A Class of Greedy Algorithms And Its Relation to Greedoids

Srinivas Nedunuri
Dept. of Computer Sciences
University of Texas at Austin
nedunuri@cs.utexas.edu
, Douglas R. Smith
Kestrel Institute
smith@kestrel.edu
, and William R. Cook
Dept. of Computer Sciences
University of Texas at Austin
cook@cs.utexas.edu

No Institute Given

Abstract. We define a class of greedy algorithms as a specialization of a form of Branch and Bound called Global Search. We show that our class generalizes a well-known characterization of greedy problems called greedoids, which are themselves a generalization of matroids. Finally, we derive a characteristic recurrence from a statement of optimality, which can then be transformed into a program for our greedy class, analogous to the Greedy Algorithm for matroids and greedoids.

1 Introduction

A greedy algorithm repeatedly makes a locally optimal choice. For some problems this can lead to a globally optimal solution. In addition to developing individual greedy algorithms, there has been long-term interest in finding a general characterization of greedy algorithms that highlights their common structure. Edmonds [Edm71] characterized greedy algorithms in terms of *matroids*. In 1981, Korte and Lovasz generalized matroids to define *greedoids*, [KLS91]. The question of whether a greedy algorithm exists for a particular problem reduces to whether there exists a translation of the problem into a matroid/greedoid. However, there are several problems for which a matroid/greedoid formulation either does not exist or is very difficult to construct. For example, no known greedoid formulations exist for problems such as Huffman Prefix-free encoding or Activity Selection, [CLRS01].

An alternative approach to constructing algorithms is to take a very general program schema and specialize it with problem-specific information. The result can be a very efficient algorithm for the given problem, [SPW95,NC09]. One such class of algorithms, Global Search [Smi88], operates by controlled search, where

at each level in the search tree there are a number of choices to be explored. Under certain conditions, this collection of choices reduces to a single locally optimal choice, which is the essence of a greedy algorithm. In this paper we axiomatically characterize those conditions. We call our specialization of Global Search *Greedy Global Search (GGS)*. We also show that this characterization of greedy algorithms generalizes greedoids, and therefore also matroids. Our proof does not rely on any particular algorithm, such as the greedy algorithm, but is based solely on the properties of greedoid theory and GGS theory. Finally, we derive a recurrence equation from the statement of correctness of GGS which can be transformed into an executable program through correctness-preserving program transformations. Such a program plays the same role for GGS theory as the Greedy Algorithm does for greedoids.

2 Background

2.1 Specifications and Morphisms

We briefly review some of the standard terminology and definitions from algebra. A *signature* $\Sigma = (S, \mathcal{F})$ consists of a set of sort symbols S and a family $\mathcal{F} = \{F_{v,s}\}$ of finite disjoint sets indexed by $S^* \times S$, where $F_{v,s}$ is the set of operation symbols of rank (v, s). We write $f : v \rightarrow s$ to denote $f \in F_{v,s}$ for $v \in S^*, s \in S$ when the signature is clear from context. For any signature Σ the Σ-terms are inductively defined in the usual way as the well-sorted composition of operator symbols and variables. A Σ-formula is a boolean valued term built from Σ-terms and the quantifiers $\forall$ and $\exists$. A Σ-sentence is a closed Σ-formula. A *specification* $T = \langle S, \mathcal{F}, A\rangle$ comprises a signature $\Sigma = (S, \mathcal{F})$ and a set of Σ-sentences A called *axioms*. The generic term *expression* is used to refer to a term, formula, or sentence. A specification $T' = \langle S', \mathcal{F}', A'\rangle$ *extends* $T = \langle S, \mathcal{F}, A\rangle$ if $S \subseteq S', F_{v,s} \subseteq F'_{v,s}$ for every $v \in S^*, s \in S$, and $A \subseteq A'$. Alternatively, we say that T' is an extension of T. A model for T is a structure for $(S, \mathcal{F})$ that satisfies the axioms.We shall use modus ponens, substitution of equals/equivalents, and other natural rules of inference in T. The *theory* of T is the set of sentences closed under the rules of inference from the axioms of T. We shall sometimes loosely refer to T as a theory. A sentence s is a *theorem* of T, written $T \vdash s$ if s is in the theory of T.

A *signature morphism* $f : (S, \mathcal{F}) \rightarrow (S', \mathcal{F}')$ maps S to S' and $\mathcal{F}$ to $\mathcal{F}$' such that the ranks of operations are preserved. A signature morphism extends in a unique way to a translation of expressions (as a homomorphism between term algebras) or sets of expressions. A *specification morphism* is a signature morphism that preserves theorems. Let $T = \langle S, \mathcal{F}, A\rangle$ and $T' = \langle S', \mathcal{F},' A'\rangle$ be specifications and let $f : (S, \mathcal{F}) \rightarrow (S', \mathcal{F}')$ be a signature morphism between them. f is a specification morphism if for every axiom $a \in A$, $f(a)$ is a theorem of T', ie. $T' \vdash f(a)$. It follows that a specification morphism translates theorems of the source specification to theorems of the target specification. The semantics of a specification morphism is given by a model construction: If $f : T \rightarrow T'$ is a specification morphism then every model $\mathcal{M}'$ of T' can be made into a model

of T by simply "forgetting" some structure of $\mathcal{M}'$. We say that T' *specializes* T. Practically, this means that any problem that can be expressed in T' can be expressed in T.

It is convenient to generalize the definition of signature morphism slightly to allow the translations of operator symbols to be expressions in the target specification and the translations of sort symbols to be constructions (e.g. products) over the target sorts. A symbol-to-expression morphism is called an *interpretation,* notated $i : T \Rightarrow T'$ where T and T' are the source and target resp. of the morphism.

Finally we note that specifications and signature morphisms form a *category.* Colimits in this category are easily computed.

2.2 Matroids And Greedoids

Matroids date back to the work of Whitney in the 1930's. Greedoids are a generalization of matroids proposed by Korte and Lovasz, [KLS91]. Both have been extensively studied as important algebraic structures with applications in a variety of areas, [BZ92]. Underlying both structures is the notion of a set system:

Definition 21. A *set system* is a pair $\langle S, I\rangle$ where S is a finite nonempty set and I is a nonempty collection of subsets of S

A matroid introduces constraints on I:

Definition 22. A *matroid* is a set system $\langle S, I\rangle$, where the elements of I are called the *independent* subsets, satisfying the following axioms:

Hereditary $\forall Y \in I, \forall X \subseteq Y.\ X \in I$
Exchange $\forall X, Y \in I.\ \|X\| < \|Y\| \Rightarrow \exists a \in Y - X.\ X \cup \{a\} \in I$

The Hereditary axiom requires that every subset of an independent set is also independent. The Exchange axiom implies that all maximal (ordered by $\subseteq$) independent sets are the same size. Such sets are called *bases.* The classic example of a matroid (and indeed the inspiration for matroids) is the set of independent vectors (I) in a vector space (S). Another example is the collection of acyclic subgraphs (I) of a an undirected graph (S). By associating a weight function $w{:}S \rightarrow Nat$ assigning a weight to each item in S, there is a Greedy Algorithm [Edm71] that will compute a (necessarily maximal) weighted independent set $z^* \in I$, i.e. z^* such that $z^* \in I \wedge (\forall z' \in I \cdot c(x, z^*) \geq c(x, z'))$ where $c(z) = \sum_{i \in z} w(i)$.

