# From Dynamic Programming to Greedy Algorithms

*Richard Bird and Oege de Moor*⋆

Programming Research Group
11 Keble Road
Oxford OX1 3QD
United Kingdom

**Abstract**
A calculus of relations is used to reason about specifications and algorithms for optimisation problems. It is shown how certain greedy algorithms can be seen as refinements of dynamic programming. Throughout, the *maximum lateness* problem is used as a motivating example.

## 1 Introduction

An optimisation problem can be solved by dynamic programming if an optimal solution is composed of optimal solutions to subproblems. This property, which is known as the *principle of optimality*, can be formalised as a monotonicity condition. If the principle of optimality is satisfied, one can compute a solution by decomposing the input in all possible ways, recursively solving the subproblems, and then combining optimal solutions to subproblems into an optimal solution for the whole problem. By contrast, a *greedy* algorithm considers only one decomposition of the argument. This decomposition is usually unbalanced, and greedy in the sense that at each step the algorithm reduces the input as much as possible. If the decomposition has a more balanced character, the algorithm is commonly classified as an instance of the divide–and–conquer paradigm.

Certain greedy algorithms can be seen as refinements of dynamic programming. If the principle of optimality is satisfied, dynamic programming is applicable, and if an additional monotonicity condition is satisfied, then we can narrow the choice of decompositions to a single candidate, thus obtaining a greedy algorithm. This idea was formalised in [3] for a restricted class of optimisation problems, namely those involving list partitions. Although it was suggested that the technique might be more widely applicable, it proved difficult to formulate the general theorem in the framework of that paper. The conclusion was that a more flexible framework was needed, and this observation motivated one of us to undertake an in–depth study of dynamic programming in a categorical setting [15]. The present paper summarises the results on dynamic programming, and shows how they can be extended to a theory of greedy algorithms.

The structure of the paper is as follows. First we study a typical application of the greedy strategy, the so–called *maximum lateness* problem from operations research. This example serves to explain the objectives of the paper and provides motivation for the

⋆ Research supported by a studentship from British Petroleum International.

subsequent calculus. After introducing this calculus, we show how dynamic programming applies to the problem. We then go on to prove an abstract result about greedy algorithms, and show how a greedy algorithm can be derived for our example. We conclude with a brief discussion of the implications of the research.

## 2 Example: Maximum Lateness

Maximum Lateness is a scheduling problem: given a bag $x$ of jobs, we want to find a permutation $y$, called a schedule, of $x$ that minimises a certain function $c$, called the cost function. This cost function returns the so-called *maximum lateness* of a schedule, and this explains the name of the problem [7]. The maximum lateness problem is specified as a relation $mt$, where

$$mt = min(c) \cdot perms.$$

The function *perms* returns the set of all permutations of a bag. That is,

$$perms\ y = \{\, x \mid bagify\ x = y \,\},$$

where *bagify* is the function that turns a sequence of jobs into a bag. The relation $min(c)$ holds between a set of schedules and those of minimum cost $c$:

$$y\,(min(c))\ ys = y \in ys\ \wedge\ (\forall z \in ys\ :\ c\,y \leq c\,z).$$

The cost function $c$ returns the *maximum lateness* of a schedule. With each job $a$ are associated three nonnegative quantities: a processing time $time\ a$, a due time $due\ a$, and a weight $weight\ a$. The processing time is a measure of the relative time it takes to complete a job. The due time gives the absolute time when a job should be finished. Finally, the weight of a job indicates the importance of finishing this job in time. An important job has a high weight, and an unimportant one a low weight.

The cost function is defined by the following equations (we write $(+\!\!+)$ for concatenation of sequences and $[a]$ for the singleton sequence with element $a$)

$$\begin{aligned} c\,y &= max\,\{\, lateness\ z \mid \exists v : z +\!\!+ v = y \wedge z \neq [\,] \,\} \\ lateness\,(z +\!\!+ [a]) &= weight\ a \times (length\,(z +\!\!+ [a]) - due\ a) \\ length\ z &= \sum_{a \in z} (time\ a). \end{aligned}$$

In words, the length of a (partial) schedule is the total time taken to complete it, which is the sum of the individual processing times. The lateness of a job $a$ coming after a partial schedule $z$ is a weighted measure of the time by which $length\,(z +\!\!+ [a])$ exceeds the due time of $a$ (a negative quantity if $a$ is completed before it is due). Finally, the cost of a schedule is the maximum, taken over all nonempty prefixes $z$ of the schedule, of the lateness associated with $z$.

We can also define the cost $c$ recursively by the equations (writing $[\,]$ for the empty sequence)

$$\begin{aligned} c\,[\,] &= -\infty \\ c(y \mathbin{+\!\!+} [a]) &= c\;y \sqcup ((\mathit{bagify}\;y) \otimes a) \\ y \otimes a &= \mathit{wt}\;a \times (\sum_{b \in y} (\mathit{time}\;b) + \mathit{time}\;a - \mathit{due}\;a). \end{aligned}$$

Here $(\sqcup)$ stands for the binary operator that returns the maximum of its arguments. We have defined $y \otimes a$ for a bag $y$ rather than a list, because the order of elements is unimportant. This observation will be useful when we derive an efficient algorithm for $mt$.

*An example.* The table below displays an instance of the maximum lateness problem. There are four jobs given, named $a$, $b$, $c$ and $d$. The respective values of *time*, *due* and *weight* are given in the subsequent columns. The minimum cost of a schedule is 20, and this minimum is realized by two schedules: *bcda* and *cbda*.

| *name* | *time* | *due* | *weight* |
|---|---|---|---|
| $a$ | 10 | 13 | 4 |
| $b$ | 3 | 3 | 5 |
| $c$ | 1 | 2 | 6 |
| $d$ | 4 | 5 | 3 |

*Dynamic programming solution.* A typical dynamic programming solution for the maximum lateness problem is given by

$$mt\;x = \begin{cases} [\,], & \text{if } x = \langle\,\rangle \\ \mathit{min}(c)\{mt\;y \mathbin{+\!\!+} [a] \mid y + \langle a \rangle = x\}, & \text{otherwise} \end{cases}$$

Here we use $\langle\,\rangle$ to denote the empty bag, $\langle a \rangle$ for the singleton bag with element $a$, and $+$ for bag union. This dynamic programming solution for $mt$ takes time exponential in the size of $x$, even when the recursive calls of $mt$ are tabulated. Admittedly, this description of dynamic programming is informal, since the notation suggests that $mt$ is a function while it is really a relation. To give a rigorous formulation, we shall need various concepts from the relational calculus, which will be introduced in Section 3 below.

