A comparison of well-quasi orders on trees

Mogensen, Torben Ægidius

Published in:
Semantics, Abstract Interpretation, and Reasoning about Programs

DOI:
10.4204/EPTCS.129.3

Publication date:
2013

Document version
Publisher's PDF, also known as Version of record

Citation for published version (APA):
Mogensen, T. Æ. (2013). A comparison of well-quasi orders on trees. In A. Banerjee, O. Danvy, K-G. Doh, & J. Hatcliff (Eds.), *Semantics, Abstract Interpretation, and Reasoning about Programs: essays dedicated to David A. Schmidt on the occasion of his sixtieth birthday, Manhattan, Kansas, USA, 19-20th September 2013* (pp. 30-40). Electronic Proceedings in Theoretical Computer Science, Vol.. 129 https://doi.org/10.4204/EPTCS.129.3

A Comparison of Well-Quasi Orders on Trees

Torben Æ. Mogensen
DIKU, University of Copenhagen
torbenm@diku.dk

Well-quasi orders such as homeomorphic embedding are commonly used to ensure termination of program analysis and program transformation, in particular supercompilation.

We compare eight well-quasi orders on how discriminative they are and their computational complexity. The studied well-quasi orders comprise two very simple examples, two examples from literature on supercompilation and four new proposed by the author.

We also discuss combining several well-quasi orders to get well-quasi orders of higher discriminative power. This adds 19 more well-quasi orders to the list.

1 Introduction

A quasi order $(S, \leq)$ is a set S with a reflexive and transitive binary relation $\leq$ on $S \times S$.

A well-quasi order $(S, \trianglelefteq)$ is a quasi order such that for any infinite sequence of elements $s_0, s_1, \ldots$ where $\forall i \in \mathbb{N} : s_i \in S$, there exists $i, j \in \mathbb{N}$ such that $i < j$ and $s_i \trianglelefteq s_j$. We abbreviate "well-quasi order" to WQO. When the set S is clear from the context, we will sometimes refer to a WQO only by the relation $\trianglelefteq$.

Well-quasi orders are often used to ensure termination of program analyses and program transformations (such as partial evaluation and supercompilation). The idea is that when a sequence of values is produced, production of a new value s_j is seen as "safe" if there is no previous $s_i,\ i < j$ such that $s_i \trianglelefteq s_j$. If a value is found so $s_i \trianglelefteq s_j$, s_i is replaced by a combination of s_i and s_j in a process called *widening* or *generalisation* and the sequence is recomputed from this point in the hope that the sequence will be regular (i.e, that it can "loop back" to create a finite cycle). See [4, 5] for a detailed discussion on the use of a particular well-quasi order called *homeomorphic embedding* for online termination control.

Sequences of trees are often used in program transformation (and sometimes in program analysis), and since the set of trees over a finite signature is usually infinite, some form of widening or generalisation is required to ensure termination of such sequences.

We can compare different well-quasi orders for the same set by *discriminative power*: A WQO $(S, \trianglelefteq_1)$ is more discriminative than a WQO $(S, \trianglelefteq_2)$ if $\forall s_1, s_2 \in S:\ s_1 \trianglelefteq_1 s_2 \Rightarrow s_1 \trianglelefteq_2 s_2$, and it is strictly more discriminative if, additionally, $\exists s_1, s_2 \in S:\ s_1 \not\trianglelefteq_1 s_2 \wedge s_1 \trianglelefteq_2 s_2$.

A more discriminative WQO will allow longer sequences before generalisation or widening is applied, but (being a WQO) will still avoid infinite sequences. Since widening or generalisation imply loss of information, this can give more precise analysis and stronger program transformations. On the flip side, longer sequences before generalisation or widening is applied imply longer running time and higher memory use of the analysis or transformation, and transformed programs can also be larger, as fewer parts of the transformed program are merged. So there is a trade off involved. By presenting a range of WQOs with different discriminative power and computational cost, we hope to give researchers a basis for choosing WQOs that works well for their analyses or transformation systems.

We will for simplicity of the presentation assume that all trees are built from a finite signature: A finite set of constructors with finite arities. It is, however, not hard to generalise to the trees where constructors can have multiple (or even unbounded) arities. The size of a tree can be computed in several ways, for example the number of nodes, the number of edges or the sum of these two. We will use the number of nodes (constructor occurrences) as our measure.

2 Properties of well-quasi orders

We will review a few properties of WQOs that we will use in this paper.

Theorem 1 *If S is a finite set, $(S,=)$ is a WQO.*

Proof: Any infinite sequence of elements from a finite set will repeat elements. □

Theorem 2 *If $(S_2, \trianglelefteq)$ is a WQO and f is a total function from S_1 to S_2, then $(S_1, \trianglelefteq^f)$ is a WQO, where $\trianglelefteq^f$ is defined by $x \trianglelefteq^f y$ iff $f(x) \trianglelefteq f(y)$.*

Proof: If $s_0, s_1, \dots$ is an infinite sequence of elements from S_1, then $f(s_0), f(s_1), \dots$ is an infinite sequence of elements from S_2. Since $(S_2, \trianglelefteq)$ is a WQO, there must be $i < j$ such that $f(s_i) \trianglelefteq f(s_j)$ which is the definition of $s_i \trianglelefteq^f s_j$. □

Theorem 3 (Kruskal, 1960) *If T is a set of finite trees, $(T, \trianglelefteq_H)$ is a WQO, where $s \trianglelefteq_H t$ is defined by the rules:*

$$\begin{array}{lcl} c(s_1,\dots,s_n) \trianglelefteq_H c(t_1,\dots,t_n) & \Leftarrow & s_1 \trianglelefteq_H t_1 \wedge \cdots \wedge s_n \trianglelefteq_H t_n \\ s \trianglelefteq_H c(t_1,\dots,t_n) & \Leftarrow & \exists 1 \le i \le n : s \trianglelefteq_H t_i \\ s \not\trianglelefteq_H t & , & \textit{otherwise} \end{array}$$

Basically, $s \trianglelefteq_H t$ if s can be obtained from t by repeatedly replacing a subtree x of t by one of the subtrees of x. This ordering is called *homeomorphic embedding*.

