

Synthesis of Greedy Algorithms Using Dominance Relations

Srinivas Nedunuri, William R. Cook
University of Texas at Austin

Douglas R. Smith
Kestrel Institute


April 14, 2010

What's the problem?

There is currently no straightforward process for constructing greedy algorithms!

What's our solution

- We have defined an algorithm class called *Greedy Global Search (GGS)*¹
- GGS supplies a *program schema* (template) containing *operators* whose semantics is *axiomatically* defined
- Operators must be instantiated by the user (developer) in accordance with the axioms.

¹S. Nedunuri, D.R. Smith, W.R. Cook, "A class of greedy algorithms and its relation to greedoids", Submitted to *Intl. Colloq. on Theoretical Aspects of Computing, 2010* 

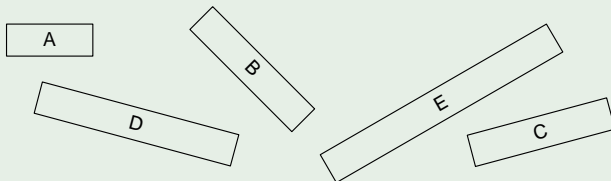
This talk will cover:

- How to *calculate* instantiations for the operators that satisfy the axioms
- Showing the calculation reduces to simple proofs (that could potentially be automated)
- In cases where the calculation is not straightforward we introduce a tactic that aids in the process

The result is a greedy algorithm that is correct by construction

Example: Machine Scheduling

Schedule jobs to minimize sum of completion times



Jobs



Schedule

First specify the problem!

Structure of Specification

- An output condition (postcondition), $o : D \times R \rightarrow \text{Boolean}$
- A cost function, $c : D \times R \rightarrow C$

(along with definitions for the types D , R and C)

$D \mapsto \text{Job} \rightarrow \text{Duration}$

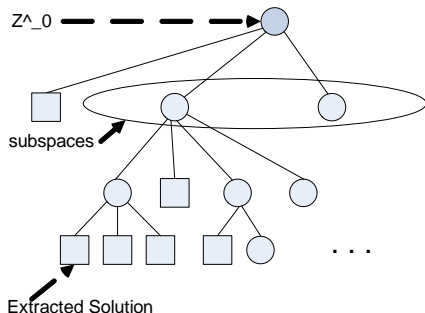
$R \mapsto [\text{Job}]$

$o \mapsto \lambda(x, z) \cdot \text{asSet}(z) = \text{dom}(x)$

$c \mapsto \lambda(x, z) \cdot \sum_{i=1}^{\#z} \text{ct}(z, i)$
 $\text{ct}(z, i) = \sum_{j=1}^i x(z[j])$

GGs solves these problems using search

- Takes the *solution space* (potentially infinite) and partitions it
- Each element of the partition is called a *subspace*, and is recursively partitioned until a singleton space is encountered, called a *solution*²
- This recursive procedure is implemented by a program schema



- \hat{z}_0 op constructs the initial space
- \leq op defines the subspaces
- χ op extracts a solution (if one exists)

²based on N. Agin, "Optimum Seeking with Branch and Bound", Mgmt. Sci. 1966

Step 2: Instantiate the operators for the Scheduling problem

$\widehat{z}_0, \triangleleft, \chi$

- The initial space (\widehat{z}_0) is just an empty list
- Form a subspace (\triangleleft) by adding an as yet unscheduled task to the current partial solution
- extract a solution (χ) by just returning the current list

Formal Definitions

$$\widehat{z}_0 \mapsto \lambda x \cdot []$$

$$\triangleleft \mapsto \lambda(x, \widehat{z}, \widehat{z}') \cdot \exists a \in x - \text{asSet}(\widehat{z}) \cdot \widehat{z}' = \widehat{z}++[a]$$

$$\chi \mapsto \lambda(z, \widehat{z}) \cdot z = \widehat{z}$$

These are expressed as predicates and not as partial functions to make it easier to verify they satisfy the axioms

Step 2: Instantiate the operators for the Scheduling problem

$\widehat{z}_0, \triangleleft, \chi$

- The initial space (\widehat{z}_0) is just an empty list
- Form a subspace (\triangleleft) by adding an as yet unscheduled task to the current partial solution
- extract a solution (χ) by just returning the current list

Formal Definitions

$\widehat{z}_0 \mapsto \lambda x \cdot []$

$\triangleleft \mapsto \lambda(x, \widehat{z}, \widehat{z}') \cdot \exists a \in x - \text{asSet}(\widehat{z}) \cdot \widehat{z}' = \widehat{z}++[a]$

$\chi \mapsto \lambda(z, \widehat{z}) \cdot z = \widehat{z}$

These are expressed as predicates and not as partial functions to make it easier to verify they satisfy the axioms

Remember the contract:

- If you instantiate the types D, R, C and operators $o, c, \widehat{z}_0, \chi, \leq$
 - In the program schema
 - In a way that satisfies the GGS axioms
 - Then you have a correct program that will solve your problem (meet the specification)

Remember the contract:

- If you instantiate the types D, R, C and operators $o, c, \hat{z}_0, \chi, \leq$
- In the program schema
- In a way that satisfies the GGS axioms
- Then you have a correct program that will solve your problem (meet the specification)

Remember the contract:

- If you instantiate the types D, R, C and operators $o, c, \hat{z}_0, \chi, \leq$
- In the program schema
- In a way that satisfies the GGS axioms
- Then you have a correct program that will solve your problem (meet the specification)

Remember the contract:

- If you instantiate the types D, R, C and operators $o, c, \widehat{z}_0, \chi, \leq$
- In the program schema
- In a way that satisfies the GGS axioms
- Then you have a correct program that will solve your problem (meet the specification)

Constraint Satisfaction Problems and the \oplus operator

We will focus on a class of problems (constraint satisfaction) in which the search space is split (partitioned) by assigning values to variables

Definition: Partial Solution or Space (\hat{z})

An assignment to some of the variables. Can be extended into a (complete) solution by assigning to all the variables

- Let $\hat{z} \oplus e$ denote the result of combining a partial solution \hat{z} with an *extension* e using a left-associative operator \oplus
- When $\hat{z} \oplus e$ is a (feasible) complete solution, e is called the (*feasible*) *completion* of \hat{z} .

Machine scheduling example

The \oplus operator is just $\hat{z} \oplus e = \hat{z}++e$

Constraint Satisfaction Problems and the \oplus operator

We will focus on a class of problems (constraint satisfaction) in which the search space is split (partitioned) by assigning values to variables

Definition: Partial Solution or Space (\hat{z})

An assignment to some of the variables. Can be extended into a (complete) solution by assigning to all the variables

- Let $\hat{z} \oplus e$ denote the result of combining a partial solution \hat{z} with an *extension* e using a left-associative operator \oplus
- When $\hat{z} \oplus e$ is a (feasible) complete solution, e is called the (*feasible*) *completion* of \hat{z} .

