to programs P. This "meaning" arises by showing how programs compute to values, which is specified in a "computational" or "operational" manner—hence the adjective *operational.* Finally, *structured* refers to the finite tree structure of P: whenever we have $P \Downarrow V$, we can see which rules might have been used to deduce $P \Downarrow V$ by looking at the outermost constructor of P.

You should note that the rules in Figure 4 yield a **lazy** operational semantics for functions and lists. In general, *lazy* means that "subterms of programs are only computed if absolutely necessary". For a general program of the form

$$P \equiv C(M_1, M_2, \ldots, M_n)$$

where C is a program constructor, we only evaluate those M_i to values necessary for the evaluation of P. We illustrate by example:

Consider $P\,Q$. Let us write this as the finite tree $\mathsf{ap}(P, Q)$ (as originally defined) where ap is the program constructor. In order to evaluate $\mathsf{ap}(P, Q)$, we *must* evaluate P to a function, say $\lambda x.M$. But now we are lazy!! We do not bother to evaluate Q before passing it to $\lambda x.M$. Thus the next step of the computation is to evaluate $M[Q/x]$. If now $M[Q/x]$ evaluates to a value, say V, then so too does $\mathsf{ap}(P, Q)$. Now look at rule AP, and see how it captures our intended operational semantics!

The same idea applies to lists. Consider $H : T$, that is $\mathsf{cons}(H, T)$ where cons is the program constructor. We regard this as a fully evaluated program—very lazy!! We only compute the subterms H or T if they are extracted by taking a head or tail. Thus to evaluate $\mathsf{hd}(P)$, we first evaluate the list P to a value of the form $\mathsf{cons}(P', Q)$, but then we only bother (lazy) to evaluate P' to a value, say V. Thus $\mathsf{hd}(P)$ evaluates to V, and there is no need to evaluate Q. Now look at rules HD and TL.

We have seen that if P is a program, V is a value, and $P \Downarrow V$, the latter means that "the program P evaluates to the value V". But what would happen if there was another value V' for which $P \Downarrow V'$? This would mean that one program could compute to two different values. In fact, thankfully, this cannot happen! The relation $\Downarrow$ is *deterministic*, meaning that a program can only compute to one value.

Theorem 2.9 The relation $\Downarrow$ is **deterministic**: For any program P and values V and V', if $P \Downarrow V$ and $P \Downarrow V'$, then[1] $V = V'$.

Proof We shall show that the set

$$S \stackrel{\text{def}}{=} \{\ (P, V) \in \Downarrow\ \mid\ \forall V'\ (P \Downarrow V' \Longrightarrow V = V')\ \}$$

is closed under the rules in Figure 4.

(*Closure under* COND$_2$): The inductive hypotheses are

H1 for all V', if $P \Downarrow V'$ then $\underline{F} = V'$, and

H2 for all V', if $Q' \Downarrow V'$ then $V = V'$.

We have to prove that

C for any V', if $\mathsf{if}\ P\ \mathsf{then}\ Q\ \mathsf{else}\ Q' \Downarrow V'$ then $V = V'$.

Pick an arbitrary V' for which $\mathsf{if}\ P\ \mathsf{then}\ Q\ \mathsf{else}\ Q' \Downarrow V'$—(*). Now (*) could be deduced from an application of either COND$_1$ or COND$_2$. If it were the former, then $P \Downarrow \underline{T}$. So using **H1**, we would have $\underline{F} = \underline{T}$, a contradiction. Hence (*) must be a conclusion to an instance of COND$_2$, say

$$\frac{P \Downarrow \underline{F} \quad Q' \Downarrow V'}{\mathsf{if}\ P\ \mathsf{then}\ Q\ \mathsf{else}\ Q' \Downarrow V'}$$

Hence $Q' \Downarrow V'$ for some program Q'. But using **H2**, it follows that $V = V'$ as required.

(*Closure under* REC): The inductive hypothesis is

H for all V', if $M[\mathsf{rec}\, x.M/x] \Downarrow V'$ then $V = V'$.

We have to prove that

C for any V', $\mathsf{rec}\, x.M \Downarrow V'$ implies $V = V'$.

Pick an arbitrary V' for which $\mathsf{rec}\, x.M \Downarrow V'$. This last relation *must* arise through the rule REC, and so $M[\mathsf{rec}\, x.M/x] \Downarrow V'$. That $V = V'$ then follows from **H**.

Exercise: finish the proof. □

[1]Do not forget that = here denotes equality of the two α-equivalence classes represented by V and V'.

Examples 2.10 Prove that $(\lambda z.z\underline{*}2)\,\underline{3} \Downarrow \underline{6}$, that is $((\lambda z.z\underline{*}2)\,\underline{3}, \underline{6}) \in \Downarrow$. To do this, we produce a deduction tree (see Theorem 1.3). First note that the program being evaluated is an application. So it *must* arise by the rule AP, hence we need to show that $\lambda z.z\underline{*}2 \Downarrow \lambda x.M$ for some x and M, and that $M[\underline{3}/x] \Downarrow \underline{6}$. The first of these is easy, being an instance of VAL with $x \equiv z$ and $M \equiv z\underline{*}2$. The second, namely $\underline{3}\underline{*}2 \Downarrow \underline{6}$, is also easy following from OP. Putting this altogether we get

$$\dfrac{\dfrac{}{\lambda z.z\underline{*}2 \Downarrow \lambda z.z\underline{*}2}\,\text{VAL} \qquad \dfrac{\dfrac{}{\underline{3} \Downarrow \underline{3}}\,\text{VAL} \qquad \dfrac{}{\underline{2} \Downarrow \underline{2}}\,\text{VAL}}{(z\underline{*}2)[\underline{3}/z] \equiv \underline{3}\underline{*}2 \Downarrow \underline{6}}\,\text{OP}}{(\lambda z.z\underline{*}2)\,\underline{3} \Downarrow \underline{6}}\,\text{AP}$$

Prove that $\mathsf{hd}((\lambda x.x+\underline{2})\,\underline{3} : \mathsf{nil}) \Downarrow \underline{5}$. To do this, we derive a deduction tree:

$$\dfrac{T \qquad \dfrac{\dfrac{}{\lambda x.x+\underline{2} \Downarrow \lambda x.x+\underline{2}}\,\text{VAL} \qquad \dfrac{\dfrac{}{\underline{3} \Downarrow \underline{3}}\,\text{VAL} \qquad \dfrac{}{\underline{2} \Downarrow \underline{2}}\,\text{VAL}}{(x+\underline{2})[\underline{3}/x] \equiv \underline{3}+\underline{2} \Downarrow \underline{5}}\,\text{OP}}{(\lambda x.x+\underline{2})\,\underline{3} \Downarrow \underline{5}}\,\text{AP}}{\mathsf{hd}((\lambda x.x+\underline{2})\,\underline{3} : \mathsf{nil}) \Downarrow \underline{5}}\,\text{HD}$$

where T is the tree

$$\dfrac{}{(\lambda x.x+\underline{2})\,\underline{3} : \mathsf{nil} \Downarrow (\lambda x.x+\underline{2})\,\underline{3} : \mathsf{nil}}\,\text{VAL}$$

2.7 Transitions: One Step Computations

Not all programs compute to values! An example is $\mathsf{rec}\,x.x$. However, when a program P does compute to a value, how can we calculate that value? The rules for deriving the relation $P \Downarrow V$ do not lend themselves to direct calculation. To overcome this problem, we shall define a new relation between programs, written $P \rightsquigarrow Q$. The intuitive idea is that if P and Q are related by $\rightsquigarrow$, then P "computes in one step" to Q. For example,

$$(\lambda x.x+\underline{2})\underline{3} \rightsquigarrow \underline{3}+\underline{2} \qquad \text{and} \qquad \underline{3}+\underline{2} \rightsquigarrow \underline{5}.$$

Now we give the full definition:

We shall define a **transition relation** between programs, that is a binary relation on $\mathcal{P}$. It takes the form $P \rightsquigarrow Q$ and is inductively defined by the rules in Figure 5. If $P \rightsquigarrow Q$ we say that P **computes in one step** to Q.