*Greedy solution.* The greedy algorithm for maximum lateness can be described by the following recursion equation:

$$mt\;x = \begin{cases} [\,], & \text{if } x = \langle\,\rangle \\ mt\;y \mathbin{+\!\!+} [a], & \text{otherwise.} \\ \text{where } (y, a) = \mathit{split}\;x. & \end{cases}$$

The expression $\mathit{split}\;x$ yields a pair $(y, a)$, consisting of a bag $y$ and an element $a$ such that $y + \langle a \rangle = x$. Furthermore, $y$ and $a$ are chosen to minimise the value of $y \otimes a$. In this sense, the algorithm is greedy: *split* finds an optimal splitting. A straightforward implementation of the greedy algorithm takes cubic time: it takes quadratic time to find the optimum split, because there is a linear number of splits $(y, a)$, and one may compute $y \otimes a$ in linear time. Using well-known program transformations, this naive program can be transformed into a quadratic time program, thus obtaining Lawler's algorithm [10]. There is a yet more efficient implementation of the same greedy strategy, which only requires $\mathcal{O}(n \log^2 n)$ computation steps [7].

The general questions we are interested in are these: how is the dynamic programming solution derived from the initial problem statement, and what extra conditions are necessary to ensure that a greedy algorithm also solves the problem? To answer these questions we need a calculus of relations suitable for expressing and manipulating specifications of optimisation problems.

## 3 A Calculus of Relations

This section gives a brief introduction to a calculus of relations designed for the purpose of solving optimisation problems [15]. The exposition makes use of some elementary notions from category theory, namely category, functor, terminal object, product, coproduct, and algebra for an endofunctor. Readers not familiar with this material can find it (for example) in the textbook by Barr and Wells [2]. There are also a number of introductions that focus on applications to program derivation [6, 11, 12, 14, 17]; these are especially suited as background for the present paper.

### 3.1 Relations

Sets are denoted by upper case identifiers: $A, B, C$. A *relation* between $A$ and $B$ is a subset of the cartesian product $A \times B$ and we write $a(R)b$ as shorthand for $(a, b) \in R$. The category *Rel* of sets and relations has sets as objects and relations as arrows. We write $R : A \leftarrow B$ for a relation to $A$ from $B$. The set $A$ is called the *target* of $R$ and $B$ the *source*. Composition of relations is defined by

$$a(R \cdot S)c \equiv (\exists b : a(R)b \ \wedge \ b(S)c),$$

and the converse $R^\circ$ by $b(R^\circ)a \equiv a(R)b$. A relation $R : A \leftarrow B$ is said to be *simple* if $R \cdot R^\circ \subseteq id_A$, and *entire* if $id_B \subseteq R^\circ \cdot R$. Simple relations are also known as imps, partial functions or partial maps; and entire relations are also called total relations. A relation is a *function* if it is both entire and simple. Functions will be denoted by lower case identifiers. The two inequations which state that $f$ is a function can also be phrased as an equivalence: $f$ is a function if and only if

$$(R \cdot f^\circ \subseteq S) \equiv (R \subseteq S \cdot f)$$

for all $R$ and $S$. Equivalently, $f$ is a function if and only if

$$(f \cdot R \subseteq S) \equiv (R \subseteq f^\circ \cdot S)$$

for all $R$ and $S$. We shall refer to these equivalences as the *shunting* rules for functions. Sets and functions form a subcategory *Fun* of *Rel*.

*Intersection and union.* Given two relations $R, S : A \leftarrow B$ the intersection $(R \cap S)$ is defined by the equivalence

$$T \subseteq (R \cap S) \equiv (T \subseteq R) \ \wedge \ (T \subseteq S).$$

In other words, $R \cap S$ is the greatest lower bound of $R$ and $S$. Intersection and converse are related by the so–called *modular law*,

$$(R \cdot S) \cap T \subseteq (R \cap (T \cdot S^{\circ})) \cdot S,$$

which is also known as Dedekind's rule. An importance consequence is that composition with simple relations distributes over intersection:

$$(R \cap T) \cdot S = (R \cdot S) \cap (T \cdot S), \quad \text{provided } S \text{ is simple.}$$

The inclusion $(\subseteq)$ is an instance of monotonicity, and the containment $(\supseteq)$ follows from the modular law and simplicity:

$$(R \cdot S) \cap (T \cdot S) \subseteq (R \cap (T \cdot S \cdot S^{\circ})) \cdot S \subseteq (R \cap T) \cdot S.$$

The union $R \cup S$ of two relations $R, S : A \leftarrow B$ is their least upper bound:

$$R \cup S \subseteq T \equiv (R \subseteq T) \ \wedge \ (S \subseteq T).$$

In contrast to intersection, we have

$$(R \cup T) \cdot S = (R \cdot S) \cup (T \cdot S)$$

without any restriction on $S$.

*Knaster-Tarski.* For any two sets $A$ and $B$, the relations $A \leftarrow B$ form a complete lattice. We can therefore appeal to the well-known theorem of Knaster and Tarski for solving recursion equations. A modern proof of this theorem can be found in [5].

**Theorem 1.** *Let $(\mathcal{L}, \leq)$ be a complete lattice, and let $\phi : \mathcal{L} \leftarrow \mathcal{L}$ be a monotonic function. Then the equation $\phi\, x = x$ has a least solution which is also the least solution of $\phi\, x \leq x$.*

*Quotient.* Suppose $R : A \leftarrow B$ and $S : C \leftarrow B$ are relations with a common source. Then the quotient $R/S : A \leftarrow C$ is defined by the equivalence

$$(T \subseteq R/S) \equiv (T \cdot S \subseteq R).$$

One can also construct $R/S$ explicitly:

$$c(R/S)b = (\forall a : b(S)a \Rightarrow c(R)a).$$

Using the characterisation of quotients and the shunting rules for functions, we get

$$T \subseteq (R/S) \cdot f \ \equiv \ T \cdot f^{\circ} \subseteq R/S \ \equiv \ T \cdot f^{\circ} \cdot S \subseteq R \ \equiv \ T \subseteq R/(f^{\circ} \cdot S).$$

Hence $(R/S) \cdot f = R/(f^{\circ} \cdot S)$. Similar reasoning gives $f^{\circ} \cdot (R/S) = (f^{\circ} \cdot R)/S$.