Machine scheduling example

The \oplus operator is just $\hat{z} \oplus e = \hat{z}++e$

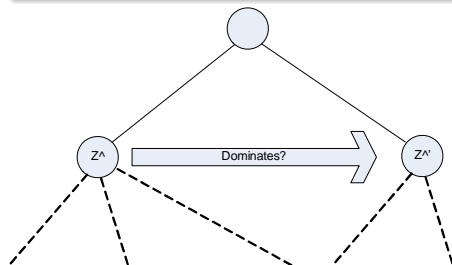
Are we done?

- Once the operators are instantiated, we have a program that is correct, but its not very efficient!
- There are several refinements to the search that we can introduce

Dominance Relations

What are dominance relations?

- Enables the comparison of one partial solution with another to determine if one of them can be discarded
- Given \hat{z} and \hat{z}' if the best possible solution in \hat{z} is better than the best possible solution in \hat{z}' then \hat{z}' can be discarded



Axioms capture this

A1: Find a δ_x satisfying:

$$\widehat{z} \delta_x Z \Rightarrow (\exists z \in \widehat{z}, o(x, z), \forall z' \in Z, \forall z' \in \widehat{z}' \cdot o(x, z') \Rightarrow \wedge c(x, z) \geq c(x, z'))$$

If a partial solution \widehat{z} dominates a set of partial solutions Z then: there is some solution z that can be obtained by extending \widehat{z} which is cheaper than any solution obtained from any partial solution in Z

A2: Find a z^* satisfying:

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

An optimal feasible solution z^* must be extractable from a partial solution \widehat{y} (that contains any feasible solutions) or a subspace \widehat{z}^* of \widehat{y} must dominate all the subspaces of \widehat{y} . \widehat{z}^* is the *greedy choice*

There are others but these are ones specific to greedy global search

Axioms capture this

A1: Find a δ_x satisfying:

$$\widehat{z} \delta_x Z \Rightarrow (\exists z \in \widehat{z}, o(x, z), \forall z' \in Z, \forall z' \in \widehat{z}' \cdot o(x, z') \Rightarrow \wedge c(x, z) \geq c(x, z'))$$

If a partial solution \widehat{z} dominates a set of partial solutions Z then: there is some solution z that can be obtained by extending \widehat{z} which is cheaper than any solution obtained from any partial solution in Z

A2: Find a z^* satisfying:

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

An optimal feasible solution z^* must be extractable from a partial solution \widehat{y} (that contains any feasible solutions) or a subspace \widehat{z}^* of \widehat{y} must dominate all the subspaces of \widehat{y} . \widehat{z}^* is the *greedy choice*

There are others but these are ones specific to greedy global search

Axioms capture this

A1: Find a δ_x satisfying:

$$\widehat{z} \delta_x Z \Rightarrow (\exists z \in \widehat{z}, o(x, z), \forall z' \in Z, \forall z' \in \widehat{z}' \cdot o(x, z') \Rightarrow \wedge c(x, z) \geq c(x, z'))$$

If a partial solution \widehat{z} dominates a set of partial solutions Z then: there is some solution z that can be obtained by extending \widehat{z} which is cheaper than any solution obtained from any partial solution in Z

A2: Find a z^* satisfying:

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

An optimal feasible solution z^* must be extractable from a partial solution \widehat{y} (that contains any feasible solutions) or a subspace \widehat{z}^* of \widehat{y} must dominate all the subspaces of \widehat{y} . \widehat{z}^* is the *greedy choice*

There are others but these are ones specific to greedy global search

- Dominance can be proven by intersecting two relations called *semi-congruence* and *extension-dominance*
- if \hat{z} is semi-congruent to \hat{z}' , then all correct extensions of \hat{z}' are also correct extensions of \hat{z}
- Semi-congruence tries to push feasibility into a partial solution

Step 4: Calculate the Semi-Congruence Condition

$$\forall e, \hat{z}, \hat{z}' \cdot \hat{z} \rightsquigarrow_x \hat{z}' \Rightarrow o(\hat{z}' \oplus e) \Rightarrow o(\hat{z} \oplus e)$$

Extension-dominance condition for Scheduling Problem

$$\begin{aligned} & o(x, \hat{z} \oplus e) \\ = & \{\text{defn of } o \text{ applied to a partial soln}\} \\ & \exists z \cdot \chi(z, \hat{z} \oplus e) \wedge o(x, z) \\ = & \{\text{defn of } \chi, o\} \\ & \exists z \cdot z = \hat{z} \wedge \text{asSet}(z) = \text{dom}(x) \\ = & \{\text{quantifier elimination}\} \\ & \text{asSet}(\hat{z}) = \text{dom}(x) \\ = & \{o(x, \hat{z}' \oplus e) \text{ ie. } \text{asSet}(\hat{z}') = \text{dom}(x)\} \\ & \text{asSet}(\hat{z}) = \text{asSet}(\hat{z}') \end{aligned}$$

The proofs are intended to be simple enough that an automated theorem prover (suitably equipped) could carry them out

Step 4: Calculate the Semi-Congruence Condition

$$\forall e, \hat{z}, \hat{z}' \cdot \hat{z} \rightsquigarrow_x \hat{z}' \Rightarrow o(\hat{z}' \oplus e) \Rightarrow o(\hat{z} \oplus e)$$

Extension-dominance condition for Scheduling Problem

$$\begin{aligned} & o(x, \hat{z} \oplus e) \\ = & \{ \text{defn of } o \text{ applied to a partial soln} \} \\ & \exists z \cdot \chi(z, \hat{z} \oplus e) \wedge o(x, z) \\ = & \{ \text{defn of } \chi, o \} \\ & \exists z \cdot z = \hat{z} \wedge \text{asSet}(z) = \text{dom}(x) \\ = & \{ \text{quantifier elimination} \} \\ & \text{asSet}(\hat{z}) = \text{dom}(x) \\ = & \{ o(x, \hat{z}' \oplus e) \text{ ie. } \text{asSet}(\hat{z}') = \text{dom}(x) \} \\ & \text{asSet}(\hat{z}) = \text{asSet}(\hat{z}') \end{aligned}$$

The proofs are intended to be simple enough that an automated theorem prover (suitably equipped) could carry them out

Step 4: Calculate the Semi-Congruence Condition

$$\forall e, \hat{z}, \hat{z}' \cdot \hat{z} \rightsquigarrow_x \hat{z}' \Rightarrow o(\hat{z}' \oplus e) \Rightarrow o(\hat{z} \oplus e)$$

