Solving Optimisation Problems with Catamorphisms

Richard S. Bird and Oege de Moor*

Oxford University Programming Research Group
11 Keble Road, Oxford OX1 3QD

Abstract. This paper contributes to an ongoing effort to construct a calculus for deriving programs for optimisation problems. The calculus is built around the notion of initial data types and *catamorphisms* which are homomorphisms on initial data types. It is shown how certain optimisation problems, which are specified in terms of a relational catamorphism, can be solved by means of a functional catamorphism. The result is illustrated with a derivation of Kruskal's algorithm for finding a minimum spanning tree in a connected graph.

1 Introduction

Efficient algorithms for solving optimisation problems can sometimes be expressed as homomorphisms on initial data types. Such homomorphisms, which correspond to the familiar *fold* operators in functional programming, are called *catamorphisms*. In this paper, we give conditions under which an optimisation problem can be solved by a catamorphism. Our results are a natural generalisation of earlier work by Jeuring [5, 6], who considered the same problem in a slightly less abstract setting. The main contribution of the paper is to show how several seemingly disparate results about subsequences, permutations, sequence partitions, subtrees, and so on, can all be captured in a single theorem.

The specification of an optimisation problem is usually split into three components, called the *generator*, *filter*, and *selector*, respectively. Consider, for example, the construction of a longest ascending subsequence of a sequence of numbers. This problem might be specified as

$$las = max(\#) \cdot up\triangleleft \cdot subs.$$

The function *subs* is the generator and returns the set of all subsequences of a list. The filter $up\triangleleft$ retains only those subsequences which are ascending. The remaining term $max(\#)$ is the selector; it is a relation such that $a(max(\#))x$ holds if a is a longest sequence in x.

We can develop an algorithm from a specification like the above in two steps. First, we express the composition of the filter and generator as a catamorphism. The conditions under which this is possible are given by a generalisation of Malcolm's *promotion* theorem [8]. Second, we take the resulting catamorphism g and try to

* Research supported by the Dutch Organisation for Scientific Research, grant NFI 62–518.

express the composition $max(\#) \cdot g$ as a second catamorphism. This is also achieved by promotion but, as we shall see, the conditions are much more restrictive.

Both the generator and the result of the first step are catamorphisms returning sets of values. Using the fact that set–valued functions are isomorphic to relations, we can express many set–valued catamorphisms as *relational* catamorphisms. Briefly, these are catamorphisms to algebras in which the operators are arbitrary relations rather than functions. It turns out that this observation allows us to simplify the formal treatment substantially.

The structure of the rest of the paper is as follows. In Section 2, we discuss the basic framework of sets and functions, and define the notion of catamorphism formally. We also state and prove the promotion theorem. In Section 3 we exploit the isomorphism between relations and set–valued functions to extend this material to relations. In Section 4 we define the notion of a relational catamorphism. Examples of such catamorphisms are given in Section 5. The next step in the development is to study the properties of the relation $max(R)$, and we do this in Section 6. In particular, we will need a weak form of monotonicity, called *maxotonicity*. With this machinery we can state and prove the main theorem of the paper. In the last section we show how the theorem applies to one particular example, namely the derivation of Kruskal's algorithm for minimum cost spanning trees.

2 Preliminaries

Throughout, we assume that we are working in one of two categories: the category *Fun* of sets and total functions, and (later on) the category *Rel* of sets and relations. The restriction to set theory is for purely expository reasons, as we could have started with a suitable topos and its category of relations instead of *Fun* and *Rel* [9]. It is possible to phrase the axioms of such a topos in terms of elementary identities between relations [4], and most of our results can be derived from these identities by equational reasoning. However, rather than taking such an abstract course, we will explain ideas using set–theoretic concepts. In particular, only a passing acquaintance with category theory is assumed, and we shall review all definitions essential to the exposition.

Functions and functors. Total functions will be denoted by lower case letters. We write $f : A \to B$ to indicate that f has source type A and target type B. The identity function of type $A \to A$ is written id_A, though we usually omit the subscript. Functional composition is denoted by a dot, so $(f \cdot g)\ a = f(g\ a)$.

We shall also need the notion of a *functor*. Functors, which will be denoted by sans serif letters, are homomorphisms between categories that preserve identity arrows and composition. For example, $\mathsf{F} : \mathit{Fun} \to \mathit{Fun}$ is a functor if $\mathsf{F}A$ is an object (i.e. set or 'type') of *Fun* for each object A of *Fun* and $\mathsf{F}f : \mathsf{F}A \to \mathsf{F}B$ for each arrow (or 'function') $f : A \to B$ of *Fun*. Moreover, $\mathsf{F}id = id$ and $\mathsf{F}(f \cdot g) = \mathsf{F}f \cdot \mathsf{F}g$. Functor composition is denoted simply by juxtaposition, so we have $(\mathsf{FG})f = \mathsf{F}(\mathsf{G}f)$. Usually, we write $\mathsf{FG}f$ without any brackets.

One important example of a functor is the *power* functor P. Here, $\mathsf{P}A$ denotes the power set of A, and for $f : A \to B$ the function $\mathsf{P}f : \mathsf{P}A \to \mathsf{P}B$ is defined by

$$(\mathsf{P}f)x = \{f\ a \mid a \in x\}.$$

In words, $\mathsf{P}f$ applies f to every element of a set. The verification that P is indeed a functor is straightforward.

Products. The *product* of two sets B and C is a set $B \times C$ together with two *projection* functions, $\pi_1 : B \times C \to B$ and $\pi_2 : B \times C \to C$, and a binary operator $\langle \text{-}, \text{-} \rangle$ called *split*. Given two functions $f : A \to B$ and $g : A \to C$, we have $\langle f, g \rangle : A \to B \times C$. These operators satisfy

$$h = \langle f, g \rangle \equiv (\pi_1 \cdot h = f \ \wedge \ \pi_2 \cdot h = g)$$

for all $h : A \to B \times C$. This property characterises product up to isomorphism, so we are free to choose any representative from the isomorphism class determined by the definition. In set theory, the canonical choice for $B \times C$ is cartesian product, and the split operator is defined by

$$\langle f, g \rangle a = (f\ a, g\ a).$$

The operator $\times$ on sets can also be defined on functions: $f \times g = \langle f \cdot \pi_1, g \cdot \pi_2 \rangle$. This turns $\times$ into a functor (strictly speaking, a *bifunctor*), because $\times$ preserves composition and identities:

$$(f \times g) \cdot (h \times k) = (f \cdot h) \times (g \cdot k) \quad \text{and} \quad id_A \times id_B = id_{A \times B} \ .$$

Coproducts. Analogous to the definition of *product*, we can define the notion of *coproduct*. The coproduct of two sets B and C is a set $B + C$ together with two *injection* functions $\iota_1 : B \to (B + C)$ and $\iota_2 : C \to (B + C)$, and a binary operator $[\text{-}, \text{-}]$ called *join*. Given two functions $f : B \to A$ and $g : C \to A$, the function $[f, g]$ has type $B + C \to A$. These operators satisfy

$$h = [f, g] \equiv (h \cdot \iota_1 = f \ \wedge \ h \cdot \iota_2 = g).$$

This property characterises coproduct up to isomorphism. In set theory, the canonical choice for $B + C$ is the disjoint sum, defined by

$$B + C \ = \ \{ \, (1, b) \mid b \in B \, \} \cup \{ \, (2, c) \mid c \in C \, \} \ .$$

The join operator is defined by

$$[f, g](1, b) = f\ b \text{ and } [f, g](2, c) = g\ c.$$

Just like $\times$, we can make $+$ into a functor by defining

$$f + g = [\iota_1 \cdot f, \ \iota_2 \cdot g].$$

Polynomial Functors. The class of *polynomial* functors in *Fun* $\to$ *Fun* is defined inductively by the following clauses:

1. The identity functor and constant functors are polynomial.
2. If F and G are polynomial functors, then so are their composition FG, their sum $\mathsf{F} + \mathsf{G}$, and their product $\mathsf{F} \times \mathsf{G}$, where

$$(\mathsf{F} + \mathsf{G})(k) = \mathsf{F}(k) + \mathsf{G}(k)$$
$$(\mathsf{F} \times \mathsf{G})(k) = \mathsf{F}(k) \times \mathsf{G}(k).$$

As we shall see, polynomial functors enjoy a number of useful properties. One example of a functor that is not polynomial is the power functor P.

