Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Science of Computer Programming 49 (2003) 125–157
Science of Computer Programming
www.elsevier.com/locate/scico

The classification of greedy algorithms

S.A. Curtis

Department of Computing, Oxford Brookes University, Oxford OX33 1HX, UK

Received 22 April 2003; received in revised form 5 September 2003; accepted 22 September 2003

Abstract

This paper presents principles for the classification of greedy algorithms for optimization problems. These principles are made precise by their expression in the relational calculus, and illustrated by various examples. A discussion compares this work to other greedy algorithms theory.

MSC: 68R05

Keywords: Greedy algorithms

1. Introduction

What makes an algorithm *greedy*? Definitions in the literature vary slightly, but most describe a greedy algorithm as one that makes a sequence of choices, each choice being in some way the best available at that time (the term greedy refers to choosing the best). When making the sequence of choices, a greedy algorithm never goes back on earlier decisions.

Because of their simplicity, greedy algorithms are frequently straightforward and efficient. They are also very versatile, being useful for different problems in many varied areas of combinatorics and beyond. Some examples of applications include data compression and DNA sequencing [16,23,25], finding minimum-cost spanning trees of weighted undirected graphs [24,30,34], computational geometry [17], and routing through networks [20], to name but a few.

Some problems are impractical to solve exactly, but may have a greedy algorithm that can be used as a heuristic, to find solutions which are close to optimal. For other problems, greedy algorithms may produce an exact solution. Unfortunately, for any

E-mail address: sharoncurtis@brookes.ac.uk (S.A. Curtis).

0167-6423/$ - see front matter
doi:10.1016/j.scico.2003.09.001

particular problem, there is no guarantee that a greedy algorithm exists to solve it exactly. Therefore the algorithm designer who thinks up plausible greedy strategies to solve a problem may find theory about greedy algorithms useful: correctness conditions can be tested to see whether a particular greedy algorithm provides an exact solution.

Existing greedy theories have frequently addressed the following concerns:

- Expression of greedy algorithms.
- Correctness proofs of greedy algorithms.
- Characterization of greedy data structures.
- Synthesis of greedy algorithms.
- Coverage of as many greedy algorithms as possible.

Different greedy theories have concentrated on different selections of the above concerns. The theory of matroids [15,37], and later greedoids [27–29], models greedy algorithms using set systems, concentrating heavily on the characterization of greedy data structures (problem structures for which the greedy algorithm produces an optimal solution), but does not consider the synthesis of greedy algorithms at all. Some theories, like that of Helman's k-ary dominance relations [18], and his work with Moret and Shapiro [19], attempt to cover more greedy algorithms, by generalizing the greedy data structures considered. Other theories have concentrated on the expression of greedy algorithms, such as the work of Charlier [9], and that of Bird and de Moor [2,3,4,6]. The latter characterizes greedy structures by using categorical datatypes, and focuses on the use of catamorphisms and anamorphisms on those datatypes to express and develop greedy algorithms.

To date, greedy theories have made good progress in addressing the first four of the above concerns, but none has yet succeeded in incorporating *all* greedy algorithms.

The goals for the work in this paper were (in order):

(1) Coverage of all greedy algorithms that solve optimization problems.
(2) Characterization of greedy structures.
(3) Correctness proofs and design inspiration for algorithm development.
(4) Expressing greedy algorithms simply.

This paper presents the results of that work: a comprehensive theory of greedy algorithms for the solution of optimization problems, which provides four general principles that both classify greedy algorithms and assist with proofs of their correctness. The theory does not cover greedy heuristics, nor does it address problems that are not optimization problems: its scope is that of greedy algorithms providing optimal solutions to optimization problems.

The presentation of this work is in three parts. In Section 2, the four greedy principles are presented informally to provide a readable introduction at an abstract level. Section 3 formally expresses these principles in the relational calculus, illustrated with several examples of greedy algorithms. Finally, in Section 4, this work is discussed and evaluated.

2. Informal overview

In this section, the four greedy principles are stated informally with examples. This separate overview emphasizes that the principles are not reliant on any particular formalism in which a greedy algorithm might be expressed.

2.1. Terminology for greedy algorithms

An optimization problem is typically specified by the description of its potential solutions, together with a criterion used to judge which is optimal. This criterion will be called the *global optimality criterion*.

As described previously, a greedy algorithm makes a sequence of choices, each being in some way the best at that time. The criterion to judge what is best for the greedy choice will be called the *local optimality criterion*.

As the greedy algorithm progresses, each choice involves taking a step towards the construction of a solution to the problem. Such a step will be called the *construction step*. It is intended that the role of the construction step (independent of the way it is used within the greedy algorithm) is to be able to generate all potential solutions to the optimization problem, by repeatedly applying the construction step to the input in all possible ways.

With regards to this construction, the term *completed solution* will be used to denote potential solutions to the problem that are fully completed, which may or may not be optimal with respect to the global optimality criterion. In contrast, the term *partial solution* will refer to a partially constructed potential solution (which may or may not be fully constructed).

Here are two examples to illustrate:

Kruskal's algorithm (see [30]) is a well-known greedy algorithm that finds a spanning tree of minimum cost in an undirected (connected) graph with edge costs. It proceeds by starting with an empty set of edges, and at each stage, adds an edge to the set so that no cycles are created amongst the edges collected so far. The edge chosen at each step is one of minimum cost.

For this algorithm, the acyclic edge sets are the partial solutions, and the edge sets forming spanning trees of the graph are the completed solutions. In addition,

Construction Step: Add a non-cycle-creating edge to the set of edges.

Local Optimality Criterion: The edge chosen should be of minimum cost.

Global Optimality Criterion: The sum of the costs of the set of edges should be as small as possible.

For Kruskal's algorithm, the local optimality criterion could have been rephrased to be the same as the global optimality criterion. This is because when choosing to add a non-cycle-creating edge to the set, minimizing the sum of the edge costs is the same goal as minimizing the cost of the edge being added. Whilst it is often possible to rephrase the local criterion in such a way that it is the same as the global criterion, sometimes it is not. The following example demonstrates that the two optimality criteria may be very different:

The coin changing problem (see [7,8]) concerns the choosing of as few coins as possible to add up to a given value. For certain currencies (for example, with denominations of $\{1,2,5\} \times 10^n$, $n \in \{0,1,2,3,4\}$), the following greedy algorithm works: at each step choose the greatest valued coin that does not exceed the amount remaining. For example, to make up change to the value of 45*c*, the coins 20*c*, 20*c*, 5*c* would be chosen, in that order.

For this algorithm, coin collections not exceeding the given value are the partial solutions, and coin collections totalling precisely the given value are the completed solutions. In addition,

Construction Step: Add a coin to those chosen so far, such that the value of the chosen coin does not bring the total to more than the given value.

Local Optimality Criterion: The coin chosen should be of maximum value.

Global Optimality Criterion: The total number of coins chosen should be as small as possible.

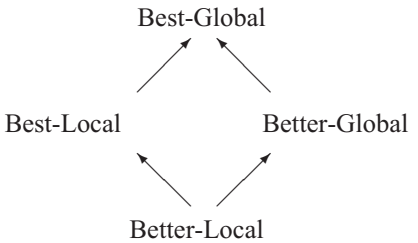
In the greedy coin-changing algorithm, the global optimality criterion is very different from the local optimality criterion, and illustrates a common theme: an optimal solution requires as few construction steps as possible, and in contrast, the greedy choice tries to make as much progress as it can.

There are many problems solvable by greedy algorithms with different global and local criteria. For example, the following problems also use global optimality criteria minimizing the number of construction steps: the Smallest Upravels [1], Dictionary Coding [5,36], Professor Midas [10] (also see later), and the Rally Driver [11,14]. In contrast, some problems use global optimality criteria maximizing the number of construction steps, for example the Activity Selection [10] and Marble Mingling [11–13] problems. If the global criterion either minimizes or maximizes the number of construction steps, then the local criterion must necessarily be different from the global criterion. Furthermore, some greedy algorithms have different global and local optimality criteria where the global criterion has nothing to do with the construction step, for example the Amoeba Fight Show problem [32].

The fact that the local optimality criterion often differs from the global optimality criterion is important: several theories do not include this possibility and thus exclude many greedy algorithms.

2.2. Four greedy principles

The following diagram names and relates the four greedy principles (the arrows are implications):



The principles describe possible relationships between the optimality criteria and the construction step. Every greedy algorithm that produces an optimal solution for an optimization problem will satisfy the Best-Global principle, but may or may not satisfy any of the others. These principles are now individually described informally and discussed briefly.

2.2.1. Best-Global

The Best-Global principle is really *the* condition that says that the greedy algorithm works in the first place. That is, every greedy algorithm that finds optimal solutions to optimization problems complies with this principle. It can be described in words as follows:

Best-Global: Consider a partial solution that is the immediate result of a greedy choice (that is, a partial solution that is *best* with respect to the local optimality criterion, of the alternative ways of performing the construction step). Compared to completions of any of the other alternatives, there is a completion of this best partial solution that is at least as good with respect to the global optimality criterion.

This condition is frequently used as a method of proof for individual greedy algorithms: typically a completed solution is transformed into a second completed solution, where the second solution was constructed by starting with a greedy choice. It is then shown that the second is no worse than the first, with respect to the global optimality criterion, and in this way, it can be shown that greedily chosen partial solutions can be completed to be at least as good as other completed solutions.

One important feature about this global condition is that optimality is only guaranteed for completed solutions: it may be that a partial solution that is produced at an intermediate stage of the greedy algorithm is *not* optimal.

Every greedy algorithm that produces an optimal solution to an optimization problem satisfies this Best-Global principle, and there are numerous examples of correctness proofs following this pattern throughout the literature. For example, in Horowitz et al.'s book [22], a specific instance of the above principle is given as a way to prove a greedy algorithm correct, and this is used to show the correctness of Kruskal's algorithm.

2.2.2. Better-Global

The Better-Global principle is similar to the Best-Global, but stronger:

Better-Global: Consider two partial solutions (of the possible alternatives when making the greedy choice), the first *better* than the second with respect to the local optimality criterion. Given any completion of the second alternative, there is a completion of the first alternative that is at least as good with respect to the global optimality criterion.

The Better-Global condition, like the Best-Global condition, considers how the relative merits of partial solutions affect their completions. However, the Best-Global condition addresses the best of alternatives, whereas the Better-Global condition addresses two of the alternatives, one better than the other. Thus the Better-Global principle is rarely used for correctness proofs, because it is a stronger condition (the better of the

two partial solutions cannot also be assumed to be the best) and therefore the proofs are no easier.