Note again that $\rightsquigarrow$ is **lazy**. In order to compute $P\,Q$, we have to (deterministically!) apply rule AP_1 until P reduces to a value $\lambda x.M$ and then apply rule AP_2 which substitutes the function argument Q straight into M without first evaluating Q to a value:

$$P\,Q \rightsquigarrow_{\text{AP}_1} P'\,Q \rightsquigarrow \;\ldots\; \rightsquigarrow_{\text{AP}_1} (\lambda x.M)\,Q \rightsquigarrow_{\text{AP}_2} M[Q/x]$$

Some programs cannot compute in one step to another program. An example is $\underline{2} + \underline{T}$ for which there is no program Q with

$$(\underline{2} + \underline{T}) \rightsquigarrow Q.$$

One can see this by inspecting the rules for generating $\rightsquigarrow$. We say that such programs are **terminal**. However, when a program P is not terminal, there is a *unique* program Q for which $P \rightsquigarrow Q$. Thus, the relation $\rightsquigarrow$ is, like $\Downarrow$, *deterministic*.

Theorem 2.11 The relation $\rightsquigarrow$ is **deterministic**: If P, Q and Q' are any programs, then if $P \rightsquigarrow Q$ and $P \rightsquigarrow Q'$ we have $Q = Q'$.

$$\frac{P \rightsquigarrow P'}{P\ op\ Q \rightsquigarrow P'\ op\ Q}\ \text{OP}_1 \qquad \frac{Q \rightsquigarrow Q'}{\underline{n}\ op\ Q \rightsquigarrow \underline{n}\ op\ Q'}\ \text{OP}_2 \qquad \frac{}{\underline{n}\ op\ \underline{m} \rightsquigarrow \underline{n\ op\ m}}\ \text{OP}_3$$

$$\frac{P \rightsquigarrow P'}{\mathsf{if}\ P\ \mathsf{then}\ Q\ \mathsf{else}\ Q' \rightsquigarrow \mathsf{if}\ P'\ \mathsf{then}\ Q\ \mathsf{else}\ Q'}\ \text{COND}$$

$$\frac{}{\mathsf{if}\ \underline{T}\ \mathsf{then}\ P\ \mathsf{else}\ Q \rightsquigarrow P}\ \text{COND}_1 \qquad \frac{}{\mathsf{if}\ \underline{F}\ \mathsf{then}\ P\ \mathsf{else}\ Q \rightsquigarrow Q}\ \text{COND}_2$$

$$\frac{P \rightsquigarrow P'}{P\,Q \rightsquigarrow P'\,Q}\ \text{AP}_1 \qquad \frac{}{(\lambda x.M)\,Q \rightsquigarrow M[Q/x]}\ \text{AP}_2 \qquad \frac{}{\mathsf{rec}\,x.M \rightsquigarrow M[\mathsf{rec}\,x.M/x]}\ \text{REC}$$

$$\frac{P \rightsquigarrow P'}{\mathsf{hd}(P) \rightsquigarrow \mathsf{hd}(P')}\ \text{HD}_1 \qquad \frac{}{\mathsf{hd}(P : Q) \rightsquigarrow P}\ \text{HD}_2$$

$$\frac{P \rightsquigarrow P'}{\mathsf{tl}(P) \rightsquigarrow \mathsf{tl}(P')}\ \text{TL}_1 \qquad \frac{}{\mathsf{tl}(P : Q) \rightsquigarrow Q}\ \text{TL}_2$$

$$\frac{P \rightsquigarrow P'}{\mathsf{elist}(P) \rightsquigarrow \mathsf{elist}(P')}\ \text{ELIST}_1 \qquad \frac{}{\mathsf{elist}(\mathsf{nil}) \rightsquigarrow \underline{T}}\ \text{ELIST}_2 \qquad \frac{}{\mathsf{elist}(P : Q) \rightsquigarrow \underline{F}}\ \text{ELIST}_3$$

Figure 5: Rules for Inductively Generating the Transition Relation $P \rightsquigarrow Q$ in $\mathbb{UL}$

Proof Let

$$S \stackrel{\text{def}}{=} \{\, (P,Q) \in \rightsquigarrow \;\mid\; \forall Q'(P \rightsquigarrow Q' \Longrightarrow Q = Q') \,\}$$

So by the Principle of Induction, we show closure of S for the rules in Figure 5. This routine exercise is left to the reader. □

2.8 Relating Evaluation and Transition Relations

We need to find a connection between $\Downarrow$ and $\rightsquigarrow$. Consider

$$\begin{array}{rcl} \mathsf{hd}((\lambda x.x + \underline{2})\,\underline{3} : \mathsf{nil}) & \rightsquigarrow & (\lambda x.x + \underline{2})\,\underline{3} \\ & \rightsquigarrow & \underline{3} + \underline{2} \\ & \rightsquigarrow & \underline{5} \end{array}$$

and (see Examples 2.10)

$$\mathsf{hd}((\lambda x.x + \underline{2})\,\underline{3} : \mathsf{nil}) \;\Downarrow\; \underline{5}$$

It appears that a program will compute to a value if there is a sequence of one-step transitions from the program to the value. This suggests that $\Downarrow$ might be the transitive closure of $\rightsquigarrow$. In fact $\Downarrow$ is (more-or-less) the reflexive transitive closure $\rightsquigarrow^*$—reflexivity arises from the fact that for any value V, we have $V \Downarrow V$.

Theorem 2.12 For every program P and value V in $\mathbb{UL}$, we have

$$P \Downarrow V \Longleftrightarrow P \rightsquigarrow^* V.$$

Proof

($\Rightarrow$) We use the Principle of Induction for $\Downarrow$ to prove $\{\, P \Downarrow V \mid P \rightsquigarrow^* V \,\}$ is all of $\Downarrow$. The details are an exercise.

($\Leftarrow$) We can show that

$$X \stackrel{\text{def}}{=} \{\, (P,Q) \mid \forall V.\, (Q \Downarrow V \Longrightarrow P \Downarrow V) \,\}$$

is closed under the rules in Figure 5 which define $\rightsquigarrow$.

(*Closure under* $\textsc{hd}_1$): Suppose $(P, P') \in X$—(*). We have to prove $(\mathsf{hd}(P), \mathsf{hd}(P')) \in X$, that is

$$\forall V.\, (\mathsf{hd}(P') \Downarrow V \Longrightarrow \mathsf{hd}(P) \Downarrow V) \qquad (\dagger)$$

Pick an arbitrary value V and suppose that $\mathsf{hd}(P') \Downarrow V$. Then from rule $\textsc{hd}$ of Figure 4 we know that there must be programs Q and Q' for which $P' \Downarrow Q : Q'$—(**) and $Q \Downarrow V$. Now, $Q : Q'$ is a value in $\mathbb{UL}$, so using supposition $(*)$, and $(**)$, we have $P \Downarrow Q : Q'$. Hence

$$\frac{P \Downarrow Q : Q' \quad Q \Downarrow V}{\mathsf{hd}(P) \Downarrow V}$$

As V was arbitrary, (†) holds.