Algebras. Let $\mathcal{E}$ be a category and $\mathsf{F} : \mathcal{E} \to \mathcal{E}$ a functor. By definition, an F–*algebra* is an arrow $f : \mathsf{F}A \to A$ of $\mathcal{E}$. For example, consider the set L of finite sequences with elements of type E. Among other ways, we can construct all finite sequences with the help of two functions: the constant function $\nu : \mathbf{1} \to L$, which returns the empty sequence $[\,]$, and the binary operator $+\!\!+\!\!< : L \times E \to L$ (pronounced *snoc*), which takes a sequence and an element, and appends the element to the sequence:

$$[e_0, e_1, \ldots, e_{n-1}] +\!\!+\!\!< e_n = [e_0, e_1, \ldots, e_{n-1}, e_n].$$

(The set $\mathbf{1}$ in the type of ν stands for some distinguished one–element set and plays the role of *unit type*.)

The join $[\nu, +\!\!+\!\!<]$ therefore has type type $\mathbf{1} + (L \times E) \to L$. This join is an F–algebra, where $\mathsf{F} : \mathit{Fun} \to \mathit{Fun}$ is given by by

$$\begin{aligned} \mathsf{F}A &= \mathbf{1} + (A \times E) \\ \mathsf{F}h &= \mathit{id} + (h \times \mathit{id}) \ . \end{aligned}$$

Homomorphisms, initial algebras, and catamorphisms. Let $f : \mathsf{F}A \to A$ and $g : \mathsf{F}B \to B$ be F–algebras. By definition, an F–*homomorphism* from f to g is an arrow $h : A \to B$ such that

$$h \cdot f = g \cdot \mathsf{F}h \ .$$

The composition of two F–homomorphisms is again an F–homomorphism, so the F–algebras in $\mathcal{E}$ form the objects of a category $\mathcal{E}_{\mathsf{F}}$ in which the arrows are F–homomorphisms. For many functors F this category has an *initial object*, which we denote by $\mu\mathsf{F}$. To say that $\mu\mathsf{F}$ is initial means that for any other F–algebra f, there exists a unique homomorphism, which we denote by $(\!|f|\!)_{\mathsf{F}}$, from $\mu\mathsf{F}$ to f. Arrows of the form $(\!|h|\!)_{\mathsf{F}}$ are called *catamorphisms*.

The defining property of catamorphisms is called the *unique extension* property and is formalised by the equivalence

$$g = (\!|f|\!)_{\mathsf{F}} \equiv g \cdot \mu\mathsf{F} = f \cdot \mathsf{F}g \ .$$

From now on we shall write $(\!|f|\!)$ instead of $(\!|f|\!)_{\mathsf{F}}$, the functor F being understood from context.

An example of an initial F–algebra is the data type of lists discussed above. The so–called *fold–left* operator $\mathit{foldl}\ (\oplus)\ e$ common in functional programming can be defined by two recursion equations

$$\begin{aligned} \mathit{foldl}\ (\oplus)\ e\ [\,] &= e \\ \mathit{foldl}\ (\oplus)\ e\ (x +\!\!+\!\!< a) &= (\mathit{foldl}\ (\oplus)\ e\ x) \oplus a \ . \end{aligned}$$

These equations are equivalent to the statement that $\mathit{foldl}\ (\oplus)\ e$ is an F–homomorphism from $[\nu, +\!\!+\!\!<]$ to $[e, \oplus]$. Since $[\nu, +\!\!+\!\!<]$ is the initial F–algebra $\mu\mathsf{F}$, we can write $\mathit{foldl}\ (\oplus)\ e = (\!|[e, \oplus]|\!)$. For legibility we will omit the inner brackets in such expressions, writing $(\!|e, \oplus|\!)$ instead.

Promotion. The following theorem is called the *promotion* theorem. In its statement we use the abbreviation $f = g \pmod{h}$ for $f \cdot h = g \cdot h$.

Theorem 1. *For any functor* $\mathsf{F} : Fun \to Fun$ *with an initial algebra we have*

$$f \cdot (\![g]\!) = (\![h]\!) \equiv f \cdot g = h \cdot \mathsf{F}f \pmod{\mathsf{F}(\![g]\!)}.$$

Proof. The proof is simple:

$$\begin{array}{ll} & f \cdot (\![g]\!) = (\![h]\!) \\ \equiv & \{\text{unique extension property}\} \\ & f \cdot (\![g]\!) \cdot \mu\mathsf{F} = h \cdot \mathsf{F}(f \cdot (\![g]\!)) \\ \equiv & \{\mathsf{F} \text{ is a functor}\} \\ & f \cdot (\![g]\!) \cdot \mu\mathsf{F} = h \cdot \mathsf{F}f \cdot \mathsf{F}(\![g]\!) \\ \equiv & \{\text{unique extension property}\} \\ & f \cdot g \cdot \mathsf{F}(\![g]\!) = h \cdot \mathsf{F}f \cdot \mathsf{F}(\![g]\!). \end{array}$$

□

3 Relations

So far we have been working in the category *Fun* of sets and total functions. Now we turn to *Rel*, the category of sets and relations. Our objective is to see how and when the above theory of functions can be extended to relations.

Relations will be denoted by upper case letters $R, S, \ldots$. We write $R : A \to B$ to denote the fact that R has source A and target B. For such an R we write $b(R)a$ to denote the fact that b stands in the relation R to a. Functions are special kinds of relations, namely total and single–valued ones, so if f is a function, then $b(f)a$ means $b = f\ a$. We again use a dot for relational composition, so $c(R \cdot S)a$ holds if there is a b with $c(R)b$ and $b(S)a$. We will assume familiarity with the operations of the relational calculus displayed in Figure 1.

operator	pronunciation	meaning
R°	R converse	$b(R^\circ)a = a(R)b$
$R \cap S$	R and S	$b(R \cap S)a = b(R)a \wedge b(S)a$
$R \cup S$	R or S	$b(R \cup S)a = b(R)a \vee b(S)a$
$R \Rightarrow S$	R implies S	$b(R \Rightarrow S)a = b(R)a \Rightarrow b(S)a$
R/S	R divided by S	$b(R/S)a = (\forall c : a(S)c \Rightarrow b(R)c)$
$p?$	test p	$b(p?)a = (b = a) \wedge (p\ a)$
$R\Box$	domain of R	$b(R\Box)a = (b = a) \wedge (\exists c : c(R)a)$
$\Box R$	range of R	$b(\Box R)a = (b = a) \wedge (\exists c : b(R)c)$

Fig. 1. Operators of the relational calculus.