An important point to note is that "better" is used in a non-strict way: the description "at least as good as" would be more apt, but less succinct.

One example of a greedy algorithm satisfying the Better-Global condition is as follows:

Professor Midas' Driving problem (from [10]) concerns the plan of a car journey along a fixed route. In particular, the Professor wishes to choose which service stations to stop at along the route, in order to fill up the tank of the car with petrol. The car's tank can hold enough petrol to travel up to some fixed distance, and overall Professor Midas would like to make as few stops as possible.

If the plan for the journey is constructed by deciding which service stations to stop at, making the decisions in order from the start point, then the greedy algorithm of "Always go as far as you can before stopping for petrol" works.

This satisfies the Better-Global condition: when the Professor is at a service station contemplating the next stop, the choice of a further away station, as compared to a nearer one, offers the possibility of a journey with a lesser (or equal) number of stops. (See later for the proof.)

Another example of a greedy algorithm satisfying the Better-Global condition is one solving the Maximum Tardiness problem (see [21,31]), and this is given later as an illustration for Theorem 2 (Better-Global).

2.2.3. *Best-Local*

The two remaining principles are more concerned with the local optimality criterion, and how optimality is maintained with respect to that criterion as the greedy algorithm progresses.

Best-Local: During the running of the greedy algorithm, a partial solution that is the *best* so far, with respect to the local optimality criterion, remains the best so far after applying the greedy step. In addition, optimality with respect to the local optimality criterion must imply that global optimality is also achieved.

One important point to note is that not all greedy algorithms can be considered for compliance with the Best-Local principle. The greedy step only considers immediate possible choices from a single partial solution, and so the local optimality criterion is only required to be able to compare such choices; however the above principle requires a local optimality criterion that can compare partial solutions in a more general way. This is best illustrated with examples. First, an example where the local optimality criterion can be generalized, and the Best-Local principle does hold:

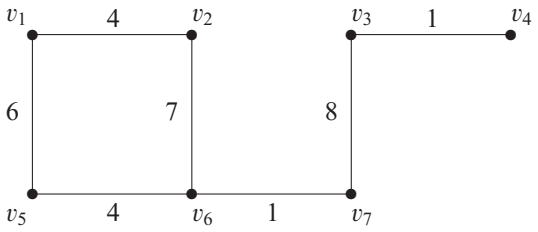
Kruskal's Algorithm (as previously described in Section 2.1) finds a minimum cost spanning tree of a graph by repeatedly selecting an non-cycle-creating edge of minimum cost, to add to those selected so far. Here, the local (and global) optimality criterion is the minimizing of the sum of the edge costs. Kruskal's algorithm satisfies the Best-Local principle, because after n steps of the algorithm, the acyclic subgraph formed is one with minimum cost, out of all possible ways of choosing n edges to form acyclic subgraphs. This therefore leads to a minimum cost spanning tree for the whole (connected) graph.

Secondly, here is an example where the local optimality criterion can be generalized, but the Best-Local principle does not hold:

Prim's Algorithm (see [24,34]) is another well-known greedy algorithm for finding a minimum cost spanning tree in an undirected connected graph with edge costs. It too proceeds by starting with an empty set of edges, at each stage adding a minimum cost edge to its set collected so far. However, unlike Kruskal's algorithm, it selects its non-cycle-creating edge only from edges adjoining those collected so far.

The local and global optimality criteria are the same as those for Kruskal's algorithm, namely the minimizing of the sum of the costs of the edges collected so far. It is the construction step that differs, by the restriction of edge choices.

Prim's algorithm does not satisfy the Best-Local principle, as the tree created after n steps is not necessarily the tree of minimum cost out of all those with n edges. For example, consider the following graph, with a starting node nominated as v_6:



After three steps of Prim's algorithm, the tree $\{(v_6, v_7), (v_5, v_6), (v_1, v_5)\}$ will be formed. Yet the tree $\{(v_6, v_7), (v_7, v_3), (v_3, v_4)\}$ also has three edges, sprouts from v_6, and is of lower total cost.

Finally, here is an example where the Best-Local principle is not applicable, as the local optimality criterion cannot be generalized:

Marble Mingling (see [11–13]). Given a collection of coloured marbles, it is desired to select as many sets of marbles of size k as possible from the collection, where no set contains two marbles of the same colour. Assuming that the marbles have been arranged into jars by their colour, a greedy algorithm to solve this problem involves the repeated selection of k marbles from the k fullest jars.

Here the local optimality criterion prefers the selection of marbles from jars which are fullest *at that step* in the algorithm. This criterion is not rephrasable to cover partial solutions more generally, and so the Best-Local principle is not applicable.

2.2.4. Better-Local

The Better-Local principle is the strongest of the four principles:

Better-Local: Suppose one partial solution is *better* than a second, with respect to the local optimality criterion. Given the result of a construction step on the second, there is a way of performing a construction step on the first that is at least as good with respect to the local optimality criterion. In addition, optimality with respect to the local optimality criterion must imply that global optimality is also achieved.

(Again, "better" is translated to mean "at least as good as".) The Better-Local principle is essentially a monotonicity condition, saying that the construction step is monotonic with respect to the local optimality criterion. This can be rephrased in the words from a well-known song: "Anything you can do, I can do better".

Although this condition is the strongest, and not many greedy algorithms satisfy it, it has the advantage of being relatively easy to prove, for those greedy algorithms that do satisfy it. One example of a greedy algorithm satisfying the Better-Local principle is the following:

Professor Midas' *Driving problem*, as previously described, uses the plan of "Always go as far as you can before stopping for petrol". Thus the local optimality criterion prefers service stations that are further along the route.

This satisfies the Better-Local condition: imagine a friend of Professor Midas who drives an identical car along the same route. If the Professor is currently stopped at a service station, and he is at least as far along the route as his friend (also currently stopped at a service station), then however far his friend gets before next stopping, the Professor will be able to get at least as far as his friend, on his next stop. (See later for the proof.)

Other examples of greedy algorithms which comply with the Better-Local principle include solutions of the following problems: Dictionary Coding [5,36], Activity Selection [10], Bank Deliveries [5,38] and the Lexicographically Largest Subsequence [11].

2.3. Discussion

The above four principles describe different ways that the construction step may relate to the global and local optimality criteria, for greedy algorithms.

The earlier statements of the principles are lengthy, in an effort to make them precise and unambiguous. Here is a summary of all four, for comparison and reference. These statements are shorter and more memorable, but less precise:

Best-Global: Making a *best* local choice can ultimately lead to a better solution.

Better-Global: Making a *better* local choice can ultimately lead to a better solution.

Best-Local: Repeatedly making a *best* local choice always results in a partial solution that is best so far.

Better-Local: A *better* partial solution can lead to one that is still better after the next construction step.

The two *-Global principles are concerned with how choices with respect to the local optimality criterion affect how completed solutions relate using the global optimality criterion. For algorithms complying with the *-Global principles, optimality is achieved ultimately, but is not insisted upon for intermediate stages. In contrast, the two *-Local principles primarily focus on how choices with respect to the local optimality criterion affect partial solutions and their relationship with respect to the local optimality criterion. The two *-Local principles do insist on optimality at intermediate stages of the greedy algorithm. An analogy could be runners in a race: a runner who wins the race, having led the other runners all the way through, is following a local principle, whereas a runner who wins, having not always been in the lead, is following a global principle.

The two Better-* principles are concerned with whether making a better choice is sufficient to affect optimality, whereas the two Best-* principles consider how the best choices affect optimality.

2.3.1. Classification

The four greedy principles can also be viewed as different ways one might prove the correctness of a greedy algorithm: if a greedy algorithm complies with one or more of the principles, then it solves its optimization problem. Conversely, every greedy algorithm solving an optimization problem complies with at least one of these principles, and this, together with the implications between the principles (described in the diagram in Section 2.2), results in the classification of greedy algorithms in five different ways:

(1) *Best-Global* only
(2) *Better-Global* and *Best-Global* only
(3) *Best-Local* and *Best-Global* only
(4) *Best-Local*, *Better-Global* and *Best-Global* only
(5) *Better-Local*, *Best-Local*, *Better-Global* and *Best-Global*

This classification results in a variety of possible ways for an algorithm designer to prove the correctness of a greedy algorithm. Whilst a greedy algorithm may comply with several of the four principles, its compliance with one principle may be easier to demonstrate than that of another.

The four greedy principles have been informally expressed, to illustrate their general applicability and independence of any particular formalism. To add substance to their description, and to justify the assertions made, a translation of them into the relational calculus is now presented, together with examples of assorted greedy algorithms.

3. Greedy principles and the relational calculus

Although greedy algorithms may be expressed in many different formalisms, it is advantageous to use relations for specifications and algorithms, as will be discussed later on. What follows is an introduction to the main relational concepts used in this paper; a more extensive reference on relational theory can be found, for example, in [6].

3.1. Relational notation

A binary relation, being a set of pairs of elements, needs notation to express which elements are related and what type(s) they are. The statement aRb asserts that a is related to b by relation R, just as for the relation $\leqslant$ one might say $a \leqslant b$. More formally, the set-theoretic meaning of aRb is $(a, b) \in R$. The type of a relation is given as $R : A \leftarrow B$ (to mean $R \subseteq A \times B$), emphasizing the idea that relations can be regarded as non-deterministic partial functions, with inputs of type B, and outputs of type A.

3.1.1. Basic operators

Two relations are composed in a way similar to functions: for $a \in A$, $c \in C$, and $R : A \leftarrow B$, $S : B \leftarrow C$,

$$a(R.\ S)c \quad \equiv \quad \exists b\ .\ aRb\ \wedge\ bSc$$

The relation $R.S$ can be thought of as applying S to an input (in a non-deterministic fashion), then applying R to the result. For example, the relation *SquareRoot* . *Square* relates any input number x to both $+x$ and $-x$.

Sometimes it is convenient to use an arrow notation to describe related elements. For example, $xRy \wedge ySz$ can be written:

$$x \overset{R}{\leftarrow} y \overset{S}{\leftarrow} z$$

This differs from merely stating $x(R\ .\ S)z$, as the intermediate element y is also identified.

The *converse* of a relation is denoted by ^{o}, so that $aR^o b \equiv bRa$. In the arrow notation, converse is represented by a reversed arrow, so that (for example),

$$x \overset{R}{\leftarrow} y \overset{S}{\rightarrow} z$$

abbreviates $xRy \wedge yS^o z$.