*Powersets.* The representation of a relation $R : A \leftarrow B$ as a subset of the cartesian product $A \times B$ is the traditional one, but it is also possible to consider a relation as a set–valued function $\Lambda R : \mathsf{P}A \leftarrow B$, where $\mathsf{P}A$ denotes the power set of $A$. The isomorphism between relations and set–valued functions is described by the equivalence

$$(f = \Lambda R) \equiv (\in \cdot f = R),$$

where the function $\Lambda R$, called the *power transpose* of $R$, is defined by

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

and $\in : B \leftarrow \mathsf{P}B$ is the membership relation. Various useful identities can be derived from the above equivalence. For instance, by taking $f = id$ and $R = \in$ in the right-hand side, one finds that

$$\Lambda\!\in\, = id.$$

By taking $f = \Lambda R$ in the left-hand side, we obtain

$$\in \cdot \Lambda R = R.$$

### 3.2 Minimum Elements

For $R : A \leftarrow A$, the relation $min(R) : A \leftarrow \mathsf{P}A$ is defined by

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

where $\ni$ denotes the converse of $\in$. In words, $a(min(R))x$ if $a$ is an element of $x$, and for all $b$, if $b \in x$, then $a(R)b$. We can define $max(R) = min(R^\circ)$, so the restriction to minimum elements is not important. We will need various properties of $min(R)$, the first of which is

$$min(R) \cdot \Lambda S = S \cap R/S^\circ.$$

Note that this specializes to the definition of $min$ when $S = \in$, because $\Lambda\!\in\, = id$. This result may be proved as follows

$$\begin{array}{cl} & min(R) \cdot \Lambda S \\ = & \quad \{\text{definition of } min(R)\} \\ & (\in \cap\, R/\ni) \cdot \Lambda S \\ = & \quad \{\text{since } \Lambda S \text{ is simple}\} \\ & (\in \cdot \Lambda S) \cap (R/\ni) \cdot \Lambda S \\ = & \quad \{\text{since } \in \text{ cancels } \Lambda\} \\ & S \cap (R/\ni) \cdot \Lambda S \\ = & \quad \{\text{quotient, } \Lambda S \text{ function}\} \\ & S \cap R/((\Lambda S)^\circ \cdot \ni) \\ = & \quad \{\text{converse and } \in \text{ cancels } \Lambda\} \\ & S \cap R/S^\circ. \end{array}$$

A useful fact in applications is the following result which says that we can always constrain $min(R)$ to take account of context.

**Proposition 2.** *If $S$ is simple, then $min(R) \cdot S = min(R') \cdot S$, where $R' = (\in \cdot S) \cdot (\in \cdot S)^\circ \cap R$.*

**Proof**. Observe that if $S$ is simple, then (by left-distributivity of $(\cdot S)$ over $\cap$)

$$\begin{aligned} min(R) \cdot S &= \in \cdot S \cap (R/\ni) \cdot S \\ min(R') \cdot S &= \in \cdot S \cap ((\in \cdot S) \cdot (\in \cdot S)^\circ)/\ni \cdot S \cap (R/\ni) \cdot S \end{aligned}$$

It is therefore sufficient to show that

$$\in \cdot S \subseteq ((\in \cdot S) \cdot (\in \cdot S)^\circ)/\ni \cdot S.$$

For any $S$ we have (by the modular law) $S \subseteq S \cdot S^\circ \cdot S$, and therefore $\in \cdot S \subseteq \in \cdot S \cdot S^\circ \cdot S$. Since (by the definition of quotients)

$$\in \cdot S \cdot S^\circ \subseteq ((\in \cdot S) \cdot (\in \cdot S)^\circ)/\ni,$$

the claim is established.

**Corollary 3.** *We have*

$$min(R) \cdot \Lambda S^\circ = min(S^\circ \cdot S \cap R) \cdot \Lambda S^\circ.$$

*Moreover, $S^\circ \cdot S \cap R$ is transitive if $R$ is transitive and $S$ is simple.*

**Proof**. The first part is immediate since $\Lambda S^\circ$ is simple and $\in \cdot \Lambda S^\circ = S^\circ$. For the second part we argue

$$\begin{array}{ll} & (S^\circ \cdot S \cap R) \cdot (S^\circ \cdot S \cap R) \\ \subseteq & \{\text{monotonicity}\} \\ & S^\circ \cdot S \cdot S^\circ \cdot S \cap R \cdot R \\ \subseteq & \{\text{since } S \text{ is simple}\} \\ & S^\circ \cdot S \cap R \cdot R \\ \subseteq & \{\text{since } R \text{ is transitive}\} \\ & S^\circ \cdot S \cap R. \end{array}$$

### 3.3 Relators

The class of monotonic functors $Rel \leftarrow Rel$ plays a fundamental role in the calculus of relations. A functor $\mathsf{F} : Rel \leftarrow Rel$ is monotonic if $R \subseteq S$ implies $\mathsf{F}R \subseteq \mathsf{F}S$. The following theorem states the most important properties of monotonic functors. A detailed proof can be found in the paper by Carboni, Kelly and Wood [4].

**Proposition 4.** *Suppose that $\mathsf{F} : Rel \leftarrow Rel$ is monotonic. Then*

- $\mathsf{F}$ *preserves functions,* i.e. *for all $f$ in Fun, $\mathsf{F}f$ is a function.*
- $\mathsf{F}$ *preserves converse,* i.e. $\mathsf{F}(R^\circ) = (\mathsf{F}R)^\circ$.

– $\mathsf{F}$ *is determined by its action on functions. That is, if* $\mathsf{G} : Rel \leftarrow Rel$ *is another monotonic functor, the statement*

$$\mathsf{F}f = \mathsf{G}f \quad \textit{for all } f \textit{ in } Fun$$

*is equivalent to* $\mathsf{F} = \mathsf{G}$.

In words, the last property says that each monotonic functor is the unique extension of some functor $\mathsf{F} : Fun \leftarrow Fun$ to relations. We will use the same letter $\mathsf{F}$ to denote both a functor on *Fun* and its extension to *Rel*. When a functor on *Fun* has an extension to relations, it is said to be a *relator*.

**Proposition 5.** *A functor* $\mathsf{F} : Fun \leftarrow Fun$ *is a relator if and only if the following condition is satisfied:* $f \cdot g^\circ = h^\circ \cdot k$ *implies* $\mathsf{F}f \cdot (\mathsf{F}g)^\circ = (\mathsf{F}h)^\circ \cdot \mathsf{F}k$.