Power Transpose. In set theory there are various representations of relations, of which the most well–known is to represent a relation $R : A \to B$ as a set of pairs (a, b) with $a \in A$ and $b \in B$. However, one can also think of a relation $R : A \to B$ as a set–valued function $A \to \mathsf{P}B$, where $\mathsf{P}B$ denotes the power set of B. We will take the view that a relation is a set of pairs, so that $b(R)a$ means $(a, b) \in R$. The operator Λ that sends a relation to the corresponding set–valued function is called *power transpose* and is defined by the equation

$$(\Lambda R)\, a = \{b \mid b(R)a\}.$$

In particular, $(\Lambda f)a = \{f\ a\}$. The function Λ is an isomorphism and so the inverse function Λ^{-1} is well–defined. For $R : A \to B$ and $f : A \to \mathsf{P}B$ we have

$$\in_B \cdot \Lambda R \;=\; R \quad \text{and} \quad \Lambda(\in_B \cdot f) \;=\; f,$$

where $\in_B : \mathsf{P}B \to B$ is the membership relation. Usually, we omit the subscript.

Existential Image. Readers familiar with category theory will recognise that power transpose defines a right adjoint of the inclusion functor $Fun \to Rel$. This right adjoint is known as the *existential image*, and the adjunction is called the *power adjunction*. In fact, most of set theory can be derived from the existence of the power adjunction, an observation which lies at the heart of topos theory.

Less abstractly, the existential image of a relation $R : A \to B$ is a function $\mathsf{E}R : \mathsf{P}A \to \mathsf{P}B$, defined by

$$(\mathsf{E}R)\, x = \{b \mid (\exists a \in x : b(R)a)\}.$$

The existential image preserves identity and composition and, since we can take $\mathsf{E}A = \mathsf{P}A$, we have that E is a functor from *Rel* to *Fun*.

Note that $\mathsf{E}f = \mathsf{P}f$ for all functions f; in words, E and P coincide on functions. However, E is not the only functor of *Rel* that coincides with P on functions. In the next section we shall describe another and more useful one.

The power transpose Λ and the existential image E can each be defined in terms of the other. For $R : A \to B$ we have

$$\Lambda R \;=\; \mathsf{E}R \cdot \tau_A \quad \text{and} \quad \mathsf{E}R \;=\; \Lambda(R \cdot \in_A)\ ,$$

where $\tau_A : A \to \mathsf{P}A$ is the function which sends an element a in A to the singleton set $\{a\}$. From the first identity we obtain the law of composition

$$\Lambda(R \cdot S) = \mathsf{E}R \cdot \Lambda S.$$

Singleton and Membership. The singleton function τ (again, we omit the subscript) can be defined in terms of power transpose: $\tau = \Lambda id$. The collection of singleton formers is 'polymorphic', a fact which is expressed by the identity $\mathsf{P}f \cdot \tau = \tau \cdot f$. Category theorists refer to this equation by saying that τ is a *natural transformation* from the identity functor to P.

The membership relation is dual to singleton in that $\in = \Lambda^{-1} id$. Membership is also a natural transformation; in symbols, $\in \cdot \mathsf{E}R = R \cdot \in$. This duality is no accident

and stems from the fact that membership and singleton form the unit and counit of the power adjunction.

The 'big union' function $\bigcup$ is defined by $\bigcup = \mathsf{E}\in$. Big union is also a natural transformation, the proof being

$$\bigcup \cdot \mathsf{E}\mathsf{E}R = \mathsf{E}\in \cdot \mathsf{E}\mathsf{E}R = \mathsf{E}(\in \cdot \mathsf{E}R) = \mathsf{E}(R \cdot \in) = \mathsf{E}R \cdot \mathsf{E}\in = \mathsf{E}R \cdot \bigcup.$$

We also have

$$\mathsf{E}R = \mathsf{E}(\in \cdot \Lambda R) = \bigcup \cdot \mathsf{E}(\Lambda R).$$

(For those familiar with monads, $(\mathsf{E}, \tau, \bigcup)$ is the monad defined by the power adjunction.)

Filter. Many other set–theoretic operators can be defined in terms of power transpose and existential image. One important example is the *filter* operator. Given a Boolean–valued function $p : A \to \Omega$, we define $p\triangleleft : \mathsf{P}A \to \mathsf{P}A$ by $p\triangleleft = \mathsf{E}(p?)$. The advantage of such definitions is that they are easier to manipulate than traditional set comprehensions. For instance, here is the very short proof that filter distributes through union:

$$p\triangleleft \cdot \bigcup = \mathsf{E}(p?) \cdot \bigcup = \bigcup \cdot \mathsf{E}\mathsf{E}(p?) = \bigcup \cdot \mathsf{E}(p\triangleleft).$$

4 Relators and Cross–operators

Let us now turn to the question of whether each functor F of *Fun* can be extended to a functor of *Rel*. An important consideration is that we would like $\mathsf{F} : \mathit{Rel} \to \mathit{Rel}$ to be *monotonic*, i.e.

$$R \subseteq S \Rightarrow \mathsf{F}R \subseteq \mathsf{F}S.$$

Inclusion rather than equality is the 'natural' way of comparing relations. Of course, for total functions inclusion means equality. Fortunately, we have

Proposition 2. *For each functor* $\mathsf{F} : \mathit{Fun} \to \mathit{Fun}$ *there exists at most one monotonic functor in* $\mathit{Rel} \to \mathit{Rel}$ *that coincides with* F *on functions.*

If F does have such an extension, then we say that F is a *relator* and we use the same symbol for both functors. We can characterise relators as those functors F that satisfy the condition

$$f^\circ \cdot g = h \cdot k^\circ \Rightarrow (\mathsf{F}f)^\circ \cdot \mathsf{F}g = \mathsf{F}h \cdot (\mathsf{F}k)^\circ$$

It can be shown that every monotonic functor on relations maps functions to functions, so every monotonic functor is the extension of a relator.

For example, all polynomial functors are relators. The list functor L is also a relator; this functor sends a set A to the set of finite sequences with elements from A. On functions, L is defined by the equation

$$(\mathsf{L}f)[a_1, a_2, \ldots, a_n] = [fa_1, fa_2, \ldots, fa_n].$$

Thus L is the familiar *map* operator from functional programming and is similar to P except that it acts on lists rather than sets. The explicit definition of $\mathsf{L} : \mathit{Rel} \to \mathit{Rel}$ is rather messy:

$$[b_1, \ldots, b_n](\mathsf{L}R)[a_1, \ldots, a_m] = (n = m) \ \wedge \ (\forall i : 1 \leq i \leq n : b_i(R)a_i).$$

The functor $\mathsf{P} : \mathit{Fun} \to \mathit{Fun}$ also satisfies the above conditions, although it is not polynomial. The extension $\mathsf{P} : \mathit{Rel} \to \mathit{Rel}$ is defined explicitly by

$$\begin{aligned} y(\mathsf{P}R)x = \ & (\forall a \in x : \exists b \in y : b(R)a) \wedge \\ & (\forall b \in y : \exists a \in x : b(R)a) \ . \end{aligned}$$

Hence $\mathsf{P}R$ corresponds to the *Egli–Milner* ordering induced by R. Note that $\mathsf{P} : \mathit{Rel} \to \mathit{Rel}$ is *not* the same as $\mathsf{E} : \mathit{Rel} \to \mathit{Rel}$ since P returns relations and is monotonic, while E returns functions and is not monotonic.

Cross–operators. Relators can be characterised in another useful way, namely in terms of the existence of so–called *cross–operators*. Cross–operators are a generalisation of the polymorphic cartesian product function.