The *intersection* of two relations $R, S : B \leftarrow A$ is defined as their set-theoretic intersection, $R \cap S$. This captures the idea of the Boolean operator $\wedge$, as $x(R \cap S)y$ when $xRy \wedge xSy$. For example, $\leqslant \cap \geqslant$ is the equality relation. The following universal property gives an alternative definition: for all R, S, T

$$R\ \subseteq\ S \cap T \quad \equiv \quad R \subseteq S\ \wedge\ R \subseteq T$$

Similarly, the union of two relations R and S is defined to be their set-theoretic union $R \cup S$, which captures the idea of the Boolean operator $\vee$, as $x(R \cup S)y$ when $xRy \vee xSy$. For example, $\langle \cup \rangle$ is the inequality relation. Union also has an alternative definition by a universal property: for all R, S, T

$$R \cup S\ \subseteq\ T \quad \equiv \quad R \subseteq T\ \wedge\ S \subseteq T$$

3.1.2. Identity and coreflexives

To express equality, the *identity relation* is used, denoted Id, because the symbol '=' is potentially confusing within equations. Thus $Id_A : A \leftarrow A$ is defined to be $Id_A = \{(x,x) \mid x \in A\}$, and the subscript is often omitted and inferred from context. For example, the identity relation can be found in the definition of a preorder: if relation R is a preorder, it must be reflexive and transitive, expressed as

$$Id \subseteq R \quad \text{(reflexivity)}$$

$$R\ .\ R \subseteq R \quad \text{(transitivity)}$$

A *coreflexive* is a relation included in (a subset of) Id. Such relations can be used to represent predicates: given a predicate $p : Boolean \leftarrow A$, it is represented by the

coreflexive $p? : A \leftarrow A$, where

$$p? \quad = \quad \{(x,x) \mid p\ x\}$$

For example, one predicate commonly used in this paper concerns whether an element is in the *domain* of a relation. If $R : A \leftarrow B$, then the domain of R is the set $\{x \mid \exists y\ .\ yRx\}$. The coreflexive which corresponds to elements in the domain of R is denoted $R\Box$. An alternative definition for $R\Box$ is

$$R\Box \quad = \quad Id \cap R^o.\ R$$

Another useful coreflexive is $R\boxtimes$, to indicate that an element is not in the domain of a relation, and this can be defined (using set-theoretic notation) by

$$R\boxtimes \quad = \quad Id - R\Box$$

From the above definitions follow properties such as

$$R\Box \ \cup \ R\boxtimes = Id$$

$$R\ .\ R\Box = R$$

3.1.3. Optimal selections

In the same way that the intersection $\cap$ and union $\cup$ operators capture the idea of $\wedge$ and $\vee$, the *quotient* $/$ operator captures the idea of implication. It can be defined by the universal property that for all relations R, S, T

$$R \ \subseteq \ S/T \quad \equiv \quad R\ .\ T \ \subseteq \ S$$

From this follows the property that explains the name "quotient": for all R, S

$$R/S\ .\ S \subseteq R$$

However, the equivalence $x(R/S)y \equiv (\forall z.ySz \Rightarrow xRz)$ better illuminates the connection with implication. Thus for example, if x is an upper bound on the set y, this can be expressed by $x(\geqslant / \ni)y$, as for all z in the set y, $x \geqslant z$.

This leads onto the extraction of an optimal element from a set. Recall the definition of the maximum of a set: the maximum must be an upper bound for the set and also in the set. This inspires the following definition for obtaining an optimum of a set with respect to a relation R:

$$opt\ R \quad = \quad \in \ \cap \ \ R/\ni$$

To repeat this in words, if an element x is the best in a set with respect to R, this means that it must be a member of the set, and it must be better (with respect to R) than any other element in the set. For example, $opt \geqslant$ can be used to find the maximum of a set, as $x(opt \geqslant)s$ when x is the maximum of the set s. Similarly, $opt \leqslant$ returns the minimum of a set.

The operator *opt* is not only used with linear orderings: for example, defining $\leqslant_f$ to be $x \leqslant_f y \equiv f(x) \leqslant f(y)$, the relation $opt \leqslant_f$ chooses an element from a set that is minimal with respect to the function f.

The Λ operator converts a relation to the corresponding set-valued function:

$$(\Lambda\ R)\ x \quad = \quad \{y \mid y\ R\ x\}$$

This operator is useful to model choice: whilst a relation R can be thought of as applying R to an input to yield a result, ΛR can be thought of as returning the set of all possible results of applying R to that input.

One useful property concerning *opt* and Λ is the following:

$$opt\ R\ .\ \Lambda T \quad = \quad T\ \cap\ R/T^o$$

Expressing this in words, $y(opt\ R\ .\ \Lambda T)x$ says that out of all possible ways of applying T to x, y is optimal with respect to R. On the right hand side, $y(T \cap R/T^o)x$ says that y is a result of applying T to x, and furthermore, for any result of applying T to x, y is at least as good as with respect to R.

3.1.4. Repetition

The last concept needed is that of repetition. One form of repetition applies a relation a fixed number of times in succession. This can be denoted R^n, where

$$R^n \quad = \quad \overbrace{R\ .\ R\ .\ \ \ldots\ \ .\ R}^{n}$$

Another form of repetition specifies that the relation is to be applied until this is no longer possible. This is denoted by *rep* R, where

$$rep\ R \quad = \quad R \boxtimes\ \cup\ rep\ R\ .\ R \tag{1}$$

To understand this, consider $y(rep\ R)x$ for some x and y. If x is not in the domain of R, then R cannot be performed, and $y(R\boxtimes)x$ with $x = y$. Alternatively, x is in the domain of R, and thus R is first performed once on x, and then repeatedly applied until this is no longer possible, to obtain y. Thus *rep* R is the union of $R\boxtimes$ and *rep* R . R.

The operator *rep* is typically used when some form of loop is required. For example, consider the relation $R : (Bag\ E \times List\ E) \leftarrow (Bag\ E \times List\ E)$, where for any bag b, list s and element e,

$$(b, [e] +\!\!+ s)\ R\ (b + \lfloor e \rfloor, s)$$

Thus R removes an element e from a bag $b + \lfloor e \rfloor$ (the $\lfloor$ and $\rfloor$ denote bag brackets, and $+$ denotes bag addition), and adds it onto the front of a list s, to yield $[e] +\!\!+ s$ (the [and] are list brackets, and the $+\!\!+$ is list concatenation). This can be used to produce permutations: given an input $(b, [\ ])$, repeatedly applying R will result in a permutation of the elements in b, so that $(\lfloor\ \rfloor, s)rep\,R(b, [\ ])$ when the elements in s form the bag b.

Whilst (1) illustrates the idea of *rep*, it does not uniquely specify *rep* R: in general, there may be several such relations which satisfy $X = F(X)$, with

$$F(X) \quad = \quad R \boxtimes\ \cup\ X\ .\ R \tag{2}$$

The relation *rep R* is defined to be the least fixpoint of Eq. (2). As a consequence of this definition

$$F(S) \subseteq S \quad \Rrightarrow \quad rep\ R \subseteq S$$

Further useful properties of *rep* are given below, along with explanations:

$$rep\ R\ .\ R\boxtimes = R\boxtimes \tag{3}$$

$$rep\ R\ .\ R\Box = rep\ R\ .\ R \tag{4}$$

$$rep\ R\ .\ R \subseteq rep\ R \tag{5}$$

$$R\boxtimes\ .\ rep\ R = rep\ R \tag{6}$$

$$R\boxtimes\ .\ R^n \subseteq rep\ R \tag{7}$$

$$R\boxtimes\ .\ R^* = rep\ R \tag{8}$$

where R^* represents the reflexive transitive closure of R. Paraphrasing the above properties, (3) says that attempting to repeatedly apply R to an element not in the domain of R has no effect. On the other hand, for elements in the domain of R, repeatedly applying R (until it can be applied no more) begins with an initial application of R (4). Eq. (5) says that if R is applied once before repeating it, then the result is one that could have been obtained by repeatedly applying R in the first place. Eq. (6) points out that applying *rep R* always gives results not in the domain of R. Eq. (7) asserts that if R is applied n times to an input, yielding an output not in the domain of R, then the input and output are related by *rep R*. Furthermore, the relation *rep R* consists of the union of all such $R\boxtimes\ .\ R^n$, for $n \geqslant 0$, because $R^* = \bigcup_{n\geqslant 0} R^n$ (8).

3.2. Modelling greedy algorithms and optimization problems

A single step of the greedy algorithm makes a choice with respect to the local optimality criterion, amongst possible ways to perform a construction step. Let $S : P \leftarrow P$ be the construction step, with P a datatype suitable for representing partial solutions, so that $t'St$ when t' is a possible result of performing a construction step on the partial solution t. Thus the function ΛS returns the set of all possible ways a construction step can be performed on a partial solution. If the local optimality criterion is denoted by $L : P \leftarrow P$, then $t_1 L t_2$ when t_1 is at least as good as t_2 with respect to the local optimality criterion, and thus taking an optimum with respect to L can be performed by *opt L*. The greedy step can thus be defined as

$$Greedy(L, S) = opt\ L\ .\ \Lambda S$$

The greedy algorithm is simply a repetition of the greedy step, so that a greedy algorithm is modelled by

$$rep\ \ Greedy(L, S)$$

Optimization problems also need to be modelled. Repeatedly applying the construction step to the input gives a possible solution to the problem, so that *rep S* is the relation

that relates an input to a possible completed solution. Thus $\Lambda(rep\ S)$ is the function taking the input and returning all possible completed solutions to the problem. The best of these can then be selected with $opt\ C$, where $C : P \leftarrow P$ is the global optimality criterion. Optimization problems that can be solved by greedy algorithms are thus specified in the following way:

$$opt\ C\ .\ \Lambda(rep\ S)$$

Thus, a statement that a greedy algorithm solves such an optimization problem can be expressed as

$$rep\ Greedy(L,S) \quad \subseteq \quad opt\ C\ .\ \Lambda(rep\ S)$$

The above models of greedy algorithm and optimization problem seem to be introduced the wrong way round, with the problem statement only being specified after the greedy algorithm to solve it! In practice, the algorithm designer will start with a problem statement looking like

$$opt\ C'\ .\ \Lambda Soln$$

and will then consider various ways in which solutions can be repeatedly constructed, that is, how $Soln$ can be expressed as $rep\ S$ for some relation $S : P \leftarrow P$. Different ways of expressing $Soln$ as a repetition will result in the consideration of different greedy algorithms for the problem. Once a repetition for $Soln$ is chosen, the input datatype for the problem will typically not be of the same datatype as that of partial solutions, P, so the input will need to be transformed into such a form. Also, the global optimality criterion will need to be expressed so that it is of type $P \leftarrow P$, and possibly the output may need to be extracted afterwards from the optimal completed solution (discarding data only used during the execution of the greedy algorithm). The problem statement is thus transformed to read

$$extract\ .\ opt\ C\ .\ \Lambda(rep\ S)\ .\ initialize$$

and the designer of greedy algorithms then concentrates further work on the portion $opt\ C\ .\ \Lambda(rep\ S)$.