Proof: The proof that $(T, \trianglelefteq_H)$ is a well-quasi order can be found in [3].

Theorem 4 *If $Q = S^*$ is the set of finite sequences over a finite set S, then $(Q, \ll)$, where $\ll$ is the subsequence relation, is a WQO.*

Proof: We map Q to a set of trees T using the mapping q:

$$\begin{array}{lcl} q(\varepsilon) & = & \text{a leaf node labeled } \varepsilon \\ q(aw) & = & \text{a node labeled } a \text{ with a single child } q(w) \end{array}$$

We note that $w_1 \ll w_2$ iff $q(w_1) \trianglelefteq_H q(w_2)$, so by Theorems 3 and 2, $(Q, \ll) = (Q, \trianglelefteq_H^q)$ is a WQO. □

Theorem 5 *If B is the set of bags (multisets) over a finite set S, then $(B, \subseteq)$, where $\subseteq$ is the subset relation on multisets, is a WQO.*

Proof: We map multisets to sequences by sorting the elements based on any total ordering of S. We then note that $b_1 \subseteq b_2$ iff $sort(b_1) \ll sort(b_2)$, so by Theorems 4 and 2, $(B, \subseteq) = (B, \ll^{sort})$ is a WQO. An alternative proof uses that $(B, \subseteq)$ is a *multiset extension* [9] of $(S,=)$. □

Theorem 6 *If $\trianglelefteq$ is a well-quasi order, then any infinite sequence $s_0, s_1, \dots$ contains an infinite increasing subsequence $s_{i_0} \trianglelefteq s_{i_1} \trianglelefteq \dots$*

Proof: Assume that the set $M = \{i \mid \nexists k > i : s_i \trianglelefteq s_k\}$ is infinite. As $\trianglelefteq$ is a well-quasi order, there must be $i, j \in M$ such that $i < j$ and $s_i \trianglelefteq s_j$. But this contradicts the definition of M. Hence, M is finite and has a maximal element m. Now assume that the longest increasing subsequence starting from i_{m+1} is finite. If so, it has a maximal element s_p, such that there are no $q > p : s_p \trianglelefteq s_q$. But that would make $p \in M$, which, because $p > m$, contradicts the fact that m is maximal in M. Hence, we have an infinite increasing subsequence starting from i_{m+1}. □

Theorem 7 *Given two WQOs* $(S, \trianglelefteq_1)$ *and* $(S, \trianglelefteq_2)$*, then* $(S, \trianglelefteq_1 \cap \trianglelefteq_2)$ *defined by* $x\,(\trianglelefteq_1 \cap \trianglelefteq_2)\,y \Leftrightarrow x \trianglelefteq_1 y \wedge x \trianglelefteq_2 y$ *is a WQO.*

Proof: Any infinite sequence $s_0, s_1, \ldots$ will due to Theorem 6 have an infinite increasing subsequence $s_{i_0} \trianglelefteq_1 s_{i_1} \trianglelefteq_1 \ldots$, which (because $\trianglelefteq_2$ is a WQO) will have a pair $s_{i_j} \trianglelefteq_2 s_{i_k}$, where $j < k$. It follows that $s_{i_j}\,(\trianglelefteq_1 \cap \trianglelefteq_2)\,s_{i_k}$. □

3 A selection of well-quasi orders on trees

In the following, we will describe six WQOs on trees with finite signatures. We divide these WQOs into groups based on whether they are defined directly on the trees or by mapping trees to another WQO using Theorem 2.

We use T to denote an otherwise unspecified set of trees over a finite signature Σ. We will also use Σ to denote the set (alphabet) of constructor symbols in Σ. The context should make it clear which meaning is used.

3.1 Well-quasi orders defined directly on trees

$\trianglelefteq_S$: A simple WQO for trees is based on comparison of size: For any two trees t_1, t_2, we define $t_1 \trianglelefteq_S t_2$ iff $t_1 = t_2$ or $|t_1| < |t_2|$, where $|t|$ is the size of the tree t. It is clear that an infinite sequence of trees must either repeat specific trees or increase the size of trees: There are only finitely many different trees of a given size. Hence, $(T, \trianglelefteq_S)$ is a WQO.

$\trianglelefteq_H$: A WQO that is often used for controlling termination of program transformation is *homeomorphic embedding* $(T, \trianglelefteq_H)$ as defined as in Theorem 3.

Homeomorphic embedding has been used for termination proofs for term-rewriting systems [2]. Using homeomorphic embedding to control termination of supercompilation was first proposed in [8].

3.2 Well-quasi orders defined by mapping to sets

$\trianglelefteq_Z$: If S is a finite set, Theorem 1 gives that $(S, =)$ is a WQO. Since T uses a finite signature Σ, $(2^\Sigma, =)$ is a WQO, where 2^Σ is the set of subsets of Σ.