Greedoids [KLS91] are a generalization of matroids in which the Hereditary axiom $\forall Y \in I, \forall X \subseteq Y.\ X \in I$ is replaced with a weaker requirement called *Accessibility.*

Definition 23. A *greedoid* is a set system $\langle S, I\rangle$, where the elements of I are called the *feasible* subsets, satisfying the following axioms:

Accessibility $X \in I.\ X \neq \emptyset \Rightarrow \exists a \in X.\ X - \{a\} \in I$

Exchange $\forall X, Y \in I.\ \|X\| < \|Y\| \Rightarrow \exists a \in Y - X.\ X \cup \{a\} \in I$

Remark. The Hereditary and Accessibility axioms are easier to compare if the Hereditary Axiom is written as:

$$\forall X \in I, \forall a \in X.\ X - \{a\} \in I$$

which can be shown to be equivalent to the original formulation by induction.

Why are greedoids important? Consider the problem of finding spanning trees. It is true that given a matroid $\langle S, I \rangle$ where S is a set of edges forming a connected graph and I is the set of acyclic subgraphs on that graph, the Greedy Algorithm (see Section 2.4) instantiated on this matroid with an appropriate cost function, is equivalent to Kruskal's algorithm, [CLRS01] and returns a minimum spanning tree. However, the collection of *trees* (that is, connected acyclic subgraphs) over a graph does not form a matroid, because the Hereditary Axiom does not hold for a tree. To see this, consider a set system where S is the set of edges $\{(a, b), (a, c), (b, d)\}$ (see Fig. 2.1) and I is the set of trees on this graph. Clearly S is feasible but the subset of edges $\{(a, c), (b, d)\}$ is not. However, the weaker Accessibility Axiom does hold, so $\langle S, I \rangle$ where S is as above, and I is the set of trees on S forms a greedoid. Instantiated with this greedoid representation of the problem, the Greedy Algorithm is equivalent to Prim's algorithm for MSTs,[CLRS01].

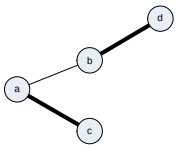

Fig. 2.1. When the Hereditary axiom does not hold

2.3 Greedoid Languages

The implication of the weaker Accessibility axiom for greedoids is that feasible sets should be constructed in an ordered manner, since it is no longer guaranteed that a particular feasible set is reachable from any subset. There is an alternative formulation of greedoids that makes this order explicit [BZ92] which we will utilize. In what follows, a *simple word* over an alphabet S is any word in which no letter occurs more than once and S_s^* is the (finite) set of simple words in S^*.

Definition 24. A *greedoid language* is a pair $\langle S, L \rangle$ where S is a finite ground set and L is a simple language $L \subseteq S_s^*$ satisfying the following conditions:

Hereditary $\forall XY \in L \cdot X \in L$
Exchange $\forall X, Y \in L \cdot \|X\| < \|Y\| \Rightarrow \exists a \in Y.\ Xa \in L$

The hereditary and exchange axioms are analogous to the corresponding axioms for matroids, subject to their application to words. That is, the hereditary axiom requires that any prefix of a feasible word is also a feasible. The exchange axiom requires that a shorter feasible word can be extended to a longer feasible one by appending a letter contained in the longer word to the shorter one. As a consequence, all maximal words in L have the same length.

Bjorner and Ziegler [BZ92] show that the set and language formulations of greedoids are equivalent, that is for every greedoid there is a unique isomorphic greedoid language and v.v. Intuitively, this is because the language version of the greedoid is just enforcing the construction order implied by the feasible set of the greedoid.

2.4 The Greedy Algorithm and Admissible Cost Functions

The greedy algorithm, due to Edmonds [Edm71], is a program schema that is parametrized on a suitable structure such as a matroid or greedoid. The following shows the structure of a pseudo-Haskell program for the greedy algorithm that has been parametrized on a greedoid language. First we define the concept of a feasible extension

Definition 25. Given a greedoid language $\langle S, L\rangle$, the set of *feasible extensions* of a word $A \in L$, written $ext(A)$ is the set $\{a \mid Aa \in L\}$.

```
ga(x,y,w) =
    in if exts(ya) = ∅
       then y
       else let m = arbPick(opt(w, exts(ya))) in ga(x,ym,w)
opt(w, s) = {a: ∀a'∈ s . w(a) >= w(a')}
```

where `arbPick` is a function that picks some element from its argument set. For the the greedy algorithm to be optimal, the cost function must be compatible with the particular structure, or *admissible*. Linear functions are admissible for matroids, but unfortunately not for all greedoids. Admissibility for all greedoids is defined as follows:

Definition 26. Given a greedoid language $\langle S, L\rangle$, a cost function $c : L \to \mathbb{R}$ is *admissible* if, for any $A \in L$, $a \in ext(A)$, whenever $\forall b \in ext(A) \cdot c(Aa) \geq c(Ab)$, the following two conditions hold:

$$\forall b \in S, \forall B, C \in S^* \cdot ABaC \in L \wedge ABbC \in L \Rightarrow c(ABaC) \geq c(ABbC) \quad (2.1)$$

and

$$\forall b \in S, \forall B, C \in S^* \cdot AaBbC \in L \wedge AbBaC \in L \Rightarrow c(AaBbC) \geq c(AbBaC) \quad (2.2)$$

The first condition states that if a is the best choice immediately after A then it continues to be the best choice. The second condition states that a first and b later is better than b first and a later. A cost function that does not depend

on the order of elements in a word immediately satisfies the second condition. Bottleneck functions (functions of the form $\min\{w(X) \mid X \in \mathcal{S}\}$) are an example of admissible functions. Any admissible cost function with a greedoid structure is optimized by the greedy algorithm scheme.

Definition 24 of a greedoid language along with Definition 26of an admissible cost function is what we call Greedoid Language Theory (GL).

2.5 Global Search and Problem Specifications

Global Search with Optimality (GSO) is a class of algorithms that operate by controlled search. GSO has an axiomatic characterization as a specification, [Smi88]. In the same way that the greedy algorithm is parametrized on a matroid or greedoid specification, the GSO class has an associated program schema that is parametrized on the GSO specification. We will formalize a specification of GGS that specializes GSO. Before doing so, we will describe a root specification that GSO itself specializes, called an *optimization problem specification* (P).

P is a 6-tuple $\langle D, R, C, i, o, c\rangle$ specifying the problem to be solved. D is the type of problem inputs, R is the type of problem outputs, augmented with the distinguished value $None$. $(C, \leq)$ is a total order representing some cost domain. $i : D \to Boolean$ is an input condition characterizing valid problem inputs over the domain D, $o : D \times R \to Boolean$ is the output condition characterizing *valid* or *feasible* solutions and $c : D \times R \to C$ is a cost function that returns the cost of a solution. The intent is that any function that meets this specification will take any input $x : D$ that satisfies i and return a $z : R$ that satisfies o for the given x.