There are some implicit conditions on L, C and S. Whilst in practice, relations L and C representing optimality criteria are often linear orderings, typically of the form $\leqslant_f$ for some cost function f, linearity is not necessary. It is required only that they are preorders (reflexive and transitive), and that they can be used to find an optimum in the context in which they are used. For example, the Marble Mingling problem as discussed in the previous section, has a greedy algorithm with a non-linear local optimality criterion.

Upon S there is an implicit condition that $rep\ S$ should terminate, otherwise the greedy algorithm may go for ever! Whilst of course it is important to ensure termination, this condition on S is not explicitly addressed in the theory: in practice, it is very simple to prove the termination of greedy algorithm implementations by existing methods (e.g. that of variants, see [33]), so no special theory is needed here.

Having modelled optimization problems and the greedy algorithms to solve them, the following sections look at the expression of the greedy principles in the relational calculus, exploring the relationships between C, S and L.

3.3. Four greedy principles

3.3.1. Best-Global

The following theorem expresses the Best-Global condition in the relational calculus:

Theorem 1. *Given $L, C, S : P \leftarrow P$, with L and C preorders, if the following conditions hold*:

$$Greedy(L,S)\Box \quad = \quad S\Box \tag{9}$$

$$Greedy(L,S) \;.\; (rep\ S)^o \quad \subseteq \quad (rep\ S)^o \;.\; C, \tag{10}$$

then

$$rep\ \ Greedy(L,S) \quad \subseteq \quad opt\ \ C \;.\; \Lambda(rep\ \ S)$$

Condition (9) concerns the feasibility of taking an optimum with respect to L, by insisting that whenever it is possible to perform a construction step (i.e. $S\Box$ holds), it is also possible to take an optimum of all such choices (i.e. $Greedy(L,S)\Box$ holds). The main Best-Global condition is (10). This insists that if a partial solution (say x) has a greedy step performed on it (say this produces the partial solution g), then for any other completion of x (to x', say):

$$g \xleftarrow{Greedy(L,S)} x \xrightarrow{rep\ \ S} x',$$

g can be completed to yield a better solution (g', say) than x':

$$g \xrightarrow{rep\ \ S} g' \xleftarrow{C} x'$$

In other words, taking a greedy step ultimately guarantees the possibility of arriving at a completed solution which is at least as good with respect to C as any other possible solution to the problem. See the Appendix for the proof.

Here is an example of a greedy algorithm to illustrate the Best-Global condition:

Prim's algorithm. It is possible to model the problem of finding a minimum cost spanning tree of a connected graph $G = (V,E)$ by

$$opt \leqslant_{cost} \;.\; \Lambda SpanningTree$$

The relation *SpanningTree* can be expressed as *rep S*, where

$$t \cup \{e\}\ \ S\ \ t, \quad \text{if } tree\ (t \cup \{e\}) \quad \wedge \quad e \in E \quad \wedge \quad e \notin t$$

The *cost* function is defined as

$$cost\ t\ =\ \sum_{e \in t}\ edgecost\ e$$

The greedy algorithm is then $rep\ Greedy(\leqslant_{cost}, S)$, which is given the partial solution $\{\ \}$ as input.

Condition (9) is trivially true. For condition (10), suppose that

$$t \cup \{e\} \xleftarrow{Greedy(\leqslant_{cost}, S)} t \xrightarrow{rep\ \ S} t \cup s$$

A completion of $t \cup \{e\}$ is needed, with no greater cost than $t \cup s$. If $e \in s$, then

$$t \cup \{e\} \xrightarrow{rep\ \ S} t \cup s \xleftarrow{\leqslant_{cost}} t \cup s$$

Otherwise, suppose $e = (u, v)$, with u being a node that already occurs within t, and v a new node. As $t \cup s$ is a spanning tree, it must contain some path from u to v, which must contain some edge $e' = (u', v')$ such that u' is within t, and v' is not. The edge e' is at least as costly as e, because of the greedy algorithm's choice. Thus $t \cup s \cup \{e\} - \{e'\}$ is also a spanning tree, and

$$t \cup \{e\} \xrightarrow{rep\ \ S} t \cup s \cup \{e\} - \{e'\} \xleftarrow{\leqslant_{cost}} t \cup s$$

Thus Prim's algorithm is correct, and satisfies the Best-Global condition.

3.3.2. Better-Global

The Better-Global Theorem is very similar to the Best-Global:

Theorem 2. *If $L, C, S : P \leftarrow P$, such that L and C are preorders, and the following conditions hold*:

$$Greedy(L, S)\Box\ =\ S\Box \tag{11}$$

$$L\ .\ (rep\ \ S)^o\ \subseteq\ (rep\ \ S)^o\ .\ C \tag{12}$$

then

$$rep\ \ Greedy(L, S)\ \subseteq\ opt\ \ C\ .\ \Lambda(rep\ \ S)$$

The only difference between Theorem 1 (Best-Global) and the above is the subtle change in the main condition from $Greedy(L, S)$ in (10) to L in (12). Rather than looking at the *best* with respect to L, of all possible local choices, this Better-Global condition looks at two of the possible alternatives, and requires that if one is at least as good as the other with respect to L, then ultimately, the better alternative can overall result in a better completion with respect to C.

This is a stronger requirement: indeed, the proof (given in the Appendix) directly shows that condition (12) implies (10).

An example of a greedy algorithm that satisfies the Better-Global principle is the following:

The Maximum Tardiness problem (see [4,21,31]) concerns ordering of jobs into a schedule, where n jobs are to be processed one after another on a single machine. Each job j takes time t_j to process, has a due time d_j, and a positive penalty factor p_j for late completion. If job j is scheduled to occur immediately after the processing of all the jobs in a set js, the completion time of the job is defined to be

$$time\ \ j\ \ js\ \ =\ \ t_j + \sum_{i \in js}\ \ t_i$$

The tardiness of the job is its lateness, weighted by its penalty factor:

$$tardiness\ \ j\ \ js\ \ =\ \ p_i(0\ \ \sqcup\ \ (time\ \ j\ \ js\ \ -\ \ d_j))$$

(where $\sqcup$ denotes maximum). Overall, the maximum tardiness for the whole schedule is mt, where

$$mt\ \ js\ \ =\ \ \sqcup_{0 \leqslant i < n}\ \ (tardiness\ \ j\ \ js[0 \ldots i-1])$$

It is desired to minimize the maximum tardiness for all the jobs, so the problem is specified

$$opt\ \ \leqslant_{mt}\ \ .\ \Lambda Schedule$$

To solve this problem, the well-known greedy algorithm of Lawler [31] allocates the jobs to a schedule in reverse order, repeatedly selecting the job with minimum tardiness when scheduled at the end of those jobs currently remaining.

Expressing this algorithm in relations, partial solutions will be of the form (js, s), where js is a bag of jobs remaining to be scheduled, and s is the schedule so far, listing the jobs to be performed after those in js. As discussed earlier, a schedule can be generated by $rep\ S$, where S is defined as

$$(js, [j] +\!\!+ s)\ \ S\ \ (js + \lfloor j \rfloor, s)$$

The local optimality criterion L prefers jobs with lesser tardinesses, so that

$$(js + \lfloor y \rfloor, [x] +\!\!+ s)\ \ L\ \ (js + \lfloor x \rfloor, [y] +\!\!+ s)$$

$$\equiv\ \ \ tardiness\ \ x\ \ (js + \lfloor y \rfloor) \leqslant tardiness\ \ y\ \ (js + \lfloor x \rfloor)$$

To show that the above theorem applies, first note that condition (11) is trivially true, as it is always possible to select a minimally tardy job when there are jobs to select from. For condition (12), consider the situation where

$$(js + \lfloor y \rfloor, [x] +\!\!+ s)\ \ \xleftarrow{L}\ \ (js + \lfloor x \rfloor, [y] +\!\!+ s)\ \ \xrightarrow{rep\ \ S}\ \ (\lfloor\ \ \rfloor, r +\!\!+ [y] +\!\!+ s)$$

If x is the same job as y, then trivially

$$(js + \lfloor y \rfloor, [x] +\!\!+ s)\ \ \xrightarrow{rep\ \ S}\ \ (\lfloor\ \ \rfloor, r +\!\!+ [x] +\!\!+ s)\ \ \xleftarrow{\leqslant_{mt}}\ \ (\lfloor\ \ \rfloor, r +\!\!+ [y] +\!\!+ s)$$

Otherwise, x must appear in the schedule $r +\!\!+[y] +\!\!+s$, which must be of the form $f +\!\!+[x] +\!\!+b +\!\!+[y] +\!\!+s$. It is claimed that

$$(js + \lfloor y \rfloor, [x] +\!\!+ s) \xrightarrow{rep\ \ S}$$
$$(\lfloor\ \rfloor, f +\!\!+ b +\!\!+ [y,x] +\!\!+ s) \xleftarrow{\leqslant_{mt}} (\lfloor\ \rfloor, f +\!\!+ [x] +\!\!+ b +\!\!+ [y] +\!\!+ s)$$

To justify this claim, note that in this suggested schedule $f +\!\!+ b +\!\!+ [y,x] +\!\!+ s$, compared to $r +\!\!+[y] +\!\!+s$, the jobs in f and s are unaffected as they occur at the same time as before. The jobs in $b +\!\!+[y]$ all occur earlier in the schedule, so if anything, their tardinesses are lessened. The only job that has moved later is x. But it is already known that $tardiness\ x(js + \lfloor y \rfloor) \leqslant tardiness\ y(js + \lfloor x \rfloor)$, and thus the maximum tardiness of the suggested schedule is no worse than that of $r +\!\!+[y] +\!\!+s$.

Thus the Maximum Tardiness problem yields a greedy algorithm that satisfies the Better-Global principle.