We can show closure under the other rules similarly, and the details are omitted. Hence by leastness for $\rightsquigarrow$, we have $\rightsquigarrow \subseteq X$, that is for any P and Q,

$$P \rightsquigarrow Q \Longrightarrow \forall V\, (Q \Downarrow V \Longrightarrow P \Downarrow V).$$

Note that X is in fact a reflexive and transitive relation between programs—exercise: check this!! But, by definition, $\rightsquigarrow^*$ is the smallest such relation which contains $\rightsquigarrow$. Hence $\rightsquigarrow^* \subseteq X$, and so for any P and Q,

$$P \rightsquigarrow^* Q \Longrightarrow \forall V\, (Q \Downarrow V \Longrightarrow P \Downarrow V).$$

If we take $Q \stackrel{\text{def}}{=} V$, and note that $V \Downarrow V$, then we have

$$P \rightsquigarrow^* V \Longrightarrow P \Downarrow V$$

as required. □

Suppose that P is a program. We say that P has a **finite transition sequence** if there is a transition sequence of the form

$$P \equiv P_0 \rightsquigarrow P_1 \rightsquigarrow P_2 \rightsquigarrow \ldots \rightsquigarrow P_m$$

for some natural number $m \geq 0$ for which P_m is terminal. Note that appealing to Theorem 2.11, any transition sequence must be *unique*, and hence that m must be unique too. We shall also call such P **convergent**. In such a case we call

$$P_0 \rightsquigarrow P_1 \rightsquigarrow P_2 \rightsquigarrow \ldots \rightsquigarrow P_m$$

the **full transition sequence** of P. The case that m is 0 is simply saying P itself is terminal.

If such finite m does not exist, we say that P has an **infinite transition sequence**, which must be of the form

$$P \rightsquigarrow P_1 \rightsquigarrow P_2 \rightsquigarrow \ldots \rightsquigarrow P_n \rightsquigarrow \ldots$$

where each P_n is non-terminal. We say that P is **divergent** or **loops**, and indicate this by $P \rightsquigarrow^{\omega}$. Note that by Theorem 2.11, all programs are either convergent or divergent, but cannot be both. It is easy to see from the definition of $\rightsquigarrow$ that a value V is terminal, and hence convergent.

Let $M \overset{\text{def}}{=} \mathsf{if}\ x = \underline{1}\ \mathsf{then}\ \underline{1}\ \mathsf{else}\ x + f\,(x - \underline{1})$, $F \overset{\text{def}}{=} \lambda x.M$ and $R \overset{\text{def}}{=} \mathsf{rec}\, f.F$. We give the full transition sequence of $R\,\underline{2}$ in $\mathbb{UL}$.

$$\begin{array}{lll}
R\,\underline{2} & \rightsquigarrow & F[R/f]\,\underline{2} \ \equiv\ (\lambda x.M[R/f])\,\underline{2} \\
& \rightsquigarrow & M[R/f][\underline{2}/x] \ \equiv\ \mathsf{if}\ \underline{2} = \underline{1}\ \mathsf{then}\ \underline{1}\ \mathsf{else}\ \underline{2} + R\,(\underline{2} - \underline{1}) \\
& \rightsquigarrow & \mathsf{if}\ \underline{F}\ \mathsf{then}\ \underline{1}\ \mathsf{else}\ \underline{2} + R\,(\underline{2} - \underline{1}) \\
& \rightsquigarrow & \underline{2} + R\,(\underline{2} - \underline{1}) \\
& \rightsquigarrow_{(1)} & \underline{2} + (\lambda x.M[R/f])\,(\underline{2} - \underline{1}) \\
& \rightsquigarrow & \underline{2} + M[R/f][\underline{2} - \underline{1}/x] \\
& \equiv & \underline{2} + (\mathsf{if}\ (\underline{2} - \underline{1}) = \underline{1}\ \mathsf{then}\ \underline{1}\ \mathsf{else}\ (\underline{2} - \underline{1}) + R\,((\underline{2} - \underline{1}) - \underline{1})) \\
& \rightsquigarrow & \underline{2} + (\mathsf{if}\ \underline{1} = \underline{1}\ \mathsf{then}\ \underline{1}\ \mathsf{else}\ (\underline{2} - \underline{1}) + R\,((\underline{2} - \underline{1}) - \underline{1})) \\
& \rightsquigarrow & \underline{2} + (\mathsf{if}\ \underline{T}\ \mathsf{then}\ \underline{1}\ \mathsf{else}\ (\underline{2} - \underline{1}) + R\,((\underline{2} - \underline{1}) - \underline{1})) \\
& \rightsquigarrow & \underline{2} + \underline{1} \\
& \rightsquigarrow & \underline{3}
\end{array}$$

It is not too difficult to verify each of the transition steps. For example, the step $\rightsquigarrow_{(1)}$ is valid because:

$$\dfrac{\dfrac{\overline{R \rightsquigarrow F[R/f]}\ \text{REC}}{R\,(\underline{2} - \underline{1}) \rightsquigarrow (\lambda x.M[R/f])\,(\underline{2} - \underline{1})}\ \text{AP}_1}{\underline{2} + R\,(\underline{2} - \underline{1}) \rightsquigarrow_{(1)} \underline{2} + (\lambda x.M[R/f])\,(\underline{2} - \underline{1})}\ \text{OP}_2$$

where of course $F[R/f] = \lambda x.M[R/f]$.

2.9 Coinductively Characterizing Divergence

In fact we can give a coinductive definition of divergence based solely on the evaluation relation. Let us coinductively define a subset of $\mathcal{P}$, denoted by $\Uparrow$, by the rules in Figure 6.

Then we have the following theorem.

Theorem 2.13 A program P diverges just in case $P \Uparrow$.

$$\frac{P \Uparrow}{P\ op\ Q \Uparrow} \qquad \frac{Q \Uparrow}{P\ op\ Q \Uparrow}\ P \Downarrow \underline{n}$$

$$\frac{P \Uparrow}{\mathsf{if}\ P\ \mathsf{then}\ Q\ \mathsf{else}\ Q' \Uparrow} \qquad \frac{Q \Uparrow}{\mathsf{if}\ P\ \mathsf{then}\ Q\ \mathsf{else}\ Q' \Uparrow}\ P \Downarrow \underline{T} \qquad \frac{Q' \Uparrow}{\mathsf{if}\ P\ \mathsf{then}\ Q\ \mathsf{else}\ Q' \Uparrow}\ P \Downarrow \underline{F}$$

$$\frac{P \Uparrow}{P\,Q \Uparrow} \qquad \frac{M[Q/x] \Uparrow}{P\,Q \Uparrow}\ P \Downarrow \lambda x.M \qquad \frac{M[\mathsf{rec}\,x.M/x] \Uparrow}{\mathsf{rec}\,x.M \Uparrow}$$

$$\frac{P \Uparrow}{\mathsf{hd}(P) \Uparrow} \qquad \frac{P \Uparrow}{\mathsf{tl}(P) \Uparrow} \qquad \frac{P \Uparrow}{\mathsf{elist}(P) \Uparrow}$$

Figure 6: Rules for Coinductively Generating the Divergence Predicate $P \Uparrow$ in $\mathbb{UL}$

Proof Let us write $\mathcal{D}$ for the set of divergent programs, that is $\mathcal{D} \stackrel{\text{def}}{=} \{ D \in \mathcal{P} \mid D \rightsquigarrow^{\omega} \}$. Then we have to prove that $\mathcal{D} = \nu\,\Phi_{\Uparrow}$, where $\Phi_{\Uparrow}\colon \mathbb{P}(\mathcal{P}) \to \mathbb{P}(\mathcal{P})$ is the name of the set of rules $\mathcal{R}_{\Uparrow}$ in Figure 6.

We shall verify that $\mathcal{D}$ is in fact a greatest $\mathcal{R}_{\Uparrow}$ dense set, and is thus $\nu\,\Phi_{\Uparrow}$.