Extension-dominance condition for Scheduling Problem

$$\begin{aligned} & o(x, \hat{z} \oplus e) \\ = & \{\text{defn of } o \text{ applied to a partial soln}\} \\ & \exists z \cdot \chi(z, \hat{z} \oplus e) \wedge o(x, z) \\ = & \{\text{defn of } \chi, o\} \\ & \exists z \cdot z = \hat{z} \wedge \text{asSet}(z) = \text{dom}(x) \\ = & \{\text{quantifier elimination}\} \\ & \text{asSet}(\hat{z}) = \text{dom}(x) \\ = & \{o(x, \hat{z}' \oplus e) \text{ ie. } \text{asSet}(\hat{z}') = \text{dom}(x)\} \\ & \text{asSet}(\hat{z}) = \text{asSet}(\hat{z}') \end{aligned}$$

The proofs are intended to be simple enough that an automated theorem prover (suitably equipped) could carry them out

Step 4: Calculate the Semi-Congruence Condition

$$\forall e, \hat{z}, \hat{z}' \cdot \hat{z} \rightsquigarrow_x \hat{z}' \Rightarrow o(\hat{z}' \oplus e) \Rightarrow o(\hat{z} \oplus e)$$

Extension-dominance condition for Scheduling Problem

$$\begin{aligned} & o(x, \hat{z} \oplus e) \\ = & \{\text{defn of } o \text{ applied to a partial soln}\} \\ & \exists z \cdot \chi(z, \hat{z} \oplus e) \wedge o(x, z) \\ = & \{\text{defn of } \chi, o\} \\ & \exists z \cdot z = \hat{z} \wedge \text{asSet}(z) = \text{dom}(x) \\ = & \{\text{quantifier elimination}\} \\ & \text{asSet}(\hat{z}) = \text{dom}(x) \\ = & \{o(x, \hat{z}' \oplus e) \text{ ie. } \text{asSet}(\hat{z}') = \text{dom}(x)\} \\ & \text{asSet}(\hat{z}) = \text{asSet}(\hat{z}') \end{aligned}$$

The proofs are intended to be simple enough that an automated theorem prover (suitably equipped) could carry them out

Step 4: Calculate the Semi-Congruence Condition

$$\forall e, \hat{z}, \hat{z}' \cdot \hat{z} \rightsquigarrow_x \hat{z}' \Rightarrow o(\hat{z}' \oplus e) \Rightarrow o(\hat{z} \oplus e)$$

Extension-dominance condition for Scheduling Problem

$$\begin{aligned} & o(x, \hat{z} \oplus e) \\ = & \{\text{defn of } o \text{ applied to a partial soln}\} \\ & \exists z \cdot \chi(z, \hat{z} \oplus e) \wedge o(x, z) \\ = & \{\text{defn of } \chi, o\} \\ & \exists z \cdot z = \hat{z} \wedge \text{asSet}(z) = \text{dom}(x) \\ = & \{\text{quantifier elimination}\} \\ & \text{asSet}(\hat{z}) = \text{dom}(x) \\ = & \{o(x, \hat{z}' \oplus e) \text{ ie. } \text{asSet}(\hat{z}') = \text{dom}(x)\} \\ & \text{asSet}(\hat{z}) = \text{asSet}(\hat{z}') \end{aligned}$$

The proofs are intended to be simple enough that an automated theorem prover (suitably equipped) could carry them out

Step 4: Calculate the Semi-Congruence Condition

$$\forall e, \hat{z}, \hat{z}' \cdot \hat{z} \rightsquigarrow_x \hat{z}' \Rightarrow o(\hat{z}' \oplus e) \Rightarrow o(\hat{z} \oplus e)$$

Extension-dominance condition for Scheduling Problem

$$\begin{aligned} & o(x, \hat{z} \oplus e) \\ = & \{\text{defn of } o \text{ applied to a partial soln}\} \\ & \exists z \cdot \chi(z, \hat{z} \oplus e) \wedge o(x, z) \\ = & \{\text{defn of } \chi, o\} \\ & \exists z \cdot z = \hat{z} \wedge \text{asSet}(z) = \text{dom}(x) \\ = & \{\text{quantifier elimination}\} \\ & \text{asSet}(\hat{z}) = \text{dom}(x) \\ = & \{o(x, \hat{z}' \oplus e) \text{ ie. } \text{asSet}(\hat{z}') = \text{dom}(x)\} \\ & \text{asSet}(\hat{z}) = \text{asSet}(\hat{z}') \end{aligned}$$

The proofs are intended to be simple enough that an automated theorem prover (suitably equipped) could carry them out

Step 4: Calculate the Semi-Congruence Condition

$$\forall e, \hat{z}, \hat{z}' \cdot \hat{z} \rightsquigarrow_x \hat{z}' \Rightarrow o(\hat{z}' \oplus e) \Rightarrow o(\hat{z} \oplus e)$$

Extension-dominance condition for Scheduling Problem

$$\begin{aligned} & o(x, \hat{z} \oplus e) \\ = & \{\text{defn of } o \text{ applied to a partial soln}\} \\ & \exists z \cdot \chi(z, \hat{z} \oplus e) \wedge o(x, z) \\ = & \{\text{defn of } \chi, o\} \\ & \exists z \cdot z = \hat{z} \wedge \text{asSet}(z) = \text{dom}(x) \\ = & \{\text{quantifier elimination}\} \\ & \text{asSet}(\hat{z}) = \text{dom}(x) \\ = & \{o(x, \hat{z}' \oplus e) \text{ ie. } \text{asSet}(\hat{z}') = \text{dom}(x)\} \\ & \text{asSet}(\hat{z}) = \text{asSet}(\hat{z}') \end{aligned}$$

The proofs are intended to be simple enough that an automated theorem prover (suitably equipped) could carry them out

Step 4 (contd)

- For the extension-dominance condition, turns out its sufficient to determine when $\widehat{c}(x, \widehat{z}) \leq \widehat{c}(x, \widehat{z}')$
- However, if we try to calculate such a condition as we have done for semi-congruence we will end up with an expression that involves the values of individual task task times 🤖

Is there a simpler form?

Introducing.. Exchange Tactic

- Tactics provide hints to the developer about where to begin based on the shape of the formula
- Analogous in intent to tactics used in integration eg. “integration by parts”, “integration by partial fractions”, “integration by change of variable”, etc.