3.3.3. Best-Local

Theorem 3. *If $L, C, S : P \leftarrow P$, such that L and C are preorders, and the following conditions hold*:

$$Greedy(L,S)\square \ = \ S\square \tag{13}$$
$$\forall n \in \mathbb{N}\ .\ (Greedy(L,S))^n \ \subseteq \ opt\ L\ .\ \Lambda S^n \tag{14}$$
$$S \boxtimes\ .\ L\ .\ S^o \ \subseteq \ L \tag{15}$$
$$L\ .\ S\boxtimes \ \subseteq \ (rep\ S)^o\ .\ C \tag{16}$$

then

$$rep\ \ Greedy(L,S) \ \subseteq \ opt\ \ C\ .\ \Lambda(rep\ \ S)$$

Condition (13) is the same as before. Condition (14) is the main condition that expresses the notion of "best-locality" for uncompleted solutions, and the last two conditions are those that relate the local optimality criterion to the global optimality criterion for completed solutions. The translation of (15) is that if a completed solution is better than an uncompleted solution with respect to L, then performing a construction step on the uncompleted solution does not alter that relationship. Condition (16) expresses that if an uncompleted solution is better with respect to L than a completed solution, then there is a way of completing the uncompleted solution to give a better result overall (with respect to C).

The translation of the main condition (14) is that performing n greedy steps is optimal with respect to L, for any performing of n construction steps. Writing the condition in this form is clearer, but slightly clumsy to use. Following from properties of *opt* and Λ, it can be rewritten for ease of use:

$$(Greedy(L,S))^n\ .\ (S^n)^o \quad \subseteq \quad L$$

The following is an example of an algorithm which satisfies the Best-Local principle but not the Better-Local:

Kruskal's algorithm (see earlier). The specification of the problem of finding a spanning tree of minimum cost in a connected graph $G=(V,E)$ is the same as for Prim's algorithm, namely

$$opt \leqslant_{cost} \; . \; \Lambda SpanningTree$$

where

$$cost \;\; t \;\; = \;\; \sum_{e \in t} \;\; edgecost \;\; e$$

However, spanning trees are constructed differently in Kruskal's algorithm. Now the definition $SpanningTree = rep\ S$ is used, where

$$es \cup \{e\} \;\; S \;\; es, \quad \text{if } e \in E - es \quad \wedge \quad acyclic \;\; (es \cup \{e\})$$

The partial solution $\{\ \}$ is given as input. The global criterion C is $\leqslant_{cost}$; the local criterion L is essentially the same, but also needs to contain some context information (compared edge sets contain the same number of edges and are from the same graph):

$$\begin{aligned} es_1 \;\; L \;\; es_2 \equiv\; & cost \;\; es_1 \leqslant cost \;\; es_2 \\ & \wedge \quad |es_1| \;\; = \;\; |es_2| \\ & \wedge \quad es_1 \;\; \subseteq E \quad \wedge \quad acyclic \;\; es_1 \\ & \wedge \quad es_2 \;\; \subseteq E \quad \wedge \quad acyclic \;\; es_2 \end{aligned}$$

Condition (13) is trivially true. For (14), suppose that

$$es \cup \{e_1 \ldots e_n\} \xleftarrow{Greedy(L,S)^n} es \xrightarrow{S^n} es \cup \{e'_1 \ldots e'_n\}$$

where $e_1 \ldots e_n$ were added in that order, and $e'_1 \ldots e'_n$ are also labelled in non-decreasing order. Further define $ds = \{e_1 \ldots e_n\} - \{e'_1 \ldots e'_n\}$ and $ds' = \{e'_1 \ldots e'_n\} - \{e_1 \ldots e_n\}$. If $ds \neq \{\ \}$, let i be the lowest integer such that $e_i \in ds$. Adding e_i to the edge set $es \cup \{e'_1 \ldots e'_n\}$ may form a cycle. If so, remove from $es \cup \{e'_1 \ldots e'_n\}$ another edge from the cycle that is in ds'. If no cycle was formed, then remove any edge out of those in ds'. As by the greedy choice, the cost of edge e_i is no greater than any edge in ds', this edge swap does not increase the sum of the edge costs. Repeating this swap until $ds = \{\ \}$ results in the formation of $es \cup \{e_1 \ldots e_n\}$, along with the guarantee that its cost is no more than that of $es \cup \{e'_1 \ldots e'_n\}$. Thus condition (14) holds.

As for the last two conditions, first note that $es \in dom\ S$ is equivalent to $|es| = |V| - 1$. This makes (15) trivially true, as $S \boxtimes \;\; . \; L \; . \; S\Box = \{\ \}$, from the insistence in the definition of L that related sets are the same size. Likewise $S\Box \; . \; L \; . \; S \boxtimes = \{\ \}$, leaving only $S \boxtimes \;\; . \; L \; . \; S \boxtimes \subseteq (rep\ S)^o \; . \; C$ to check, which follows from $S \boxtimes \subseteq (rep\ S)^o$ and $L \subseteq C$ (for this problem).

Therefore Kruskal's Algorithm satisfies the Best-Local principle.

Other examples of greedy algorithms which satisfy the Best-Local principle included the Coin Changing problem (see earlier, also [7,8]), and Huffman Coding (see [23,25]).

3.3.4. Better-Local

Theorem 4. *If* $L, C, S : P \leftarrow P$, *such that* L *and* C *are preorders, and the following conditions hold*:

$$Greedy(L,S)\square \;=\; S\square \tag{17}$$

$$S\square \;.\; L \;.\; S^o \;\subseteq\; S^o \;.\; L \tag{18}$$

$$S\boxtimes \;.\; L \;.\; S^o \;\subseteq\; L \tag{19}$$

$$L \;.\; S\boxtimes \;\subseteq\; (rep\ S)^o \;.\; C \tag{20}$$

then

$$rep\ \ Greedy(L,S) \;\subseteq\; opt\ \ C \;.\; \Lambda(rep\ \ S)$$

Note that conditions (19) and (20), concerning completed solutions, are the same as for Theorem 3 (Best-Local). The only difference is condition (18), which is a straightforward monotonicity condition for non-completed solutions.

Any greedy algorithm satisfying the conditions of this theorem also satisfies those of Theorems 1 (Best-Global), 2 (Better-Global), and 3 (Best-Local) (see the Appendix for the proofs). Although this is the strongest of the theorems given, the monotonicity condition is one of the easiest to verify, for greedy algorithms which satisfy it. Here is an example:

Professor Midas' Driving problem (as promised earlier). Let the distances that the Professor has to travel between service stations be given as a list of strictly positive distances, *dists*, and the distance that can be travelled on a full tank of petrol be D. A plan of the journey can be represented by a partition of *dists*, generated by *rep S* from input $([\,], dists)$, where

$$(plan +\!\!+ [s], ds)\ \ S\ \ (plan, s +\!\!+ ds), \quad \text{if } s \neq [\,] \ \wedge\ sum\ \ s \leqslant D$$

The global criterion must reflect the Professor's preference for as few stops as possible, so

$$(plan_1, [\,])\ \ C\ \ (plan_2, [\,]) \;\equiv\; length\ \ plan_1 \leqslant length\ \ plan_2$$

and the Professor's problem is now modelled as $opt\ \ C \;.\; \Lambda(rep\ \ S)$.

The greedy algorithm (as discussed above) prefers going as far as possible before stopping for petrol, and thus

$$\begin{aligned}
&(plan_1, ds_1)\ \ L\ \ (plan_2, ds_2)\\
&\quad\equiv\; concat\ \ plan_1 +\!\!+\ ds_1 \;=\; concat\ \ plan_2 +\!\!+\ ds_2\\
&\qquad\wedge\quad length\ \ plan_1 \leqslant length\ \ plan_2\\
&\qquad\wedge\quad sum\ \ ds_1 \leqslant sum\ \ ds_2,
\end{aligned}$$

where the last line expresses the local optimality criterion in the form of having less far to go, and the first two conditions are just context information which have no effect on the making of the greedy choice.

In satisfying the conditions, (17) is trivially true. Turning to the monotonicity condition, suppose that

$$(plan_1, ds_1) \xleftarrow{L} (plan_2, s_2 +\!\!+ ds_2) \xrightarrow{S} (plan_2 +\!\!+ [s_2], ds_2)$$

with $ds_1 \neq [\,]$. As $(plan_1, ds_1)\ L\ (plan_2, s_2 +\!\!+ ds_2)$, ds_1 must be a suffix of $s_2 +\!\!+ ds_2$. If $length\, ds_1 \leqslant length\, ds_2$, then

$$(plan_1, ds_1) \xrightarrow{S} (plan_1 +\!\!+ [head\ \ ds_1], tail\ \ ds_1) \xleftarrow{L} (plan_2 +\!\!+ [s_2], ds_2)$$

Otherwise, let s_1 be a prefix of ds_1 such that $s_1 +\!\!+ ds_2 = ds_1$, and then

$$(plan_1, ds_1) \xrightarrow{S} (plan_1 +\!\!+ [s_1], ds_2) \xleftarrow{L} (plan_2 +\!\!+ [s_2], ds_2)$$

For condition (19), suppose that

$$(plan_1, [\,]) \xleftarrow{L} (plan_2, s_2 +\!\!+ ds_2) \xrightarrow{S} (plan_2 +\!\!+ [s_2], ds_2)$$

Then clearly $(plan_1, [\,])\ L\ (plan_2 +\!\!+ [s_2], ds_2)$, from the definition of L.

Finally, the betterment of a completed solution must be of the form $(plan_1, [\,])\ (L\ .\ S\boxtimes)(plan_2, [\,])$, and from the definitions of S and C, it is clear that

$$(plan_1, [\,]) \xrightarrow{rep\ \ S} (plan_1, [\,]) \xleftarrow{C} (plan_2, [\,])$$

from the context conditions of L.

3.4. Discussion of relational theory

The four greedy principles, as stated in Section 2, describe the full variety of greedy algorithms, in an informal way. Their expression in Section 3.3, using the relational calculus, is not so comprehensive, as the theorems are slightly simplified. This section discusses the use of relations to express greedy algorithms, and generalizes the theorems to yield a more comprehensive theory.

3.4.1. Use of relations

Relations express this greedy theory well. In the modelling of optimization problems, the finding of an optimum is a relation: there may well be many optima, just one, or none at all. Furthermore, when considering an optimality criterion, although optimization problems frequently use some cost function f which is to be minimized using $opt \leqslant_f$ (or maximized using $opt \geqslant_f$) sometimes a cost function is not appropriate. For example, the Marble Mingling problem from [11–13] has a greedy step which selects one marble from each of the k fullest jars. Expressing this choice in the form $opt \geqslant_f$ results in a contrived and awkward cost function f. Thus not all optimality criteria can be easily modelled using cost functions, and those greedy theories requiring the use of cost functions are unnecessarily restrictive.

In addition, there are frequently many ways to construct possible solutions to a problem, and relations model the choices involved well.

