garbage collection: Framework and implementations. In *ACM Symposium on Principles of Programming Langauges*, pages 261–269, January 1990.

[GT87] H. W. Glaser and P. Thopmson. Lazy garbage collection. *Software Practice and Experience*, 17(1):1–4, January 1987.

[Han77] David R. Hanson. Storage management for an implementation of snobol4. *Software: practice and experience*, 7(2):179–192, March 1977.

[LH83] Henry Lieberman and Carl Hewitt. A real-time garbage collector based on the lifetimes of objects. *Communications of the ACM*, 26(6):419–429, June 1983.

[Moo84] David A. Moon. Garbage collection in a large lisp system. In *ACM Symposium on Lisp and Functional Languages*, pages 235–246, August 1984.

[Rov85] Paul Rovner. On adding garbage collection and runtime types to a strongly-typed, statically-checked, concurrent language. Technical Report CSL-84-7, Xerox PARC, 1985.

[Ste75] Guy L. Steele Jr. Multiprocessing compactifying garbage collecting. *Communications of the ACM*, 18(9):495–508, September 1975.

[Ung84] David Ungar. Generation scavenging: A non-disruptive high performance storage reclamation algorithm. In *ACM SIGSOFT/SIGPLAN Practical Programming Conference*, pages 157–167, April 1984.

[WDH89] Mark Weiser, Alan Demers, and Carl Hauser. The portable common runtime approach to interoperability. In *Proceedings 13th ACM Symposium on Operating Systems Principles*, December 1989.

[WW87] Paul Watson and Ian Watson. *An Efficient Garbage Collection Scheme for Parallel Computer Architectures*, pages 432–443. Number 259 in Lecture Notes in Computer Science. Springer Verlag, 1987.

of scheduling, paging, memory lay-out, virtual-memory mapping, and a host of other operating systems parameters, as well as the benchmarks chosen to test the system, so the estimates from the model will at best be approximate. These predictions provide a way to factor out the effects of the benchmarks and the particular implimentation of the collector.

4 Acknowledgements

This paper is a reworking of a previous paper presented at the 1990 POPL conference [DWH+90] and I owe a debt to the other authors of that paper: Alan Demers, Mark Weiser, Hans Boehm, Danny Bobrow, and Scott Shenkar.

References

[AEL88] Andrew W. Appel, John R. Ellis, and Kai Li. Real-time concurrent collection on stock multiprocessors. *Proceedings of the SIGPLAN '88 Conference on Programming Language Design and Implimentation, SIGPLAN Notices*, 23(7):11–20, July 1988.

[Bak78] Henry G. Baker, Jr. List-processing in real time on a serial computer. *Communications of the ACM*, 21(4):280–294, April 1978.

[Bar88] Joel F. Bartlett. Compacting garbage collection with ambiguous roots. Technical Report 88/2, Digital Equipment Corp., Western Research Laboratory, February 88.

[Bev87] D. I. Bevan. *Distributed Garbage Collection Using Reference Counting*, pages 176–185. Number 259 in Lecture Notes in Computer Science. Springer Verlag, 1987.

[BW88] Hans-Juergen Boehm and Mark Weiser. Garbage collection in an uncooperative environment. *Software: practice and Experience*, pages 495–508, 1988. to appear.

[CDG+88] Luca Cardelli, James Donahue, Lucille Glassman, Mick Jordan, Bill Kalsow, and Nelson Greg. Modula-3 report. Technical Report ORC-1, Olivetti Research Center, 1988.

[DB76] L. P. Deutsch and D. G. Bobrow. An efficient incremental automatic garbage collector. *Communications of the ACM*, 19(9):522–526, September 1976.

[DWH+90] Alan Demers, Mark Weiser, Barry Hayes, Hans Boehm, Daniel Bobrow, and Scott Shenker. Combining generational and conservative

it is isomorphic to an unordered set with one element for each area. For time-based generations, it is isomorphic to a total order with one element for each generation. The filters and ideals on these are very simple, and so is the check for rescuers.

Theorem 15 *For a threatened set T, if $a \notin T$ is a rescuer then $\mathrm{A}_{\prec}(b) \in \mathrm{F}_{\prec}(a)$ for some $b \in T$.*

> If a is a rescuer, then there is an element $b \in T$ such that $\langle a, b\rangle \in P$. The filter $\mathcal{F}_{\langle\{c|\langle a,c\rangle \in P\},\prec\rangle}$ must be a subset of $F(a)$ for any rescuing function F, and so the homomorphic image of the filter must be a subset of the homomorphic image of $F(a)$, which is $F_{\prec}(a)$. But since $\langle a, b\rangle \in P$, b must be in the filter, and the image of b under the homomorphism, $A_{\prec}(b)$, must be in the image of the filter,$F_{\prec}(a)$.