Most functors that occur in programming problems are relators, and we now consider some examples.

*Product.* The extension of the product functor $\times : Fun \leftarrow (Fun \times Fun)$ is given by

$$R \times S = (\pi_1^\circ \cdot R \cdot \pi_1) \cap (\pi_2^\circ \cdot S \cdot \pi_2)$$

where $\pi_1 : A \leftarrow A \times B$ and $\pi_2 : B \leftarrow A \times B$ are the left and right projection functions. However, the extended product $\times$ does *not* define a categorical product in *Rel*, so here the decision to use the same notation for the extension to relations is misleading.

*Coproduct.* The coproduct functor $+ : Fun \leftarrow (Fun \times Fun)$ also extends to relations; we have

$$R + S = \iota_1 \cdot R \cdot \iota_1^\circ \cup \iota_2 \cdot S \cdot \iota_2^\circ,$$

where $\iota_1 : A + B \leftarrow A$ and $\iota_2 : A + B \leftarrow B$ are the coproduct injections. Unlike product, $+$ does define a categorical coproduct in *Rel*, so the use of the same notation is harmless. Since *Rel* is isomorphic to its own opposite, $+$ also defines a product in *Rel*.

*List.* The list functor $\mathsf{L} : Fun \leftarrow Fun$ takes a set $A$ and returns the set $A^*$ of all finite sequences with elements from $A$. On arrows, $(\mathsf{L}f)$ is the function that applies $f$ to all elements of a sequence:

$$\mathsf{L}f\,[a_1, a_2, \ldots, a_n] = [f\ a_1, f\ a_2, \ldots, f\ a_n].$$

The extension of $\mathsf{L}$ to relations is defined by

$$\begin{array}{l}[a_1, a_2, \ldots, a_n](\mathsf{L}R)[b_1, b_2, \ldots, b_m] = \\ \quad (n = m) \ \wedge\ (\forall i \,:\, 1 \le i \le n \,:\, a_i(R)b_i).\end{array}$$

*Powerset.* Finally, consider the covariant powerset functor $\mathsf{P} : \mathit{Fun} \leftarrow \mathit{Fun}$ that sends a function to its existential image. Here, $\mathsf{P}A$ is the powerset of $A$ and $(\mathsf{P}f)\,x = \{f\ a \mid a \in x\}$. The extension of $\mathsf{P}$ to relations is defined by

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

Note that $\mathsf{P} : \mathit{Rel} \leftarrow \mathit{Rel}$ is *not* the same as the existential image functor $\mathsf{E} : \mathit{Rel} \leftarrow \mathit{Rel}$ defined by $\mathsf{E}A = \mathsf{P}A$ and

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

The functor $\mathsf{P}$ returns relations and is monotonic, while $\mathsf{E}$ returns functions and is not monotonic. Since $\mathsf{P}$ and $\mathsf{E}$ coincide on functions, we may conclude that the restriction to monotonic functors in proposition 4 is necessary.

Since for every $A$ we have a relation $\in : A \leftarrow \mathsf{P}A$, one might expect it to be some sort of natural transformation, both with respect to $\mathsf{E}$ and $\mathsf{P}$. Indeed, we have

$$\in \cdot\, \mathsf{E}\,R = R \cdot \in \ \text{ and } \in \cdot\, \mathsf{P}\,R \subseteq R \cdot \in.$$

As an application of these facts, we prove a technical proposition that will be useful in later proofs. It states a rule for eliminating *min*, $\mathsf{P}$ and $\Lambda$.

**Proposition 6.**

$$\mathit{min}(R) \cdot \mathsf{P}S \cdot \Lambda\, T^{\circ} \cdot T \subseteq R \cdot S$$

**Proof**. First observe that

$$\begin{array}{ll} & \Lambda\, T^{\circ} \cdot T \subseteq \ni \\ \equiv & \{\Lambda\, T^{\circ} \text{ function, shunting}\} \\ & T \subseteq (\Lambda\, T^{\circ})^{\circ} \cdot \ni \\ \equiv & \{\text{converse}\} \\ & T \subseteq (\in \cdot\, \Lambda\, T^{\circ})^{\circ} \\ \equiv & \{\in \text{ cancels } \Lambda\} \\ & T \subseteq (T^{\circ})^{\circ} \\ \equiv & \{\text{converse is an involution}\} \\ & \mathit{true}. \end{array}$$

Using this auxiliary result, we can prove the proposition:

$$\begin{array}{ll} & \mathit{min}(R) \cdot \mathsf{P}S \cdot \Lambda\, T^{\circ} \cdot T \\ \subseteq & \{\text{above}\} \\ & \mathit{min}(R) \cdot \mathsf{P}S \cdot \ni \end{array}$$

$$
\begin{array}{ll}
\subseteq & \{\text{naturality of } \in \text{ (see below)}\} \\
& min(R) \cdot \ni \cdot S \\
\subseteq & \{\text{def. } min\} \\
& R/\ni \cdot \ni \cdot S \\
\subseteq & \{\text{quotient}\} \\
& R \cdot S.
\end{array}
$$

In the second step, we exploited the naturality of $\in$ in the following way:

$$
\begin{array}{ll}
& \mathsf{P}S \cdot \ni \\
= & \{\text{converse}\} \\
& (\in \cdot (\mathsf{P}S)^\circ)^\circ \\
= & \{\mathsf{P} \text{ relator, Prop. 4}\} \\
& (\in \cdot \mathsf{P}\, S^\circ)^\circ \\
\subseteq & \{\text{naturality of } \in\} \\
& (S^\circ \cdot \in)^\circ \\
= & \{\text{converse}\} \\
& \ni \cdot S
\end{array}
$$

This completes the proof.

Once it is known how a functor can be extended from functions to relations, it is easy to extend other operators as well. Consider for instance the *split* operator $\langle \_, \_ \rangle$, which is defined on two functions with a common source by the equation

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

For relations $R$ and $S$ we have

$$\langle R, S \rangle = (R \times S) \cdot \langle id, id \rangle.$$

Such derived operators do not necessarily satisfy the same properties as their functional counterparts. For example, we have that $\pi_1 \cdot \langle R, S \rangle \subseteq R$ but $\subseteq$ cannot be replaced by $=$.

### 3.4 Algebras and Catamorphisms

Let $\mathsf{F} : \mathcal{A} \leftarrow \mathcal{A}$ be a functor on some category $\mathcal{A}$. By definition, an $\mathsf{F}$–*algebra* is an arrow $f : A \leftarrow \mathsf{F}A$. The object $A$ is said to be the *carrier* of $f$.