3.4.2. Context information

The theorems, as stated in Section 3.3, omit the use of context information, which would help satisfy the correctness conditions. When comparing possible alternatives in the greedy step, context information is known.

Firstly, these alternatives are not just any partial solutions: those compared in the greedy step have all emanated from the same partial solution, one construction step ago. To take account of this, the conditions for the two global greedy theorems could be re-written using L', where $L' = L \cap S.S^o$ instead of L. This has no effect on the Best-Global theorem, as $Greedy(L,S) = Greedy(L',S)$, but condition (12) now reads

$$(L \ \cap \ S.S^o) \ . \ (rep \ S)^o \ \subseteq \ (rep \ S)^o \ . \ C$$

(See [11] for more details.) This does not apply to the two *-Local theorems, as these require a more general L, able to compare two partial solutions that are not related by $S.S^o$. However, two partial solutions, when compared, will be related using $S^*.S^{*o}$, and so the *-Local theorems could be re-written to use different context information.

Secondly, more context information is known about the partial solution obtained after each greedy step: up to that point in the algorithm, the greedy step has been used repeatedly. The theorems as stated make no use of this information. Whilst it is possible to state this explicitly by adding the assertion $\Box Greedy(L,S)^*$, it is easier in practice to use an invariant. The invariant can assert whichever consequence of $\Box Greedy(L,S)^*$ is needed, and can usually be expressed in a simpler form than $\Box Greedy(L,S)^*$. The generalized version of Theorem 1 (Best-Global) is

Theorem 5. *Given $L, C, S, I : P \leftarrow P$, with L and C preorders, and I coreflexive, if the following conditions hold*

$$S\Box \ \cap \ I \quad \subseteq \quad Greedy(L,S)\Box \tag{21}$$

$$Greedy(L,S) \ . \ I \quad \subseteq \quad I \ . \ Greedy(L,S) \tag{22}$$

$$Greedy(L,S) \ . \ I \ . \ (rep \ S)^o \quad \subseteq \quad (rep \ S)^o \ . \ C, \tag{23}$$

then

$$rep \ Greedy(L,S) \ . \ I \quad \subseteq \quad opt \ C \ . \ \Lambda(rep \ S)$$

This theorem is proved in [11]. A similar generalization can be applied to the Best-Local theorem, however the use of invariants is not suitable for the Better-* theorems, as they concern the comparison of partial solutions for which $\Box Greedy(L,S)^*$ does not apply.

An example of a greedy algorithm which requires Theorem 5 is the Dartboards algorithm from [11].

There may also be other context information necessary to prove the conditions of the theorems, however it is a trade-off between putting the context explicitly in the

theorem conditions (and thus obfuscating the theorems by giving multiple versions of them with extensive conditions), or having to explicitly state context information in the definition of the relation L for use in a particular problem. In this paper, the examples have explicit context information.

3.4.3. Alternative proof conditions

The theorems as given are not the only possible translation of the greedy principles into the relational calculus, as the proof conditions can be expressed slightly differently.

For example, the main condition for Theorem 1 (Best Global) is

$$Greedy(L,S)\ .\ (rep\ \ S)^o \quad \subseteq \quad (rep\ \ S)^o\ .\ C$$

and an alternative phrasing is

$$rep\ \ Greedy(L,S)\ .\ (rep\ \ S)^o \quad \subseteq \quad C$$

Under the conditions of Theorem 1 (Best-Global), these two inequalities are equivalent (proof omitted). Another example of an alternative proof condition was given already for Theorem 3; there are various other possible rephrasings of the main conditions of the theorems, for example the conditions in the *-Local theorems expressing the relationship between L and C. Thus, these translations of the principles into relations are not unique.

3.4.4. Characterization of completed solutions

The given relational model constructs potential solutions with *rep S*, and makes the assumption that the completion of a solution is precisely denoted by $S\boxtimes$. This is the case for most greedy algorithms, but not all. However, adaptation is relatively easy, as shown in the following paragraphs.

Some problems have partial solutions which are in *dom S*, and which could be regarded as potential solutions to the optimization problem, but the global optimality criterion is the maximization of the number of construction steps, so solutions in *dom S* are of no interest, and no adaptation is necessary. Examples include the Marble Mingling [11–13] and Activity Selection [10] problems.

In contrast, some problems have potential "completed" solutions which are in *dom S*, but the global optimality criterion is the minimization of the number of construction steps. Such a problem uses a feasibility predicate to characterize potential solutions correctly, and the construction step can then be altered to use this predicate. Examples of such problems include that of the Rally Driver [11], and the following:

Knuth's TeX problem (see [26]) concerns the conversion of an integer multiple of $1/2^{16}$ to a reasonably accurate decimal fraction with as few digits as possible. For example, 0.49999 is a better fraction approximating to 1/2 than 0.499999999000 is. If potential solutions are constructed by adding feasible digits from left to right after the decimal point, then potential solutions are not characterized by not being in the domain of the construction relation, as any feasible digit sequence may always have a 0 added to the end. Instead, as this global criterion insists on the minimum number of construction steps, for Knuth's problem, the characterization of potential solutions

is that they represent the given fraction to the desired accuracy. Thus the construction step is altered from "Append a digit that may result in a feasible solution", to "Append a digit that may result in a feasible solution, if the sequence of digits so far does not already represent a feasible solution". (By the way, the greedy algorithm to solve the problem, whenever there is a choice of digits to add, adds the larger. So, for example, this would lead to 0.5 as being a better representation for $1/2$ than 0.49999.)

In general, whilst using *rep* makes the theorems look simpler, an operator such as *until* expresses some problems more accurately, where $until(p,S) = p? \,.\, S$, with $p?$ a coreflexive characterizing completed solutions. It is possible to incorporate this generalization into the greedy theorems as given, as under certain (reasonable) conditions, $until(p,S)$ can be re-expressed as $rep(S \,.\, (\neg p)?)$ which can be used instead in the greedy theorems instead of *rep S*. Thus problems where *dom S* does not coincide with the set of potential solutions, can still be solved within this theory. See [11] for further details.

4. Conclusions

4.1. Greedy theories

As stated earlier, the goals for the work in this paper were (in order):

(1) Coverage of all greedy algorithms that solve optimization problems
(2) Characterization of greedy structures
(3) Correctness proofs and design inspiration for algorithm development
(4) Expressing greedy algorithms simply

In existing greedy theories, there is a trade-off: the more specific a theory is about structure, the fewer greedy algorithms the theory covers. This seems to be true when characterizing the structure of both greedy algorithms and datatypes for which the greedy algorithm works. This trade-off is illustrated by the developments from matroid to greedoid theory: greedoids generalize matroids, modelling more greedy algorithms as a result.

Given this trade-off, a higher level of abstraction was used in an attempt to attain the goal of covering all greedy algorithms that solve optimization problems. This abstraction inevitably leads to consequences for an algorithm designer: being less specific about data structures gives the algorithm designer less detailed guidance. This also means that the algorithm designer is not led away from a correct route (leading to a greedy algorithm) by overspecific guidance. In addition, if the abstraction leads to a theory covering *all* greedy algorithms, the algorithm designer can feel happier using that theory because it is more comprehensive.

4.2. Theory in Sections 2 and 3

The theory presented in this paper has been tested practically by its application to the author's extensive collection of greedy algorithms that solve optimization problems

(a report of which, unfortunately, the margin is too small to contain). The given theory appears to meet the above goals.

Greedy algorithms are modelled with two key elements: a local optimality criterion, and a construction step. This fits in well with the given model of optimization problems, for which the key elements are the global optimality criterion and potential solutions (constructed by repeating the construction step). This modelling works both for the informal description of the greedy principles, and for their explicit formalization as relations, which model well the non-determinism involved. The relational expression of the greedy algorithm $rep\ Greedy(L, S)$ is also reassuringly simple, given the simplicity of the concept of greedy algorithms.

Concerning the characterization of greedy algorithms according to the four principles, all greedy algorithms known by the author fit into one of the five classifications. This is perhaps hardly surprising, since the Best-Global principle is really *the* condition that says that the greedy algorithm works in the first place. Given this fact, it could be said that the Best-Global principle does not say anything interesting, as it is obviously true for all greedy algorithms. However, the Best-Global principle is indeed of interest, as it helps distinguish between a greedy algorithm belonging to the weaker "satisfies Best-Global only" class of greedy algorithms, and one belonging to a class for which a stronger principle holds.

The revelation that greedy algorithms have differing relationships between the optimality criteria and the construction step, the expression of these relationships in the four greedy principles, and the diamond-shaped implications between the principles, these all provide characterization of greedy structures, one of the goals for this work. In the author's opinion, these principles elegantly capture the fundamental relationships between the components of greedy algorithms, highlighting the similarities and contrasts between different greedy algorithms. These relationships have not been covered in depth before in the literature: most formalisms barely even acknowledge the possibility of different local and global optimality criteria, yet, within the author's collection of greedy algorithms, those which can be expressed with the local criterion the same as the global, are in a minority. Bird and de Moor (see [2–4,6]) partly investigated these relationships, but their work leads to a restricted view: the compliance of any anamorphically expressed greedy algorithm with either of the *-Local principles is not visible, and, dually, neither is the compliance of any catamorphically expressed algorithm with the *-Global principles.

Furthermore, these four principles help the algorithm designer. Whichever formalism the designer is working in, this theory gives four possible ways to attempt a proof of the algorithm's correctness. Some are easier to prove than others for particular problems, so having the choice is potentially helpful. In addition, the explicit formalism in the relational calculus models the construction step as $rep\ S$, and this can help to suggest first steps in the development of a greedy algorithm. As mentioned before, the relational calculus is not central to the main ideas here, but it competently expresses optimization problems and their solution by greedy algorithms. Relations capture the non-determinism implicit in many greedy algorithms, and the use of relations to specify optimality criteria, rather than objective functions, allows the theory to be more inclusive.

One area not addressed by this work, is that of datatypes. Here, no help is given to the algorithm designer, and this work doesn't offer any insight into greedy structures beyond the relationships given in the four greedy principles. However, as discussed, this was felt to be a necessary trade-off in order to achieve sufficient generality to include *all* greedy algorithms that solve optimization problems. Other theories use more specific data structures and the class of greedy algorithms covered is not as inclusive.

Having met the above goals, it remains to be seen whether these results have more general applications, and future work will focus on the application of greedy algorithms to non-optimization problems, and problems for which greedy algorithms do not produce an optimal solution, only an approximation.

Acknowledgements

Thanks are due to Charles Bryant, for useful comments and suggestions.