The utility of theorem 15 is in its contrapositive: for a threatened set T and $a \notin T$, if $A_{\prec}(b) \notin F_{\prec}(a)$ for all $b \in T$ then a is not a rescuer. Once a threatened set has been established, the functions $F_{\prec}$ and $A_{\prec}$ can be used to find a set of objects which are neither threatened nor rescuers, and these objects can be ignored for the collection.

The goal of designing a collector can be seen as choosing a pre-order $\prec$ and a rescuer F. The choice for the pre-order must strive for a balance between making $A_{\prec}$ efficient to encode and allowing exactly the collections desired. The choice of F then determines $F_{\prec}$, which also needs an efficient encoding, but should allow a large set of object to be ignored at each collection.

3 Conclusions

The model presented by pointer-augmented collections is not meant to be prescriptive, but rather descriptive and weakly predictive. Any collector can be described in the primatives presented in this paper, and the algebra of collectors allows them to be combined. The common descriptive language of a pre-order on the objects and a threatening function allows implementers to explore the space of combinations of features, and focuses on the cheif design decisions involved in a collector.

The costs of collection fall roughly into the categories defined by the primitives. There is a cost to maintaining the pre-order on the objects, there is a cost of finding the rescuers for the threatened set, and there is a cost of tracing the objects in the threatened set. Estimates of the cost of collection in these terms may give approximations to the cost of collections in a real system.

Unfortunately, in a real operating system storage management will have complex interactions with the other hardware and software. The overall performance is almost impossible to predict without considering the combined effects

Definition 19 *A function F is defined to be a* rescuer function *if for all $a \in O$, $F(a) \supseteq \mathcal{F}_{\langle \{b | \langle a,b \rangle \in P\}, \prec \rangle}$.*

The rescuer function allows Theorem 12 to hold in the direction desired, but gives a collection system freedom to liberalize the search for rescuers.

Theorem 13 *For any storage state $\langle AS, P \rangle$, and any rescuer function F, if $\langle a, b \rangle \in P$ then $b \in F(a)$.*

This is an immediate consequence of Definition 19.

Since like ideals filters can often be expressed compactly there is a reason to look at the space of collectors where the rescuer function maps not to general subsets of O but to filters under the pre-order. Theorem 13 will still apply, and the filters may be liberal representations of the true rescuers.

2.7 A/F Collectors

When the pre-orders are fairly simple and the threatening functions are all filters, there is another way of talking about the lattice of pointer augmentations. Any pre-order on the objects breaks into two components: a partition $\Pi_\prec$ of O into cliques and a partial order on the cliques in $\Pi_\prec$.

Definition 20 *Let $\mathrm{A}_\prec$ be the function mapping an object in O to the clique containing that object under pre-order $\prec$. For any two partition elements $A, B \in \Pi_\prec$, the partial order $A \leq B$ means that for some $a \in A$ and some $b \in B$, $a \prec b$.*

The reason that the function $A_\prec$ is useful is that it in mapping the objects to the cliques the ordering is preserved, so it is a homomorphism. Homomorphisms not only preserve order but they map ideals to ideals and filters to filters.

Theorem 14 $\mathrm{A}_\prec$ *is a homomorphism.*

For any two objects a and b in O, $a \in A_\prec(a)$ and $b \in A_\prec(b)$, so if $a \prec b$ then $A_\prec(a) \leq A_\prec(b)$. This all follows from Definition 20.

A threatened set, being an ideal of O, is mapped on to an ideal of $\Pi_\prec$. Likewise if the rescuing function maps from the objects to the filters of O, the homomorphism can be combined with it to make a function from the objects to the filters of $\Pi_\prec$.

Definition 21 *If F is a rescuer function, let $\mathrm{F}_\prec(a)$ be $\{\mathrm{A}_\prec(b) | b \in F(a)\}$.*

If the rescuer function F maps only to the filters of O then $F_\prec$ will map to the filters of $\Pi_\prec$.