By definition, a *cross–operator* on $\mathsf{F} : \mathit{Fun} \to \mathit{Fun}$ is a collection of functions $\mathsf{F}\dagger_A : \mathsf{FP}A \to \mathsf{PF}A$ that satisfies the following four axioms:

1. Crossing is polymorphic, i.e. $\mathsf{F}\dagger \cdot \mathsf{FP}f = \mathsf{PF}f \cdot \mathsf{F}\dagger$.
2. Crossing singletons gives a singleton, i.e. $\mathsf{F}\dagger \cdot \mathsf{F}\tau = \tau$.
3. Crossing distributes over union, i.e. $\mathsf{F}\dagger \cdot \mathsf{F}\bigcup = \bigcup \cdot \mathsf{PF}\dagger \cdot \mathsf{F}\dagger$.
4. Crossing is monotonic, i.e. $f \subseteq g \Rightarrow \mathsf{F}\dagger \cdot \mathsf{F}f \subseteq \mathsf{F}\dagger \cdot \mathsf{F}g$, where $f \subseteq g$ means pointwise inclusion of set–valued functions f and g.

Proposition 3. *For each functor* $\mathsf{F} : \mathit{Fun} \to \mathit{Fun}$ *there exists at most one cross–operator* $\mathsf{F}\dagger$. *This cross–operator exists if and only if* F *is a relator, in which case we have* $\mathsf{F}\dagger = \Lambda(\mathsf{F}\in)$.

For an example of a particular cross–operator, recall the list functor L. The cross–operator $\mathsf{L}\dagger$ is the polymorphic function that sends a sequence of sets to its cartesian product:

$$\mathsf{L}\dagger[x_1, x_2, \ldots, x_n] = \{[a_1, a_2, \ldots, a_n] \mid (\forall i : a_i \in x_i)\}.$$

It is possible to develop a little calculus of cross–operators for synthesising the cross–operator of a composite functor by simple calculation. Reluctantly, we omit a discussion of this calculus for reasons of space.

Relational Catamorphisms. So far we have only considered algebras and catamorphisms in the category *Fun* of sets and total functions. This is sufficient for many purposes, but for a satisfactory treatment of optimisation problems, and in particular for the selector $\mathit{min}R$, we would like to move to relations. So, how can the above theory be extended to algebras and catamorphisms in *Rel*? The obvious setting for relational algebras is the category $\mathit{Rel}_{\mathsf{F}}$ for some relator $\mathsf{F} : \mathit{Fun} \to \mathit{Fun}$ with an initial algebra. Fortunately, the initial algebra of $\mathsf{F} : \mathit{Rel} \to \mathit{Rel}$ coincides with the initial algebra of $\mathsf{F} : \mathit{Fun} \to \mathit{Fun}$.

Proposition 4. (Eilenberg and Wright [3]) *Let* $\mathsf{F} : \mathit{Fun} \to \mathit{Fun}$ *be a relator with initial algebra* $\mu\mathsf{F}$. *Then* $\mu\mathsf{F}$ *is also an initial algebra of the extension of* F *to* $\mathit{Rel} \to \mathit{Rel}$. *Furthermore,* $\Lambda(\llbracket R \rrbracket) = (\!|\mathsf{E}R \cdot \mathsf{F}\dagger|\!)$.

We also have the following generalisation of the promotion theorem.

Theorem 5. (Backhouse *et al.* [1]) *Let* F *be a relator and possess an initial algebra. Then*

$$R \cdot (\!|S|\!) \subseteq (\!|T|\!) \Leftarrow R \cdot S \subseteq T \cdot \mathsf{F}R \pmod{\mathsf{F}\llbracket S \rrbracket}.$$

Moreover, the same implication holds when $\subseteq$ *is replaced by* $\supseteq$ *(and so also when* $\subseteq$ *is replaced by* $=$*).*

We will not prove the generalisation, but we will prove the following corollary:

Corollary 6.

$$p\triangleleft \cdot \Lambda(\llbracket R \rrbracket) = \Lambda(\llbracket S \rrbracket) \Leftarrow p? \cdot R = S \cdot \mathsf{F}(p?) \pmod{\mathsf{F}\llbracket R \rrbracket}.$$

Proof. We have

$$\begin{array}{ll} & p\triangleleft \cdot \Lambda(\llbracket R \rrbracket) \\ = & \quad \{\text{definition of filter}\} \\ & \mathsf{E}(p?) \cdot \Lambda(\llbracket R \rrbracket) \\ = & \quad \{\text{power transpose of composition}\} \\ & \Lambda(p? \cdot (\!|\llbracket R \rrbracket|\!)) \\ = & \quad \{\text{promotion}\} \\ & \Lambda(\llbracket S \rrbracket)\ . \end{array}$$

5 Examples

Let us now give some examples of relational catamorphisms. The first three are catamorphisms on the data type of finite sequences described above.

Subsequences. Consider a finite sequence x. By leaving out some of the elements of x, but retaining the original left–to–right order, we obtain a *subsequence* of x. Formally, y is a subsequence of x if

$$y\ (\!|\nu,\ \pi_1 \cup +\!\!\!\prec|\!)\ x\ .$$

One can think of $(\pi_1 \cup +\!\!\!\prec)$ as a non–deterministic operator which, when given a pair (x, a), either returns x or appends a to x. The catamorphism $(\!|\nu,\ \pi_1 \cup +\!\!\!\prec|\!)$ is then a relation that holds between a sequence x and a subsequence of x. It follows that the function *subs*, which returns the set of all subsequences of a sequence, is defined by

$$\mathit{subs} = \Lambda(\llbracket \nu,\ \pi_1 \cup +\!\!\!\prec \rrbracket)\ .$$

We can use Proposition 4 to eliminate the power transpose from the right–hand side of this equation. After simplifying, we obtain a characterisation of *subs* expressed in terms of functions: $subs = (\!|\tau \cdot \nu,\ \otimes|\!)$, where $\tau \cdot \nu$ is the constant function returning $\{[\,]\}$, and

$$xs \otimes a = xs \cup \{x +\!\!\prec a \mid x \in xs\}.$$

The relational characterisation of *subs* seems preferable simply because it is shorter.

Let us now instantiate the corollary of the promotion theorem to the function *subs*. We shall need the following property of predicates. A predicate p is said to be *prefix–closed* with *derivative* q if $p\,[\,]$ holds and $p\,(x +\!\!\prec a) = p\,x \wedge q(x,a)$. More shortly, p is prefix–closed with derivative q if

$$\begin{aligned} p? \cdot \nu &= \nu \\ p? \cdot +\!\!\prec &= +\!\!\prec \cdot q? \cdot (p? \times id). \end{aligned}$$

Assuming this property of p we get

$$\begin{array}{ll} & p? \cdot [\nu,\ \pi_1 \cup +\!\!\prec] \\ = & \quad \{\text{coproduct and composition over union}\} \\ & [p? \cdot \nu,\ (p? \cdot \pi_1) \cup (p? \cdot +\!\!\prec)] \\ = & \quad \{p \text{ is prefix–closed}\} \\ & [\nu,\ (p? \cdot \pi_1) \cup (+\!\!\prec \cdot q? \cdot (p? \times id)] \\ = & \quad \{\text{since } p? \cdot \pi_1 = \pi_1 \cdot (p? \times id)\} \\ & [\nu,\ (\pi_1 \cup (+\!\!\prec \cdot q?))(p? \times id)] \\ = & \quad \{\text{coproduct}\} \\ & [\nu,\ \pi_1 \cup (+\!\!\prec \cdot q?)] \cdot (id + (p? \times id)) \\ = & \quad \{\text{definition of } \mathsf{F}\} \\ & [\nu,\ \pi_1 \cup (+\!\!\prec \cdot q?)] \cdot \mathsf{F}(p?). \end{array}$$

Hence if p is prefix–closed with derivative q, then

$$p\triangleleft \cdot subs = \Lambda(\!|\nu,\ \pi_1 \cup (+\!\!\prec \cdot q?)|\!).$$

Eliminating Λ, we get that $p\triangleleft \cdot subs = (\!|\tau \cdot \nu,\ \otimes|\!)$, where

$$xs \otimes a = xs \cup \{x +\!\!\prec a \mid x \in xs\ \wedge\ q(x,a)\}.$$

We will use this result in our final example, and in the appendix we give a more abstract version of the above argument.