Appendix

Proof of Theorem 1 (Best-Global): Writing $G = Greedy(L,S)$ for brevity,

$$
\begin{array}{ll}
& rep\ G \subseteq opt\ C\ .\ \Lambda(rep\ S) \\
\Leftarrow & \{\text{Repetition}\} \\
& G \boxtimes \cup\ opt\ C\ .\ \Lambda(rep\ S)\ .\ G \subseteq opt\ C\ .\ \Lambda(rep\ S) \\
\equiv & \{\text{Optimum, Intersection}\} \\
& G \boxtimes \cup\ (rep\ S \cap C/(rep\ S)^o)\ .\ G \subseteq rep\ S \\
& \wedge\ G \boxtimes \cup\ (rep\ S \cap C/(rep\ S)^o)\ .\ G \subseteq C/(rep\ S)^o \\
\Leftarrow & \{\text{Intersection, Converse, Union}\} \\
& G \boxtimes \cup\ (rep\ S)\ .\ G \subseteq rep\ S \\
& \wedge\ G \boxtimes \cup\ (C/(rep\ S)^o)\ .\ G \subseteq C/(rep\ S)^o \\
\Leftarrow & \{\text{Condition (9), Definition of } Greedy(L,S)\} \\
& S \boxtimes \cup\ (rep\ S)\ .\ opt\ L\ .\ \Lambda S \subseteq rep\ S \\
& \wedge\ S \boxtimes \cup\ (C/(rep\ S)^o)\ .\ G \subseteq C/(rep\ S)^o \\
\Leftarrow & \{\text{Optimum, Intersection, Quotient}\} \\
& S \boxtimes \cup\ (rep\ S)\ .\ S \subseteq rep\ S
\end{array}
$$

$$\wedge \quad S \boxtimes \ \cup (C/(rep\ S)^o)\ .\ G\ .\ (rep\ S)^o \quad \subseteq \quad C$$

$$\Leftarrow \quad \{\text{Repetition, Condition (10)}\}$$

$$S \boxtimes \ \cup (C/(rep\ S)^o)\ .\ (rep\ S)^o\ .\ C \quad \subseteq \quad C$$

$$\Leftarrow \quad \{\text{Domains, Quotient}\}$$

$$Id \ \cup \ C\ .\ C \quad \subseteq \quad C$$

$$\equiv \quad \{C \text{ is a preorder}\}$$

$$true$$

□

Proof of Theorem 2 (Better-Global): It suffices to demonstrate condition (10) of Theorem 1:

$$Greedy(L,S)\ .\ (rep\ S)^o$$

$$= \quad \{\text{Domains, Condition (11)}\}$$

$$Greedy(L,S)\ .\ S\Box\ .\ (rep\ S)^o$$

$$= \quad \{\text{Repetition, Converse}\}$$

$$Greedy(L,S)\ .\ S^o\ .\ (rep\ S)^o$$

$$\subseteq \quad \{\text{Definition of } Greedy, \text{ Intersection}\}$$

$$L/S^o\ .\ S^o\ .\ (rep\ S)^o$$

$$\subseteq \quad \{\text{Quotient}\}$$

$$L\ .\ (rep\ S)^o$$

$$\subseteq \quad \{\text{Condition (12)}\}$$

$$(rep\ S)^o\ .\ C$$

□

Proof of Theorem 3 (Best-Local): The proof established condition (10) of Theorem 1 (Best-Global)

Writing $G = Greedy(L,S)$ for brevity, suppose $a(G\ .\ (rep\ S)^o)b$. It will be shown that $a((rep\ S)^o\ .\ C)b$.

As $a(G \,.\, (rep\ S)^o)b$, $\exists c \,.\, a \overset{G}{\leftarrow} c \xrightarrow{rep\ S} b$, and furthermore $\exists n \in \mathbb{N} \,.\, c \xrightarrow{S^n \,.\, S\boxtimes} b$. Either $a \in dom\ G^{n-1}$ or not. If not, let m be the least integer in the range $1 \ldots n$ such that $a \in dom\ G^{m-1}$ and $a \notin dom\ G^m$. It is established that either $a(G^{n-1}\Box \,.\, G \,.\, (S^n)^o \,.\, S\boxtimes)b$ or $a(G \boxtimes \,.\, G^{m-1})\Box \,.\, G \,.\, (S^n)^o \,.\, S\boxtimes)b$. Calculation gives that

$$
\begin{array}{ll}
& G^{n-1}\Box \,.\, G \,.\, (S^n)^o \,.\, S \boxtimes \\
\cup & (G \boxtimes \,.\, G^{m-1})\Box \,.\, G \,.\, (S^n)^o \,.\, S \boxtimes \\
\subseteq & \{\text{Domains, Converse}\} \\
& (G^{n-1})^o \,.\, G^{n-1} \,.\, G \,.\, (S^n)^o \,.\, S \boxtimes \\
\cup & (G^{m-1})^o \,.\, G \boxtimes \,.\, G \boxtimes \,.\, G^{m-1} \,.\, G \,.\, (S^n)^o \,.\, S \boxtimes \\
\subseteq & \{\text{Composition, Definition of } \textit{Greedy}\text{, Condition (13)}\} \\
& (S^{n-1})^o \,.\, G^n \,.\, (S^n)^o \,.\, S \boxtimes \\
\cup & (S^{m-1})^o \,.\, S \boxtimes \,.\, S \boxtimes \,.\, G^m \,.\, (S^n)^o \,.\, S \boxtimes \\
\subseteq & \{\text{Condition (14), Optimum, Composition}\} \\
& (S^{n-1})^o \,.\, L/(S^n)^o \,.\, (S^n)^o \,.\, S \boxtimes \\
\cup & (S^{m-1})^o \,.\, S \boxtimes \,.\, S \boxtimes \,.\, L/(S^m)^o \,.\, (S^m)^o \,.\, (S^{n-m})^o \,.\, S \boxtimes \\
\subseteq & \{\text{Quotient}\} \\
& (S^{n-1})^o \,.\, L \,.\, S \boxtimes \\
\cup & (S^{m-1})^o \,.\, S \boxtimes \,.\, S \boxtimes \,.\, L \,.\, (S^{n-m})^o \,.\, S \boxtimes \\
\subseteq & \{\text{Repetition}\} \\
& (S^{n-1})^o \,.\, L \,.\, S \boxtimes \\
\cup & (S^{m-1})^o \,.\, S \boxtimes \,.\, S \boxtimes \,.\, L \,.\, (rep\ S)^o \\
\subseteq & \{\text{Condition (16), Claim}\} \\
& (S^{n-1})^o \,.\, (rep\ S)^o \,.\, C \\
\cup & (S^{m-1})^o \,.\, S \boxtimes \,.\, C \\
\subseteq & \{\text{Repetition}\} \\
& (rep\ S)^o \,.\, C
\end{array}
$$

The claim above is that $S \boxtimes \ . L \ . (rep\ S)^o \subseteq C$, which is proved:

$$
\begin{array}{ll}
S \boxtimes \ . L \ . (rep\ S)^o & \\
= & \{\text{Repetition}\} \\
\quad S \boxtimes \ . L \ . (rep\ S)^o \ . S \boxtimes & \\
\subseteq & \{\text{Claim}\} \\
\quad S \boxtimes \ . L \ . S \boxtimes & \\
\subseteq & \{\text{Condition (16)}\} \\
\quad S \boxtimes \ . (rep\ S)^o \ . C & \\
= & \{\text{Repetition}\} \\
\quad S \boxtimes \ . C & \\
\subseteq & \{\text{Domains}\} \\
\quad C &
\end{array}
$$

The remaining claim is that $S \boxtimes \ . L \ . (rep\ S)^o \subseteq S \boxtimes \ . L$, which is proved:

$$
\begin{array}{ll}
S \boxtimes \ . L \ . (rep\ S)^o \quad \subseteq \quad S \boxtimes \ . L & \\
\equiv & \{\text{Converse}\} \\
\quad rep\ S \ . L^o \ . S \boxtimes \quad \subseteq \quad L^o \ . S \boxtimes & \\
\equiv & \{\text{Quotient}\} \\
\quad rep\ S \quad \subseteq \quad (L^o \ . S \boxtimes)/(L^o \ . S \boxtimes) & \\
\Leftarrow & \{\text{Repetition, Union}\} \\
\quad S \boxtimes \quad \subseteq \quad (L^o \ . S \boxtimes)/(L^o \ . S \boxtimes) & \\
\quad \wedge \quad (L^o \ . S \boxtimes)/(L^o \ . S \boxtimes) \ . S \quad \subseteq \quad (L^o \ . S \boxtimes)/(L^o \ . S \boxtimes) & \\
\equiv & \{\text{Quotient}\} \\
\quad S \boxtimes \ . L^o \ . S \boxtimes \quad \subseteq \quad L^o \ . S \boxtimes & \\
\quad \wedge \quad (L^o \ . S \boxtimes)/(L^o \ . S \boxtimes) \ . S \ . L^o \ . S \boxtimes \quad \subseteq \quad L^o \ . S \boxtimes &
\end{array}
$$

$$
\begin{array}{ll}
\equiv & \{\text{Domains}\} \\
 & (L^o \;.\; S\boxtimes)/(L^o \;.\; S\boxtimes) \;.\; S \;.\; L^o \;.\; S\boxtimes \;.\; S\boxtimes \quad \subseteq \quad L^o \;.\; S\boxtimes \\
\Leftarrow & \{\text{Condition (15)}\} \\
 & (L^o \;.\; S\boxtimes)/(L^o \;.\; S\boxtimes) \;.\; L^o \;.\; S\boxtimes \quad \subseteq \quad L^o \;.\; S\boxtimes \\
\equiv & \{\text{Quotient}\} \\
 & \mathit{true}
\end{array}
$$