The reason for examining the $A_\prec$ and $F_\prec$ functions is that for many simple applications the partial order on the cliques has a very simple form. For areas,

another statistical observation to cut down on this search: there are relatively few old objects that point to young objects. This set is usually small enough so that it can be stored explicitly, kept accurate efficiently, and examined quickly.

Definition 16 *A* rescuing object *in a storage state* $\langle AS, P\rangle$ *under threatened set* T *is an object* $a \notin T$ *such that* $\langle a, t\rangle \in P$ *for some* $t \in T$.

If there is no structure to the threatened set there is nothing that can be done to make the search for rescuers easier; each object must be examined in turn to see if it points to a threatened object. But in the special case of generational collection schemes the threatened set is always an ideal of a pre-order of the allocated objects. This restriction on the threatened set produces structure on the rescuers and this structure is related to the dual of the ideals, the filters on the pre-order.

Definition 17 *A* filter F *of a pre-order* $\prec$ *on a set* S *is a subset of* F *such that* $a \succ b \wedge b \in S \Rightarrow a \in F$.

Definition 18 *The upward closure of a set* $B \subseteq O$ *under pre-order* $\prec$ *is* $\mathcal{F}_{\langle B,\prec\rangle} = \{o \in O | \exists b \in B \text{ such that } o \succ b\}$

Since the threatened objects always form an ideal under the pre-order, if $a \prec b$ and b is in the threatened set then a will be in it as well. An object r that points to both a and b will be a rescuer if and only if a is threatened, and so the pointer to b can be ignored.

Theorem 12 *In a generational scheme of* $\langle AS, P\rangle$ *with pre-order* $\prec$ *and threatened ideal* I, *an object* $a \notin I$ *is a rescuer if and only if* $\mathcal{F}_{\langle P_a,\prec\rangle} \cap I$ *is not empty, where* $P_a = \{b | \langle a, b\rangle \in P\}$

> If some object $a \notin I$ is a rescuer, then there is some $\langle a, b\rangle \in P$ such that $b \in I$. But then $b \in P_a$, and so $b \in \mathcal{F}_{\langle P_a,\prec\rangle}$.
>
> If $\mathcal{F}_{\langle P_a,\prec\rangle} \cap I$ is not empty then there is some minimal b in the intersection, so that there is no $b' \succ b$ in the intersection. But $\mathcal{F}_{\langle P_a,\prec\rangle}$ is a filter with lower boundry $\langle P_a, \prec\rangle$ and b is minimal, so $b \in P_a$. Thus $\langle a, b\rangle \in P$, and a is a rescuer.

To decide if an immune object is a rescuer the only pointers a collector needs to keep track of are those to minimal objects under the pre-order. If the pre-order is well chosen this set can be very small and efficiently expressed.

A collector is only obliged not to miss any rescuers, so it can be liberal in its search and examine objects that are not in fact rescuers. It may end up examining more objects in a search for pointers to the threatened set but will never miss the pointers it needs and the over-estimation is safe, even if not efficient. Such over-estimation may be useful if computing the exact set of rescuers would take more time or space than is wasted by examining some extra objects.

Above it in the lattice is a pre-order where the set C has one element for every subset of $O - C$ and the pre-order records the exact set of objects that point to any given object. This pre-order is akin to a collection system that keeps back-pointers for every pointer.

An area-based collector can even be combined with a reference counting collector to create a collector that keeps several counts for each object, but each count reflects the number of references to the object from each of several partitions of the allocated objects. Collectors operating in this way have been proposed [DB76], distinguishing between pointers from objects on the program stack and pointers from other objects. The counts of the pointers from the stack are only computed occasionally and if these objects are very active the cost of keeping the reference counts goes down.

2.5.5 Creole Collectors

Having the full lattice of possible collectors should open the eyes of designers to more interesting and useful combinations. For instance, conservative and time-based generational collectors have been combined with promising results [DWH+90].

The success of time-based generational collectors is based on their ability to collect well on the young objects, but the accumulation of relatively old objects must be faced. Most of these collectors occasionally collect over all the allocated objects and try to schedule this for a time when real-time demands on the system are low. Perhaps another scheme is more applicable to old objects.

For example, if objects are being created in many processes in an operating system and the processes themselves have very little intercommunication and object sharing, then partitioning the objects by process and collecting the old objects in a process when that process terminates would work well.

Threating all objects is a fairly expensive way to collect the old objects, and perhaps a creole using time-based generations for the young objects and reference counting for the old objects would find the garbage in the old objects efficiently. This is very similar to keeping distinct counts for pointers from the stack objects [DB76] and is another way to control the growth of the old objects.