A given problem can be classified as an optimization problem by giving an interpretation from the symbols of P to the terms and definitions of the given problem. Here for example is a morphism from P to the Minimum Spanning Tree (MST) problem. The input is a set of edges, where each edge is a pair of nodes with a weight, and nodes are represented by numbers.

$$\begin{aligned} D &\mapsto \{Edge\} \\ &\quad\ Edge \doteq \{a : Node, b : Node, w : Nat\} \\ &\quad\ Node \doteq Nat \\ R &\mapsto \{Edge\} \\ C &\mapsto Nat \\ i &\mapsto \lambda x.\ true \\ o &\mapsto \lambda x, z.\ connected(z) \wedge acyclic(z) \\ c &\mapsto \lambda(x, z).\ \textstyle\sum_{e \in x} e.w \end{aligned} \tag{2.3}$$

Appropriate definitions of connected and acyclic are assumed. Note that an optimal solution to this problem (one that satisfies o and maximizes c) is automatically a spanning tree.

3 Greedy Global Search Theory

We first give an axiomatic specification of GGS. The interested reader may refer to Section 3.4 for the associated program schema that is parametrized on this theory.

Sorts The sorts of a GGS theory are D, R, $\hat{R}$ and C, where D, R, and C are inherited from P, the optimization problem theory, and $\hat{R}$ is the sort of space *descriptors*. A space descriptor is a compact representation of a space and represents all the possible solutions in that space. It is common to make $\widehat{R} = R$.

Operations In addition to i, o, c which are inherited from P, GGS theory adds additional operators, as befits being a richer theory. As with P, a given problem can be classified as a GGS problem by providing a morphism from the symbols of GGS to the given problem. The operator $\lessdot$ corresponds to the *split* operation mentioned in section 2.5 and χ to the *extract* operation. Note that χ and γ are defined as predicates for uniformity of reasoning in proofs. They are more intuitively thought of as partial functions, one possibly extracting a solution from a space and the other possibly greedily choosing a subspace of a space.

$\hat{z}_0 : D \to \hat{R}$	initial space
$\in : R \times \hat{R} \to bool$	is the solution contained in the space?
$\lessdot : D \times \hat{R} \times \hat{R} \to bool$	is the 1st space a subspace of the 2nd space?
$\chi : R \times \hat{R} \to bool$	is the solution extractable from the space?
$\gamma : D \times \widehat{R} \times \{\widehat{R}\} \to bool$	sufficient cond for the space to greedily dominate the set

For ease of reading, ternary operators that take the input x as one of their arguments will from here on be often written in a subscripted infix form. For example,$\gamma(x, \widehat{z}, Z)$ will be written $\widehat{z}\,\gamma_x\, Z$.

Axioms Finally, the following axioms serve to define the semantics of the operations. $\lessdot^*$ denotes a finite number of applications of the $\lessdot$ operator and is defined as

$$\widehat{s} \lessdot_x^* \widehat{r} = \exists i \geq 0 \cdot \widehat{s} \lessdot_x^i \widehat{r}$$

where $\widehat{s} \lessdot_x^0 \widehat{r} = (\widehat{r} = \widehat{s})$ and $s \lessdot_x^{i+1} \widehat{r} = \exists \widehat{t} \cdot \widehat{t} \lessdot_x \widehat{r} \wedge \widehat{s} \lessdot_x^i \widehat{t}$. All free variables are universally quantified, and all variables are assumed to have their appropriate type.

A1. $$i(x) \wedge o(x, z) \Rightarrow z \in \widehat{z}_0(x)$$

A2. $$i(x) \Rightarrow (z \in \widehat{y} \Leftrightarrow \exists \widehat{z} \cdot \widehat{z} \lessdot_x^* \widehat{y} \wedge \chi(z, \widehat{z}))$$

A3. $$\widehat{z}\,\gamma_x\, ss(x, \widehat{y}) \Rightarrow (\exists z \in \widehat{z}, o(x, z), \forall \widehat{z'} \in ss(x, \widehat{y}), \forall z' \in \widehat{z'} \cdot o(x, z') \Rightarrow \wedge c(x, z) \geq c(x, z'))$$

A4. $$i(x) \wedge (\exists z \in \widehat{y} \cdot o(x, z)) \Rightarrow$$
$$(\exists z^* \cdot \chi(z^*, \widehat{y}) \wedge o(x, z^*) \wedge c(x, z^*) = c^*(\widehat{y})) \vee \exists \widehat{z^*} \lessdot_x \widehat{y} \cdot \widehat{z^*}\,\gamma_x\, ss(\widehat{y})$$

A1 provides the semantics for the initial space - it states that all feasible solutions are contained in the initial space.

A2 provides the semantics for the subspace operator $\lessdot$ - namely an output object z is in the space denoted by $\widehat{y}$ iff z can be extracted after finitely many applications of $\lessdot$ to $\widehat{y}$. For convenience it is useful to define a function $ss(x, \widehat{y}) = \{\widehat{z} : \widehat{z} \lessdot_x \widehat{y}\}$.

A3 constrains γ to be a *greedy dominance relation.* (Dominance relations have a long history in algorithm development and provide a way of quickly eliminating subspaces that cannot possibly lead to optimal solutions, [BS74],[ANCK08], [NC09]). That is, $\widehat{z}\,\gamma_x\, Z$ is sufficient to ensure that $\widehat{z}$ will always lead to at least one feasible solution better than any feasible solution in any space $\widehat{z}'$ in Z. As we will shortly demonstrate, A3 also provides a way of *calculating* the desired γ by a process called *derived antecedents.*

A4 places an additional constraint on γ when applied to the subspaces of $\widehat{y}$: An optimal feasible solution in a space $\widehat{y}$ that contains feasible solutions must be immediately extractable or a subspace of $\widehat{y}$ must greedily dominate the subspaces of $\widehat{y}$. Note that extract is not confined to leaves of the search tree: it is possible that a solution can be extracted from a space that can also be split into subspaces.

Remark. A4 is a little stronger than necessary. In fact, in the case that an optimal feasible solution cannot be immediately extracted from a space, some subspace of that space need only greedily dominate other subspaces in the case that the (parent) space was itself the result of a series of greedy choices. Weakening A4 in this way would complicate its statement without, we felt, much of a benefit in practice.

We will show that the class of problems solvable by GGS-theory generalizes the class of problems for which a greedoid representation exists. The way in which this is done is by defining a signature morphism from GGS theory to GL theory, showing the signature morphism is a specification morphism, and then composing that with the isomorphism between Greedoid Languages and Greedoids allowing us to conclude that GGS generalizes Greedoids.

3.1 A Signature Morphism From GGS theory to Greedoid Languages

The signature morphism from GGS to GL is shown in two parts - first the translation of symbols in GGS inherited from P and then the translation of symbols introduced by GGS. Assume the target is a greedoid language $\langle S, L\rangle$ with associated weight function w and objective function c. The translation of

P symbols is[1]: (the [] notation denotes the type of words over an alphabet)

$$
\begin{aligned}
D &\mapsto \{S : \{Id\}, L : \{[Id]\}, w : Id \to C\} \\
R &\mapsto [Id] \\
C &\mapsto Nat \\
i &\mapsto \lambda x.\ \mathit{finite}(x.S) \wedge x.S \neq \emptyset \wedge x.L \subseteq (x.S)_s^* \wedge x.L \neq \emptyset \wedge \mathit{hered}(x.L) \wedge \mathit{exchg}(x.L) \\
&\quad \mathit{hered}(L) = \forall XY \in L \cdot X \in L \\
&\quad \mathit{exchg}(L) = \forall X, Y \in L \cdot \|X\| < \|Y\| \Rightarrow \exists a \in Y.\ Xa \in L \\
o &\mapsto \lambda x, z.\ z \in x.L \\
c &\mapsto c
\end{aligned}
$$

The domain D along with the restriction i captures the type of greedoids, and the range R the type of a result, namely some set of objects from the greedoid. The weight of a solution is calculated by c as the sum of the weights of the elements in the solution.