We define $(T, \trianglelefteq_Z)$ by the function f from T to 2^Σ that maps a tree t to the set of constructors used in t, so $s \trianglelefteq_Z t$ iff s and t use the same set of constructors. Since$(2^\Sigma, =)$ is a WQO, $(T, \trianglelefteq_Z) = (T, =^f)$ is by Theorem 2 also a WQO.

$\trianglelefteq_Y$: We propose a variant of $\trianglelefteq_Z$ by mapping a tree to a set of constructors using a different mapping: w maps a tree t to the set of constructors used *at least twice* in t. By the same reasoning as above, $(T, \trianglelefteq_Y) = (T, =^w)$ is WQO. Obviously, this can be generalised to sets of constructors that are used at least 3, 4 or more times.

$\trianglelefteq_Y$ is appropriate for supercompilation and related transformations, as an infinite sequence of trees usually copies some part of an earlier initial tree in more and more copies. Such copying will, eventually, create a tree with the same set of constructors that are used at least twice as an earlier tree. But $\trianglelefteq_Y$ will not stop temporary growth that does not preserve this set, such as replacing a subtree $c(a,a,b)$ by $c(a,b,b)$. Obviously, larger trees will trigger more false positives, so $\trianglelefteq_Y$ works best in combination with other WQOs.

3.3 Well-quasi orders defined by mapping to multisets

For these WQOs we map a tree to the multiset (bag) of its constructors. More precisely, we use a function g from T to the set $B = \mathbb{N}^\Sigma$ of multisets over Σ defined in the following way:

$$\begin{array}{lcl} g(c) & = & \{c\} \\ g(c(s_1,\ldots,s_n)) & = & \{c\} \cup \bigcup_{i=1}^{n} g(s_i) \end{array}$$

where $\cup$ is union on multisets.

$\trianglelefteq_B$: We propose a new WQO $(T, \trianglelefteq_B)$ that directly uses g: $s \trianglelefteq_B t \Leftrightarrow g(s) \subseteq g(t)$.

$(B, \subseteq)$ is a WQO due to Theorem 5, so $(T, \trianglelefteq_B) = (T, \subseteq^g)$ is a WQO by Theorem 2.

We deem $\trianglelefteq_B$ appropriate for supercompilation and related transformations because an infinite sequence usually involves copying nodes in the tree more and more times, which would make the multiset of constructors in a new tree a superset of those in a previous tree.

$\trianglelefteq_M$: Another WQO that has been used for supercompilation [6] is also based on multisets over Σ.

Given two multisets $b_1, b_2 \in B$, we define $b_1 \leq b_2 \Leftrightarrow b_1 = b_2 \vee set(b_1) = set(b_2) \wedge |b_1| < |b_2|$, where $set(b)$ is the set of different elements in b and $|b|$ is the total number of elements in b.[1]

Since there are only finitely many different sets over a finite alphabet, any infinite sequence must have infinitely many bags with the same underlying sets. Since there are only finitely many bags with the same size, this means that, in any infinite sequence, the size of the bags must increase. Hence, $(B, \leq)$ is a WQO.

We define $(T, \trianglelefteq_M)$ to be the quasi order on trees derived from $(B, \leq)$ by the mapping g. In other words, $s \trianglelefteq_M t \Leftrightarrow g(s) \leq g(t)$. By Theorem 2, $(T, \trianglelefteq_M) = (T, \leq^g)$ is a WQO. Basically, $\trianglelefteq_M$ refines $\trianglelefteq_S$ by making trees incomparable if they have different underlying sets.

In [6], subexpressions of the original program are named, and new expressions inherit the names of their progenitors. So any expression will be assigned a multiset of names. Since names of new expressions are based on where in the original program their progenitor expressions occur, two identical expression trees can have different multisets of names. While adding names to nodes in the trees adds information that a plain expression tree does not have, the mapping from trees to multisets is basically the same as g, except that a node with multiple names is mapped to a multiset of all these names instead of to a multiset with just one name.

3.4 Well-quasi orders defined by mapping to strings

$\trianglelefteq_P$: We map trees into strings over a finite alphabet and compare these strings. We map a tree t to a string w by a mapping *Pre* from T to Σ^*, i.e, the set of finite strings over the alphabet Σ. $Pre(t)$ is the string of constructors in t in preorder-traversal order:

[1]Mitchell uses a negated form of this relation and negates the test for termination detection.

$$\begin{array}{lcl} Pre(c) & = & c \\ Pre(c(t_1,\dots,t_n) & = & c\,Pre(t_1)\cdots Pre(t_n) \end{array}$$

Note that, since constructors have fixed arities, *Pre* is injective: Different trees map to different strings.

We compare the strings using the subsequence order: $w_1 \ll w_2$ if w_1 is a subsequence of w_2. By Theorem 4, $(\Sigma^*, \ll)$ is a WQO.

We now define the quasi order $(T, \trianglelefteq_P)$ by $t_1 \trianglelefteq_P t_2 \Leftrightarrow Pre(t_1) \ll Pre(t_2)$. Theorem 2 gives that $(T, \trianglelefteq_P) = (T, \ll^{Pre})$ is a WQO.

$\trianglelefteq_P$ is appropriate for supercompilation because it approximates $\trianglelefteq_H$, which has proven successful for controlling termination in supercompilation, but (as we shall see) $\trianglelefteq_P$ is less costly to compute.