2.6 Rescuers

Pre-orders on the allocated set and collections over ideals of that pre-order help restrict the set of objects to be examined at collection time by placing a limit on the search for reachable objects. Unfortunately, that is only part of the story. The collector must still find all the pointers from non-threatened objects to threatened objects, and examination of all immune objects will usually be too time consuming.

Collectors need a way to cut down on the search for rescuing objects without searching the entire immune set. Time-based collectors take advantage of

cliques. A collector using areas can threaten any area or collection of areas, recovering garbage in those chosen areas and ignoring garbage in the others.

2.5.3 Ordered Generations

When the cliques that come about from areas are themselves ordered the result is that no longer can an arbitrary subset of the areas be threatened, but a threat to one area may require other areas to be threatened as well.

The simplest example of this is time-based generations. The cliques are totally ordered and a threat to any generation implies that all the younger generations are threatened as well.

But the order on the cliques need not be total, as in the example combining time-based generations with areas. In fact, the order need not be time-based at all. Consider partitioning the objects by data-type. There may be "leaf" data-types that contain no pointers, and "root" data-type that point to leaf objects. If the root objects are static, the only way a leaf object can become unreachable is if a root has become unreachable. A collection which tried to collect only the leaf objects would find no garbage. If an ordering is imposed on the partitions such that a threat to the leaves implies a threat to the root as well, these useless collections would be prohibited.

Further specialization for various kinds of roots and leaves could produce a collector with a "bushy" partial order based on data-types, where leaves of various types could be collected at the same time as the roots that use them.

2.5.4 Reference Counting

If the threatened set is $\{o|\langle o,o\rangle \notin P^*\}$ then the garbage is exactly those objects that would be recovered by a reference counting system. Although it produces the same result, this is hardly in the same spirit, since it requires calculation of P^*.

A more appropriate way to describe reference counting garbage collectors is to use the pre-order to model the counters. If there is a countable infinite set $C \subseteq O - AS$ where each element models a non-negative integer, then defining $a \prec b$ if and only if $a \in C$, $b \notin C$, and a models the size of $\{c \in AS|\langle c,b\rangle \in P\}$ will produce a pre-order. In other words, the pre-order relates objects and their reference counts. The collector could threaten any ideal but the ideal of interest is the downward-closure of the object associated with zero. All the allocated objects in this ideal have a reference count of zero and can be freed immediately. The cost of such a scheme is in adjusting the pre-order to reflect the size of the back-pointer set, and is analogous to the costs of reference-counting collection.

Give the poor suckers a diagram! Heaven knows, I keep drawing it myself.

Of course, this is only one of a family of reference-counting pre-orders. Below it in the lattice are, among other things, pre-orders that rather than having an infinite set C use a finite set and use one special element for those objects that have overflowed their reference count.

one partition is related to any element in another. There are eight ideals in this pre-order, since each of the three partitions can be in the ideal or not.

The time-based scheme has two generations with three objects in each. The relation in this case is "not-older-than," only one of edges relating a younger object to an older is shown in the diagram, but since it is transitive, this implies that any young object is "not-older-than" any old object. Of course, two objects in the same generation are "not-older-than" each other. There are only three ideals in this pre-order: the empty set, the set of just the young objects, and the set of all the objects.

Using the $\vee$ operation, all the choices for ideals that were available under either pre-order are available under the combined pre-order. A collector which is using a combination of several strategies can make all the choices available under any one of them and more.

Theorem 11 $\mathcal{R}_{\langle\langle AS,P\rangle,M_\prec \vee N_\prec\rangle} \supseteq \mathcal{R}_{\langle\langle AS,P\rangle,M_\prec\rangle}$.

Since $M_\prec \vee N_\prec \geq M_\prec$, Theorem 10 applies, and the result follows immediately.

The example in Figure 2 is a case where more options are available after the combination than before. All of the ideals previously present are still ideals, but new ideals are also created which were not present present in either the area or time-based scheme. For example, any subset of the young objects can be collected even if it was not an ideal under either scheme alone.

2.5 Examples

While the inspiration for describing garbage collections in terms of pre-orders and threatened ideals is to get a handle on "stop and copy", "trace and sweep", and other tracing garbage collection algorithms operating in standard area- and time-based generations, the framework is powerful enough to describe a large class of collection algorithms including reference counting, and orderings that are neither area- nor time-based.