First we check that it is $\mathcal{R}_{\Uparrow}$ dense, that is, $\mathcal{D} \subseteq \Phi_{\Uparrow}(\mathcal{D})$. To do this, we first write down a description of the set $\Phi_{\Uparrow}(\mathcal{D})$ using the definition of $\Phi_{\Uparrow}$. Let X range over $\Phi_{\Uparrow}(\mathcal{D})$. Then each such X takes the form

$$\begin{array}{rcll}
X & \equiv & D\ op\ Q & \\
 & \mid & C_{\underline{n}}\ op\ D & \\
 & \mid & \mathsf{if}\ C_{\underline{T}}\ \mathsf{then}\ D\ \mathsf{else}\ Q' & \\
 & \mid & \mathsf{if}\ C_{\underline{F}}\ \mathsf{then}\ Q\ \mathsf{else}\ D & \\
 & \mid & D\,Q & \\
 & \mid & C_{\lambda x.M}\,Q & \text{provided that } M[Q/x] \in \mathcal{D} \\
 & \mid & \mathsf{rec}\,x.M & \text{provided that } M[\mathsf{rec}\,x.M/x] \in \mathcal{D} \\
 & \mid & \mathsf{hd}(D) & \\
 & \mid & \mathsf{hd}(C_{D:Q}) & \\
 & \mid & \mathsf{tl}(D) & \\
 & \mid & \mathsf{tl}(C_{P:D}) & \\
 & \mid & \mathsf{elist}(D) &
\end{array}$$

where $M \in \mathcal{E}$, $P, Q, Q' \in \mathcal{P}$, $D \in \mathcal{D}$ and C_V denotes a convergent term whose full transition sequence terminates at a value V.

We shall show that if $X \in \mathcal{D}$ then $X \in \Phi_{\Uparrow}(\mathcal{D})$ by a structural case analysis of X. First note that X can't be either x or $\underline{c}$.

$\boxed{\text{Case } X \text{ is } P\ op\ Q}$

If $P \in \mathcal{D}$, then $X \in \Phi_{\Uparrow}(\mathcal{D})$. If this is not the case, then P converges to a terminal, say T, and of course $X \rightsquigarrow^* T\ op\ Q$. Hence as X diverges, T must be of the form $\underline{m}$ so that P is of the form $C_{\underline{m}}$, and thus $X \rightsquigarrow^* \underline{m}\ op\ Q \rightsquigarrow^{\omega}$ as transition sequences are unique. It follows that $Q \rightsquigarrow^{\omega}$ as required.

$\boxed{\text{Case } X \text{ is } P\,Q}$

If $P \in \mathcal{D}$, then $X \in \Phi_{\Uparrow}(\mathcal{D})$. If this is not the case, then P converges to a terminal, say T, and of course $X \rightsquigarrow^* T\,Q$. Hence as X diverges, T must be of the form $\lambda x.M$ so that P is of the form $C_{\lambda x.M}$. Now note that

$$X \rightsquigarrow^* (\lambda x.M)\,Q \rightsquigarrow M[Q/x] \rightsquigarrow^{\omega}$$

because X diverges, and the transition sequence is unique. Thus $M[Q/x] \in \mathcal{D}$.

$\boxed{\text{Case } X \text{ is } \mathsf{rec}\, x.M}$

This is trivial, because $\mathsf{rec}\, x.M \rightsquigarrow M[\mathsf{rec}\, x.M/x]$, and as the former diverges, so too must the latter.

We leave the remaining cases as an exercise. So $\mathcal{D}$ is $\mathcal{R}_{\Uparrow}$ dense.

Let $\mathcal{S} \subseteq \Phi_{\Uparrow}(\mathcal{S})$. We shall now show that $\mathcal{D}$ is greatest among all such $\mathcal{R}_{\Uparrow}$ dense sets. To do this, we shall first prove that

$$\forall S \in \mathcal{E}. \quad S \in \mathcal{S} \Longrightarrow \exists S' \in \mathcal{S}. S \rightsquigarrow S' \qquad (\dagger)$$

We use the Principle of Induction for the set of rules in Figure 1 to show that

$$\{\, S \in \mathcal{T} \mid S \in \mathcal{S} \Longrightarrow \exists S' \in \mathcal{S}. S \rightsquigarrow S' \,\}$$

is all of $\mathcal{T}$. Then † follows.

(*Closure under* VAR): Of course $x \notin \mathcal{S} \subseteq \mathcal{P}$.

(*Closure under* OP): Denote the rule by $\frac{P \quad Q}{P op Q}$. Suppose that $P\ op\ Q \in \mathcal{S}$. Note that because $\mathcal{S} \subseteq \Phi_{\Uparrow}(\mathcal{S})$, then either $P \in \mathcal{S}$, or $P \equiv C_{\underline{m}}$ and $Q \in \mathcal{S}$. In the former case, by the induction hypothesis $P \rightsquigarrow P'$ for some P', and hence $P\ op\ Q \rightsquigarrow P'\ op\ Q$. In the latter case, if $P \rightsquigarrow P'$ for some P' we are similarly done, and otherwise P must be $\underline{m}$, in which case $P\ op\ Q \rightsquigarrow P\ op\ Q'$ by the induction hypothesis for some Q'.

(*Closure under* AP): Denote the rule by $\frac{P \quad Q}{P\,Q}$. If $P \in \mathcal{S}$, then $P\,Q \rightsquigarrow P'\,Q$ follows by the induction hypothesis. If not, then $P \equiv C_{\lambda x.M}$ and $Q \in \mathcal{S}$. If $P \rightsquigarrow P'$, then we are done. Otherwise, $P \equiv \lambda x.M$, whence $P\,Q \rightsquigarrow M[Q/x]$.

We omit the remaining cases (exercise!).

It is quite easy to conclude from †, Theorem 2.11, and the definition of $\mathcal{D}$, that $\mathcal{S} \subseteq \mathcal{D}$. Thus, as $\nu\,\Phi_{\Uparrow} \subseteq \Phi_{\Uparrow}(\nu\,\Phi_{\Uparrow})$ we have $\nu\,\Phi_{\Uparrow} \subseteq \mathcal{D}$, and as $\mathcal{D}$ is indeed $\mathcal{R}_{\Uparrow}$ dense, equality follows.

$\square$

Examples 2.14 It follows from the Principle of Coinduction and Theorem 2.13, that to prove $P \rightsquigarrow^{\omega}$, it is enough to find a set $\mathcal{S} \subseteq \mathcal{P}$ which is dense under the rules in Figure 6, such that $P \in \mathcal{S}$.

1. The program $\mathsf{rec}\, x.x$ diverges. Take $\mathcal{S} \overset{\text{def}}{=} \{\, \mathsf{rec}\, x.x \,\}$, which is clearly dense.

2. The program $R \overset{\text{def}}{=} \mathsf{rec}\, x.x\ op\ \underline{2}$ diverges. Take $\mathcal{S} \overset{\text{def}}{=} \{R, R\ op\ \underline{2}\}$. Then $\mathcal{S}$ is dense if $R, R\ op\ \underline{2} \in \Phi_{\Uparrow}(\mathcal{S})$, which is true if (respectively) $R\ op\ \underline{2}, R \in \mathcal{S}$.

3. Fix $k \in \mathbb{Z}$. The program $R \overset{\text{def}}{=} (\mathsf{rec}\, f.\underline{n} * f\ (\underline{n} - 1))\ \underline{k}$ diverges. Let $P^k_0 \overset{\text{def}}{=} \underline{k}$ and $P^k_{m+1} \overset{\text{def}}{=} P^k_m - \underline{1}$ for $m \geq 0$. Take $\mathcal{S} \overset{\text{def}}{=} \{\, R\,\underline{k}, P^k_m * (R\,P^k_{m+1}) \mid m \geq 0 \,\}$. Then

$$\begin{array}{lll} \mathcal{S} \text{ is dense} & \text{iff} & R\,\underline{k} \in \Phi_{\Uparrow}(\mathcal{S}) \text{ and } P^k_m * (R\,P^k_{m+1}) \in \Phi_{\Uparrow}(\mathcal{S}) \\ & \text{if} & P^k_0 * (R\,P^k_1) \in \mathcal{S} \text{ and } P^k_{m+1} * (R\,P^k_{m+2}) \in \mathcal{S} \end{array}$$

3 Algebras and Coalgebras

We shall assume that the reader is familiar with categories, functors, natural transformations, and simple limits and colimits.