$\trianglelefteq_E$: We can refine $\trianglelefteq_P$ by mapping trees to strings that contain more information about the structure of the tree: In a preorder traversal, a constructor node is visited only once, before its children. But we can add additional visits of the node between and after traversing its children. This is sometimes called an Euler-tour traversal. We define this through a modified traversal function:

$$\begin{array}{lcl} Eul(c) & = & c \\ Eul(c(t_1,\dots,t_n) & = & c_0\,Eul(t_1)\,c_1\cdots Eul(t_n)\,c_n \end{array}$$

where c_i is a symbol that marks that i children of a constructor node labelled c have been traversed.

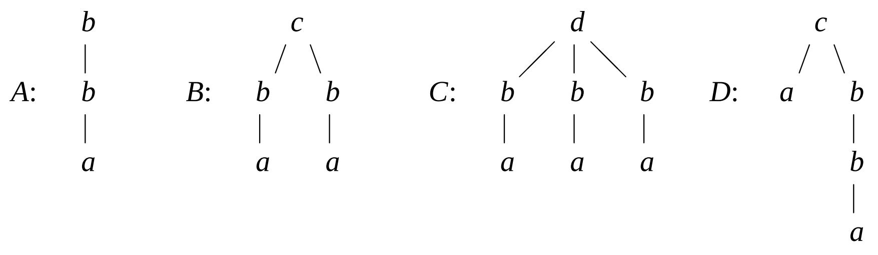
Theorems 4 and 2 give that $(T, \trianglelefteq_E) = (T, \ll^{Eul})$ is a WQO.

$\trianglelefteq_E$ is appropriate for supercompilation for the same reason that $\trianglelefteq_P$ is.

4 Comparing well-quasi orders by discriminative power

We recall that a WQO $\trianglelefteq_1$ is more discriminative than a WQO $\trianglelefteq_2$ if $\forall s_1, s_2 \in S:\ s_1 \trianglelefteq_1 s_2 \Rightarrow s_1 \trianglelefteq_2 s_2$, and it is strictly more discriminative if, additionally, $\exists s_1, s_2 \in S:\ s_1 \not\trianglelefteq_1 s_2 \wedge s_1 \trianglelefteq_2 s_2$.

We will use the following trees in our discussion about the relative discriminative power of WQOs:



It is not hard to see that $t_1 \trianglelefteq_H t_2 \Rightarrow Eul(t_1) \ll Eul(t_2) \Leftrightarrow t_1 \trianglelefteq_E t_2$, so $\trianglelefteq_H$ is more discriminative than $\trianglelefteq_E$. It is also *strictly* more discriminative:

$$Eul(A) = b_0b_0ab_1b_1 \ \ll\ d_0b_0ab_1d_1b_0ab_1d_2b_0ab_1d_3 = Eul(C)$$

so $A \trianglelefteq_E C$, but it is clear from inspection of the trees that $A \not\trianglelefteq_H C$.

It is also clear that $\trianglelefteq_E$ is more discriminative than $\trianglelefteq_P$: If $Eul(t_1) \ll Eul(t_2)$ then, clearly, $Pre(t_1) \ll Pre(t_2)$. It is also strictly more discriminative: $Eul(A) = b_0b_0ab_1b_1 \not\ll c_0b_0ab_1c_1b_0ab_1c_2 = Eul(B)$, but $Pre(A) = bba \ll cbabab = Pre(B)$, so $A \not\trianglelefteq_E B$ while $A \trianglelefteq_P B$.

It is easy to see that $\trianglelefteq_P$ is more discriminative than $\trianglelefteq_B$: If $Pre(t_1) \ll Pre(t_1)$, then $bag(Pre(t_1)) \subseteq bag(Pre(t_1))$, where *bag* maps a string to a bag by ignoring the order of elements. It is also strictly more discriminative, as $B \ntrianglelefteq_P D$ but $B \trianglelefteq_B D$, since the trees B and D map to the same bag of constructors $\{a,a,b,b,c\}$.

It is, however, not the case that $\trianglelefteq_B$ is more discriminative than $\trianglelefteq_M$. We see this by observing that $(B, \subseteq)$ and $(B, \leq)$ have incomparable discriminative power:

On the one hand, $\{a,b,b\} \leq \{a,a,a,b\}$, since their underlying sets are the same and $|\{a,b,b\}| < |\{a,a,a,b\}|$, but on the other hand $\{a,b,b\} \not\subseteq \{a,a,a,b\}$. Also, $\{a\} \subseteq \{a,b\}$ but $\{a\} \not\leq \{a,b\}$, because the underlying sets are different. In fact, because $\trianglelefteq_M$ makes trees with different underlying sets of constructors incomparable, its discriminative power is also incomparable to $\trianglelefteq_P$ and $\trianglelefteq_H$.

It is trivial to see that $\trianglelefteq_M$ is strictly more discriminative than $\trianglelefteq_S$, as $\trianglelefteq_M$ is effectively size comparison restricted to trees with the same underlying set of constructors.

$\trianglelefteq_B$ has incomparable discriminative power to $\trianglelefteq_S$ also: Two different trees of the same size are incomparable by $\trianglelefteq_S$, but if they are built from the same multiset of constructors, they are comparable by $\trianglelefteq_B$.

But $\trianglelefteq_S$ has strictly less discriminative power than $\trianglelefteq_P$ (and, hence, also $\trianglelefteq_E$ and $\trianglelefteq_H$): Since constructors have fixed arity, two trees have the same preorder traversal strings only if they are identical. Two same-size trees will have same-length preorder-traversal strings, and same-length strings can be in the subsequence relation only if they are equal. So two non-identical same-size trees will be incomparable by both $\trianglelefteq_P$ and $\trianglelefteq_S$. But where trees of different size are always comparable by $\trianglelefteq_S$, they need not be comparable by $\trianglelefteq_P$.