2.5.1 Collect Everything

A garbage collector that always tries to collect every object in O comes about if the pre-order relates every object to every other object. There are only two ideals of this pre-order: the empty set and the set O. A collector can choose to threaten no object and recover nothing, or threaten all the objects and recover all the unreachable storage.

2.5.2 Areas

If the pre-order is symmetric it defines a partition of O, dividing it into cliques of objects. The ideals of this pre-order are all the possible unions of all the

augmentation formed when the threatened set is the downward closure of L under $N_\prec$, $\mathcal{I}_{\langle L, N_\prec\rangle} \triangleright \langle AS, P\rangle$. But since $M_\prec \subseteq N_\prec$, the downward closure of L under $N_\prec$ is also downward closed under $M_\prec$, and so $\langle AS, Q\rangle \in \mathcal{R}_{\langle\langle AS,P\rangle, M_\prec\rangle}$.

For any pair of objects, if $a \prec b$ by ordering $M_\prec$ then any ideal under $M_\prec$ that contains b must also contain a. But if $\mathcal{R}_{\langle\langle AS,P\rangle, N_\prec\rangle} \subseteq \mathcal{R}_{\langle\langle AS,P\rangle, M_\prec\rangle}$ then the ideals of $N_\prec$ must also have this property. If every ideal of $N_\prec$ that contains b must also contain a then $a \prec b$ by ordering $N_\prec$.

The correspondence encourages a partial ordering on the pre-orders and a derived ordering on the generational schemes of the objects.

Definition 14 *Given two pre-orders $M_\prec$ and $N_\prec$ on a set O, $M_\prec \leq N_\prec$ if $M_\prec \subseteq N_\prec$.*

Since the ordering is a subset inclusion ordering intersection gives a definition for a least upper bound. The intersection of two pre-orders will always be reflexive and transitive and so will always be a pre-order. The new pre-order can be used as a new generational scheme.

Definition 15 *For any storage state $\langle AS, P\rangle$ and any two pre-orders relations $M_\prec$ and $N_\prec$ on O the pre-order $M_\prec \vee N_\prec$ is $M_\prec \cap N_\prec$.*

Thus given two generational schemes, they can be intersected as pre-order relations to give a new generational scheme with at least all the ideals and threatened sets allowed by either. In general, new threatened sets will also be allowed in this combined scheme.

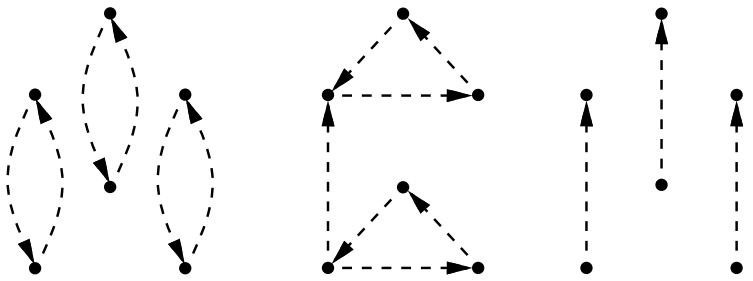

Figure 2: An Area Scheme, a Time-based Scheme, and Their Least Upper Bound

For example, consider the least upper bound of an area scheme and a time-based generational scheme, as in Figure 2. The area scheme is a three-way partition with each of the partitions containing two elements, each related to the other under the $\prec$ pre-order indicated by the dashed arrows. No element in

Definition 13 *The* restricted augmentations $\mathcal{R}_{\langle\langle AS,P\rangle,\prec\rangle}$ *of a storage state* $\langle AS,P\rangle$ *to a pre-order* $\prec$ *on* O *is the set of pointer augmentations* $\{T \triangleright \langle AS,P\rangle | T = \mathcal{I}_{\langle L,\prec\rangle}$ *for some* $L \subseteq O\}$.

While the inspiration for much of this theory comes from time-based generational collection, other generational collection schemes have been proposed and implemented. The MIT LISP machine and the PL/I language partitioned the allocated storage into *areas*, where each area could be collected independently.

This scheme is slightly less restrictive than time-based generational collection since no area's collection requires the collection of another area. Some collectors partition objects by age into unrelated areas rather than ordered generations to allow collection of a set of middle-aged objects without collecting the young objects at the same time.