Partitions. A *partition* of a sequence x is a division of x into non–empty contiguous subsequences. For example, the sequence of sequences

$$[[1,2],[3],[4,5,6]]$$

is a partition of $[1,2,3,4,5,6]$. The function *parts*, which returns all partitions of its argument, is the power transpose of a relational catamorphism

$$parts = \Lambda(\!|\nu,\ \oplus \cup \otimes|\!),$$

where $\oplus$ is a total function defined by $xs \oplus a = xs +\!\!+\!\!\prec [a]$ and $\otimes$ is the partial function defined on nonempty lists by

$$(xs +\!\!+\!\!\prec x) \oplus a = xs +\!\!+\!\!\prec (x +\!\!+\!\!\prec a)\ .$$

In words, we get a partition of $x +\!\!+\!\!\prec a$ by taking some partition xs of x, and either appending $[a]$ to xs (represented by the term $xs \oplus a$), or appending a to the last component of xs (represented by the term $xs \otimes a$). The power–transpose can be eliminated in a similar manner as before, but we omit details.

Permutations. The function *perms*, which returns all permutations of a sequence, is also the power transpose of a relational catamorphism

$$perms = \Lambda(\!|\nu,\ (\oplus \cdot +\!\!+^{\circ} \times id)|\!),$$

where $+\!\!+$ denotes concatenation, and $(y,z) \oplus a = y +\!\!+ [a] +\!\!+ z$. Thus,

$$((y,z),a)(+\!\!+^{\circ} \times id)(x,a) \ \text{ if } \ x = y +\!\!+ z.$$

Eliminating Λ from this expression, we get $perms = (\!|\nu\,;\tau,\ \otimes|\!)$, where $\otimes$ is defined by the set comprehension

$$xs \otimes a = \{y +\!\!+ [a] +\!\!+ z \mid (\exists x : x \in xs \ \wedge\ y +\!\!+ z = x)\}.$$

Tree Pruning. Finally, we briefly describe a relational catamorphism over a different data type. Consider the type of *binary labelled trees* in which both the leaves and the internal labels are natural numbers. This type is defined as the initial F–algebra, where F is the polynomial functor given by

$$\begin{aligned} \mathsf{F}A &= \mathrm{N} + (A \times \mathrm{N} \times A) \\ \mathsf{F}k &= id_{\mathrm{N}} + (k \times id_{\mathrm{N}} \times k)\ . \end{aligned}$$

We will call the components of the initial F–algebra *Tip* and *Node*, so

$$\mu\mathsf{F} = [\,Tip, Node]\ .$$

Now consider the operator *prune* which takes a tree and prunes away some of its subtrees in a non–deterministic fashion. Formally, we define *prune* as a relational catamorphism:

$$prune = (\!|\,Tip, Node \cup (Tip \cdot \pi_2^3)|\!),$$

where π_2^3 is the projection function that returns the middle component of a triple. The power transpose of *prune* is the function that returns all possible prunings of its arguments. This function was used by Jeuring [5] to solve various optimisation problems on trees.

6 Minimisation

Now we consider the selection of an optimal element from a set of candidate solutions to an optimisation problem.

Let $R : A \to A$ be a *preorder*, so R is assumed to be reflexive (i.e. $id \subseteq R$) and transitive (i.e. $R \cdot R \subseteq R$). The relation $minR : \mathsf{P}A \to A$ relates a set to its minimum elements under R. Formally, we define

$$minR = \in \cap\ (R/\ni),$$

where $\ni = \in^\circ$. To understand this succinct definition, put $R = (\leq)$. We then get that $b(min(\leq))x$ if and only if b is both an element of x and a lower bound of x, i.e. for all a if $a \in x$, then $b \leq a$. Note that $maxR = minR^\circ$.

From the assumption that R is reflexive we get $minR \cdot \tau = id$. We give the proof:

$$\begin{array}{ll} & minR \cdot \tau \\ = & \quad \{\text{definition of } minR\} \\ & (\in \cap\, (R/\ni)) \cdot \tau \\ = & \quad \{\text{composition (with a function) distributes over } \cap\} \\ & (\in \cdot \tau) \cap (R/\ni) \cdot \tau \\ = & \quad \{\in \cdot \tau = id \text{ and } (R/\ni) \cdot \tau = R\} \\ & id \cap R \\ = & \quad \{R \text{ is reflexive}\} \\ & id. \end{array}$$

From the assumption that R is transitive, we get

$$minR \cdot \bigcup \supseteq minR \cdot \mathsf{P}minR.$$

We have $\supseteq$ rather than $=$ in this assertion since not every set has a minimum element — in particular, the empty set does not have one. However, we can replace $\supseteq$ by $=$ provided we compose the left–hand side with $\mathsf{P}((minR)\Box)$, so excluding sets without minimum elements under R. For reasons of space, we omit proof of these assertions.

Another useful result is that minimisation operators can always be composed:

$$minS \cdot \Lambda(minR) = min\, T,$$

where $T = R \cap (R^\circ \Rightarrow S)$. The proof of this identity, which we omit, has been vigorously studied by a number of researchers in the proof methods community *e.g.* [10].

Monotonicity and distributivity. In order to combine our results about relational catamorphisms and minimisation, we need to know how $minR$ interacts with F–algebras. Consider, for example, addition of natural numbers $+ : (\mathrm{N} \times \mathrm{N}) \to \mathrm{N}$. Addition is an algebra of the functor F given by

$$\mathsf{F}A = A \times A$$
$$\mathsf{F}k = k \times k.$$

Addition is monotonic with respect to $\leq$, a fact we can express succinctly by the assertion $(+) \cdot \mathsf{F}(\leq) \subseteq (\leq) \cdot (+)$. This assertion translates as

$$c = a + b \ \wedge \ a \leq a' \ \wedge \ b \leq b' \Rightarrow c \leq a' + b'.$$

More generally, $f : \mathsf{F}A \to A$ is *monotonic* with respect to R if

$$f \cdot \mathsf{F}R \subseteq R \cdot f.$$

Monotonicity is equivalent to distributivity. For example, writing **min** for the function that returns the (unique) minimum element of a set of numbers, we have

$$\mathbf{min}\, x + \mathbf{min}\, y = \mathbf{min}\, \{a + b \mid a \in x \ \wedge \ b \in y\},$$

provided the left-hand side is well-defined. The above implication can be expressed more shortly as

$$(+) \cdot \mathsf{F}min(\leq) \subseteq min(\leq) \cdot \mathsf{P}(+) \cdot \mathsf{F}\dagger.$$

Generalising to an arbitrary preorder R, we have

$$f \cdot \mathsf{F}R \subseteq f \cdot R \equiv f \cdot \mathsf{F}minR \subseteq minR \cdot \mathsf{P}f \cdot \mathsf{F}\dagger.$$