Let $F: \mathcal{C} \to \mathcal{C}$ be an endofunctor. An **algebra** for the functor F is specified by a pair (A, α^A) where A is an object of $\mathcal{C}$ and $\alpha^A: FA \to A$ is a morphism. We define the **category of F-algebras**, denoted by $\mathcal{C}^F$, to have objects the algebras of F, and a morphism $f: (A, \alpha^A) \to (B, \alpha^B)$ is a morphism $f: A \to B$ in $\mathcal{C}$ for which the diagram

$$\begin{tikzcd}
FA \arrow[r, "Ff"] \arrow[d, "\alpha^A"'] & FB \arrow[d, "\alpha^B"] \\
A \arrow[r, "f"'] & B
\end{tikzcd}$$

commutes in $\mathcal{C}$. Dually we can define a **coalgebra** for F as a pair (A, α^A) where $\alpha^A: A \to FA$ is a morphism in $\mathcal{C}$, and the **category of F-coalgebras** $\mathcal{C}_F$ is defined similarly to the category of algebras. We also define an **initial** F-algebra (I, α^I) to be an initial object in $\mathcal{C}^F$; and a **final** F-coalgebra (C, α^C) to be a terminal object in $\mathcal{C}_F$. If (A, f) is an F-algebra, we write $\overline{f}: (I, \alpha^I) \to (A, f)$ for the unique mediating morphism, and similarly $\overline{f}: (A, f) \to (C, \alpha^C)$ if (A, f) is an F-coalgebra.

Let I be an arbitrary indexing set, and suppose that $\mathcal{C}$ has I-indexed products and coproducts. We shall write $\Pi_{i \in I}: \mathcal{C} \to \mathcal{C}$ for the functor which maps an object A to the I-fold product of copies of A; in the case that I is finite, we write Π_m for $\Pi_{i \in I}$, and A^m for $\Pi_{i \in I} A$, where m is the cardinality of I. Similarly, $\Sigma_{i \in I}$ denotes I-fold coproduct.

Given families of morphisms $(f_i: A \to B_i \mid i \in I)$ and $(g_i: C_i \to D \mid i \in I)$, then

$$\langle f_i \mid i \in I \rangle: A \to \Pi_{i \in I} B_i \quad \text{and} \quad [g_i \mid i \in I]: \Sigma_{i \in I} C_i \to D$$

denote pairing and copairing. Projections are denoted by $\pi_j: \Pi_{i \in I} B_i \to B_j$ and insertions by $ins_j: C_j \to \Sigma_{i \in I} C_i$.

Given a morphism $f: A \to B$ in a category $\mathcal{C}$ with pullbacks, the **kernel** K_f of f is given by the pullback square

$$\begin{tikzcd}
K_f \arrow[r, "\pi_1"] \arrow[d, "\pi_2"'] & A \arrow[d, "f"] \\
A \arrow[r, "f"'] & B
\end{tikzcd}$$

In the case that $f = id_A$, then $\pi_1 = \pi_2$ $(= \pi$, say) and K_{id_A} is denoted by Eq_A. Clearly π is monic, and the subobject $\langle \pi, \pi \rangle: Eq_A \to A \times A$ is called the **equality relation** on A.

We now give a few examples of initial algebras and final coalgebras.

3.1 The Functor $A + (-): \mathcal{S}et \longrightarrow \mathcal{S}et$

Let A be a set, $+$ denote coproduct. Then the functor $A + (-)$ has an initial algebra $(A \times \mathbb{N}, \alpha^{A \times \mathbb{N}})$ where $\alpha^{A \times \mathbb{N}}: A + (A \times \mathbb{N}) \to A \times \mathbb{N}$ is defined by

$$\begin{aligned}
\alpha^{A \times \mathbb{N}}(\xi) \quad \stackrel{\text{def}}{=} \quad & \text{case } \xi \text{ of} \\
& \quad ins_L(a) \mapsto (a, 0) \\
& \quad ins_R(a, n) \mapsto (a, n+1)
\end{aligned}$$

where ins_L and ins_R are the left and right coproduct insertions, and $n \geq 0$.
Then given any function $f\colon A+S \to S$, we can define $\overline{f}$ where

$$\begin{tikzcd}
A+(A\times\mathbb{N}) \arrow[r, "\alpha^{A\times\mathbb{N}}"] \arrow[d, "A+\overline{f}"'] & A\times\mathbb{N} \arrow[d, "\overline{f}"] \\
A+S \arrow[r, "f"'] & S
\end{tikzcd}$$

by setting

$$\begin{aligned}
\overline{f}(a,0) &\stackrel{\text{def}}{=} f(ins_L(a)) \\
\overline{f}(a,n+1) &\stackrel{\text{def}}{=} f(ins_R(\overline{f}(a,n)))
\end{aligned}$$

Let $(A\times\mathbb{N})^\infty \stackrel{\text{def}}{=} (A\times\mathbb{N})\cup\{\infty\}$. The functor $A+(-)$ has a final coalgebra $((A\times\mathbb{N})^\infty, \alpha^{(A\times\mathbb{N})^\infty})$ where

$$p \stackrel{\text{def}}{=} \alpha^{(A\times\mathbb{N})^\infty}\colon (A\times\mathbb{N})^\infty \to A+(A\times\mathbb{N})^\infty$$

is defined (for $n \geq 0$) by

$$\begin{aligned}
p(a,0) &\stackrel{\text{def}}{=} ins_L(a) \\
p(a,n+1) &\stackrel{\text{def}}{=} ins_R(a,n) \\
p(\infty) &\stackrel{\text{def}}{=} ins_R(\infty)
\end{aligned}$$

Note that given any function $f\colon S \to A+S$, we can write for $r \geq 1$,

$$\begin{aligned}
f^1(s) &\stackrel{\text{def}}{=} f(s) \\
f^{r+1}(s) &\stackrel{\text{def}}{=} \text{case } f^r(s) \text{ of} \\
&\qquad ins_R(s') \mapsto f(s') \\
&\qquad \text{otherwise} \mapsto \textit{undefined}
\end{aligned}$$

It is easy to see that if there exists $m \in \mathbb{N}$ and $a \in A$ for which $f^{m+1}(s) = ins_L(a)$, then m and a must be unique. Hence we can define $\overline{f}$ where

$$\begin{tikzcd}
S \arrow[r, "f"] \arrow[d, "\overline{f}"'] & A+S \arrow[d, "A+\overline{f}"] \\
(A\times\mathbb{N})^\infty \arrow[r, "p"'] & A+(A\times\mathbb{N})^\infty
\end{tikzcd}$$

by setting

$$\overline{f}(s) \stackrel{\text{def}}{=} \begin{cases} (a,m) \text{ if these exist as above} \\ \infty \text{ otherwise} \end{cases}$$

3.2 The Functor $1 + (A \times -)\colon \mathcal{S}et \to \mathcal{S}et$

For $k \geq 1$ we shall define the set A^k to be the collection of k-tuples of elements of A. If $l = (a_1, \dots, a_k) \in A^k$, then we shall regard l as a partial function $\{1, \dots, k\} \rightharpoonup A$. If $a \in A$ and $l \in A^k$, then we define $al \in A^{k+1}$ by $al(1) \stackrel{\text{def}}{=} a$ and $al(r) \stackrel{\text{def}}{=} l(r-1)$ for $r \geq 2$.