If $f : A \leftarrow \mathsf{F}A$ and $g : B \leftarrow \mathsf{F}B$ are $\mathsf{F}$–algebras, then an $\mathsf{F}$–*homomorphism* from $f$ to $g$ is an arrow $h : A \leftarrow B$ of $\mathcal{A}$ such that $h \cdot f = g \cdot \mathsf{F}h$. The composition of two $\mathsf{F}$–homomorphisms is again an $\mathsf{F}$–homomorphism, so the $\mathsf{F}$–algebras in $\mathcal{A}$ form a category in which the objects are $\mathsf{F}$–algebras and the arrows are $\mathsf{F}$–homomorphisms. When this category has an initial object $\alpha$, and $f$ is another $\mathsf{F}$–algebra, we write $(\!|f|\!)$ to denote the unique $\mathsf{F}$–homomorphism from $\alpha$ to $f$. Homorphisms of the form $(\![f]\!)$ are called *catamorphisms*. Initiality can thus be phrased as the equivalence

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

Not all functors $Fun \leftarrow Fun$ have an initial algebra, for example, the existential image functor $\mathsf{P}$ does not. However, all *polynomial* functors $Fun \leftarrow Fun$ do (see [13]). The class of polynomial functors is inductively defined by the following clauses:

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

$$\begin{aligned}(\mathsf{F} + \mathsf{G})\, f &= \mathsf{F}f + \mathsf{G}f \\ (\mathsf{F} \times \mathsf{G})\, f &= \mathsf{F}f \times \mathsf{G}f.\end{aligned}$$

All polynomial functors are relators. For any relator $F$, the initial algebra $\alpha$ of $\mathsf{F} : Fun \leftarrow Fun$ is also an initial algebra of $\mathsf{F} : Rel \leftarrow Rel$. We also have that $(\![R]\!)$ is simple if $R$ is. Proofs of these facts can be found in [1, 15].

In the category of functions, the universal property of the initial algebra $\alpha$ can only be specified as an equation, for equality is the only way of comparing two functions. For relations, the situation is different: here we can also talk in terms of inclusion. The following result, which is an easy consequence of the Knaster–Tarski fixpoint theorem, shows how the universal property of $\alpha$ can be weakened to deal with inclusions.

**Proposition 7.** *For all $R$ and $S$ we have*

$$\begin{aligned}(R \cdot \alpha = S \cdot \mathsf{F}R) &\equiv (R = (\![S]\!)) \\ (R \cdot \alpha \subseteq S \cdot \mathsf{F}R) &\Rightarrow (R \subseteq (\![S]\!)) \\ (R \cdot \alpha \supseteq S \cdot \mathsf{F}R) &\Rightarrow (R \supseteq (\![S]\!)).\end{aligned}$$

It is well-known that the initial algebra $\alpha$ is in fact an isomorphism [9], and therefore we have, for instance,

$$(R = S \cdot \mathsf{F}R \cdot \alpha^\circ) \equiv (R = (\![S]\!)).$$

## 4 Dynamic Programming

In this section we restate a result of De Moor [15]. Throughout, we assume $\mathsf{F}$ is a relator and $\alpha$ is its initial algebra.

We shall need the following definition. A function $f : A \leftarrow \mathsf{F}A$ is *monotonic* with respect to a relation $R : A \leftarrow A$ if

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

To illustrate this definition, consider numerical addition $+ : N \leftarrow (N \times N)$, where $N$ is the set of natural numbers. Addition is an algebra of the functor $\mathsf{F}$ given by $\mathsf{F}A = A \times A$

and $\mathsf{F}f = f \times f$. Now, addition is monotonic with respect to $\leq$. The definition above translates to

$$(\exists a, b : c = a + b \ \wedge \ a \leq a' \ \wedge \ b \leq b') \Rightarrow c \leq a' + b',$$

and corresponds to the normal definition of monotonicity of $+$ in both arguments.

**Theorem 8.** (De Moor [15]) *Let $R$ be a preorder, and let $P$ be an $\mathsf{F}$-algebra. Define $T = min(R) \cdot \Lambda (\![P]\!)^\circ$. If $\alpha$ is monotonic with respect to $R$, then the least solution $D$ of the equation*

$$D = min(R) \cdot \mathsf{P}(\alpha \cdot \mathsf{F}D) \cdot \Lambda P^\circ$$

*satisfies $D \subseteq T$.*

Before giving the proof of this theorem, let us briefly consider its intuitive interpretation. The recursion equation for $D$ is in line with the operational description of dynamic programming in the introduction to this paper. The function $\Lambda P^\circ$ splits the argument in all possible ways. This yields a set of decompositions, and for each of these decompositions, we recursively compute solutions to subproblems. The expression

$$\mathsf{P}(\alpha \cdot \mathsf{F}D) \cdot \Lambda P^\circ$$

generates a set of candidate solutions, and $min(R)$ selects a minimum element.

Turning to the proof of the Dynamic Programming Theorem, we note that (by Knaster-Tarski) it suffices to show

$$min(R) \cdot \mathsf{P}(\alpha \cdot \mathsf{F}T) \cdot \Lambda P^\circ \subseteq T.$$

Since $T = min(R) \cdot \Lambda (\![P]\!)^\circ = (\![P]\!)^\circ \cap R/(\![P]\!)$, this proof obligation can be split into two simpler conjuncts:

$$min(R) \cdot \mathsf{P}(\alpha \cdot \mathsf{F}T) \cdot \Lambda P^\circ \subseteq (\![P]\!)^\circ$$

and

$$min(R) \cdot \mathsf{P}(\alpha \cdot \mathsf{F}T) \cdot \Lambda P^\circ \cdot (\![P]\!) \subseteq R.$$