The next proposition is generalisation of this result. It deals with a weaker property than monotonicity, called *minotonicity*. A function $f : \mathsf{F}A \to A$ is said to be minotonic (with respect to R) on the range of a set–valued function $g : B \to \mathsf{P}A$ if

$$f \cdot \mathsf{F}(minR \cdot \Box g \cdot \ni) \subseteq R \cdot f.$$

Taking F as the identity functor, this reads

$$c = f\ b \ \wedge \ b(minR)x \ \wedge \ x(\Box g)x \ \wedge a \in x \Rightarrow c(R)(f\ a).$$

Since

$$\begin{array}{ll} & minR \cdot \Box g \cdot \ni \\ \subseteq & \quad \{\text{since } \Box g \subseteq id\} \\ & minR \cdot \ni \\ = & \quad \{\text{definition of } minR\} \\ & (\in \cap (R/\ni)) \cdot \ni \\ \subseteq & \quad \{\text{calculus}\} \\ & (\in \cdot \ni) \cap (R/\ni) \cdot \ni \\ \subseteq & \quad \{\text{since } (R/S) \cdot S \subseteq R\} \\ & R \end{array}$$

we have that $\mathsf{F}(minR \cdot \Box g \cdot \ni) \subseteq \mathsf{F}R$, and so monotonicity implies minotonicity. The following result is called the *minotonicity lemma* and the proof can be found in [2].

Lemma 7. *The function $f : \mathsf{F}A \to A$ is minotonic with respect to R on the range of $g : B \to \mathsf{P}A$ if and only if*

$$f \cdot \mathsf{F}minR \subseteq minR \cdot \mathsf{P}f \cdot \mathsf{F}\dagger \quad (\mathrm{mod}\ \mathsf{F}g).$$

7 Main Theorem

Now we come to the main theorem. Let F be a fixed relator in what follows, so $(\!|\text{-}|\!) = (\!|\text{-}|\!)_{\mathsf{F}}$. The theorem addresses the question of when it is possible to find a program for computing $minR \cdot \Lambda(\!|S|\!)$ that can be expressed as a catamorphism. We take the view that a program is a total function, so the task is to determine when

$$(\!|f|\!) \subseteq minR \cdot \Lambda(\!|S|\!)$$

for some total function f.

Theorem 8. *Suppose f is minotonic with respect to a preorder R on the range of $\Lambda(\!|S|\!)$. Then*

$$f \subseteq minR \cdot \Lambda S \Rightarrow (\!|f|\!) \subseteq minR \cdot \Lambda(\!|S|\!).$$

Proof. We calculate

$$\begin{array}{ll} & (\!|f|\!) \subseteq minR \cdot \Lambda(\!|S|\!) \\ \Leftrightarrow & \{\text{Proposition 4}\} \\ & (\!|f|\!) \subseteq minR \cdot (\!|\mathsf{E}S \cdot \mathsf{F}\dagger|\!) \\ \Leftarrow & \{\text{promotion}\} \\ & f \cdot \mathsf{F}minR \subseteq minR \cdot \mathsf{E}S \cdot \mathsf{F}\dagger \quad (\mathrm{mod}\ \mathsf{F}(\!|\mathsf{E}S \cdot \mathsf{F}\dagger|\!)) \\ \Leftarrow & \{\text{minotonicity}\} \\ & minR \cdot \mathsf{P}f \cdot \mathsf{F}\dagger \subseteq minR \cdot \mathsf{E}S \cdot \mathsf{F}\dagger \quad (\mathrm{mod}\ \mathsf{F}(\!|\mathsf{E}S \cdot \mathsf{F}\dagger|\!)) \\ \Leftarrow & \{\text{logical entailment}\} \\ & minR \cdot \mathsf{P}f \subseteq minR \cdot \mathsf{E}S. \end{array}$$

Now we have

$$\begin{array}{ll} & minR \cdot \mathsf{P}f \\ \subseteq & \{\text{given assumption, and monotonicity of } \mathsf{P}\} \\ & minR \cdot \mathsf{P}(minR \cdot \Lambda S) \\ = & \{\text{since } \mathsf{P} \text{ is a functor}\} \\ & minR \cdot \mathsf{P}minR \cdot \mathsf{P}(\Lambda S) \\ \subseteq & \{\text{since } R \text{ is transitive}\} \\ & minR \cdot \bigcup \cdot \mathsf{P}(\Lambda S) \\ = & \{\text{since } \bigcup \cdot \mathsf{P}(\Lambda S) = \mathsf{E}S \text{ for any } S\} \\ & minR \cdot \mathsf{E}S\ . \end{array}$$

8 Application: Kruskal's Algorithm

Let us now see how the above theorem applies to one particular example, namely the construction of a minimum cost spanning tree in a connected graph. Each edge of the graph has an associated cost, and we will write *cost e* for the cost of e. We will assume that the graph is given as a sequence of edges in ascending order of cost, an assumption that will allow us to develop Kruskal's algorithm.

The problem can be specified as follows:

$$mcst = min\, C \cdot \Lambda(max(\#)) \cdot acyclic^{\triangleleft} \cdot subs.$$

Here, *subs* returns the set of subsequences of the given sequence of edges, $acyclic^{\triangleleft}$ removes all subsequences containing a cycle, and $\Lambda(max(\#))$ returns the set of acyclic sequences of edges of greatest length, i.e. the spanning trees of the graph. The function $\#$ returns the length of a sequence. It is an abuse of notation to write $max(\#)$, since $\#$ is not an ordering; strictly speaking, we should write

$$max(\#^{\circ} \cdot (\leq) \cdot \#) \ .$$

With the same understanding, C is also a function and is defined as the sum of the costs of the edges in a given sequence. Putting all this together, *mcst* is a relation that holds between a minimum cost spanning tree and the graph.

As specified, the spanning tree problem involves the composition of two optimal selections. We can use an earlier result about combining minimisation operators to show that

$$min\, C \cdot \Lambda(max(\#)) = min\, R,$$

where the preorder R is given by

$$y(R)x = (\#y > \#x) \vee (\#y = \#x \wedge C\ y \leq C\ x) \ .$$

It follows that the spanning tree problem is solved by minimising with respect to R. Note that

$$y(R)x \Rightarrow (y +\!\!\!\prec a)(R)(x +\!\!\!\prec a)$$

for all a.

The next step is to instantiate the main theorem for problems about subsequences. In the appendix, we have stated a 'categorical' version of the corollary, and we have spelled out the proof in detail. Here we just state the result as it is applied in practice.

Corollary 9. *Assume that R is a preorder, and suppose p is prefix–closed with derivative q and satisfies*

$$q\,(x, a) \Rightarrow (x +\!\!\!\prec a)(R)x \ .$$

Furthermore, assume that

$$\begin{aligned} q(y, a) \wedge q(x, a) &\Rightarrow (y +\!\!\!\prec a)(R)(x +\!\!\!\prec a) \\ \neg\, q(y, a) \wedge q(x, a) &\Rightarrow y(R)(x +\!\!\!\prec a), \end{aligned}$$

for all x and y satisfying

$$y(minR \cdot p\triangleleft \cdot subs)z \ \wedge \ x(\in \cdot p\triangleleft \cdot subs)z$$

for some z. Then $([\nu, \oplus]) \subseteq minR \cdot p\triangleleft \cdot subs$, where $x \oplus a = x +\!\!+\!\!< a$ if $q(x, a)$, and x otherwise.