Try to derive a dominance relation by comparing a partial solution $\hat{y} \oplus a \oplus \alpha \oplus b$ to a variant obtained by exchanging a pair of terms, that is, $\hat{y} \oplus b \oplus \alpha \oplus a$

Back to Step 4: Apply the tactic and calculate

Partial Sched z^{\wedge}



Partial Sched z^{\wedge}'



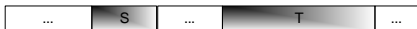
Extension-dominance condition for Scheduling Problem

$$\begin{aligned} & c(x, \hat{z}) \leq c(x, \hat{z}') \\ = & \{\text{unfold defn of } c\} \\ & \sum_{i=1}^m ct(\hat{z}, i) + ct(\hat{z}, a) + ct(\hat{z}, a+1 \dots b-1) + ct(\hat{z}, b) + \sum_{i=b+1}^n ct(\hat{z}, i) \\ & \leq \\ & \sum_{i=1}^m ct(\hat{z}', i) + ct(\hat{z}', a) + ct(\hat{z}', a+1 \dots b-1) + ct(\hat{z}', b) + \sum_{i=b+1}^n ct(\hat{z}', i) \\ = & \{\text{unfold defn of } ct\} \\ & \dots \\ = & \{\text{use that } \hat{z}(i) = \hat{z}'(i) \text{ for all } i \neq a, b \text{ \& some simple algebra}\} \\ & x(\hat{z}_a) \leq x(\hat{z}_b) \end{aligned}$$

That is, a^{th} job before b^{th} job is better than b^{th} job before a^{th} job if the duration of the a^{th} job is less than the duration of the b^{th} job

Back to Step 4: Apply the tactic and calculate

Partial Sched z^\wedge



Partial Sched z^\wedge'



Extension-dominance condition for Scheduling Problem

$$c(x, \hat{z}) \leq c(x, \hat{z}')$$

= {unfold defn of c }

$$\sum_{i=1}^m ct(\hat{z}, i) + ct(\hat{z}, a) + ct(\hat{z}, a+1 \dots b-1) + ct(\hat{z}, b) + \sum_{i=b+1}^n ct(\hat{z}, i)$$

$$\leq \sum_{i=1}^m ct(\hat{z}', i) + ct(\hat{z}', a) + ct(\hat{z}', a+1 \dots b-1) + ct(\hat{z}', b) + \sum_{i=b+1}^n ct(\hat{z}', i)$$

= {unfold defn of ct }

...

= {use that $\hat{z}(i) = \hat{z}'(i)$ for all $i \neq a, b$ & some simple algebra}

$$x(\hat{z}_a) \leq x(\hat{z}_b)$$

That is, a^{th} job before b^{th} job is better than b^{th} job before a^{th} job if the duration of the a^{th} job is less than the duration of the b^{th} job

Back to Step 4: Apply the tactic and calculate



Extension-dominance condition for Scheduling Problem

$$\begin{aligned} & c(x, \hat{z}) \leq c(x, \hat{z}') \\ = & \{\text{unfold defn of } c\} \\ & \sum_{i=1}^m ct(\hat{z}, i) + ct(\hat{z}, a) + ct(\hat{z}, a+1 \dots b-1) + ct(\hat{z}, b) + \sum_{i=b+1}^n ct(\hat{z}, i) \\ & \leq \\ & \sum_{i=1}^m ct(\hat{z}', i) + ct(\hat{z}', a) + ct(\hat{z}', a+1 \dots b-1) + ct(\hat{z}', b) + \sum_{i=b+1}^n ct(\hat{z}', i) \\ = & \{\text{unfold defn of } ct\} \\ & \dots \\ = & \{\text{use that } \hat{z}(i) = \hat{z}'(i) \text{ for all } i \neq a, b \text{ \& some simple algebra}\} \\ & x(\hat{z}_a) \leq x(\hat{z}_b) \end{aligned}$$

That is, a^{th} job before b^{th} job is better than b^{th} job before a^{th} job if the duration of the a^{th} job is less than the duration of the b^{th} job

Back to Step 4: Apply the tactic and calculate



Extension-dominance condition for Scheduling Problem

$$\begin{aligned} & c(x, \hat{z}) \leq c(x, \hat{z}') \\ = & \{\text{unfold defn of } c\} \\ & \sum_{i=1}^m ct(\hat{z}, i) + ct(\hat{z}, a) + ct(\hat{z}, a+1 \dots b-1) + ct(\hat{z}, b) + \sum_{i=b+1}^n ct(\hat{z}, i) \\ & \leq \\ & \sum_{i=1}^m ct(\hat{z}', i) + ct(\hat{z}', a) + ct(\hat{z}', a+1 \dots b-1) + ct(\hat{z}', b) + \sum_{i=b+1}^n ct(\hat{z}', i) \\ = & \{\text{unfold defn of } ct\} \\ & \dots \\ = & \{\text{use that } \hat{z}(i) = \hat{z}'(i) \text{ for all } i \neq a, b \text{ \& some simple algebra}\} \\ & x(\hat{z}_a) \leq x(\hat{z}_b) \end{aligned}$$

That is, a^{th} job before b^{th} job is better than b^{th} job before a^{th} job if the duration of the a^{th} job is less than the duration of the b^{th} job

Back to Step 4: Apply the tactic and calculate



Extension-dominance condition for Scheduling Problem

$$\begin{aligned} & c(x, \hat{z}) \leq c(x, \hat{z}') \\ = & \{\text{unfold defn of } c\} \\ & \sum_{i=1}^m ct(\hat{z}, i) + ct(\hat{z}, a) + ct(\hat{z}, a+1 \dots b-1) + ct(\hat{z}, b) + \sum_{i=b+1}^n ct(\hat{z}, i) \\ & \leq \\ & \sum_{i=1}^m ct(\hat{z}', i) + ct(\hat{z}', a) + ct(\hat{z}', a+1 \dots b-1) + ct(\hat{z}', b) + \sum_{i=b+1}^n ct(\hat{z}', i) \\ = & \{\text{unfold defn of } ct\} \\ & \dots \\ = & \{\text{use that } \hat{z}(i) = \hat{z}'(i) \text{ for all } i \neq a, b \text{ \& some simple algebra}\} \\ & x(\hat{z}_a) \leq x(\hat{z}_b) \end{aligned}$$

That is, a^{th} job before b^{th} job is better than b^{th} job before a^{th} job if the duration of the a^{th} job is less than the duration of the b^{th} job

Back to Step 4: Apply the tactic and calculate



Extension-dominance condition for Scheduling Problem

$$\begin{aligned} & c(x, \hat{z}) \leq c(x, \hat{z}') \\ = & \{\text{unfold defn of } c\} \\ & \sum_{i=1}^m ct(\hat{z}, i) + ct(\hat{z}, a) + ct(\hat{z}, a+1 \dots b-1) + ct(\hat{z}, b) + \sum_{i=b+1}^n ct(\hat{z}, i) \\ & \leq \\ & \sum_{i=1}^m ct(\hat{z}', i) + ct(\hat{z}', a) + ct(\hat{z}', a+1 \dots b-1) + ct(\hat{z}', b) + \sum_{i=b+1}^n ct(\hat{z}', i) \\ = & \{\text{unfold defn of } ct\} \\ & \dots \\ = & \{\text{use that } \hat{z}(i) = \hat{z}'(i) \text{ for all } i \neq a, b \text{ \& some simple algebra}\} \\ & x(\hat{z}_a) \leq x(\hat{z}_b) \end{aligned}$$

That is, a^{th} job before b^{th} job is better than b^{th} job before a^{th} job if the duration of the a^{th} job is less than the duration of the b^{th} job

And finally

- By using the tactic we got an expression that a theorem prover (suitably equipped with a basic algebra) could simplify
- By combining semi-congruence and extension-dominance, we have verified Axiom A1
- A similar process for constructively verifying A2 gives us that the greedy choice (\hat{z}^*) is the task with the shortest processing time
- This is the well-known Shortest Processing Time rule discovered by W.E. Smith in 1956.