The translation for the additional symbols introduced by GGS is as follows:

$$
\begin{aligned}
\widehat{R} &\mapsto [Id] \\
\widehat{z}_0 &\mapsto [] \\
\in &\mapsto \lambda z, \widehat{z} \cdot \exists u \in (x.S - \widehat{z})^* \cdot z = \widehat{z}u \\
\lessdot &\mapsto \lambda x, \widehat{z}, \widehat{y} \cdot \exists a \in x.S - \widehat{y} \cdot \widehat{z} = \widehat{y}a \\
\chi &\mapsto \lambda z, \widehat{z} \cdot z = \widehat{z} \\
\gamma &\mapsto ?
\end{aligned}
$$

To complete the morphism, a translation for γ has to be found, which we will do as part of the process of verifying this morphism is indeed a specification morphism.

3.2 Verifying the morphism is a specification morphism

To complete the signature morphism and show it is a specification morphism, the translation of the GGS axioms must be provable in GL theory. Axiom A1 holds trivially because the empty list prefixes any list, and A2 can be proven by induction.

A3 To demonstrate A3, we will reason backwards from the consequent. The required assumption will form the greedy dominance relation. We must show the existence of some $z \in \widehat{z}$ for which $\forall \widehat{z}' \in ss_x(\widehat{y}), \forall z' \in \widehat{z}' \cdot o(x, z') \Rightarrow o(x, z) \wedge c(x, z) \geq c(x, z')$. We will first show $\forall \widehat{z}' \in ss_x(\widehat{y}), \forall z' \in \widehat{z}', \exists z \in \widehat{z} \cdot o(x, z') \Rightarrow o(x, z) \wedge c(x, z) \geq c(x, z')$ and then show the existence of a z that does not depend on $\widehat{z}'$ and z'. Let $\widehat{z} = \widehat{y}a$ for some $a \in x.S - \widehat{y}$, similarly $\widehat{z}' = \widehat{y}u_1'$ for some $u_1' \in x.S - \widehat{y}$. Now let $z' = \widehat{z}'U'$ for some $U' \in S^*$ be any solution contained in $\widehat{z}'$ (See Fig 3.1). Reasoning forwards:

[1] This is greedoid theory from an optimization perspective. Of course, other uses of greedoid theory exist

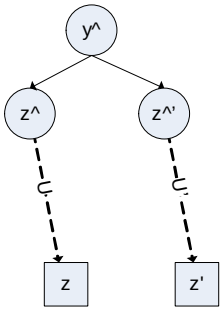

Fig. 3.1. A solution z in $\widehat{z}$ compared with a solution z' in $\widehat{z}'$

$o(x, z')$
$= \{\text{unfold defn}\}$
$z' \in x.L$
$= \{\text{abbreviation above}\}$
$\widehat{y}u_1'U' \in x.L$
$\Rightarrow \{\text{let } U' = U_1'u_2'U_2' \cdots u_n'U_n', \text{ apply Lemma 31 for } j = 0, \text{ if } a \in ext(\widehat{y})\}$
$\widehat{y}aU_1'u_1'U_2'u_2' \cdots U_{n-1}'u_{n-1}'U_n' \in x.L$
$\Rightarrow \{\text{let } z = \widehat{y}aU_1'u_1'U_2'u_2' \cdots U_{n-1}'u_{n-1}'U_n'\}$
$\exists z \in \widehat{z} \cdot o(x, z)$

Next we show that z is better than z'

Under the assumption $a \in ext(\widehat{y}) \wedge \forall a' \in ext(\widehat{y}) \cdot c(x, \widehat{y}a) \geq c(x, \widehat{y}a')$, the following statements can all be shown: By Lemma 31, and property 2.2,

$$c(x, \widehat{y}aU_1'u_1'U_2'u_2' \cdots U_{n-1}'u_{n-1}'U_n') \geq c(x, \widehat{y}u_1'U_1'aU_2'u_2' \cdots U_{n-1}'u_{n-1}'U_n')$$

and by By Lemma 31, and property 2.2 repeatedly,

$$c(x, \widehat{y}u_1'U_1'aU_2'u_2' \cdots U_{n-1}'u_{n-1}'U_n') \geq c(x, \widehat{y}u_1'U_1'u_2'U_2' \cdots aU_n')$$

and finally by property 2.1

$$c(x, \widehat{y}u_1'U_1'u_2'U_2' \cdots aU_n') \geq c(x, \widehat{y}u_1'U_1'u_2'U_2' \cdots u_n'U_n')$$

and so, by transitivity, $c(x, \widehat{y}aU_1'u_1'U_2'u_2' \cdots U_{n-1}'u_{n-1}'U_n') \geq c(x, \widehat{y}u_1'U_1'u_2'U_2' \cdots u_n'U_n')$ ie. $c(x, z) \geq c(x, z') \Leftarrow a \in ext(\widehat{y}) \wedge \forall a' \in ext(\widehat{y}) \cdot c(x, \widehat{y}a) \geq c(x, \widehat{y}a')$.

We can assert the existence of a single feasible z^* that is better than any feasible z' in $\widehat{z}'$ by taking such a z^* to be the best of every z derived above. Finally, collecting together the assumptions, we get a greedy dominance relation satisfying A3: $\widehat{y}a\, \delta_x\, \{\widehat{y}a'\} = a \in ext(\widehat{y}) \wedge \forall a' \in ext(\widehat{y}) \cdot c(\widehat{y}a) \geq c(\widehat{y}a)$.

Notation: In what follows, $A - B$, where A and B are words over L, denotes the asymmetric set difference of the two sets A_s and B_s where W_s is the set of symbols contained in the word W, and $\prod_{i=j}^{k} X_i$, for any $X_j, \cdots, X_k \in S^*$, denotes the concatenation $X_j \cdots X_k$.

Lemma 31. *Given a greedoid $\langle S, L\rangle$, and $Aa \in L, AB \in L$ for some $A, B \in S^*, a \in S$: B can be written $\prod_{i=1}^{n} b_i B_i$ for some $B_1, B_2, \cdots, B_n \in S^*$, such that $\forall j \in [0..n) \cdot A(\prod_{i=0}^{j} b_i B_i)a(\prod_{i=j+1}^{n-1} B_i b_i)B_n \in L$.*

Proof. See Appendix □

A4 To demonstrate A4 holds, note that if a given word $\widehat{y}$ can be feasibly extended, then, from the greedy dominance relation derived above, there will be a subspace that greedily dominates all subspaces, satisfying the second term of the disjunction. If no such extension exists, a feasible solution can be extracted at any time by taking $\chi(z, \widehat{z}) = (z = \widehat{z})$ and at least one of those will be optimal in $\widehat{z}$, satisfying the first term of the disjunction.

This completes the specialization of GGS by GL.