For the spanning tree problem, we have that the predicate *acyclic* is prefix–closed with derivative q, where $q(x, a)$ holds if a does not create a cycle when added to x. The first condition on R holds because $\#y > \#x \Rightarrow y(R)x$. The second condition holds because $y(R)x \Rightarrow (y +\!\!+\!\!< a)(R)(x +\!\!+\!\!< a)$. This leaves us with the last condition. Let x, y, and z be as stated in the corollary, so y is some minimum cost spanning tree of z and x is some acyclic subsequence of z with $y(R)x$. Suppose that $x +\!\!+\!\!< a$ is acyclic, but $y +\!\!+\!\!< a$ contains a cycle. Since y is acyclic, $y +\!\!+\!\!< a$ contains a cycle of the form

$$\{(v_0, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_n)\},$$

where $a = (v_0, v_n)$. Because $x +\!\!+\!\!< a$ is acyclic, there exists an edge $b = (v_{i-1}, v_i)$ in y which is not connected in x. Furthermore, *cost* $b \leq$ *cost* a since the graph is given as a sequence of edges in ascending order of cost. As both x and y are subsequences of z, and b is in y, it follows that there exist x_1 and x_2 with $x = x_1 +\!\!+ x_2$ and

$$(x_1 +\!\!+ [b] +\!\!+ x_2)(\in \cdot p\triangleleft \cdot subs)z.$$

Finally, we have

$$y(R)(x_1 +\!\!+ [b] +\!\!+ x_2) \ \text{ and } \ (x_1 +\!\!+ [b] +\!\!+ x_2)(R)(x_1 +\!\!+ x_2 +\!\!+ [a])$$

so $y(R)(x +\!\!+\!\!< a)$. The the corollary is therefore applicable, and the result is Kruskal's algorithm.

The spanning tree problem illustrates the strength of our claim that the foregoing abstract theory gives clear guidance on the difficult proof obligations that have to be met to obtain efficient algorithms. Indeed, apart from the last condition, the rest of the development consists essentially of instantiating the theory to problems about subsequences. The last condition is less mechanical and is the crucial property in the derivation of Kruskal's algorithm. (It is, in fact, the verification of the *matroid* property [7].) But this does not invalidate the claim, because the goal of our research is to provide *organising principles* for algorithm design, not necessarily to *mechanise* the whole design process.

We should emphasise that the main theorem has many more applications besides algorithms on subsequences. The papers by Jeuring [5, 6] discuss applications to subtrees, permutations and partitions. In a forthcoming paper [2], we review some of those applications, and show how the present work can be extended to include certain dynamic programming algorithms.

References

1. R.C. Backhouse, P. de Bruin, G. Malcolm, T.S. Voermans, and J. van der Woude. Relational catamorphisms. In B. Möller, editor, *Proceedings of the IFIP TC2/WG2.1 Working Conference on Constructing Programs*, pages 287–318. Elsevier Science Publishers B.V., 1991.
2. R.S. Bird and O. de Moor. Inductive solutions to optimisation problems. Draft, 1991.
3. S. Eilenberg and J.B. Wright. Automata in general algebras. *Information and Control*, 11(4):452–470, 1967.
4. P.J. Freyd and A. Ščedrov. *Categories, Allegories*, volume 39 of *Mathematical Library*. North–Holland, 1990.
5. J. Jeuring. Deriving algorithms on binary labelled trees. In P.M.G. Apers, D. Bosman, and J. van Leeuwen, editors, *Proceedings SION Computing Science in the Netherlands*, pages 229–249, 1989.
6. J. Jeuring. Algorithms from theorems. In M. Broy and C.B. Jones, editors, *Programming Concepts and Methods*, pages 247–266. North–Holland, 1990.
7. B. Korte, L. Lovasz, and R. Schrader. *Greedoids*, volume 4 of *Algorithms and combinatorics*. Springer–Verlag, 1991.
8. G. Malcolm. Data structures and program transformation. *Science of Computer Programming*, 14:255–279, 1990.
9. O. de Moor. Categories, relations and dynamic programming. D.Phil. thesis. Technical Monograph PRG-98, Computing Laboratory, Oxford, 1992.
10. J.C.S.P. van der Woude. Free style spec wrestling ii: Preorders. *The Squiggolist*, 2(2):48–53, 1991.

Appendix: Instantiation for Subsequences

Here we show how the main theorem may be instantiated for problems about subsequences. The proofs are spelled out in great detail, to support our claim that the instantiation is largely mechanical. Throughout, F stands for the functor

$$\begin{aligned} \mathsf{F}A &= \mathbf{1} + (A \times E) \\ \mathsf{F}f &= id + (f \times id) \ , \end{aligned}$$

and α abbreviates the initial algebra $[\nu, +\!\!\!\prec] : \mathsf{F}L \to L$, where L is the type of lists over E.

Recall that $subs = (\!|\nu, +\!\!\!\prec \cup \pi_1|\!)$. It is convenient to rewrite this in the form

$$subs = \Lambda(\!|\alpha \cup \delta|\!) \ ,$$

where $\delta = [0, \pi_1]$, and 0 is the empty relation. We have

$$[R, S] \cup [U, V] = [R \cup U, S \cup V] \ ,$$

so the two expressions for *subs* are equivalent. The advantage of naming δ is that we can refer to its algebraic properties. In particular, we have that δ is a collection of partial functions

$$\delta \cdot \delta^\circ \subseteq id \ ,$$

which furthermore satisfies

$$\delta \cdot \mathsf{F}Q = Q \cdot \delta \ .$$

The second property asserts that δ is a natural transformation. Both of these inclusions are straightforward to verify.

Our first task is to express $p\triangleleft \cdot subs$ in the form $p\triangleleft \cdot subs = \Lambda(\!|S|\!)$ for some S. Recall Corollary 1 of the promotion theorem which says that

$$p\triangleleft \cdot \Lambda(\!|R|\!) = \Lambda(\!|S|\!) \Leftarrow p? \cdot R = S \cdot \mathsf{F}p? \ .$$

Also, recall that p is prefix–closed with derivative q if

$$\begin{aligned} p? \cdot \nu &= \nu \\ p? \cdot +\!\!\!\prec &= +\!\!\!\prec \cdot q? \cdot (p? \times id) \ . \end{aligned}$$

These two identities can also be formulated as a single equation, namely

$$p? \cdot \alpha = \alpha \cdot r? \cdot \mathsf{F}p?$$

with $r = [true, q]$. Assuming the latter equation, we reason:

$$\begin{array}{ll} & p? \cdot (\alpha \cup \delta) \\ = & \quad \{\text{composition over union}\} \\ & (p? \cdot \alpha) \cup (p? \cdot \delta) \\ = & \quad \{\text{assumption: } p? \cdot \alpha = \alpha \cdot r? \cdot \mathsf{F}p? \ \} \\ & (\alpha \cdot r? \cdot \mathsf{F}p?) \cup (p? \cdot \delta) \\ = & \quad \{\text{naturality of } \delta\} \\ & (\alpha \cdot r? \cdot \mathsf{F}p?) \cup (\delta \cdot \mathsf{F}p?) \\ = & \quad \{\text{composition over union}\} \\ & ((\alpha \cdot r?) \cup \delta) \cdot \mathsf{F}p? \end{array}$$

Hence by Corollary 1,

$$p\triangleleft \cdot subs = \Lambda([\![(\alpha \cdot r?) \cup \delta]\!]) \ .$$

Note that $(\alpha \cdot r?) \cup \delta = [\nu, (+\!\!\!\prec \cdot q?) \cup \pi_1]$, giving the result cited in Section 5.