The first conjunct is proved as follows:

$$\begin{array}{cl}
 & min(R) \cdot \mathsf{P}(\alpha \cdot \mathsf{F}T) \cdot \Lambda P^\circ \\
\subseteq & \{\text{def. } min\} \\
 & \in \cdot \mathsf{P}(\alpha \cdot \mathsf{F}T) \cdot \Lambda P^\circ \\
\subseteq & \{\text{naturality of } \in\} \\
 & \alpha \cdot \mathsf{F}T \cdot \in \cdot \Lambda P^\circ \\
= & \{\in \text{ cancels } \Lambda\} \\
 & \alpha \cdot \mathsf{F}T \cdot P^\circ \\
= & \{\text{def. } T\} \\
 & \alpha \cdot \mathsf{F}(min(R) \cdot \Lambda (\![P]\!)^\circ) \cdot P^\circ
\end{array}$$

$$
\begin{array}{ll}
\subseteq & \{\text{def. } min\} \\
& \alpha \cdot \mathsf{F}(\in \cdot \Lambda (\![P]\!)^{\circ}) \cdot P^{\circ} \\
= & \{\in \text{ cancels } \Lambda\} \\
& \alpha \cdot \mathsf{F}(\![P]\!)^{\circ} \cdot P^{\circ} \\
= & \{\text{converse}\} \\
& (P \cdot \mathsf{F}(\![P]\!) \cdot \alpha^{\circ})^{\circ} \\
= & \{\text{catamorphism}\} \\
& (\![P]\!)^{\circ}.
\end{array}
$$

It remains to show that

$$min(R) \cdot \mathsf{P}(\alpha \cdot \mathsf{F}T) \cdot \Lambda P^{\circ} \cdot (\![P]\!) \subseteq R.$$

This can be done as follows:

$$
\begin{array}{ll}
& min(R) \cdot \mathsf{P}(\alpha \cdot \mathsf{F}T) \cdot \Lambda P^{\circ} \cdot (\![P]\!) \\
= & \{\text{catamorphism}\} \\
& min(R) \cdot \mathsf{P}(\alpha \cdot \mathsf{F}T) \cdot \Lambda P^{\circ} \cdot P \cdot \mathsf{F}(\![P]\!) \cdot \alpha^{\circ} \\
\subseteq & \{\text{Prop. 6}\} \\
& R \cdot \alpha \cdot \mathsf{F}T \cdot \mathsf{F}(\![P]\!) \cdot \alpha^{\circ} \\
= & \{\mathsf{F} \text{ functor}\} \\
& min(R) \cdot \ni \cdot \alpha \cdot \mathsf{F}(T \cdot (\![P]\!)) \cdot \alpha^{\circ} \\
\subseteq & \{\text{def. } T, \text{ Prop. 6}\} \\
& min(R) \cdot \ni \cdot \alpha \cdot \mathsf{F}R \cdot \alpha^{\circ} \\
\subseteq & \{\text{monotonicity}\} \\
& R \cdot R \\
\subseteq & \{R \text{ transitive}\} \\
& R.
\end{array}
$$

This completes the proof of the Dynamic Programming Theorem.

Let $T = min(R) \cdot \Lambda (\![P]\!)^{\circ}$, as in the Dynamic Programming Theorem. Using Corollary 3 we have $T = min(R') \cdot \Lambda (\![P]\!)^{\circ}$, where $R' = (\![P]\!)^{\circ} \cdot (\![P]\!) \cap R$. Furthermore, $R'$ is transitive if $R$ is and if $P$ is simple. This means we have only to establish that $\alpha$ is monotonic with respect to $R'$ in order to apply the dynamic programming theorem. In most practical applications we are not given the relation $R$ but rather a cost function $c$, *i.e.* $R = c^{\circ} \cdot S \cdot c$ for some preorder $S$. The proof of the following useful proposition is omitted.

**Proposition 9.** *Suppose $R = c^{\circ} \cdot S \cdot c$, where $c$ satisfies*

$$c \cdot \alpha = k \cdot \mathsf{F}\langle (\![P]\!), c \rangle,$$

*for some $k$ that is monotonic with respect to $S$,* i.e. $k \cdot \mathsf{F}(id \times S) \subseteq S \cdot k$. *Then, provided $P$ is entire, we have that $\alpha$ is monotonic with respect to $(\![P]\!)^{\circ} \cdot (\![P]\!) \cap R$.*

### 4.1 Application

Let us now apply the above theory to the maximum lateness problem. Let $J$ denote the type of jobs. The type of sequences over $J$ is the initial algebra of the functor $\mathsf{F} : \mathit{Fun} \leftarrow \mathit{Fun}$ given by

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

where 1 denotes the terminal object of $\mathit{Fun}$. This initial algebra will be denoted by $\alpha = [\nu, \mathbin{+\!\!\!+\!\!<}]$. Applied to the single element of 1, the function $\nu$ returns the empty sequence $[\,]$. Applied to $(x, a)$ the function $\mathbin{+\!\!\!+\!\!<}$ returns $x \mathbin{+\!\!\!+\!\!<} a = x \mathbin{+\!\!\!+} [a]$. That is, we suppose that sequences are constructed from left to right. There is of course a dual method that constructs sequences from right to left, but the left–to–right bias in the maximum lateness problem means that the given way of constructing sequences is the more appropriate.

Bags of jobs form an $F$–algebra $[\nu, \mathbin{+\!\!<}]$, where $\nu$ returns the empty bag, and $x \mathbin{+\!\!<} a$ denotes the bag formed by adding $a$ to the bag $x$. The function $\mathit{bagify} = (\!|\nu, \mathbin{+\!\!<}|\!)$ turns a sequence into a bag, and $\mathit{perms} = \Lambda\, \mathit{bagify}^\circ$ returns the set of permutations of a bag.

Recall that the cost function $c$ associated with the lateness problem satisfies

$$\begin{aligned} c\,[\,] &= -\infty \\ c(x \mathbin{+\!\!\!+} [a]) &= c\,x \sqcup (\mathit{bagify}\; x \otimes a). \end{aligned}$$

Hence

$$c \cdot \alpha = k \cdot \mathsf{F}\langle \mathit{bagify}, c \rangle$$

where $k = [-\infty, \odot]$ and $\odot$ is defined by

$$(x, n) \odot a = n \sqcup (x \otimes a).$$

In order to make use of Proposition 9 we need to check that

$$k \cdot \mathsf{F}(\mathit{id} \times (\leq)) \subseteq (\leq) \cdot k.$$

Unfolding the various definitions we get the implication

$$m \leq n \Rightarrow (x, m) \odot a \leq (x, n) \odot a.$$

But this is immediate from the definition of $\odot$. The conclusion is that dynamic programming is applicable to the maximum lateness problem.

## 5 Greedy Algorithms

Now we turn to greedy algorithms. The following theorem is similar to the dynamic programming theorem but involves an extra condition.