The functor $1+(A\times-)$ has an initial algebra (L, α^L), where we shall set $L \stackrel{\text{def}}{=} \{\mathsf{nil}\} \cup (\bigcup_{1 \leq k < \omega} A^k)$, and $\alpha^L\colon 1 + (A \times L) \to L$ is defined by

$$\begin{aligned} \alpha^L(ins_L(*)) &\stackrel{\text{def}}{=} \mathsf{nil} \\ \alpha^L(ins_R(a, nil)) &\stackrel{\text{def}}{=} a \\ \alpha^L(ins_R(a, l)) &\stackrel{\text{def}}{=} al \end{aligned}$$

It is an <u>exercise</u> to verify that this *does* yield an initial algebra.

The functor $1+(A\times-)$ has a final coalgebra (L, α^L), where we shall set $L \stackrel{\text{def}}{=} \{\mathsf{nil}\} \cup (\bigcup_{1 \leq k \leq \omega} A^k)$, and $\alpha^L\colon L \to 1 + (A \times L)$ is defined by

$$\begin{aligned} \alpha^L(\mathsf{nil}) &\stackrel{\text{def}}{=} ins_L(*) \\ \alpha^L(l) &\stackrel{\text{def}}{=} \text{case } l \text{ of} \\ &\qquad l \in A^1 \mapsto ins_R(l(1), \mathsf{nil}) \\ &\qquad l \in \textstyle\bigcup_{2 \leq k \leq \omega} A^k \mapsto ins_R(l(1), \lambda r.l(r+1)) \end{aligned}$$

Then given any function $f\colon S \to 1 + (A \times S)$ we can define $\overline{f}\colon S \to L$ by setting

$$\begin{aligned} \overline{f}(s) &\stackrel{\text{def}}{=} \text{case } f(s) \text{ of} \\ &\qquad ins_L(*) \mapsto \mathsf{nil} \\ &\qquad ins_R(a, s') \mapsto l \end{aligned}$$

where for $r \geq 1$ we set

$$\begin{aligned} l(1) &\stackrel{\text{def}}{=} a \\ l(r+1) &\stackrel{\text{def}}{=} \text{case } f^{r+1}(s) \text{ of} \\ &\qquad ins_R(a', s'') \mapsto a' \\ &\qquad \text{otherwise} \mapsto \textit{undefined} \end{aligned}$$

where for $r \geq 1$ we set

$$\begin{aligned} f^1(s) &\stackrel{\text{def}}{=} f(s) \\ f^{r+1}(s) &\stackrel{\text{def}}{=} \text{case } f^r(s) \text{ of} \\ &\qquad ins_R(a, s') \mapsto \ \text{case } f(s') \text{ of} \\ &\qquad\qquad\qquad ins_R(a', s'') \mapsto f(s') \\ &\qquad\qquad\qquad \text{otherwise} \mapsto \textit{undefined} \end{aligned}$$

3.3 The Functor $(-)_\bot\colon \omega\mathcal{CPO} \longrightarrow \omega\mathcal{CPO}$

There is an initial algebra $(\mathbb{N}^\infty_\bot, \sigma)$ where σ is defined (for all $n \geq 0$) by

$$\begin{array}{rcl} \sigma(\bot) & \stackrel{\text{def}}{=} & 0 \\ \sigma(n) & \stackrel{\text{def}}{=} & n+1 \\ \sigma(\infty) & \stackrel{\text{def}}{=} & \infty \end{array}$$

Then given any continuous function $f\colon D_\bot \to D$, we can define $\overline{f}$ where

$$\begin{tikzcd} \mathbb{N}^\infty_\bot \arrow[r, "\sigma"] \arrow[d, "\overline{f}_\bot"'] & \mathbb{N}^\infty \arrow[d, "\overline{f}"] \\ D_\bot \arrow[r, "f"'] & D \end{tikzcd}$$

by setting

$$\begin{array}{rcl} \overline{f}(n) & \stackrel{\text{def}}{=} & f^{n+1}(\bot) \\ \overline{f}(\infty) & \stackrel{\text{def}}{=} & \bigvee_{k<\omega} f^{k+1}(\bot) \end{array}$$

Note that σ is an isomorphism. In fact $(\mathbb{N}^\infty_\bot, p)$ where $p \stackrel{\text{def}}{=} \sigma^{-1}$ is a final coalgebra. Given any $f\colon D \to D_\bot$, we can define $\overline{f}$ where

$$\begin{tikzcd} D \arrow[r, "f"] \arrow[d, "\overline{f}"'] & D_\bot \arrow[d, "\overline{f}_\bot"] \\ \mathbb{N}^\infty \arrow[r, "p"'] & \mathbb{N}^\infty_\bot \end{tikzcd}$$

by setting

$$\overline{f}(d) \stackrel{\text{def}}{=} \bigwedge_{\mathbb{N}^\infty} \{\ r \mid f^{r+1}(d) = \bot \wedge r < \omega\ \}$$

4 Isomorphism Theorems

We illustrate algebraic and coalgebraic notions, by stating and proving the “group isomorphism theorems” in a general setting.

4.1 For Algebras

An (single sorted) **algebraic signature** Sg is specified by a family of **function symbols**, a typical symbol denoted by f, each of which has an **arity** $r = r(f) \in \mathbb{N}$. Recall that a **structure** in a category $\mathcal{C}$ with finite products is specified by giving a morphism $\alpha^A_f\colon A^r \to A$ in $\mathcal{C}$ for each function symbol f in Sg. We call α^A_f the **denotation** of f.

Let us now work with the category $\mathcal{S}et$ of **sets and functions**. Given an algebraic signature Sg, we define the functor $\mathbb{S} \stackrel{\text{def}}{=} \Sigma_{f\in Sg} \circ \Pi_{r(f)}$ as a composition of the sum and product functors. We then have the easy lemma

Lemma 4.1 Let *Sg* be a single sorted algebraic signature. Specifying a structure

$$(\alpha_f^A\colon A^r \to A \mid f \in Sg)$$

for *Sg* in $\mathcal{S}et$ is equivalent to specifying a $\mathbb{S}$-algebra (A, α^A).

Proof Given a structure $(\alpha_f^A\colon A^r \to A \mid f \in Sg)$, we define $\alpha^A\colon \mathbb{S}\,A \to A$ by setting

$$\alpha^A \overset{\text{def}}{=} [\alpha_f^A \mid f \in Sg]$$

that is, the structure map is the copairing of the denotations of the function symbols.

Conversely, given $\alpha^A\colon \mathbb{S}\,A \to A$, define $\alpha_f^A \overset{\text{def}}{=} \alpha^A \circ ins_f$, where

$$\begin{tikzcd} A^r \arrow[rr, "ins_f"] \arrow[dr, "\alpha_f^A"'] & & \Sigma_{f \in Sg} A^r \arrow[dl, "\alpha^A"] \\ & A & \end{tikzcd}$$

□

Let (A, α^A) and (B, α^B) be $\mathbb{S}$-algebras. We shall call a morphism $h\colon (A, \alpha^A) \to (B, \alpha^B)$ in $\mathcal{S}et^{\mathbb{S}}$ a **homomorphism**. Note that a function $h\colon A \to B$ is a homomorphism of the above form just in case for each $f \in Sg$

$$\alpha_f^B \circ \Pi_{r(f)} h = h \circ \alpha_f^A$$

We shall define a **congruence** R on an algebra (A, α^A), to be an equivalence relation on the set A for which there exists an algebra structure (R, α^R) such that the projection functions $\pi_1, \pi_2\colon R \to A$ give rise to homomorphisms

$$\pi_1, \pi_2\colon (R, \alpha^R) \longrightarrow (A, \alpha^A)$$

Note that it is not hard to see that if R is indeed a congruence, then we must have $\alpha^R = \langle \alpha^A \circ \mathbb{S}\,\pi_1, \alpha^A \circ \mathbb{S}\,\pi_2 \rangle$.