$\trianglelefteq_Z$ is clearly strictly less discriminative than $\trianglelefteq_M$, as both consider the underlying set but $\trianglelefteq_Z$ does not consider the size of the trees (or, equivalently, the size of the underlying multisets). The discriminative powers of $\trianglelefteq_Z$ and $\trianglelefteq_S$ are incomparable, as are the discriminative powers of $\trianglelefteq_Z$ and $\trianglelefteq_B$.

$\trianglelefteq_Z$ and $\trianglelefteq_Y$, though related, are of incomparable discriminative power: The trees B and C are comparable by $\trianglelefteq_Y$, as the constructors a and b (and none other) occur twice or more in both trees, but the underlying sets are different so they are not comparable by $\trianglelefteq_Z$. On the other hand D and a tree similar to D but with only one b constructor will be comparable by $\trianglelefteq_Z$ but not by $\trianglelefteq_Y$. $\trianglelefteq_Y$ is, in fact, incomparable with all the other WQOs described in Section 3.

This gives us the following hierarchy of discriminative power:

$$\begin{tikzcd}
& & & \trianglelefteq_H \arrow[d, dash] \\
& & & \trianglelefteq_E \arrow[d, dash] \\
& & \trianglelefteq_M \arrow[dl, dash] \arrow[d, dash] & \trianglelefteq_P \arrow[dl, dash] \arrow[d, dash] \\
\trianglelefteq_Y & \trianglelefteq_Z & \trianglelefteq_S & \trianglelefteq_B
\end{tikzcd}$$

4.1 Combining well-quasi orders

We can combine two or more incomparable WQOs by intersection to get a WQO of more discriminative power.

For example, we can define a WQO $(T, \trianglelefteq_{SB}) = (T, \trianglelefteq_S \cap \trianglelefteq_B)$.[2] In other words, $s \trianglelefteq_{SB} t \Leftrightarrow s \trianglelefteq_S t \wedge$

[2]When we subscript $\trianglelefteq$ with several letters, this denotes the intersection of the orderings defined by $\trianglelefteq$ subscripted with each individual letter.

$s \trianglelefteq_B t$. Since $\trianglelefteq_S$ and $\trianglelefteq_B$ have incomparable discriminative power, it clear that $\trianglelefteq_{SB}$ is strictly more discriminative than both $\trianglelefteq_S$ and $\trianglelefteq_B$. $\trianglelefteq_{SB}$ has strictly less discriminative power than $\trianglelefteq_P$: Consider a tree D' similar to D but with one extra b node in the right branch. It is clear that $B \trianglelefteq_S D'$, since D' is bigger than B and $B \trianglelefteq_B D'$ since the bag of constructors in B is a subset of the bag of constructors in D'. But $B \ntrianglelefteq_P D'$ since $Pre(B) \not\ll Pre(D')$.

From its definition, it is easy to see that $(T, \trianglelefteq_M) = (T, \trianglelefteq_{ZS})$, where $\trianglelefteq_{ZS} = (\trianglelefteq_Z \cap \trianglelefteq_S)$. All other combinations of uncomparable WQOs yield new WQOs. For example, as we saw above, $\trianglelefteq_{SB}$ is less discriminative than $\trianglelefteq_P$ but more discriminative than both $\trianglelefteq_S$ and $\trianglelefteq_B$.

Adding all combinations of incomparable WQOs introduced in Section 3 gives a hierarchy of WQOs which is induced from the partial order shown in the hierarchy above. The only non-obvious results of the combinations are that $\trianglelefteq_{SB}$ is strictly below $\trianglelefteq_P$ and that $\trianglelefteq_M = \trianglelefteq_{ZS}$, which implies, for example, $\trianglelefteq_{MP} = \trianglelefteq_{ZSP} = \trianglelefteq_{ZP}$. The complete hierarchy is too complicated to show graphically, so we show below the hierarchy of the 13 single and combined WQOs that do not involve $\trianglelefteq_Y$. For each of these, there is a more discriminative WQO obtained by combining it with $\trianglelefteq_Y$. Adding also $\trianglelefteq_Y$ itself brings the total count up to 27.

$$\begin{tikzcd}
 & \trianglelefteq_{ZH} \arrow[d, dash] \arrow[dr, dash] & \\
 & \trianglelefteq_{ZE} \arrow[d, dash] \arrow[dr, dash] & \trianglelefteq_{H} \arrow[d, dash] \\
 & \trianglelefteq_{ZP} \arrow[d, dash] \arrow[dr, dash] & \trianglelefteq_{E} \arrow[d, dash] \\
 & \trianglelefteq_{MB} \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \trianglelefteq_{P} \arrow[d, dash] \\
\trianglelefteq_{ZB} \arrow[d, dash] \arrow[drr, dash] & \trianglelefteq_{M} \arrow[dl, dash] \arrow[d, dash] & \trianglelefteq_{SB} \arrow[dl, dash] \arrow[d, dash] \\
\trianglelefteq_{Z} & \trianglelefteq_{S} & \trianglelefteq_{B}
\end{tikzcd}$$

This hierarchy says only *whether* one of the listed WQOs is more discriminative than another, but not *how much* more discriminative it is. There is no definitive measure for this, but we have made an approximate measure by generating 400 random, different trees using a signature with one nullary constructor `a`, one unary constructor `b`, one binary constructor `c` and one ternary constructor `d`, using the following probabilities for the different constructors:

`a`	50%
`b`	20%
`c`	15%
`d`	15%

The table below shows for each of the discussed WQOs and all relevant combinations of these the number of pairs (t_1, t_2) (out of 160 000) where $t_1 \trianglelefteq t_2$.