Another Example: Huffman Encoding

Example Input

Character	Frequency
a	45,000
b	13,000
c	12,000
d	16,000
e	9,000
f	5,000

Example Output

Character	Codeword
a	0
b	101
c	100
d	111
e	1101
f	1100

David Huffman devised a greedy algorithm for this in 1952³ (still used as the backend of data compression programs)

³D.A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", Proceedings of the I.R.E., September 1952

Step 1: specify the problem

Example: Huffman Encoding

$D \mapsto Char \rightarrow Frequency$

$R \mapsto Char \rightarrow [Bit]$

$o \mapsto \lambda x, z \cdot dom(z) = dom(x)$

$\wedge \forall c \neq c' \in dom(z) \cdot \neg prefixOf(z(c), z(c'))$

$prefixOf(s, t) = \exists u \cdot t = s++u \vee s = t++u$

$c \mapsto \lambda x, z \cdot \sum_{i \in dom(z)} x(i) \cdot \|z(i)\|$

Step 2: Instantiate the operators

$\widehat{R}, \widehat{z}_0, \ll, \chi$

- A good way of ensuring a condition is to fold it into a data structure
 - the rules for constructing that data structure ensure that the condition is automatically satisfied.
 - a binary tree in which the leaves are the letters, and the path from the root to the leaf provides the code for that letter, ensures that the resulting codes are automatically prefix-free
 - this gives us a type for partial solutions \widehat{R} which is different from R
- Initial config (\widehat{z}_0) is the ordered collection of leaves representing the letters
- Extract (χ) checks there is only one tree left and generates the path to each leaf
- Form a subspace (\ll) by merging together two trees

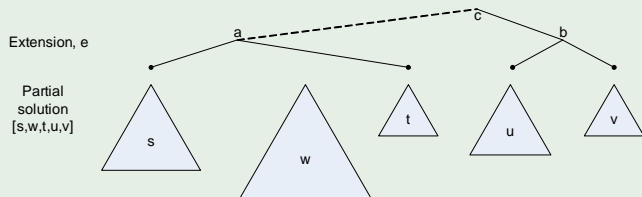
Formally:

$\hat{R} \mapsto [BinTree]$
 $\hat{z}_0 \mapsto \lambda x \cdot asList(dom(x))$
 $\triangleleft \mapsto \lambda(x, \hat{z}, \hat{z}') \cdot \exists s, t \in \hat{z}. \hat{z}' = [\langle s, t \rangle \mid (\hat{z} - s - t)]$
 $\chi \mapsto \lambda(z, \hat{z}) \cdot \|\hat{z}\| = 1 \wedge \forall p \in paths(\hat{z}) \cdot z(last(p)) = first(p)$
 $paths(\langle s, t \rangle) = (map\ prefix0\ paths(s)) ++ (map\ prefix1\ paths(t))$
 $paths(l) = [l]$
 $prefix0(p) = [0 \mid p],\ prefix1(p) = [1 \mid p]$

Step 3: What should the \oplus operator look like for Huffman?

- Should be of type $[BinTree] \times T \rightarrow [BinTree]$, for some T
- Seems natural that T should somehow specify which trees in its left argument to merge
- $\Rightarrow: \hat{z} \oplus (i, j) = [\langle \hat{z}_i, \hat{z}_j \rangle \mid \hat{z} - \hat{z}_i - \hat{z}_j]$

Example: merge s and t, merge u and v



Described by the expression $(\hat{z} \oplus (1, 3)) \oplus (3, 4) = \hat{z} \oplus (1, 3) \oplus (3, 4)$.
Extension is $(1, 3) \oplus (3, 4)$.

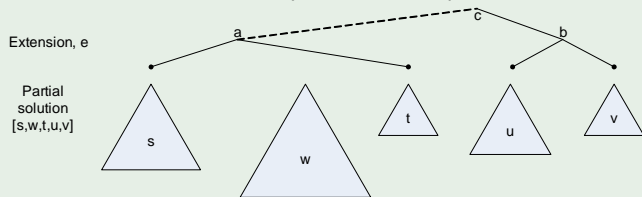
Step 4: Calculate the dominance relation

- Semi-congruence is straightforward as it was in the Scheduling problem: \hat{z} semi-congruent to \hat{z}' if $\|\hat{z}\| = \|\hat{z}'\|$.
- But if we try to calculate the extension-dominance condition the same way we end up with an expression that involves the depths of individual leaves in the trees.

Use the tactic again..

The Exchange tactic applied

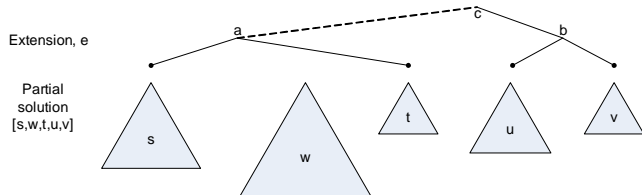
Given a partial solution \hat{y} , suppose trees s and t are merged first, and at some later point the tree containing s and t is merged with a tree formed from merging u and v (tree $satcubv$), forming a partial solution \hat{z} .



Applying the exchange tactic, when is this better than a partial solution \hat{z}' resulting from swapping the mergers in \hat{z} , ie merging u and v first and then s and t ?

Back to Step 4: Now calculate

$h = \text{depth of } s \text{ (resp. } u \text{) from the grandparent of } u \text{ (resp. } s \text{) in } \hat{z} \text{ (resp } \hat{z}')$



Calculate the extension- dominance condition for Huffman

$$\begin{aligned} & c(\hat{z}) \leq c(\hat{z}') \\ = & \{ \text{unfold defn of } c \text{ as before} \} \\ & \dots \\ \Leftarrow & \{ \text{algebra} \} \\ & \sum_{i=1}^{|lvs(s)|} x(lvs(s)_i) + \sum_{i=1}^{|lvs(t)|} x(lvs(t)_i) \leq \sum_{i=1}^{|lvs(u)|} x(lvs(u)_i) + \sum_{i=1}^{|lvs(v)|} x(lvs(v)_i) \\ & \wedge h > 2 \end{aligned}$$

Phew! Almost There

- Using a very similar technique to what we just did, we can constructively verify Axiom A2
- We will find that the greedy choice (\hat{z}^*) is just the pair of trees whose sums of letter frequencies is the least.
- This is the same criterion used by Huffman's algorithm. Of course, for efficiency, in the standard algorithm, the sums are maintained at the roots of the trees as the algorithm progresses. We would automatically arrive at a similar procedure after applying *finite differencing* transforms

Summary & Conclusions

- Have shown how to calculate instantiations for the dominance reln
- Introduced a tactic (Exchange) for when things are not so obvious
- Have used this approach to synthesize other greedy algorithms (MST, Prof. Midas' Driving Problem, etc)
- Program synthesis is an effective way of generating efficient code that is correct-by-construction
- May be of value to the verification community?
 - Instead of calculating, operators are written down by inspection and then verified
 - Verifying a small number of definitions is easier than verifying a complete program. (*Introduction to Algorithms*⁴ devotes about 7 pages to explaining the Huffman problem and proving the algorithm correct!)