Given an algebra (A, α^A) with a congruence R, we shall write A/R for the set of R-equivalence classes, each such class being denoted by $[a]$ for $a \in A$. We can endow the set A/R with an algebra structure, by defining

$$\begin{tikzcd} (A/R)^{r(f)} \arrow[r, "\alpha_f^{A/R}"] & A/R \\ ([a_1], \dots, [a_r]) \arrow[r, mapsto] & [\alpha_f^A(a_1, \dots, a_r)] \end{tikzcd}$$

It follows that the quotient function is indeed a homomorphism $q\colon (A, \alpha^A) \to (A/R, \alpha^{A/R})$.

We call the algebra (S, α^S) a **subalgebra** of (A, α^A) if S is a subset of A for which the inclusion function is a homomorphism.

Let $h\colon (A, \alpha^A) \to (B, \alpha^B)$ be a homomorphism. Then $(im(h), \alpha^{im(h)})$ is a subalgebra of (B, α^B) when $\alpha_f^{im(h)}$ is defined by restricting α_f^B.

Theorem 4.2 Let $h\colon (A, \alpha^A) \to (B, \alpha^B)$ be a homomorphism, and let K_h be the kernel of h. Then K_h is a congruence on A, and there is a diagram of the form

$$\begin{tikzcd}
(A, \alpha^A) \arrow[r, "h"] \arrow[d, "q"'] & (B, \alpha^B) \\
(A/K_h, \alpha^{A/K_h}) \arrow[r, shift left, "\phi"] \arrow[r, phantom, "\cong"] & (im(h), \alpha^{im(h)}) \arrow[l, shift left, "\psi"] \arrow[u, hook, "\iota"']
\end{tikzcd}$$

for which $h = \iota \circ \phi \circ q$, where ι is the inclusion function.

Proof There is a diagram of the form

$$\begin{tikzcd}
\mathbb{S}\,K_h \arrow[rr, "\mathbb{S}\,\pi_1"] \arrow[dd, "\mathbb{S}\,\pi_2"'] \arrow[dr, dotted, "\rho"] & & \mathbb{S}\,A \arrow[d, "\alpha^A"] \\
& K_h \arrow[r, "\pi_1"] \arrow[d, "\pi_2"'] & A \arrow[d, "h"] \\
\mathbb{S}\,A \arrow[r, "\alpha^A"'] & A \arrow[r, "h"'] & B
\end{tikzcd}$$

Exercise: verify this! The map ρ exists, being the mediating map $\langle \alpha^A \circ \mathbb{S}\,\pi_1, \alpha^A \circ \mathbb{S}\,\pi_2 \rangle$ arising from the kernel pullback square. It is then immediate that K_h admits an algebra structure for which it is a congruence. Now, all we need to do is verify the existence of an isomorphism, and check that the diagram commutes. We define ϕ by setting $\phi([a]) \stackrel{\text{def}}{=} h(a)$ and ψ by $\psi(b) \stackrel{\text{def}}{=} [a]$ where $b = h(a)$ for some $a \in A$. It is easy to verify that ϕ and ψ are homomorphisms (because, respectively, h and q are), and the functions are clearly inverse to each other.

□

Theorem 4.3 Let (A, α^A) be an algebra, (S, α^S) a **subalgebra** of (A, α^A), and R a congruence on (A, α^A). Set $Q \stackrel{\text{def}}{=} R \cap S^2$, $\overline{S} \stackrel{\text{def}}{=} (\pi_1 \circ \pi_2^{-1})S$, and $Q' \stackrel{\text{def}}{=} R \cap \overline{S}^2$. Then there is a diagram of the form

$$\begin{tikzcd}
(A, \alpha^A) \arrow[rr, "q"] & & (A/R, \alpha^{A/R}) \\
(S, \alpha^S) \arrow[u, hook, "i"] \arrow[dr, "q"'] & & (im(qi), \alpha^{im(qi)}) \arrow[u, hook] \\
& (S/Q, \alpha^{S/Q}) \arrow[ur, "\cong"'] &
\end{tikzcd}$$

and moreover $im(qi) = \overline{S}/Q'$. Thus

$$\frac{S}{R \cap S^2} \cong \frac{\overline{S}}{R \cap \overline{S}^2}$$

Proof In Theorem 4.2, take the homomorphism h to be qi. This immediately yields the above diagram, on noting that Q is exactly K_{qi}. It remains to prove that $im(qi) = \overline{S}/Q'$. To see this, note that

$$S \subseteq \overline{S} = \{ \, a \in A \mid \exists s \in S.a \; R \; s \, \}$$

Hence if $[s]_R \in im(qi)$ then $[s]_R = [s]_{Q'} \in \overline{S}/Q'$ (check!); and if $[a]_{Q'} \in \overline{S}/Q'$, then there exists $s \in S$ for which $[a]_{Q'} = [s]_R \in im(qi)$ (check!). □

4.2 For Coalgebras

We define the functor $\mathbb{P}: \mathcal{S}et \to \mathcal{S}et$ on objects by mapping a set S to its powerset $\mathbb{P}\,S$, and on morphisms $f: S \to T$ by defining $\mathbb{P}\,f: \mathbb{P}\,S \to \mathbb{P}\,T$ to be the function defined by

$$\mathbb{P}\,f(X) \stackrel{\text{def}}{=} \{ \, fx \mid x \in X \, \}$$

for each subset X of S. We shall also call a morphism of the form $h: (S, \alpha^S) \longrightarrow (T, \alpha^T)$ in $\mathcal{S}et_{\mathbb{P}}$ a **homomorphism**. Note that $h: S \to T$ is a homomorphism iff $\mathbb{P}\,h \circ \alpha^S = \alpha^T \circ h$.

If (S, α^S) is a $\mathbb{P}$-coalgebra, then we shall write $s \overset{\alpha^S}{\leadsto} s'$ to mean $s' \in \alpha^S(s)$ for any $s, s' \in S$.

Lemma 4.4 $h: (S, \alpha^S) \longrightarrow (T, \alpha^T)$ is a homomorphism just in case for any $s, s' \in S$ and $t \in T$,

- $s \overset{\alpha^S}{\leadsto} s' \Longrightarrow hs \overset{\alpha^T}{\leadsto} hs'$; and
- $hs \overset{\alpha^T}{\leadsto} t \Longrightarrow \exists s'.t = hs'$ and $s \overset{\alpha^S}{\leadsto} s'$.

Proof An easy exercise. □

We shall define a **bisimulation** R on a coalgebra (S, α^S) to be an equivalence relation on S for which there is a coalgebra (R, α^R) such that the projection functions $\pi_1, \pi_2: R \to S$ give rise to homomorphisms of the form

$$\pi_1, \pi_2: (R, \alpha^R) \longrightarrow (S, \alpha^S)$$

Lemma 4.5 An equivalence relation R on S is a bisimulation on (S, α^S) just in case for all $s, s', t \in S$, if $s \overset{\alpha^S}{\leadsto} s'$ and $s \; R \; t$, then there exists $t' \in S$ for which $t \overset{\alpha^S}{\leadsto} t'$ and $s' \; R \; t'$.