The next task is to expand the minotonicity condition

$$f \cdot \mathsf{F}(minR \cdot \Box\Lambda([\![S]\!]) \cdot \ni) \subseteq R \cdot f$$

of the main theorem, in the case $S = (\alpha \cdot r?) \cup \delta$ and $f = [\nu, \oplus]$, where

$$x \oplus a = \begin{cases} x +\!\!\!\prec a & \text{if} \quad q(x,a) \\ x & \text{otherwise} \end{cases}$$

We can rewrite f using the McCarthy conditional $(p \to R, S)$ defined by

$$(p \to R, S) = (R \cdot p?) \cup (S \cdot \neg p?) \ ,$$

where $\neg p? = (\neg p)?$. Recall that r was defined by $r = [true, q]$, so we have

$$f = (r \to \alpha, \delta) \ .$$

Writing $T = minR \cdot \Box\Lambda([\![(\alpha \cdot r?) \cup \delta]\!]) \cdot \ni$, we need conditions under which

$$(r \to \alpha, \delta) \cdot \mathsf{F}\,T \subseteq R \cdot (r \to \alpha, \delta) \tag{1}$$

holds. We can rewrite (1) using the following two properties of conditionals

$$\begin{array}{c} R \cdot (p \to S, T) = (p \to R \cdot S, R \cdot T) \\ R \subseteq (p \to S, T) \equiv (R \cdot p? \subseteq S) \wedge (R \cdot \neg p? \subseteq T) \ . \end{array}$$

Now, inclusion (1) translates to

$$\begin{array}{c} (r \to \alpha, \delta) \cdot \mathsf{F}\,T \cdot r? \subseteq R \cdot \alpha \\ (r \to \alpha, \delta) \cdot \mathsf{F}\,T \cdot \neg r? \subseteq R \cdot \delta \ . \end{array}$$

Using the definition of conditions, these inclusions expand to four more:

$$\begin{aligned}
\alpha \cdot r? \cdot \mathsf{F}\, T \cdot r? &\subseteq R \cdot \alpha \\
\delta \cdot \neg r? \cdot \mathsf{F}\, T \cdot r? &\subseteq R \cdot \alpha \\
\alpha \cdot r? \cdot \mathsf{F}\, T \cdot \neg r? &\subseteq R \cdot \delta \\
\delta \cdot \neg r? \cdot \mathsf{F}\, T \cdot \neg r? &\subseteq R \cdot \delta \ .
\end{aligned}$$

We shall assume that the first two conjuncts are satisfied. Given that $\alpha \cdot r? \subseteq R \cdot \delta$, we can prove the last two:

$$\begin{aligned}
& \alpha \cdot r? \cdot \mathsf{F}\, T \cdot \neg r? \\
\subseteq \quad & \{\text{since } \neg r? \subseteq id\} \\
& \alpha \cdot r? \cdot \mathsf{F}\, T \\
\subseteq \quad & \{\text{assumption: } \alpha \cdot r? \subseteq R \cdot \delta\} \\
& R \cdot \delta \cdot \mathsf{F}\, T \\
= \quad & \{\text{naturality of } \delta\} \\
& R \cdot T \cdot \delta \\
\subseteq \quad & \{\text{since } T \subseteq R \text{ (Section 6)}\} \\
& R \cdot R \cdot \delta \\
\subseteq \quad & \{\text{assumption: } R \text{ transitive}\} \\
& R \cdot \delta \ .
\end{aligned}$$

The proof of the last inclusion is:

$$\begin{aligned}
& \delta \cdot \neg r? \cdot \mathsf{F}\, T \cdot \neg r? \\
\subseteq \quad & \{\text{since } \neg r? \subseteq id\} \\
& \delta \cdot \mathsf{F}\, T \\
= \quad & \{\text{naturality of delta}\} \\
& T \cdot \delta \\
\subseteq \quad & \{\text{since } T \subseteq R \text{ (Section 6)}\} \\
& R \cdot \delta \ .
\end{aligned}$$

This completes the instantiation of minotonicity.

Finally we need to expand the hypothesis of the main theorem, namely

$$f \subseteq minR \cdot \Lambda S$$

where $f = (r \to \alpha, \delta)$ and $S = (\alpha \cdot r?) \cup \delta$. Using the definition of conditionals, the hypothesis reads

$$\begin{aligned}
\alpha \cdot r? &\subseteq minR \cdot \Lambda((\alpha \cdot r?) \cup \delta) \\
\delta \cdot \neg r? &\subseteq minR \cdot \Lambda((\alpha \cdot r?) \cup \delta) \ .
\end{aligned}$$

To establish these results, we use the following equivalence (which was not needed in the main text), whose proof we omit

$$U \subseteq min R \cdot \Lambda V \equiv (U \subseteq V) \wedge (U \cdot V^\circ \subseteq R) \ .$$

The hypothesis is therefore established if we can show

$$\begin{aligned} \alpha \cdot r? &\subseteq (\alpha \cdot r?) \cup \delta \\ \delta \cdot \neg r? &\subseteq (\alpha \cdot r?) \cup \delta \\ \alpha \cdot r? \cdot ((\alpha \cdot r?) \cup \delta)^\circ &\subseteq R \\ \delta \cdot \neg r? \cdot ((\alpha \cdot r?) \cup \delta)^\circ &\subseteq R \ . \end{aligned}$$

The first inclusion is immediate, and the second follows from $\neg r? \subseteq id$. For the third we reason

$$\begin{aligned} & \alpha \cdot r? \cdot ((\alpha \cdot r?) \cup \delta)^\circ \\ \subseteq \quad & \{\text{since } \alpha \cdot r? \text{ is a partial function}\} \\ & id \cup (\alpha \cdot r? \cdot \delta^\circ) \\ \subseteq \quad & \{\text{assumption: } \alpha \cdot r? \subseteq R \cdot \delta\} \\ & id \cup (R \cdot \delta \cdot \delta^\circ) \\ \subseteq \quad & \{\text{since } \delta \text{ is a partial function}\} \\ & id \cup R \\ = \quad & \{\text{assumption: } R \text{ reflexive}\} \\ & R \ . \end{aligned}$$

This proves the first inequation. For the second, we reason:

$$\begin{aligned} & \delta \cdot \neg r? \cdot ((\alpha \cdot r?) \cup \delta)^\circ \\ = \quad & \{\text{relation calculus}\} \\ & (\delta \cdot \neg r? \cdot r?^\circ \cdot \alpha^\circ) \cup (\delta \cdot \neg r? \cdot \delta^\circ) \\ = \quad & \{\text{since } \neg r? \cdot r?^\circ = 0\} \\ & \delta \cdot \neg r? \cdot \delta^\circ \\ \subseteq \quad & \{\text{since } \neg r? \subseteq id\} \\ & \delta \cdot \delta^\circ \\ \subseteq \quad & \{\text{since } \delta \text{ is a partial function}\} \\ & id \\ \subseteq \quad & \{\text{assumption: } R \text{ reflexive}\} \\ & R \ . \end{aligned}$$

This completes the instantiation of the hypothesis in the main theorem.

Let us now summarize the conditions. We have assumed R is reflexive and transitive, and so a preorder. We have supposed $p? \cdot \alpha = \alpha \cdot \mathsf{F}p? \cdot r?$, which is true when

p is prefix-closed with derivative q, for then one can take $r = [\mathit{true}, q]$. We have also supposed that

$$\begin{aligned} \alpha \cdot r? &\subseteq R \cdot \delta \\ \alpha \cdot r? \cdot \mathsf{F}\, T \cdot r? &\subseteq R \cdot \alpha \\ \delta \cdot r? \cdot \mathsf{F}\, T \cdot \neg r? &\subseteq R \cdot \alpha \ . \end{aligned}$$

With $r = [\mathit{true}, q]$, these are precisely the three conditions enunciated in Corollary 2 of the main theorem.

This article was processed using the LaTeX macro package with LLNCS style