To show a strict generalization, it is sufficient to demonstrate a problem which can be solved in GGS theory but not using greedoids. One such problem is the Activity Selection Problem [CLRS01],[NSC10]:

> Suppose we have a set $S = \{a_1, a_2, \ldots, a_n\}$ of n proposed activities that wish to use a resource, such as a lecture hall, which can be used by only one activity at a time. Each activity a_i has a start time s_i and finish time f_i where $0 \leq s_i < f_i < \infty$. If selected, activity a_i takes place in the half-open time interval $[s_i, f_i)$. Activities a_i and a_j are compatible if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap. The activity selection problem is to select a maximum-size subset of mutually compatible activities.

The input is a set of activities and a solution is subset of that set. Every activity is uniquely identified by an *id* and a start time (s) and finish time (f). The output condition requires that activities must be chosen from the input set, and that no two activities overlap. The problem specification is:

$$
\begin{aligned}
D &\mapsto \{Activity\} \\
&\quad Activity = \{id : Nat, s : Nat, f : Nat\} \\
R &\mapsto \{Activity\} \\
o &\mapsto \lambda(x, z) \cdot noOvp(x, z) \wedge z \subseteq x \\
&\quad noOvp(x, z) \doteq \forall i, j \in z \cdot i \neq j \Rightarrow i \preceq j \vee j \preceq i \\
&\quad i \preceq j = i.f \leq j.s \\
c &\mapsto \lambda(x, z) \cdot \|z\|
\end{aligned}
$$

We will now show how the problem can be solved in GGS theory. Most of the types and operators of GGS theory are straightforward to instantiate. We will just set $\widehat{R}$ to be the same as R. The initial space is just the empty set. The subspace relation $\lessdot$ splits a space by selecting an unchosen activity if one exists and adding it to the existing partial solution. The extract predicate χ can extract a solution at any time:

$$
\begin{aligned}
&\widehat{R} \mapsto R \\
&\widehat{z}_0 \mapsto \lambda x \cdot \emptyset \\
&\lessdot \mapsto \lambda(x, \widehat{z}, \widehat{z}') \cdot \exists a \in x - \widehat{z} \cdot \widehat{z}' = \widehat{z} \cup \{a\} \\
&\chi \mapsto \lambda(z, \widehat{z}) \cdot z = \widehat{z} \\
&\gamma \mapsto \lambda(x, \widehat{z}, Z) \cdot \exists \widehat{y}, a \in x \cdot Z = ss(\widehat{y}) \wedge \widehat{z} \in Z \wedge \widehat{z} = \widehat{y} \cup \{a\} \wedge \widehat{y} \preceq \{a\} \\
&\qquad \wedge \forall (\widehat{y} \cup a') \in Z \cdot \widehat{y} \preceq \{a'\} \Rightarrow a.f \leq a'.f
\end{aligned}
$$

It can be shown that this instantiation satisfies the axioms of GGS theory [NSC10]. To see that the problem cannot be solved with a greedoid representation, consider a set of three activities $\{a_1, a_2, a_3\}$ in which a_1 overlaps with both a_2 and a_3, neither of which overlap each other. Then two feasible solutions are $\{a_1\}$ and $\{a_2, a_3\}$, but neither a_2 nor a_3 can be used to feasibly extend $\{a_1\}$, thus failing to satisfy the Exchange axiom.

Finally, note that another way in which GGS generalizes greedoids is that while the Greedy Algorithm requires an admissible cost function over greedoids, GGS theory places no such restrictions a priori on the cost function.

3.3 A Program Theory for GGS

Starting from a statement of what is desired, namely to compute an optimal feasible solution, we will first formally derive a recurrence, which is then correct by construction. The recurrence can then be transformed into an executable program.

Define

$$Fgdy(z, x, \widehat{y}) = z \in opt_c\{z \mid z \in \widehat{y} \wedge o(x, z)\}$$

This is a specification of a function $Fgdy$ to be derived.opt_c is a subset of its argument that is the optimal (w.r.t. the cost function c and a w.f.o. $\geq$), defined as follows:

$$\forall z \cdot z \in opt_c S = z \in S \wedge (\forall z' \in S \cdot c(x, z) \geq c(x, z'))$$

In the sequel we will assume that the order drop the subscript c when it is clear from context

Theorem 32. *Let $\langle D, R, \widehat{R}, C, i, o, c, \widehat{z}_0, \in, \lessdot, \chi, \gamma \rangle$ be a GGS-Theory as defined above. Then the following characteristic recurrence holds for all x and z:*

$$
\begin{aligned}
&Fgdy(z, x, \widehat{y}) \Leftarrow z \in opt_c\{z \mid \\
&\quad z \in opt_c\{z \mid e(z, \widehat{y}) \wedge o(x, z))\} \vee (\exists \widehat{z} \lessdot \widehat{y} \cdot \widehat{z}\, \gamma_x\, ss(x, \widehat{y}) \wedge Fgdy(z, x, \widehat{z}))\}
\end{aligned}
$$

Proof.

$$
\begin{array}{cc}
 & Fgdy(z,x,\widehat{y}) \\
= & \{\text{unfold defn of } Fgdy\} \\
 & z \in opt_c\{z \mid z \in \widehat{y} \wedge o(x,z))\} \\
= & \{\text{provable from A2}\} \\
 & z \in opt_c\{z \mid [\chi(z,\widehat{y}) \vee (\exists \widehat{z} \cdot s(x,\widehat{y},\widehat{z}) \wedge z \in \widehat{z})] \wedge o(x,z)\} \\
= & \{\text{distributivity of set comprehension and } opt\} \\
 & z \in opt_c\{z \mid z \in opt_c\{z \mid \chi(z,\widehat{y}) \wedge o(x,z)\} \vee z \in opt_c\{z \mid \exists \widehat{z} \in ss(x,\widehat{y}) \cdot z \in \widehat{z} \wedge o(x,z)\}\} \\
\Leftarrow & \{\text{Lemma 33}\} \\
 & z \in opt_c\{z \mid z \in opt_c\{z \mid \chi(z,\widehat{y}) \wedge o(x,z)\} \vee (\exists \widehat{z} \lessdot \widehat{y} \cdot \widehat{z}\,\gamma_x\, ss(x,\widehat{y}) \wedge Fgdy(z,x,\widehat{z}))\}
\end{array}
$$

□

Lemma 33.

$$
\begin{array}{cc}
 & opt_c\{z \mid z \in opt_c\{z \mid \chi(z,\widehat{y}) \wedge o(x,z)\} \vee z \in opt_c\{z \mid \exists \widehat{z} \in ss(x,\widehat{y}) \cdot z \in \widehat{z} \wedge o(x,z)\}\} \\
\supseteq & \\
 & opt_c\{z \mid z \in opt_c\{z \mid \chi(z,\widehat{y}) \wedge o(x,z)\} \vee (\exists \widehat{z} \lessdot \widehat{y} \cdot \widehat{z}\,\gamma_x\, ss(x,\widehat{y}) \wedge Fgdy(z,x,\widehat{z}))\}
\end{array}
$$