**Theorem 10.** *Let $T = min(R) \cdot \Lambda\, (\![P]\!)^{\circ}$, where $R$ is transitive. Suppose that $\alpha$ is monotonic with respect to $R$, and $\beta = \mathsf{F}(\![P]\!) \cdot \alpha^{\circ}$ satisfies $S \cdot \beta \subseteq \beta \cdot R$ for some $S$. Then the (unique) solution $G$ of the equation*

$$G = \alpha \cdot \mathsf{F}G \cdot min(S) \cdot \Lambda\, P^{\circ}$$

*satisfies $G \subseteq T$.*

**Proof**. Let $U = min(S) \cdot \Lambda\, P^{\circ}$. We first show that $G = (\![U^{\circ}]\!)^{\circ}$ is the unique solution of the equation $G = \alpha \cdot \mathsf{F}G \cdot U$.

$$\begin{array}{ll}
 & G = (\![U^{\circ}]\!)^{\circ} \\
\equiv & \{\text{converse}\} \\
 & G^{\circ} = (\![U^{\circ}]\!) \\
\equiv & \{\text{Proposition 7}\} \\
 & G^{\circ} \cdot \alpha = U^{\circ} \cdot \mathsf{F}G^{\circ} \\
\equiv & \{\text{converse; } \mathsf{F} \text{ preserves converse}\} \\
 & \alpha^{\circ} \cdot G = \mathsf{F}G \cdot U \\
\equiv & \{\alpha \text{ is an isomorphism}\} \\
 & G = \alpha \cdot \mathsf{F}G \cdot U .
\end{array}$$

Next, we have

$$\begin{array}{ll}
 & G \subseteq T \\
\Leftarrow & \{\text{above form for } G \text{ and Proposition 7}\} \\
 & \alpha \cdot \mathsf{F}T \cdot U \subseteq T \\
\equiv & \{\text{since } T = (\![P]\!)^{\circ} \cap R/(\![P]\!)\} \\
 & \alpha \cdot \mathsf{F}T \cdot U \subseteq (\![P]\!)^{\circ} \ \text{ and } \ \alpha \cdot \mathsf{F}T \cdot U \subseteq R/(\![P]\!).
\end{array}$$

We establish these inclusions separately.

$$\begin{array}{ll}
 & \alpha \cdot \mathsf{F}T \cdot U \subseteq (\![P]\!)^{\circ} \\
\Leftarrow & \{\text{since } T \subseteq (\![P]\!)^{\circ} \text{ and } U \subseteq G^{\circ}\} \\
 & \alpha \cdot \mathsf{F}(\![G]\!)^{\circ} \cdot G^{\circ} \subseteq (\![G]\!)^{\circ} \\
\equiv & \{\text{Proposition 7}\} \\
 & \text{true.}
\end{array}$$

Second, $\alpha \cdot \mathsf{F}T \cdot U \subseteq R/(\!|P|\!)$ is equivalent to $\alpha \cdot \mathsf{F}T \cdot U \cdot (\!|P|\!) \subseteq R$. The latter inclusion may be proved as follows:

$$\begin{array}{ll}
 & \alpha \cdot \mathsf{F}T \cdot U \cdot (\!|P|\!) \\
= & \{\text{Proposition 7, } \alpha \text{ isomorphism}\} \\
 & \alpha \cdot \mathsf{F}T \cdot U \cdot P \cdot \mathsf{F}(\!|P|\!) \cdot \alpha^\circ \\
\subseteq & \{\text{def. } U\text{, Prop. 6}\} \\
 & \alpha \cdot \mathsf{F}T \cdot S \cdot \mathsf{F}(\!|P|\!) \cdot \alpha^\circ \\
\subseteq & \{\text{assumption}\} \\
 & \alpha \cdot \mathsf{F}T \cdot \mathsf{F}(\!|P|\!) \cdot \alpha^\circ \cdot R \\
= & \{\text{since } \mathsf{F} \text{ is a functor}\} \\
 & \alpha \cdot \mathsf{F}(T \cdot (\!|P|\!)) \cdot \alpha^\circ \cdot R \\
\subseteq & \{\text{def. } T\text{, Prop. 6}\} \\
 & \alpha \cdot \mathsf{F}R \cdot \alpha^\circ \cdot R \\
\subseteq & \{\text{since } \alpha \text{ is monotonic}\} \\
 & \alpha \cdot \alpha^\circ \cdot R \cdot R \\
= & \{\text{since } \alpha \text{ is an isomorphism}\} \\
 & R \cdot R \\
\subseteq & \{R \text{ is transitive}\} \\
 & \text{true.}
\end{array}$$

The proof is complete.

The conclusion of the theorem says that there is a greedy algorithm for $T$. At each step of the computation, one chooses an optimal splitting with $min(S) \cdot \Lambda P^\circ$. Subsequently, the subproblem(s) are solved by means of $\mathsf{F}G$, and the solutions are composed into a solution for the whole problem by $\alpha$. Note that we can again use Corollary 3 to rewrite $U$ in the form $U = min(S') \cdot \Lambda P^\circ$, where $S' = P^\circ \cdot P \cap S$. As in the case of monotonicity, we state a proposition that eases the task of verifying the extra condition in the greedy theorem. Like Proposition 9, its proof is omitted.

**Proposition 11.** *Suppose that* $\mathsf{F} = \mathsf{F}_0 + \mathsf{F}_1$*, and* $\alpha = [\alpha_0, \alpha_1]$*. Furthermore, assume that* $S = S_0 + S_1$*, and* $P = [P_0, P_1]$*, and* $(\!|P|\!)^\circ$ *is entire. Then*

$$(P^\circ \cdot P \cap S) \cdot \mathsf{F}(\!|P|\!) \cdot \alpha^\circ \subseteq \mathsf{F}(\!|P|\!) \cdot \alpha^\circ \cdot ((\!|P|\!)^\circ \cdot (\!|P|\!) \;\cap\; R)$$

*if and only if*

$$(P_i^\circ \cdot P_i \cap S_i) \cdot \mathsf{F}_i(\!|P|\!) \subseteq \mathsf{F}_i(\!|P|\!) \cdot \alpha_i^\circ \cdot R \cdot \alpha_i \quad \textit{for } i = 0, 1.$$

### 5.1 Application

Let us go back to the maximum lateness problem. It has already been shown that $\alpha$ is monotonic, so it remains to verify that with $P = [\nu, +\!\!<]$, and $S' = P^\circ \cdot P \cap S$ for some suitable $S$, and $R = (\mathit{bagify}^\circ \cdot \mathit{bagify}) \cap (c^\circ \cdot (\leq) \cdot c)$, we have

$$S' \cdot \mathsf{F}\mathit{bagify} \cdot [\nu, +\!\!+\!\!<]^\circ \subseteq \mathsf{F}\mathit{bagify} \cdot [\nu, +\!\!+\!\!<]^\circ \cdot R.$$