$\trianglelefteq_{YZH}$	709	$\trianglelefteq_{ZP}$	19355
$\trianglelefteq_{YH}$	899	$\trianglelefteq_{YM} = \trianglelefteq_{YZS}$	22868
$\trianglelefteq_{YZE}$	3659	$\trianglelefteq_{YS}$	24579
$\trianglelefteq_{YE}$	3875	$\trianglelefteq_{E}$	29252
$\trianglelefteq_{ZH}$	5483	$\trianglelefteq_{MB} = \trianglelefteq_{ZSB}$	37551
$\trianglelefteq_{YZP}$	6641	$\trianglelefteq_{ZB}$	38127
$\trianglelefteq_{YP}$	6999	$\trianglelefteq_{P}$	39168
$\trianglelefteq_{ZE}$	12671	$\trianglelefteq_{YZ}$	44384
$\trianglelefteq_{YMB} = \trianglelefteq_{YZSB}$	17632	$\trianglelefteq_{Y}$	47642
$\trianglelefteq_{H}$	18587	$\trianglelefteq_{M} = \trianglelefteq_{ZS}$	47915
$\trianglelefteq_{YSB}$	18662	$\trianglelefteq_{SB}$	61480
$\trianglelefteq_{YZB}$	19208	$\trianglelefteq_{B}$	62056
$\trianglelefteq_{YB}$	19238	$\trianglelefteq_{S}$	78582
		$\trianglelefteq_{Z}$	93870

Though this is a very rough comparison, as the random trees are hardly typical of what you see in program transformation and analysis, it shows a considerable difference in discriminative power between the different WQOs and also that combining two incomparable WQOs can yield a WQO with significantly higher discriminative power.

5 Comparing well-quasi orders by computational complexity

The typical scenario is that a new element in a sequence is compared to all previous elements in the sequence. This means that the total number of comparisons required for an n-long sequence is $n(n-1)/2$ or O(n^2).

So a way to reduce the overall complexity is to, if possible, split a comparison $s \trianglelefteq t$ into two steps: First computing values $f(s)$, $f(t)$ and then comparing these using a computationally simpler ordering (essentially using Theorem 2). $f(s_i)$ is computed only once per element s_i in the list, so even a small saving in comparing the results will give an overall saving, even if precomputing $f(s_i)$ is relatively expensive. We will primarily focus on the accumulated cost of building a sequence and comparing each new element to all previous elements, using any relevant precomputation on each element. We also exploit the fact that we stop building the sequence as soon as we add an element t such that there is a $s_i \trianglelefteq t$ in the sequence already.

As a simple example, consider $\trianglelefteq_S$, which is defined by $s \trianglelefteq_S t \Leftrightarrow s = t \vee |s| < |t|$. Clearly, the comparison of the sizes (once these are computed) is very fast, but time for computation of the sizes is proportional to the sizes themselves. If all n trees have sizes close to S, an implementation that computes the size at every comparison would need O($n^2 \cdot S$) time, but if the sizes are computed only once per tree, the total cost of size comparisons is only O($n \cdot S + n^2$), assuming unit cost of integer comparison. On top of the size comparison, we must compare a new element for equality to all previous elements. Though comparing for equality is in the worst case proportional to the size of the trees, it can in practice be made (near) constant time by using hashing, requiring an O(S) precomputation per tree for computing the hash. So, assuming effective hashing, the identity comparisons do not add to the overall asymptotic complexity. Additionally, since we stop adding elements to a sequence once we find an element that compares to a previous element, the sizes of elements in a sequence are non-increasing. This means that we only need to compare the size of a new element t with that of the last previously added element s_i. And comparing t for identity with each previous element can be avoided by using the hash code to look up in a table that indicates whether a tree with the same hash code has been seen before. This reduces the cost (again assuming effective hashing) to $O(S)$ for each new added element, so the total cost for an n-long sequence is $O(n \cdot S)$.

For $\trianglelefteq_Z$, we can for each tree precompute its set of constructors. The sets can be hashed or (if Σ is small) represented by a short bit vector, so we do an $O(S)$ precomputation per tree to compute a hash value or a bit vector. Much like above, we can use the hash value or bit vector as a key into a table that indicates whether we have seen the set before, so again the total cost $O(n \cdot S)$. The same analysis applies to $\trianglelefteq_Y$.

$\trianglelefteq_B$ also maps trees to multisets, which can be precomputed. A multiset can be represented as a vector of the number of occurrences of each constructor, and the comparison of two such vectors is proportional to their size, which is the size $|\Sigma|$ of the alphabet of constructors Σ. Precomputation is, again, linear in the size of the trees, so the total cost is $O(n \cdot S + |\Sigma| \cdot n^2)$.

$\trianglelefteq_P$ maps trees to strings and then compares these using the subsequence order. Mapping a tree of size S to a string is $O(S)$ and produces a string of size S. Deciding the subsequence relation for two strings v and w is $O(|v| + |w|)$. Subsequence tests are done a total of $O(n^2)$ times, so the total cost is $O(n^2 \cdot S)$, which is significantly more than for the previous WQOs. $\trianglelefteq_E$ has the same asymptotic cost as $\trianglelefteq_P$, but with a higher constant factor (about twice as high), as the generated strings are roughly twice as long.