Proof. See Appendix □

Lemma 34. *If* $\exists \widehat{z} \in ss(x,\widehat{y}), \exists z \in \widehat{z} \cdot o(x,z)$ *then for any* z^*

$$
\begin{array}{cc}
 & (\exists \widehat{z} \in ss(x,\widehat{y}) \cdot z^* \in \widehat{z} \wedge o(x,z^*)) \\
 & \wedge \\
 & (\forall \widehat{z}' \in ss(x,\widehat{y}), \forall z' \in \widehat{z}' \cdot o(x,z') \Rightarrow o(x,z^*) \wedge c(x,z^*) \geq c(x,z') \\
\Leftarrow & \\
 & \exists \widehat{z} \in ss(x,\widehat{y}) \cdot \widehat{z}\,\gamma_x\, ss(x,\widehat{y}) \wedge z^* \in opt_c\{z \mid z \in \widehat{z} \wedge o(x,z))\}
\end{array}
$$

Proof. See Appendix □

Non Triviality Finally, to demonstrate non-triviality[2] of the recurrence we need to show that if there exists an optimal solution, then one will be found. That is:

$$
\begin{array}{l}
(i(x) \wedge \exists z \cdot Fgdy(z,x,\widehat{y})) \Rightarrow \exists z \in opt_c\{z \mid \\
z \in opt_c\{z \mid (z,c,\chi(z,\widehat{y}) \wedge o(x,z))\} \vee (\exists \widehat{z} \lessdot \widehat{y} \cdot \widehat{z}\,\gamma_x\, ss(x,\widehat{y}) \wedge Fgdy(z,x,\widehat{z}))\}
\end{array}
$$

Proof. See Appendix. □

[2] This is similar but not identical to completeness. Completeness requires that every optimal solution is found by the recurrence, which we do not guarantee.

Algorithm 1 Program Schema for GGS Theory

```
--given x:D satisfying i returns optimal (wrt. cost fn c) z:R satisfying o(x,z)
function solve :: D -> {R}
solve x =
    if Φ(r̂0(x)) then (gsolve x r̂0(x) {}) else {}

function gsolve :: D -> R̂ -> {R} -> {R}
gsolve x space soln =
    let gsubs = {s | s∈subspaces x space ∧ ∀ss ∈ subspaces x space,s δx ss}
        soln' = opt c (soln ∪{z | χ(z,space) ∧ o(x,z)})
    in if gsubs = {} then soln'
       else let greedy = arbPick gsubs in gsolve x greedy soln'

function opt :: ((D,R) -> C) -> {R̂}-> {R̂}
opt c {s} = {s}
opt c {s,t} = if c(x,s)>c(x,t) then {s} else {t}
function subspaces :: D -> R̂-> {R̂}
subspaces x r̂ = {ŝ: ŝ ≪x r̂∧Φ(x,ŝ)}
```

3.4 Abstract Program

By the application of correctness preserving transforms, the recurrence proved above can be transformed into the abstract program shown in Alg. 1, written in a pseudo-Haskell style. The program for GGS belongs in a class of algorithms that operate by controlled search. That is, given a *space* of candidate *solutions* to a given problem (some of which may not be optimal or even feasible solutions), a GGS algorithm partitions (*splits*) the space into *subspaces* (also called *partial solutions*), each of which is recursively searched in turn for optimal solutions. (Such an approach is also the basis of branch-and-bound algorithms, common in AI). At any point, a solution can possibly be extracted from a space, and if correct, compared with the best solution found so far. The process terminates when no space can be further partitioned. The starting point is an *initial space* known to contain all possible solutions to the given problem. The result, if any, is an optimal solution to the problem. The key insight that makes for an efficient algorithm is the incorporation of specific search control operators at the *abstract level*, that, suitably instantiated with problem specific information can drastically reduce the amount of search and thereby lead to very efficient algorithms for a given problem. A summary of Global Search theory is contained in the Appendix. Further details are in Smith's papers, [Smi88,Smi90].

In addition, optimizations such as context-dependent simplification, finite differencing, and data structure selection often have to be carried out before arriving at a final efficient program. One such optimization is for problems (such as those which satisfy greedoid axioms) for which only maximally sized solutions need be extracted. Specware [S], a tool from Kestrel Institute, provides support for carrying out such correctness preserving program transformations.

4 Related Work

Greedoids arose when Korte and Lovasz noticed that the hereditary property required by matroids was stronger than necessary for the Greedy Algorithm of Edmonds to be optimal. However, the exact characterization of the accessible set systems for which the greedy algorithm optimized all linear functions remained an open one until Helman et al. [HMS93] showed that a structure known as a *matroid embedding* was both necessary and sufficient. Matroid embeddings relax the Exchange axiom of greedoids but add two more axioms, so they are simultaneously a generalization and a specialization of greedoids. We have shown that GGS strictly generalizes greedoids.

In earlier work, Helman [Hel89] devised a framework that unified branch-and-bound and dynamic programming. The framework also incorporated dominance relations. However, Helman's goal was the unification of the two paradigms, and not the process by which algorithms can be calculated. In fact the unification, though providing a very important insight that the two paradigms are related at a higher level, arguably makes the derivation of particular algorithms harder. Our interest is ultimately in the systematic derivation of algorithms.

Curtis [Cur03] has a classification scheme intended to cover *all* greedy algorithms. There is a top-level catch-all class and three subclasses. Each class has a some conditions that must be met for a given problem to belong to that class. In general, verifying those conditions gets easier the lower the class in the hierarchy. However, fewer problems qualify the lower in the hierarchy. Once classified, however, the greedy algorithm is then automatically correct and optimal for that problem. Unlike Curtis, we are not attempting a complete classification (although our characterization of greedy algorithms is comparable to Curtis's top level category of Best Global, and in that sense covers all greedy algorithms). Curtis also does not relate any of the greedy categories to matroids or greedoids. Another difference between our work and that of Curtis is that while Curtis's work is targeted specifically at greedy algorithms, for us greedy algorithms are just a special case of a more general problem of deriving effective global search algorithms. The same work applies to both. In the case that the dominance relation really does not lead to a singleton choice at each split, it can still prove to be highly effective. This was recently demonstrated on some Segment Sum problems we looked at. Although the dominance relation we derived for those problem did not reduce to a greedy choice, it was nonetheless key to reducing the complexity of the search (the width of the search tree was kept constant) and led to a very efficient breadth-first solution that was much faster than comparable solutions derived by program transformation, [NC09].

Another approach has been taken by Bird and de Moor [BM93] who show that under certain conditions a dynamic programming algorithm simplifies into a greedy algorithm. Our characterization can be considered an analogous specialization of (a form of) branch-and-bound. The difference is that we do not require calculation of the entire program, but specific operators, which is a less onerous task. Also, as pointed out by Curtis [Cur03], the conditions required by Bird and de Moor are not easy to meet.

Charlier [Cha95], also building on Smith's work, proposed a new algorithm class for greedy algorithms that directly embodied the matroid axioms. Using this class, he was able to synthesize Kruskal's MST algorithm and a solution to the $1/1/\sum T_i$ scheduling problem. However he reported difficulty with the equivalent of the Augmentation (comparable to the Exchange) axiom. The difficulty with a new algorithm class is often the lack of a repeatable process for synthesizing algorithms in that class, and this would appear to be what Charlier ran up against. In contrast, by specializing an existing theory (GSO), we can apply all the techniques that are available such as bounds tests, filters, propagators, etc. We are also able to handle a wider class of problems than belong in matroids.

References

[ANCK08] A Allahverdi, C T Ng, T C E Cheng, and M K Kovalyov. A survey of scheduling problems with setup times or costs. *European J. of Operational Res.*, 187:985–1032, 2008.