□

Proof of Theorem 4 (Better-Local): It suffices to show condition (12) of Theorem 2:

$$
\begin{array}{ll}
 & L \;.\; (\mathit{rep}\ S)^o \quad \subseteq \quad (\mathit{rep}\ S)^o \;.\; C \\
\equiv & \{\text{Converse, Quotient}\} \\
 & \mathit{rep}\ S \quad \subseteq \quad (C^o \;.\; \mathit{rep}\ S)/L^o \\
\Leftarrow & \{\text{Repetition}\} \\
 & S\boxtimes \;\cup\; (C^o \;.\; \mathit{rep}\ S)/L^o \;.\; S \quad \subseteq \quad (C^o \;.\; \mathit{rep}\ S)/L^o \\
\equiv & \{\text{Union, Quotient}\} \\
 & S\boxtimes \;.\; L^o \quad \subseteq \quad C^o \;.\; \mathit{rep}\ S \\
 & \wedge \quad (C^o \;.\; \mathit{rep}\ S)/L^o \;.\; S \;.\; L^o \quad \subseteq \quad C^o \;.\; \mathit{rep}\ S \\
\equiv & \{\text{Converse, Condition (20)}\} \\
 & (C^o \;.\; \mathit{rep}\ S)/L^o \;.\; S \;.\; L^o \quad \subseteq \quad C^o \;.\; \mathit{rep}\ S \\
\equiv & \{\text{Domains, Union}\} \\
 & (C^o \;.\; \mathit{rep}\ S)/L^o \;.\; S \;.\; L^o \;.\; S\Box \quad \subseteq \quad C^o \;.\; \mathit{rep}\ S \\
 & \wedge \quad (C^o \;.\; \mathit{rep}\ S)/L^o \;.\; S \;.\; L^o \;.\; S\boxtimes \quad \subseteq \quad C^o \;.\; \mathit{rep}\ S \\
\equiv & \{\text{Converse, Conditions (18), (19)}\} \\
 & (C^o \;.\; \mathit{rep}\ S)/L^o \;.\; L^o \;.\; S \quad \subseteq \quad C^o \;.\; \mathit{rep}\ S \\
 & \wedge \quad (C^o \;.\; \mathit{rep}\ S)/L^o \;.\; L^o \quad \subseteq \quad C^o \;.\; \mathit{rep}\ S
\end{array}
$$

$$
\begin{array}{ll}
\Leftarrow & \{\text{Quotient}\} \\
& C^o \ . \ rep \ S \ . \ S \quad \subseteq \quad C^o \ . \ rep \ S \\
& \wedge \quad C^o \ . \ rep \ S \quad \subseteq \quad C^o \ . \ rep \ S \\
\Leftarrow & \{\text{Repetition}\} \\
& true
\end{array}
$$

□

Proof that Better-Local implies Best-Local: It suffices to show that the conditions of Theorem 4 (Better-Local) imply condition (14) of Theorem 3 (Best-Local), and the proof is by induction on n. For $n = 0, 1$ the inequality is trivially true, and for the inductive case:

$$
\begin{array}{ll}
& (opt \ L \ . \ \Lambda S)^{n+1} \quad \subseteq \quad opt \ L \ . \ \Lambda S^{n+1} \\
\equiv & \{\text{Optimum, Intersection}\} \\
& (opt \ L \ . \ \Lambda S)^{n+1} \quad \subseteq \quad S^{n+1} \\
& \wedge \quad (opt \ L \ . \ \Lambda S)^{n+1} \quad \subseteq \quad L/(S^{n+1})^o \\
\equiv & \{\text{Quotient, Composition}\} \\
& (opt \ L \ . \ \Lambda S)^{n+1} \quad \subseteq \quad S^{n+1} \\
& \wedge \quad opt \ L \ . \ \Lambda S \ . \ (opt \ L \ . \ \Lambda S)^n \ . \ (S^n)^o \ . \ S^o \quad \subseteq \quad L \\
\Leftarrow & \{\text{Optimum, Induction Hypothesis}\} \\
& (S \ \cap \ L/S^o)^{n+1} \quad \subseteq \quad S^{n+1} \\
& \wedge \quad (S \ \cap \ L/S^o) \ . \ opt \ L \ . \ \Lambda S^n \ . \ (S^n)^o \ . \ S^o \quad \subseteq \quad L \\
\Leftarrow & \{\text{Intersection}\} \\
& (S \ \cap \ L/S^o) \ . \ opt \ L \ . \ \Lambda S^n \ . \ (S^n)^o \ . \ S^o \quad \subseteq \quad L \\
\Leftarrow & \{\text{Domains, Optimum}\} \\
& (S \ \cap \ L/S^o) \ . \ S\Box \ . \ (S \ \cap \ L/(S^n)^o) \ . \ (S^n)^o \ . \ S^o \quad \subseteq \quad L \\
\Leftarrow & \{\text{Intersection, Quotient}\} \\
& L/S^o \ . \ S\Box \ . \ L \ . \ S^o \quad \subseteq \quad L
\end{array}
$$

$$\Leftarrow \quad \{\text{Condition (18)}\}$$

$$L/S^o \;.\; S^o \;.\; L \quad \subseteq \quad L$$

$$\Leftarrow \quad \{\text{Quotient, Transitivity of } L\}$$

$$\mathit{true}$$

□

References

[1] R.S. Bird, The smallest upravel, Sci. Comput. Programming 18 (1992) 281–292.

[2] R.S. Bird, O. de Moor, Between dynamic programming and greedy: data compression, 1992. Available at: http://web.comlab.ox.ac.uk/oucl/research/areas/ap/papers/compact.ps.gz.

[3] R.S. Bird, O. de Moor, Solving optimization problems with catamorphisms, in: Mathematics of Program Construction, Springer Lecture Notes in Computer Science, Vol. 669, Springer, Berlin, 1993.

[4] R.S. Bird, O. de Moor, From dynamic programming to greedy algorithms, in: B. Möller, H. Partsch, S. Schuman (Eds.), Formal Program Development, Lecture Notes in Computer Science, Vol. 755, 1993, pp. 43–61.

[5] R.S. Bird, O. de Moor, List partitions, Formal Aspects Comput. 5 (1) (1993) 61–78.

[6] R.S. Bird, O. de Moor, The Algebra of Programming, Prentice-Hall, Englewood Cliffs, NJ, 1997.

[7] L. Chang, J.F. Korsh, Canonical coin changing and greedy solutions, J. Assoc. Comput. Machinery 23 (3) (1976) 418–422.

[8] S.K. Chang, A. Gill, Algorithmic solution of the change-making problem, J. ACM 17 (1) (1970) 113–122.

[9] B. Charlier, The greedy algorithms class: formalization, synthesis and generalization. Available at: ftp://ftp.info.ucl.ac.be/pub/reports/95/rr95-11.ps.gz.

[10] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA, 1990.

[11] S. Curtis, A relational approach to optimization problems, D.Phil. Thesis, Technical Monograph PRG-122, Oxford University Computing Laboratory, 1996.

[12] S.A. Curtis, Laziness, drugs and jam jars, in: Draft Proc. 3rd Scottish Functional Programming Workshop, Stirling, 2001.

[13] S.A. Curtis, Marble mingling, J. Funct. Programming, to appear.

[14] E.W. Dijkstra, W. Feijen, A Method of Programming, Addison-Wesley, Reading, MA, 1998.

[15] J. Edmonds, Matroids and the greedy algorithm, Math. Programming 1 (1971) 126–136.

[16] A.M. Frieze, W. Szpankowski, Greedy algorithms for the shortest common superstring that are asymptotically optimal, in: European Symp. on Algorithms, Lecture Notes in Computer Science, Vol. 1136, Springer, Berlin, 1996, pp. 194–207.

[17] L.J. Guibas, J.E. Hershberger, J.S.B. Mitchell, J.S. Snoeyink, Approximating polygons and subdivisions with minimum link paths, in: ISAAC: 2nd Internat. Symp. on Algorithms and Computation (formerly SIGAL International Symposium on Algorithms), Organized by Special Interest Group on Algorithms (SIGAL) of the Information Processing Society of Japan (IPSJ) and the Technical Group on Theoretical Foundation of Computing of the Institute of Electronics, Information and Communication Engineers (IEICE), 1991.

[18] P. Helman, A theory of greedy structures based on k-ary dominance relations, Technical Report CS89-11, Department of Computer Science, University of New Mexico, 1989.

[19] P. Helman, B. Moret, H. Shapiro, An exact characterization of greedy structures, SIAM J. Discrete Math. 6 (2) (May 1993) 274–283.

[20] M. Herbordt, J. Corbett, C. Weems, J. Spalding, Practical algorithms for online routing on SIMD meshes, Technical Report UM-CS-1991-063, Department of Computer Science, University of Massachusetts, 1991.
[21] D.S. Hochbaum, R. Shamir, An $O(n \log^2 n)$ algorithm for the maximum weighted tardiness problem, Inform. Process. Lett. 31 (1989) 215–219.
[22] E. Horowitz, S. Sahni, S. Rajasekaran, Computer Algorithms, Computer Science Press, Rockville, MD, 1998.
[23] D.A. Huffman, A method for the construction of minimum redundancy codes, Proc. IRE 40 (1952) 1098–1101.
[24] V. Jarník, O jistem problemu minimalnim, Praca Moravske Prirodovedecke Sploecnosti 6 (1930) 57–63.
[25] J.D. Kececioglu, E.W. Myers, Combinatorial algorithms for DNA sequence assembly, Algorithmica 13 (1/2) (1995) 7–51.
[26] D.E. Knuth, A simple program whose proof isn't, in: Beauty is our Business, Springer, New York, 1990.
[27] B. Korte, L. Lovász, Mathematical structures underlying greedy algorithms, in: Fundamentals of Computation Theory, Lecture Notes in Computer Science, Vol. 117, Springer, Berlin, 1981, pp. 205–209.
[28] B. Korte, L. Lovász, Greedoids and linear objective functions, SIAM J. Algebraic Discrete Methods 5 (1984) 229–238.
[29] B. Korte, L. Lovász, R. Schrader, Greedoids, Springer, Berlin, 1991.
[30] J.B. Kruskal Jr., On the shortest spanning subtree of a graph and the traveling salesman problem, Proc. Amer. Math. Soc. 7 (1) (1956) 48–50.
[31] E. Lawler, Optimal sequencing of a single machine subject to precedence constraints, Management Sci. 19 (5) (1973) 544–546.
[32] L. Meertens, Algorithmics—towards programming as a mathematical activity, in: J.W. de Bakker, M. Hazewinkel, J.K. Lenstra (Eds.), Mathematics and Computer Science, CWI Monographs, Vol. 1, North-Holland, Amsterdam, 1986, pp. 289–334.
[33] C. Morgan, Programming From Specifications, 2nd Edition, Prentice-Hall, Englewood Cliffs, NJ, 1994.
[34] R.C. Prim, Shortest connection networks and some generalizations, Bell System Technical J. 36 (1957) 1389–1401.
[35] E.S. Schwartz, An optimal encoding with minimum longest code and total number of digits, Inform. Control 7 (1) (1964) 37–44.
[36] R.A. Wagner, Common phrases and minimum-space test storage, Comm. ACM 16 (3) (1973) 148–152.
[37] H. Whitney, On the abstract properties of linear dependence, Amer. J. Math. 57 (1935) 509–533.
[38] H. Zantema, Longest segment problems, Sci. Comput. Programming 18 (1992) 39–66.