Choose $S = \mathit{id} + ((\otimes)^\circ \cdot (\leq) \cdot (\otimes))$. Proposition 11 says that the proof obligation is equivalent to

$$\begin{array}{l} (u \otimes a) \leq ((\mathit{bagify}\ y) \otimes b) \wedge (u +\!\!< a) = ((\mathit{bagify}\ y) +\!\!< b) \\ \Rightarrow \\ \exists x : u = \mathit{bagify}\ x \wedge c\,(x +\!\!+ [a]) \leq c\,(y +\!\!+ [b]). \end{array}$$

To prove this implication, we consider two cases: $a = b$ and $a \neq b$. In the case $a = b$ we have $u = \mathit{bagify}\ y$, and so we can take $x = y$. On the other hand, if $a \neq b$, we argue

$$\begin{array}{ll} & c(x +\!\!+ [a]) \leq c(y +\!\!+ [b]) \\ = & \{\text{definition of } c\} \\ & c\,x \sqcup (u \otimes a) \leq c\,y \sqcup (\mathit{bagify}\ y \otimes b) \\ = & \{\text{since } u \otimes a \leq \mathit{bagify}\ y \otimes b\} \\ & c\,x \leq c\,y \sqcup (\mathit{bagify}\ y \otimes b) \\ = & \{\text{definition of } c\} \\ & c\,x \leq c\,(y +\!\!+ [b]). \end{array}$$

Now, since $u +\!\!< a = (\mathit{bagify}\ y) +\!\!< b$ and $a \neq b$, there exist $y_0$ and $y_1$ such that $y = y_0 +\!\!+ [a] +\!\!+ y_1$. Take $x = y_0 +\!\!+ y_1 +\!\!+ [b]$. We claim that $c\,x \leq c\,(y +\!\!+ [b])$. More generally, we have $c\,(w +\!\!+ z) \leq c\,(w +\!\!+ [a] +\!\!+ z)$ for all $w$, $z$, and $a$. This follows by induction on $z$, using the result

$$\begin{array}{ll} & (w +\!\!+ z) \otimes b \\ = & \{\text{definition of } \otimes\} \\ & \mathit{wt}\ b \times ((\sum_{c \in w +\!\!+ z} \mathit{time}\ c) + \mathit{time}\ b - \mathit{due}\ b) \\ \leq & \{\text{since weights and times are nonnegative}\} \\ & \mathit{wt}\ b \times ((\sum_{c \in w +\!\!+ [a] +\!\!+ z} \mathit{time}\ c) + \mathit{time}\ b - \mathit{due}\ b) \\ = & \{\text{definition of } \otimes\} \\ & (w +\!\!+ [a] +\!\!+ z) \otimes b. \end{array}$$

The conclusion is that the maximum lateness problem can be computed by a greedy algorithm.

## 6 Conclusions

One of the aims of the paper was to clarify the relationship between dynamic programming and certain greedy algorithms. This relationship is captured in the very similar statements of the dynamic programming and greedy theorems.

It is important to note that the greedy theorem gives general conditions under which a greedy solution is possible, but does not discuss mathematical systems in which the greedy condition is valid. Two such systems are known: those of a *matroid* and those of a *greedoid* (see [8]). Essentially, the verification of the greedy condition for the maximum lateness problem is an application of greedoid theory. It remains to be seen whether our approach gives further insight into such systems.

Finally, because the greedy theorem holds for tree algebras and not just lists, it may also have applications in the derivation of divide–and–conquer algorithms. Here a comparison with Smith's approach to divide–and–conquer [16] could be a fruitful research topic.

## References

1. R.C. Backhouse, P. Hoogendijk, E. Voermans, and J.C.S.P. van der Woude. A relational theory of datatypes. Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands., June 1992.
2. M. Barr and C. Wells. *Category Theory for Computing Science*. Prentice–Hall, 1990.
3. R.S. Bird and O. de Moor. List partitions. *Formal Aspects of Computing*, 5(1):61–78, 1993.
4. A. Carboni, G.M. Kelly, and R.J. Wood. A 2–categorical approach to geometric morphisms, i. *Cahiers de Topologie et Geometrie Differentielle Categoriques*, 32(1):47–95, 1991.
5. E.W. Dijkstra. *A Discipline of Programming*. Prentice–Hall, 1976.
6. Maarten M. Fokkinga. *Law and Order in Algorithmics*. PhD thesis, Technical University Twente, The Netherlands, 1992.
7. D.S. Hochbaum and R. Shamir. An $\mathcal{O}(n \log^2 n)$ algorithm for the maximum weighted tardiness problem. *Information Processing Letters*, 31:215–219, 1989.
8. B. Korte, L. Lovasz, and R. Schrader. *Greedoids*, volume 4 of *Algorithms and combinatorics*. Springer–Verlag, 1991.
9. J. Lambek. A fixpoint theorem for complete categories. *Mathematische Zeitschrift*, 103:151–161, 1968.
10. E.L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19(5):544–546, January 1973.
11. G. Malcolm. Homomorphisms and promotability. In J. L. A. van de Snepscheut, editor, *Mathematics of Program Construction*, volume 375 of *Lecture Notes in Computer Science*, pages 335–347. Springer–Verlag, 1989.
12. G. Malcolm. Data structures and program transformation. *Science of Computer Programming*, 14:255–279, 1990.
13. E.G. Manes and M.A. Arbib. *Algebraic Approaches to Program Semantics*. Texts and Monographs in Computer Science. Springer–Verlag, 1986.
14. L. Meertens. Paramorphisms. *Formal Aspects of Computing*, 4(5):413–424, 1992.
15. O. de Moor. Categories, relations and dynamic programming. D.Phil. thesis. Technical Monograph PRG-98, Computing Laboratory, Oxford, 1992.

16. D.R. Smith. Applications of a strategy for designing divide–and–conquer algorithms. *Science of Computer Programming*, 18:213–229, 1987.
17. M. Spivey. A categorical approach to the theory of lists. In J. L. A. van de Snepscheut, editor, *Mathematics of Program Construction*, volume 375 of *Lecture Notes in Computer Science*, pages 399–408. Springer–Verlag, 1989.

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