The End!

⁴Cormen et. al, pp. 385-392, 2001

Summary & Conclusions

- Have shown how to calculate instantiations for the dominance reln
- Introduced a tactic (Exchange) for when things are not so obvious
- Have used this approach to synthesize other greedy algorithms (MST, Prof. Midas' Driving Problem, etc)
- Program synthesis is an effective way of generating efficient code that is correct-by-construction
- May be of value to the verification community?
 - Instead of calculating, operators are written down by inspection and then verified
 - Verifying a small number of definitions is easier than verifying a complete program. (*Introduction to Algorithms*⁴ devotes about 7 pages to explaining the Huffman problem and proving the algorithm correct!)

The End!

⁴Cormen et. al, pp. 385-392, 2001

Our approach to program synthesis

- *Given*: a pre/post condition specification of the problem
- *Synthesize*: an *efficient* algorithm
- *Using*: An off-the-shelf Algorithm Theory
- *..And*: *calculation* of program components

The Greedy Global Search (GGS) algorithm class

- Supplies a *program schema* (template) containing *operators* whose semantics is axiomatically defined
- Operators must be instantiated by the user (developer) in accordance with the axioms.
- Two kinds of operators: the basic *space-forming* ones and a more advanced one which carries out the *greedy* choice

Program Schema for the GGS class

```
--given x:D returns optimal (wrt. c) z:R satisfying o(x,z)
function solve :: D -> {R}
solve x = gsolve x  $\hat{z}_0(x)$  {}

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

function subspaces :: D ->  $\hat{R}$  ->  $\{\hat{R}\}$ 
subspaces x  $\hat{r}$  = { $\hat{s}$  :  $\hat{s} \prec_x \hat{r}$ }

function opt :: ((D,R) -> C) ->  $\{\hat{R}\}$  ->  $\{\hat{R}\}$ 
opt c {s} = {s}
opt c {s,t} = if c(x,s) > c(x,t) then {s} else {t}
```

The program schema is not synthesized by the developer!

Program Schema for the GGS class

```
--given x:D returns optimal (wrt. c) z:R satisfying o(x,z)
function solve :: D -> {R}
solve x = gsolve x  $\hat{z}_0(x)$  {}

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

function subspaces :: D ->  $\hat{R}$  ->  $\{\hat{R}\}$ 
subspaces x  $\hat{r}$  = {s:  $\hat{s} <_x \hat{r}$ }

function opt :: ((D,R) -> C) ->  $\{\hat{R}\}$  ->  $\{\hat{R}\}$ 
opt c {s} = {s}
opt c {s,t} = if c(x,s) > c(x,t) then {s} else {t}
```

The program schema is not synthesized by the developer!

Running Example : Optimum Prefix-Free Codes

Devise an encoding, as a binary string, for each of the characters in a given text file so as to minimize the overall size of the file. For ease of decoding, the code is required to be prefix-free, that is no encoding of a character is the prefix of the encoding of another character (e.g. assigning "0" to 'a' and "01" to 'b' would not be allowed).

- David Huffman devised a greedy algorithm for this in 1952⁵ (still used as the backend of image compression programs)
- *Introduction to Algorithms* (Cormen et. al, 2001) devotes about 5 pages to explaining the algorithm and proving its correctness

⁵D.A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", Proceedings of the I.R.E., September 1952

Running Example : Optimum Prefix-Free Codes

Devise an encoding, as a binary string, for each of the characters in a given text file so as to minimize the overall size of the file. For ease of decoding, the code is required to be prefix-free, that is no encoding of a character is the prefix of the encoding of another character (e.g. assigning "0" to 'a' and "01" to 'b' would not be allowed).

- David Huffman devised a greedy algorithm for this in 1952⁵ (still used as the backend of image compression programs)
- *Introduction to Algorithms* (Cormen et. al, 2001) devotes about 5 pages to explaining the algorithm and proving its correctness

⁵D.A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", Proceedings of the I.R.E., September 1952

The operators to be instantiated

Operator	Type	Description
\hat{z}_0	$D \rightarrow \hat{R}$	forms the initial space (root node)
χ	$D \times \hat{R} \rightarrow Bool$	can the given solution can be extracted from the given partial solution?
\leq	$D \times \hat{R} \times \hat{R} \rightarrow Bool$	is the 1st space a subspace of the 2nd?
δ_x	$D \times \hat{R} \times \hat{R} \rightarrow Bool$	does the 1st subspace dominate the 2nd?

The first 3 can usually be written down by inspection of the problem, and subsequently verified that they satisfy the axioms.

What axioms?

- A1. $i(x) \wedge o(x, z) \Rightarrow z \in \hat{z}_0(x)$
- A2. $i(x) \Rightarrow (z \in \hat{y} \Leftrightarrow \exists \hat{z} \cdot \hat{z} \triangleleft_x^* \hat{y} \wedge \chi(z, \hat{z}))$
- A3. $\hat{z} \delta_x Z \Rightarrow (\exists z \in \hat{z}, o(x, z), \forall \hat{z}' \in Z, \forall z' \in \hat{z}' \cdot o(x, z') \Rightarrow \wedge c(x, z) \geq c(x, z'))$
- A4. $i(x) \wedge (\exists z \in \hat{y} \cdot o(x, z)) \Rightarrow$
 $(\exists z^* \cdot \chi(z^*, \hat{y}) \wedge o(x, z^*) \wedge c(x, z^*) = c^*(\hat{y})) \vee \exists \hat{z}^* \triangleleft_x \hat{y} \cdot \hat{z}^* \delta_x ss(\hat{y})$

Semi-Congruence and Extension-Dominance

Definition: Semi-Congruence

is a relation $\rightsquigarrow_x \subseteq \widehat{R}^2$ such that

$$\forall e, \widehat{z}, \widehat{z}' \cdot \widehat{z} \rightsquigarrow_x \widehat{z}' \Rightarrow o(\widehat{z}' \oplus e) \Rightarrow o(\widehat{z} \oplus e)$$

Informally, if \widehat{z} is semi-congruent with \widehat{z}' then any feasible completion of \widehat{z}' is a feasible completion of \widehat{z} .

Then we need to say something about when one space is “better” than another.