Proof Easy. □

Given a bisimulation R on a coalgebra (S, α^S), we can endow S/R with a coalgebra structure by defining

$$\begin{tikzcd} S/R \arrow[r, "\alpha^{S/R}"] & \mathbb{P}\,(S/R) \\ {[s]} \arrow[r, mapsto] & (\mathbb{P}\,q \circ \alpha^S)s \end{tikzcd}$$

Note that this is well-defined precisely because R is a bisimulation. It follows that the quotient function is indeed a homomorphism $q: (S, \alpha^S) \to (S/R, \alpha^{S/R})$.

Theorem 4.6 Let $h: (S, \alpha^S) \to (T, \alpha^T)$ be a homomorphism, and let K_h be the kernel of h. Then there is a diagram of the form

$$\begin{tikzcd} (S, \alpha^S) \arrow[r, "h"] \arrow[d, "q"'] & (T, \alpha^T) \\ (S/K_h, \alpha^{S/K_h}) \arrow[r, shift left, "\phi"] \arrow[r, phantom, "\cong"'] & (im(h), \alpha^{im(h)}) \arrow[l, shift left, "\psi"] \arrow[u, hook, "\iota"'] \end{tikzcd}$$

for which $h = \iota \circ \phi \circ q$.

Proof It is an exercise to verify that K_h admits a bisimulation structure for S. All that remains is to verify the existence of an isomorphism, and check that the diagram commutes. We define ϕ by setting $i([s]) \stackrel{\text{def}}{=} hs$ and ψ by $\psi(t) \stackrel{\text{def}}{=} [s]$ where $t = hs$ for some $s \in S$. $\square$

Theorem 4.7 Let (S, α^S) be a coalgebra, (X, α^X) a **subcoalgebra** of (S, α^S), and R a bisimulation on (S, α^S). Set $Q \stackrel{\text{def}}{=} R \cap X^2$, $\overline{X} \stackrel{\text{def}}{=} (\pi_1 \circ \pi_2^{-1})X$, and $Q' \stackrel{\text{def}}{=} R \cap \overline{X}^2$. Then there is a diagram of the form

$$\begin{tikzcd}
(S, \alpha^S) \arrow[rr, "q"] & & (S/R, \alpha^{S/R}) \\
(X, \alpha^X) \arrow[u, hook, "i"'] \arrow[dr, "\overline{q}"'] & & (im(qi), \alpha^{im(qi)}) \arrow[u, hook] \\
& (X/Q, \alpha^{X/Q}) \arrow[ur, "\cong"'] &
\end{tikzcd}$$

Proof This is left as an exercise. $\square$

5 Induction and Coinduction Principles

In this section, let $\mathcal{C}$ be a category, and F an endofunctor on $\mathcal{C}$. The definitions of congruence and bisimulation already given can be seen to apply to a categorical binary relation $\iota: R \rightarrowtail A \times A$, where the projection morphisms are given by $p_i \stackrel{\text{def}}{=} \pi_i \circ \iota: R \to A$ where π_i are the binary product projections on $A \times A$.

5.1 For Algebras

Principle of Unary Induction

Let $\alpha^I: FI \to I$ be an initial algebra. If $S \rightarrowtail I$ is a subobject in $\mathcal{C}$, then to show that $S \cong I$, it is sufficient to prove that S is a subalgebra of I.

Principle of Binary Induction

Let $\alpha^I: FI \to I$ be an initial algebra. If $R \rightarrowtail I \times I$ is a binary relation on I, then to show that Eq_I is a subobject of R, it is sufficient to prove that R admits a congruence $\gamma^R: FR \to R$ on $\alpha^I: FI \to I$.

Proof We leave the proof of the first principle as an exercise. For the second, recall that such a congruence amounts to a diagram of the form

$$\begin{tikzcd}
FI \arrow[r, "\alpha^I"] & I \arrow[d, dotted, shift left=2, "\overline{\gamma^R}"] \\
FR \arrow[u, "Fp_i"] \arrow[r, "\gamma^R"'] & R \arrow[u, "p_i"]
\end{tikzcd}$$

and by uniqueness of mediating morphisms, $\pi_i \circ \overline{\gamma^R} = id_I$, implying that $\overline{\gamma^R}$ is monic. Hence, from the definition of Eq_I, we have

$$\begin{tikzcd}
Eq_I \arrow[r, tail, "\pi"] \arrow[d, "\pi"'] & I \arrow[r, tail, "\overline{\gamma^R}"] \arrow[d, "id"] & R \arrow[d, "\pi_i"] \\
I \arrow[r, "id"'] & I \arrow[r, tail, "id"'] & I
\end{tikzcd}$$

which completes the proof. $\square$

5.2 For Coalgebras

Principle of Unary Coinduction

Let $\alpha^C\colon C \to FC$ be a final coalgebra. In order to prove that x is a global element of C, it is sufficient to prove that there exists a subcoalgebra $\gamma^S\colon S \to FS$ on $\alpha^C\colon C \to FC$ for which there is a global element $x'\colon 1 \to S$ for which $x = \overline{\gamma^S} \circ x'$.

Principle of Binary Coinduction

Let $\alpha^C\colon C \to FC$ be a final coalgebra. If $\iota\colon R \rightarrowtail C \times C$ is a binary relation on C, then to show that R is a subobject of Eq_C, it is sufficient to prove that R admits a bisimulation $\gamma^R\colon R \to FR$ on $\alpha^C\colon C \to FC$. In particular, in order to prove that the global elements $x, x'\colon 1 \to C$ are equal, it is sufficient to prove that that the global element $\langle x, x'\rangle\colon 1 \to C \times C$ factors through $\iota\colon R \rightarrowtail C \times C$.

Proof If R admits such a bisimulation, we have the following diagram with $p_1 = p_2$ by uniqueness—they must both be $\overline{\gamma^R}$.

$$\begin{tikzcd}
R \arrow[r, "\gamma^R"] \arrow[d, "p_1 = p_2"'] & FR \arrow[d, "Fp_i"] \\
C \arrow[r, "\alpha^C"'] & FC
\end{tikzcd}$$

Suppose that $\langle x, x'\rangle$ factors through ι by ψ. Then we have the following diagram

$$\begin{tikzcd}
 & R \arrow[r, dotted, "\phi"] \arrow[d, tail, "\iota"] & Eq_C \arrow[r, tail, "\pi"] \arrow[d, tail, "\pi"] \arrow[dl, tail, "{\langle \pi, \pi \rangle}"'] & C \arrow[d, "id"] \\
1 \arrow[ur, "\psi"] \arrow[r, "{\langle x, x' \rangle}"'] & C \times C \arrow[r, shift left, "\pi_1"] \arrow[r, shift right, "\pi_2"'] & C \arrow[r, "id"'] & C
\end{tikzcd}$$

By definition, $p_1 \overset{\text{def}}{=} \pi_1 \circ \iota$ and $p_2 \overset{\text{def}}{=} \pi_2 \circ \iota$. Hence $\pi_1 \circ \iota = \pi_2 \circ \iota$, and as $\langle x, x' \rangle = \iota \circ \psi$ we have $x = x'$.

If we define $\phi \overset{\text{def}}{=} \langle \pi_1 \circ \iota, \pi_2 \circ \iota \rangle$, the unique mediating morphism arising from the kernel pullback square, then

$$\langle \pi, \pi \rangle \circ \phi = \langle \pi_1 \circ \iota, \pi_2 \circ \iota \rangle = \langle \pi_1, \pi_2 \rangle \circ \iota = id_C \circ \iota = \iota$$

and as ι is monic, $\phi\colon R \rightarrowtail Eq_C$ is a subobject.

$\square$

The final <u>exercises</u> are to check that when X is a set, $\mathcal{R}$ is a set of rules on X, $\mathbb{P}(X)$ is regarded as a category, and the name of $\mathcal{R}$ is regarded as a functor, the unary [co]induction principles reduce to the corresponding principles on page 3; and to think about the Binary Principles in this setting.

Index