But this is still far less than the cost of $\trianglelefteq_H$. At the time of writing, the fastest known method [1] for deciding $t_1 \trianglelefteq_H t_2$ is $O(|t_1| \cdot |t_2| / \log(|t_2|) + |t_2| \cdot \log(|t_2|))$, which makes the total cost for a sequence of n size-S trees $O(n^2 \cdot S^2 / \log(S))$, since the $O(S \cdot \log(S))$ component of the cost is asymptotically dominated by $O(n^2 \cdot S^2 / \log(S))$. There is no obvious precomputation that can reduce this time.

The table below summarises the costs both for pairwise comparison and for comparing each element in a sequence of n elements of size S to all previous elements in the sequence.

WQO	$s \trianglelefteq t$	sequence of n trees of size S
$\trianglelefteq_Z$	$O(\|s\| + \|t\|)$	$O(n \cdot S)$
$\trianglelefteq_Y$	$O(\|s\| + \|t\|)$	$O(n \cdot S)$
$\trianglelefteq_S$	$O(\|s\| + \|t\|)$	$O(n \cdot S)$
$\trianglelefteq_B$	$O(\|s\| + \|t\|)$	$O(n \cdot S + \|\Sigma\| \cdot n^2)$
$\trianglelefteq_P$	$O(\|s\| + \|t\|)$	$O(n^2 \cdot S)$
$\trianglelefteq_E$	$O(\|s\| + \|t\|)$	$O(n^2 \cdot S)$
$\trianglelefteq_H$	$O(\|s\| \cdot \|t\| / \log(\|t\|) + \|t\| \cdot \log(\|t\|))$	$O(n^2 \cdot S^2 / \log(S))$

While many of the WQOs have the same complexity $O(|s| + |t|)$ for pairwise comparison of trees, the accumulated complexity of constructing and checking a sequence differs considerably for several of these.

5.1 Cost of combined WQOs

With a few exceptions, combined WQOs have asymptotic costs that are the sums of the asymptotic costs of their components, as you can test each component WQO independently of the others.

The exceptions are combinations of $\trianglelefteq_S$ and other WQOs as we, to get the cost of testing a sequence with $\trianglelefteq_S$ down to $O(n \cdot S)$, exploited that the sizes of trees in a sequence using $\trianglelefteq_S$ as indicator would stop when the size of the new tree is larger than the immediately previous, so the sizes of trees in the sequence would be non-increasing. But when combining $\trianglelefteq_S$ with another WQO $\trianglelefteq'$, the sizes of trees in the sequence are not necessarily non-increasing, as the size of a new element t can be larger than the size of a previous element s_i as long as $s_i \ntrianglelefteq' t$. So we will generally have to compare t with all previous elements s_i. We can still precompute the sizes of the trees and compute hash codes for quick equality testing, but unless $\trianglelefteq'$ allows partitioning the sequence into subsequences of non-increasing size, we will need to compare the size of t to the sizes of all previous elements in the sequence. So without knowing more about $\trianglelefteq'$, the part of the total cost needed for testing a sequence with $\trianglelefteq_S$ is $\mathrm{O}(n \cdot S + n^2)$. So, generally, the cost of combining $\trianglelefteq_S$ with a WQO $\trianglelefteq'$ is the sum of $\mathrm{O}(n \cdot S + n^2)$ and the cost for $\trianglelefteq'$. This can, however be avoided for some instances of $\trianglelefteq'$, such as $\trianglelefteq_Y$ and $\trianglelefteq_Z$, as we shall see below.

If f is a function from T to a finite set F (such as 2^Σ), then testing a sequence for a combination of a WQO of the form $(T, =^f)$ and another WQO $\trianglelefteq'$ can be done by partitioning the sequence s_i (as it is built) by different values of $f(s_i)$. Using hashing on the values of $f(s_i)$, finding the right partition for a new element t can be done in $\mathrm{O}(|t|)$ time (to perform the hashing) and a constant-time lookup. The new element is then compared to the other elements in the partition using $\trianglelefteq'$ only, so the total cost of adding an element t to the sequence is $\mathrm{O}(|t|)$ plus the cost using $\trianglelefteq'$ of adding t to a sequence. For example, combining $\trianglelefteq_Z$ and $\trianglelefteq_S$ can be done by partitioning the sequence by different sets of constructors. Each partition will be a subsequence of the original with non-increasing sizes, so the fast method for testing a sequence for $\trianglelefteq_S$ can be used for each subsequence, which brings the total cost for building and testing a sequence with $\trianglelefteq_{ZS} = \trianglelefteq_M$ down to $\mathrm{O}(n \cdot S)$. The same analysis applies to combining $\trianglelefteq_Y$ with $\trianglelefteq_S$.

All the other WQOs presented in Section 3 require comparison of (some precomputed value of) the new tree t to all previous trees s_i, so the cost of combining two of these is a simple addition of the costs of the components. We can, however, first test a new element t against previous elements s_i using the cheaper of the two WQOs $\trianglelefteq_1$, and only when that finds an $s_i \trianglelefteq_1 t$ check if $s_i \trianglelefteq_2 t$, where $\trianglelefteq_2$ is the more costly of the two.

Similarly, less costly measures can be used to approximate more costly measures: For example, when comparing with $\trianglelefteq_{YH}$, we can first test (fairly cheaply) if $s_i \trianglelefteq_{YS} t$, and if this is true try $s_i \trianglelefteq_E t$, and only if this is also true perform the expensive test $s_i \trianglelefteq_H t$. This way, we get the full discriminative power of $\trianglelefteq_{YH}$ but we will in many (probably most) cases be able to avoid the full cost.