[BM93] R. S. Bird and O. De Moor. From dynamic programming to greedy algorithms. In *Formal Program Development, volume 755 of Lecture Notes in Computer Science*, pages 43–61. Springer-Verlag, 1993.

[BS74] K.R. Baker and Z-S. Su. Sequencing with due-dates and early start times to minimize maximum tardiness. *Naval Research Logistics*, 21(1):171–176, 1974.

[BZ92] Anders Björner and Günter M. Ziegler. Introduction to greedoids. In Neil White, editor, *Matroid Applications*. Cambridge University Press, 1992.

[Cha95] B. Charlier. The greedy algorithms class: formalization, synthesis and generalization. Technical report, 1995.

[CLRS01] T Cormen, C Leiserson, R Rivest, and C Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[Cur03] S. A. Curtis. The classification of greedy algorithms. *Sci. Comput. Program.*, 49(1-3):125–157, 2003.

[Edm71] J. Edmonds. Matroids and the greedy algorithm. *Math. Programming*, 1(1):127–136, 1971.

[Hel89] P. Helman. A common schema for dynamic programming and branch and bound algorithms. *J. ACM*, 36(1):97–128, 1989.

[HMS93] P. Helman, B. M. E. Moret, and H. D. Shapiro. An exact characterization of greedy structures. *SIAM J. on Discrete Math.*, 6:274–283, 1993.

[KLS91] B. Korte, L. Lovasz, and R. Schrader. *Greedoids*. Springer-Verlag, 1991.

[NC09] S. Nedunuri and W.R. Cook. Synthesis of fast programs for maximum segment sum problems. In *Intl. Conf. on Generative Programming and Component Engineering (GPCE)*, Oct. 2009.

[NSC10] S. Nedunuri, D. R. Smith, and W. R. Cook. Synthesis of greedy algorithms using dominance relations. *2nd NASA Symp. on Formal Methods*, 2010.

[S] Specware. http://www.specware.org.

[Smi88] D. R. Smith. Structure and design of global search algorithms. Tech. Rep. Kes.U.87.12, Kestrel Institute, 1988.

[Smi90] D. R. Smith. Kids: A semi-automatic program development system. *IEEE Trans. on Soft. Eng., Spec. Issue on Formal Methods*, 16(9):1024–1043, September 1990.

[SPW95] D. R. Smith, E. A. Parra, and S. J. Westfold. Synthesis of high-performance transportation schedulers. Technical report, Kestrel Institute, 1995.

Appendix: Proofs of Lemmas

Lemma 31:

Given a greedoid $\langle S, L\rangle$, and $Aa \in L, AB \in L$ for some $A, B \in S^*, a \in S$: B can be written $\prod_{i=1}^{n} b_i B_i$ for some $B_1, B_2, \cdots, B_n \in S^*$, such that $\forall j \in [0..n) \cdot A(\prod_{i=0}^{j} b_i B_i)a(\prod_{i=j+1}^{n-1} B_i b_i)B_n \in L$.

Proof. By induction on the length m of B.

Base case: $m = 1$: B is just a single symbol b, written $b\varepsilon$ and result holds by assumption.

Inductive case: Assume the result for B of length m, and let X^j for any $j \in [0..n)$ denote $A(\prod_{i=0}^{j} b_i B_i)a(\prod_{i=j+1}^{n-1} B_i b_i)B_n \in L$. If AB cannot be extended we are done. Otherwise, extend AB with a symbol b such that $ABb \in L$. Then since $ABb - X^j = \{b, b_n\}$ (for any $j \in [0, n)$), by the exchange axiom, a feasible extension of X^j is either b or b_n. If the extension is b then re-characterize Bb as $(\prod_{i=1}^{n-1} b_i B_i)b_n B'_n$ where $B'_n = B_n b$ and then since $X^j b \in L$, $X^j b = A(\prod_{i=0}^{j} b_i B_i)a(\prod_{i=j+1}^{n-1} B_i b_i)B'_n \in L$ for any $j \in [0, n)$ as required. If the extension of X^j (for any $j \in [0, n)$) is b_n, that is $X^j b_n \in L$, then write Bb as $b_1 B_1 b_2 B_2 \cdots b_n B_n b_{n+1} B_{n+1}$ where $b_{n+1} = b$ and $B_{n+1} = \varepsilon$ and it is clear that $X^j b_n = A(\prod_{i=0}^{j} b_i B_i)a(\prod_{i=j+1}^{n} B_i b_i)B_{n+1} \in L$ as required. To complete the proof we need to show that the $j = n$ case also holds (because the number of separators B_i is now $n + 1$), that is $A(\prod_{i=1}^{n} b_i B_i)aB_{n+1} \in L$:

$$
\begin{array}{ll}
 & A(\prod_{i=1}^{n} b_i B_i)aB_{n+1} \in L \\
= & \{B_{n+1} = \varepsilon\} \\
 & A(\prod_{i=1}^{n} b_i B_i)a \in L \\
\Leftarrow & \{\text{Exchange Axiom}\} \\
 & A(\prod_{i=1}^{n} b_i B_i) \in L \wedge A(\prod_{i=0}^{n-1} b_i B_i)aB_n \in L \\
= & \{A(\prod_{i=1}^{n} b_i B_i) = AB \in L, \text{ by assumption}\} \\
 & A(\prod_{i=0}^{n-1} b_i B_i)aB_n \in L \\
= & \{A(\prod_{i=0}^{n-1} b_i B_i)aB_n = X^{n-1}\} \\
 & true
\end{array}
$$

□

Lemma 33

$$
\begin{array}{c}
opt_c\{z \mid z \in opt_c\{z \mid \chi(z, \widehat{y}) \wedge o(x, z)\} \vee z \in opt_c\{z \mid \exists \widehat{z} \in ss(x, \widehat{y}) \cdot z \in \widehat{z} \wedge o(x, z)\}\} \\
\supseteq \\
opt_c\{z \mid z \in opt_c\{z \mid \chi(z, \widehat{y}) \wedge o(x, z)\} \vee (\exists \widehat{z} \lessdot \widehat{y} \cdot \widehat{z}\, \gamma_x\, ss(x, \widehat{y}) \wedge Fgdy(z, x, \widehat{z}))\}
\end{array}
$$

Proof. Note that $z \in opt_c\{z \mid z \in opt_c\{z \mid \chi(z, \widehat{y}) \wedge o(x, z)\} \vee z \in opt_c\{z \mid \exists \widehat{z} \in ss(x, \widehat{y}) \cdot z \in \widehat{z} \wedge o(x, z)\}\}$ iff $z \in opt_c\{z \mid \chi(z, \widehat{y})\} \wedge c(x, z) = c^*(\widehat{y})$ or $z \in opt_c\{z \mid \exists \widehat{z} \in ss(x, \widehat{y}) \cdot z \in \widehat{z} \wedge o(x, z)\} \wedge c(x, z) = c^*(\widehat{y})$. Therefore if some z

satisfies $\chi(z,\widehat{y}) \wedge o(x,z) \wedge c(x,z) = c^*(\widehat{y})$ then the result follows. Otherwise, by A4 we have $\exists \widehat{z} \lessdot \widehat{y} \cdot \widehat{z}\,\gamma_x\, ss(x,\widehat{y})$, if $i(x) \wedge \exists z \in \widehat{y} \cdot o(x,z)$. Then:

$$
\begin{array}{ll}
 & z \in opt_c\{z \mid \exists \widehat{z} \in ss(x,\widehat{y}) \cdot z \in \widehat{z} \wedge o(x,z)\} \\
= & \{\text{defn of } opt\} \\
 & (\exists \widehat{z} \in ss(x,\widehat{y}) \cdot z \in \widehat{z} \wedge o(x,z)) \\
 & \wedge \\
 & (\forall z' \cdot (\exists \widehat{z}' \in ss(x,\widehat{y}) \cdot z' \in \widehat{z}' \wedge o(x,z')) \Rightarrow c(x,z) \geq c(x,z')) \\
= & \{\text{change quantifier}\} \\
 & (\exists \widehat{z} \in ss(x,\widehat{y}) \cdot z \in \widehat{z} \wedge o(x,z)) \\
 & \wedge \\
 & (\forall \widehat{z}' \in ss(x,\widehat{y}), \forall z' \in \widehat{z}' \cdot o(x,z') \Rightarrow c(x,z) \geq c(x,z')) \\
= & \{\text{logic}\} \\
 & (\exists \widehat{z} \in ss(x,\widehat{y}) \cdot z \in \widehat{z} \wedge o(x,z)) \\
 & \wedge \\
 & (\forall \widehat{z}' \in ss(x,\widehat{y}), \forall z' \in \widehat{z}' \cdot o(x,z') \Rightarrow o(x,z) \wedge c(x,z) \geq c(x,z')) \\
\Leftarrow & \{\text{lemma 34}\} \\
 & \exists \widehat{z} \in ss(x,\widehat{y}) \cdot \widehat{z}\,\gamma_x\, ss(x,\widehat{y}) \wedge z \in opt_c\{z \mid z \in \widehat{z} \wedge o(x,z))\} \\
= & \{\text{defn of } Fgdy\} \\
 & \exists \widehat{z} \in ss(x,\widehat{y}) \cdot \widehat{z}\,\gamma_x\, ss(x,\widehat{y}) \wedge Fgdy(z,x,\widehat{z})
\end{array}
$$

□

Lemma 34:
If $\exists \widehat{z} \in ss(x,\widehat{y}), \exists z \in \widehat{z} \cdot o(x,z)$ then for any z^*

$$
\begin{array}{ll}
 & (\exists \widehat{z} \in ss(x,\widehat{y}) \cdot z^* \in \widehat{z} \wedge o(x,z^*)) \\
 & \wedge \\
 & (\forall \widehat{z}' \in ss(x,\widehat{y}), \forall z' \in \widehat{z}' \cdot o(x,z') \Rightarrow o(x,z^*) \wedge c(x,z^*) \geq c(x,z') \\
\Leftarrow & \\
 & \exists \widehat{z} \in ss(x,\widehat{y}) \cdot \widehat{z}\,\gamma_x\, ss(x,\widehat{y}) \wedge z^* \in opt_c\{z \mid z \in \widehat{z} \wedge o(x,z))\}
\end{array}
$$

Proof. From the characterization of greedy dominance (A3), $\exists \widehat{z} \in ss(x,\widehat{y}) \cdot \widehat{z}\,\gamma_x\, ss(x,\widehat{y})$ implies $\exists z \in \widehat{z}, \forall \widehat{z}' \in ss(x,\widehat{y}), \forall z' \in \widehat{z}' \cdot o(x,z') \Rightarrow o(x,z) \wedge c(x,z) \geq c(x,z')$. By assumption, some subspace of $\widehat{y}$ contains a feasible solution, so the consequent follows from $z^* \in opts_c\{z \mid z \in \widehat{z} \wedge o(x,z)\}$. □

Non-triviality

$$
\begin{array}{l}
(i(x) \wedge \exists z \cdot Fgdy(z,x,\widehat{y})) \Rightarrow \exists z \in opt_c\{z \mid \\
z \in opt_c\{z \mid (z, c, \chi(z,\widehat{y}) \wedge o(x,z))\} \vee (\exists \widehat{z} \lessdot \widehat{y} \cdot \widehat{z}\,\gamma_x\, ss(x,\widehat{y}) \wedge Fgdy(z,x,\widehat{z}))\}
\end{array}
$$

Proof.

$$
\begin{array}{ll}
 & i(x) \wedge \exists z \cdot Fgdy(z, x, \widehat{y}) \\
= & \{\text{defn of } Fgdy\} \\
 & i(x) \wedge \exists z \in opt_c\{z \mid z \in \widehat{y} \wedge o(x, z)\} \\
= & \{\text{property of } opt_c\} \\
 & \exists z \cdot i(x) \wedge z \in \widehat{y} \wedge o(x, z) \\
\Rightarrow & \{\text{Axioms A4, A2}\} \\
 & \exists z \cdot (\chi(z, \widehat{y}) \vee (\exists \widehat{z} \lessdot \widehat{y} \cdot \widehat{z}\, \gamma_x\, ss(x, \widehat{y}) \wedge z \in \widehat{z})) \wedge o(x, z) \\
= & \{\text{distributivity of } \wedge\} \\
 & (\exists z \cdot \chi(z, \widehat{y}) \wedge o(x, z)) \vee (\exists z, \widehat{z} \lessdot \widehat{y} \cdot \widehat{z}\, \gamma_x\, ss(x, \widehat{y}) \wedge z \in \widehat{z} \wedge o(x, z)) \\
= & \{\text{property of } opt_c\} \\
 & (\exists z \cdot \chi(z, \widehat{y}) \wedge o(x, z)) \vee (\exists z, \widehat{z} \lessdot \widehat{y} \cdot \widehat{z}\, \gamma_x\, ss(x, \widehat{y}) \wedge z \in opt_c\{z \mid z \in \widehat{z} \wedge o(x, z)\}) \\
= & \{\text{defn of } Fgdy\} \\
 & (\exists z \cdot \chi(z, \widehat{y}) \wedge o(x, z)) \vee (\exists z, \widehat{z} \lessdot \widehat{y} \cdot \widehat{z}\, \gamma_x\, ss(x, \widehat{y}) \wedge Fgdy(z, x, \widehat{z})) \\
= & \{\text{property of } opt_c\} \\
 & \exists z \in opt_c\{z \mid \chi(z, \widehat{y}) \wedge o(x, z)\} \vee (\exists z, \widehat{z} \lessdot \widehat{y} \cdot \widehat{z}\, \gamma_x\, ss(x, \widehat{y}) \wedge Fgdy(z, x, \widehat{z})) \\
= & \{\text{distributivity of } \exists\} \\
 & \exists z \cdot z \in opt_c\{z \mid \chi(z, \widehat{y}) \wedge o(x, z)\} \vee (\exists \widehat{z} \lessdot \widehat{y} \cdot \widehat{z}\, \gamma_x\, ss(x, \widehat{y}) \wedge Fgdy(z, x, \widehat{z})) \\
= & \{\text{property of } opt_c\} \\
 & \exists z \in opt_c\{z \mid z \in opt_c\{z \mid \chi(z, \widehat{y}) \wedge o(x, z)\} \vee (\exists \widehat{z} \lessdot \widehat{y} \cdot \widehat{z}\, \gamma_x\, ss(x, \widehat{y}) \wedge Fgdy(z, x, \widehat{z}))\}
\end{array}
$$

□