Definition: Extension-Dominance

is a relation $\widehat{\delta}_x \subseteq \widehat{R}^2$ such that

$$\forall e, \widehat{z}, \widehat{z}' \cdot \widehat{z} \widehat{\delta}_x \widehat{z}' \Rightarrow o(\widehat{z} \oplus e) \wedge o(\widehat{z}' \oplus e) \Rightarrow c(\widehat{z} \oplus e) \leq c(\widehat{z}' \oplus e)$$

If \widehat{z} weakly dominates \widehat{z}' , then any feasible completion of \widehat{z} is at least as good as the same feasible completion of \widehat{z}'

Semi-Congruence and Extension-Dominance

Definition: Semi-Congruence

is a relation $\rightsquigarrow_x \subseteq \widehat{R}^2$ such that

$$\forall e, \widehat{z}, \widehat{z}' \cdot \widehat{z} \rightsquigarrow_x \widehat{z}' \Rightarrow o(\widehat{z}' \oplus e) \Rightarrow o(\widehat{z} \oplus e)$$

Informally, if \widehat{z} is semi-congruent with \widehat{z}' then any feasible completion of \widehat{z}' is a feasible completion of \widehat{z} .

Then we need to say something about when one space is “better” than another.

Definition: Extension-Dominance

is a relation $\widehat{\delta}_x \subseteq \widehat{R}^2$ such that

$$\forall e, \widehat{z}, \widehat{z}' \cdot \widehat{z} \widehat{\delta}_x \widehat{z}' \Rightarrow o(\widehat{z} \oplus e) \wedge o(\widehat{z}' \oplus e) \Rightarrow c(\widehat{z} \oplus e) \leq c(\widehat{z}' \oplus e)$$

If \widehat{z} weakly dominates \widehat{z}' , then any feasible completion of \widehat{z} is at least as good as the same feasible completion of \widehat{z}'

Dominance Relations (contd.)

To get a dominance test, combine the two

Theorem (Dominance)

$$\forall \widehat{z}, \widehat{z}' \cdot \widehat{z} \widehat{\delta}_x \widehat{z}' \wedge \widehat{z} \rightsquigarrow \widehat{z}' \Rightarrow c^*(\widehat{z}) \leq c^*(\widehat{z}')$$

ie., if \widehat{z} is semi-congruent with \widehat{z}' and \widehat{z} extension-dominates \widehat{z}' then the cost of the best solution in \widehat{z} is at least as good as the best solution in \widehat{z}'

When $c^*(\widehat{z}) \leq c^*(\widehat{z}')$ we say \widehat{z} *dominates* \widehat{z}' , written $\widehat{z} \delta_x \widehat{z}'$

A quick and dirty way to get an extension-dominance condition:

Theorem (Cost Distribution)

If c distributes over \oplus and $c(\widehat{z}) \leq c(\widehat{z}')$ then $\widehat{z} \widehat{\delta}_x \widehat{z}'$

Dominance Relations (contd.)

To get a dominance test, combine the two

Theorem (Dominance)

$$\forall \widehat{z}, \widehat{z}' \cdot \widehat{z} \widehat{\delta}_x \widehat{z}' \wedge \widehat{z} \rightsquigarrow \widehat{z}' \Rightarrow c^*(\widehat{z}) \leq c^*(\widehat{z}')$$

ie., if \widehat{z} is semi-congruent with \widehat{z}' and \widehat{z} extension-dominates \widehat{z}' then the cost of the best solution in \widehat{z} at least as good as the best solution in \widehat{z}'

When $c^*(\widehat{z}) \leq c^*(\widehat{z}')$ we say \widehat{z} *dominates* \widehat{z}' , written $\widehat{z} \delta_x \widehat{z}'$

A quick and dirty way to get an extension-dominance condition:

Theorem (Cost Distribution)

If c distributes over \oplus and $c(\widehat{z}) \leq c(\widehat{z}')$ then $\widehat{z} \widehat{\delta}_x \widehat{z}'$

Back to Step 4: Apply the tactic and calculate

Extension-dominance condition for Scheduling Problem

Now we can derive the extension-dominance condition (m is $|dom(z.f)|$):

$$\begin{aligned} & c(z) \leq c(z') \\ = & \{\text{unfold defn of } c\} \\ & \sum_{i=1}^m ct(z, i) + ct(z, a) + ct(z, a + 1 \dots b - 1) + ct(z, b) + \sum_{i=b+1}^n ct(z, i) \\ & \leq \\ & \sum_{i=1}^m ct(z', i) + ct(z', a) + ct(z', a + 1 \dots b - 1) + ct(z', b) + \sum_{i=b+1}^n ct(z', i) \\ = & \{\text{unfold defn of } ct\} \\ & \sum_{i=1}^m \sum_{j=1}^i x(z_j) + \sum_{j=1}^{a-1} x(z_j) + x(z_a) + ct(z, a + 1 \dots b - 1) \\ & + (\sum_{j=1}^{a-1} x(z_j) + s.p + \sum_{j=a+1}^{b-1} x(z_j) + x(z_b) + \sum_{i=b+1}^n ct(z, i)) \\ & \leq \\ & \sum_{i=1}^m \sum_{j=1}^i x(z'_j) + \sum_{j=1}^{a-1} x(z'_j) + x(z_b) + ct(z', a + 1 \dots b - 1) \\ & + (\sum_{j=1}^{a-1} x(z'_j) + t.p + \sum_{j=a+1}^{b-1} x(z'_j)) + x(z_a) + \sum_{i=b+1}^n ct(z', i) \\ = & \{\text{algebra and } z.i = z'.i, i \neq a, b\} \\ & 2(x(z_s)) + x(z_b) \leq 2(x(z_b)) + x(z_a) \\ = & \\ & x(z_a) \leq x(z_b) \end{aligned}$$

That is, ??? is better than ??? if the duration of the a^{th} task is less than the duration of the b^{th} task

Thanks to the tactic, the proof has been reduced to something simple

Back to Step 4: Apply the tactic and calculate

Extension-dominance condition for Scheduling Problem

Now we can derive the extension-dominance condition (m is $|dom(z.f)|$):

$$\begin{aligned} & c(z) \leq c(z') \\ = & \{\text{unfold defn of } c\} \\ & \sum_{i=1}^m ct(z, i) + ct(z, a) + ct(z, a + 1 \dots b - 1) + ct(z, b) + \sum_{i=b+1}^n ct(z, i) \\ & \leq \\ & \sum_{i=1}^m ct(z', i) + ct(z', a) + ct(z', a + 1 \dots b - 1) + ct(z', b) + \sum_{i=b+1}^n ct(z', i) \\ = & \{\text{unfold defn of } ct\} \\ & \sum_{i=1}^m \sum_{j=1}^i x(z_j) + \sum_{j=1}^{a-1} x(z_j) + x(z_a) + ct(z, a + 1 \dots b - 1) \\ & + (\sum_{j=1}^{a-1} x(z_j) + s.p + \sum_{j=a+1}^{b-1} x(z_j) + x(z_b) + \sum_{i=b+1}^n ct(z, i)) \\ & \leq \\ & \sum_{i=1}^m \sum_{j=1}^i x(z'_j) + \sum_{j=1}^{a-1} x(z'_j) + x(z_b) + ct(z', a + 1 \dots b - 1) \\ & + (\sum_{j=1}^{a-1} x(z'_j) + t.p + \sum_{j=a+1}^{b-1} x(z'_j)) + x(z_a) + \sum_{i=b+1}^n ct(z', i) \\ = & \{\text{algebra and } z.i = z'.i, i \neq a, b\} \\ & 2(x(z_s)) + x(z_b) \leq 2(x(z_b)) + x(z_a) \\ = & \\ & x(z_a) \leq x(z_b) \end{aligned}$$