6 Conclusion

We have compared a number of well-quasi orders (WQOs) on trees for discriminative power and computational cost. The selection of WQOs include very simple WQOs ($\trianglelefteq_Z$, $\trianglelefteq_S$), several WQOs from the literature of supercompilation ($\trianglelefteq_M$, $\trianglelefteq_H$) as well as some proposed by the author ($\trianglelefteq_Y$, $\trianglelefteq_B$, $\trianglelefteq_P$, $\trianglelefteq_E$), that to the author's knowledge have not been used for termination of program analysis and transformation.

We have also looked at combining two or more WQOs of incomparable discriminative power to get a more discriminative WQO. This adds 19 more distinct WQOs ($\trianglelefteq_{YZ}$, $\trianglelefteq_{ZH}$, $\trianglelefteq_{YH}$, $\trianglelefteq_{YZH}$, $\trianglelefteq_{ZE}$, $\trianglelefteq_{YE}$, $\trianglelefteq_{YZE}$, $\trianglelefteq_{ZP}$, $\trianglelefteq_{YP}$, $\trianglelefteq_{YZP}$, $\trianglelefteq_{ZB}$, $\trianglelefteq_{YB}$, $\trianglelefteq_{YZB}$, $\trianglelefteq_{SB}$, $\trianglelefteq_{MB}$, $\trianglelefteq_{YSB}$, $\trianglelefteq_{YMB}$, $\trianglelefteq_{YS}$, $\trianglelefteq_{YM}$).

We observe that, in a typical scenario, we do not just compare a pair of trees, but compare each new element of a sequence to all previous elements. In such a scenario, time can be saved by precomputing some values for each tree as it is added to the sequence, and then using these values for the WQO comparison. This can for many WQOs dramatically reduce the overall cost.

While higher discriminative power gives more precision, there is a higher cost not only from computing the more complex WQO, but also because sequences of trees get longer before they are generalised/widened and folded back, so analyses and transformations can take longer (and transformed programs can be bigger). The choice of which WQO to use should consider all of the above.

When using a combined WQO, it can be cheaper to first compare with the cheapest component WQO and only if that succeeds, compare with the more expensive component WQO. When combining with $\trianglelefteq_Z$ and $\trianglelefteq_Y$, these can be used to partition the sequence. This idea can also be applied with comparable WQOs: If a cheaply computable WQO approximates a more expensive WQO, we can compute the cheap WQO and only if that succeeds compute the expensive WQO. In neither case is the worst-case cost lowered (and it can be somewhat increased), but the average cost can be significantly lower.

From the (admittedly naive) statistic tests, it seems combining a WQO $\trianglelefteq'$ with $\trianglelefteq_Z$ or (in particular) $\trianglelefteq_Y$ gives significantly higher discriminative power than $\trianglelefteq'$ alone, but combining with both $\trianglelefteq_Z$ and $\trianglelefteq_Y$ gives only little extra power over combining with $\trianglelefteq_Y$ alone.

7 Acknowledgements

The author would like to thank the organisers of SAIRP for inviting him to submit a paper. He also thanks the anonymous reviewers, whose suggestions greatly improved the quality of the paper.

The author would also like to thank Dave Schmidt for being an inspirational scientist and a generally fun guy to be around. The halo over his head in the photo on his personal home page [7] is well deserved.

References

[1] Philip Bille & Inge Li Gørtz (2011): *The tree inclusion problem: In linear space and faster*. *ACM Trans. Algorithms* 7(3), pp. 38:1–38:47, doi:10.1145/1978782.1978793.

[2] Nachum Dershowitz (1979): *Orderings for term-rewriting systems*. In: *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, Washington, DC, USA, pp. 123–131, doi:10.1109/SFCS.1979.32. Available at http://dl.acm.org/citation.cfm?id=1398508.1382612.

[3] Joseph B. Kruskal (1960): *Well-quasi-ordering, the Tree Theorem, and Vazsonyi's conjecture*. *Transactions of the American Mathematical Society* 95, pp. 210 – 225, doi:10.2307/1993287.

[4] Michael Leuschel (1998): *On the Power of Homeomorphic Embedding for Online Termination*. In: *Static Analysis. Proceedings of SAS'98, LNCS 1503*, Springer-Verlag, pp. 230–245, doi:10.1007/3-540-49727-7_14.

[5] Michael Leuschel (2002): *Homeomorphic embedding for online termination of symbolic methods*. In: *The essence of computation, LNCS 2566*, Springer-Verlag, pp. 379–403, doi:10.1007/3-540-36377-7_17.

[6] Neil Mitchell (2010): *Rethinking supercompilation*. In: *Proceedings of ICFP'2010, the 15th ACM SIGPLAN international conference on Functional programming*, SIGPLAN Notices 45(9), ACM, New York, NY, USA, pp. 309–320, doi:10.1145/1863543.1863588.

[7] Dave Schmidt (Sampled May 2013): *Homepage*. http://people.cis.ksu.edu/∼schmidt/. Available at http://people.cis.ksu.edu/~schmidt/.

[8] Morten H. Sørensen & Robert Glück (1995): *An Algorithm of Generalization in Positive Supercompilation*. In: *Proceedings of ILPS'95, the International Logic Programming Symposium*, MIT Press, pp. 465–479, doi:10.1.1.49.1869.

[9] Ian Wehrman (2006): *Higman's Theorem and the Multiset Order*. http://www.cs.utexas.edu/∼iwehrman/pub/ms-wqo.pdf. Available at http://www.cs.utexas.edu/~iwehrman/pub/ms-wqo.pdf.