That is, ??? is better than ??? if the duration of the a^{th} task is less than the duration of the b^{th} task

Thanks to the tactic, the proof has been reduced to something simple

Semi-congruence condition for Scheduling

$$\begin{aligned}
 & o(x, \hat{z} \oplus e) \\
 &= \{\text{push } o \text{ up into the partial solution } \hat{R}\} \\
 &\exists z \cdot \chi(z, \hat{z} \oplus e) \wedge o(x, z) \\
 &= \{\text{defn of } \chi, o\} \\
 &\exists z \cdot z = \hat{z} \wedge \text{asSet}(z) = \text{dom}(x) \\
 &= \{\text{quantifier elimination}\} \\
 &\text{asSet}(\hat{z}) = \text{dom}(x) \\
 &= \{o(x, \hat{z}' \oplus e) \text{ ie. } \text{asSet}(\hat{z}') = \text{dom}(x)\} \\
 &\text{asSet}(\hat{z}) = \text{asSet}(\hat{z}')
 \end{aligned}$$

Step 4: Calculate the semi-congruence relation

We can do this straightforwardly as we did for the Scheduling problem.

The result is $\|\hat{z}\| = \|\hat{z}'\|$

Calculate the semi-congruence condition for Huffman

$$\begin{aligned} & o(x, \hat{z} \oplus e) \\ &= \{\text{lift } o \text{ up to } \hat{R}\} \\ & \exists z \cdot \chi(z, \hat{z} \oplus e) \wedge o(x, z) \\ &= \{\text{defn of } \chi, o\} \\ & \exists z \cdot \|\hat{z} \oplus e\| = 1 \wedge \forall p \in \text{paths}(\hat{z} \oplus e) \cdot z(\text{last}(p)) = \text{first}(p) \wedge \text{dom}(z) = x \wedge \dots \\ &= \{\text{intro defn}\} \\ & \|\hat{z} \oplus e\| = 1 \wedge \text{dom}(z) = x \wedge \dots \\ & \text{where } z = \{\text{last}(p) \mapsto \text{first}(p) \mid p \in \text{paths}(\hat{z} \oplus e)\} \\ & \Leftarrow \{o(x, \hat{z}' \oplus e) \Rightarrow \text{dom}(z') = x \text{ where } z = \{\text{last}(p) \mapsto \text{first}(p) \mid p \in \text{paths}(\hat{z}' \oplus e)\}\} \\ & \|\hat{z} \oplus e\| = 1 \wedge \text{asSet}(\text{lvs}(\hat{z})) = \text{asSet}(\text{lvs}(\hat{z}')) \\ &= \{\text{split does not alter set of leaves}\} \\ & \|\hat{z} \oplus e\| = 1 \\ &= \{\|\hat{z} \oplus e\| = \|\hat{z}\| - \|e\|, \|\hat{z}' \oplus e\| = 1\} \\ & \|\hat{z}\| = \|\hat{z}'\| \end{aligned}$$

Back to Step 4: Apply the tactic and theorem

- $d(T)_i = \text{depth of leaf } i \text{ in a tree } T$
- $h = \text{depth of } s \text{ (resp. } u) \text{ from the grandparent of } u \text{ (resp. } s) \text{ in } \hat{z}$
(resp \hat{z}')

Calculate the extension- dominance condition for Huffman

$$\begin{aligned} & c(\hat{z}) \leq c(\hat{z}') \\ = & \{\text{unfold defn of } c\} \\ & \sum_{i=1}^{|\text{lvs}(s)|} (d(s)(i) + h) \cdot x(\text{lvs}(s)_i) + \sum_{i=1}^{|\text{lvs}(t)|} (d(t)(i) + h) \cdot x(\text{lvs}(t)_i) \\ & + \sum_{i=1}^{|\text{lvs}(u)|} (d(u)(i) + 2) \cdot x(\text{lvs}(u)_i) + \sum_{i=1}^{|\text{lvs}(v)|} (d(v)(i) + 2) \cdot x(\text{lvs}(v)_i) \\ & \leq \\ & \sum_{i=1}^{|\text{lvs}(u)|} (d(u)(i) + h) \cdot x(\text{lvs}(u)_i) + \sum_{i=1}^{|\text{lvs}(v)|} (d(v)(i) + h) \cdot x(\text{lvs}(v)_i) \\ & + \sum_{i=1}^{|\text{lvs}(s)|} (d(s)(i) + 2) \cdot x(\text{lvs}(s)_i) + \sum_{i=1}^{|\text{lvs}(t)|} (d(t)(i) + 2) \cdot x(\text{lvs}(t)_i) \\ = & \{\text{algebra}\} \\ & (h - 2) \cdot \sum_{i=1}^{|\text{lvs}(s)|} x(\text{lvs}(s)_i) + (h - 2) \cdot \sum_{i=1}^{|\text{lvs}(t)|} x(\text{lvs}(t)_i) \\ & \leq (h - 2) \cdot \sum_{i=1}^{|\text{lvs}(u)|} x(\text{lvs}(u)_i) + (h - 2) \cdot \sum_{i=1}^{|\text{lvs}(v)|} x(\text{lvs}(v)_i) \\ \Leftarrow & \{\text{algebra}\} \\ & \sum_{i=1}^{|\text{lvs}(s)|} x(\text{lvs}(s)_i) + \sum_{i=1}^{|\text{lvs}(t)|} x(\text{lvs}(t)_i) \leq \sum_{i=1}^{|\text{lvs}(u)|} x(\text{lvs}(u)_i) + \sum_{i=1}^{|\text{lvs}(v)|} x(\text{lvs}(v)_i) \\ & \wedge h > 2 \end{aligned}$$

So how does this help?

How does this help us derive a dominance relation between two *subspaces* after a split of a space (e.g. $[\langle s, t \rangle \mid (\hat{y} - s - t)]$ and $[\langle u, v \rangle \mid (\hat{y} - u - v)]$) ? The following theorem shows that the above condition serves as a dominance relation between the two subspaces

Theorem

Given a GGS theory for a constraint satisfaction problem,

$$(\exists \alpha \cdot (\hat{y} \oplus a \oplus \alpha \oplus b) \delta_x (\hat{y} \oplus b \oplus \alpha \oplus a)) \Rightarrow \hat{y} \oplus a \delta_x \hat{y} \oplus b$$