The pre-order on the allocated set that corresponds to an area partition makes $a \prec b$ if and only if a and b are in the same area. The restricted augmentations under this pre-order are just those augmentations that threaten some areas and make other areas immune, allowing area by area collection.

By using an ideal of some pre-order to select a threatened set the restricted augmentations of the storage state form a sub-lattice of the lattice of all pointer augmentations with the same ordering and least upper bound. See defn 20

Theorem 9 *For any storage state* $\langle AS,P\rangle$ *and pre-order* $\prec$ *on* O, $\mathcal{R}_{\langle\langle AS,P\rangle,\prec\rangle}$ *is closed under* $\vee$.

> Let $\mathcal{I}_1 \triangleright \langle AS,P\rangle$ and $\mathcal{I}_2 \triangleright \langle AS,P\rangle$ be any two elements of $\mathcal{R}_{\langle\langle AS,P\rangle,\prec\rangle}$. From Definition 7, their least upper bound is their intersection, and from Definition 8, $\cap$ distributes over $\triangleright$, so the intersection of the augmentations is $(\mathcal{I}_1 \cap \mathcal{I}_2) \triangleright \langle AS,P\rangle$. But since the intersection of two ideals is another ideal, the intersection is also in $\mathcal{R}_{\langle\langle AS,P\rangle,\prec\rangle}$.

For a given storage state and pre-order the restricted augmentations are fixed, but one pre-order may produce a restriction that is more liberal than another pre-order, allowing all the pointer augmentations of the first and permitting even more. The less liberal restriction is a proper subset of the more liberal.

But a pre-order itself is just some subset of $O \times O$ and so one pre-order may be strictly contained in another as a subset. Happily, these two views of order on pre-orders never conflict.

Theorem 10 *For any storage state* $\langle AS,P\rangle$ *and any two pre-orders* $M_\prec$ *and* $N_\prec$ *defined on* O, $\mathcal{R}_{\langle\langle AS,P\rangle,N_\prec\rangle} \subseteq \mathcal{R}_{\langle\langle AS,P\rangle,M_\prec\rangle}$ *if and only if* $M_\prec \subseteq N_\prec$.

> If the pre-orderings are nested so that $M_\prec \subseteq N_\prec$ and some particular augmentation $\langle AS,Q\rangle \in \mathcal{R}_{\langle\langle AS,P\rangle,N_\prec\rangle}$ is chosen, then from Definition 13 there is some $L \subseteq O$ such that $\langle AS,Q\rangle$ is the pointer

threatened set is well chosen it will contain nearly all the garbage and few live objects.

There has been great practical success with collectors that choose their threatened set on the basis of object creation time, collecting only those objects that were created fairly recently and making objects immune if they are relatively old and have already survived several garbage collections. The young objects, which contain a high fraction of garbage, are collected frequently. When an object has survived a number of collections the collector notes that it is less likely to be garbage and promotes it to an older generation. Objects of this age will be collected less often, and perhaps eventually promoted to yet an older generation. Since each collection has a birthday cut-off for the objects it threatens, when a generation is collected all the younger generations will also be collected. There is a pre-order on the objects since threatening old objects requires threatening the younger objects but not the other way around.

Definition 9 *A* pre-order *on a set is a reflexive, transitive relation on that set.*

In general, a collector can use any pre-order on the objects to determine when the threat to one object implies a threat to some other object.

Definition 10 *A* generational collection scheme *is a pre-order on O.*

A time-based generational collector partitions the objects into sets of objects created between two successive garbage collections. Like Thoroughbred horses, all objects created between two garbage collections have the same birthday. The pre-order associated with this scheme is "not older than." For two objects created with no intervening garbage collection, each is no older than the other. If there was an intervening collection, the object created later is not older than the one created earlier.

The collector has a choice in which objects it can threaten. It can choose to threaten all objects, or for efficiency, only those objects younger than some threshold. The notion of collecting all objects younger than a chosen age generalizes to threatening an *ideal* of the pre-order.

Definition 11 *An* ideal *I of a pre-order $\prec$ on a set S is a subset of S such that $a \prec b \wedge b \in I \Rightarrow a \in I$.*

Since an ideal is always downward closed it can often be specified concisely by a subset of maximal elements. The downward closure of any set is an ideal.

Definition 12 *The downward closure of a set $B \subseteq O$ under pre-order $\prec$ is $\mathcal{I}_{\langle B,\prec\rangle} = \{o \in O | \exists b \in B$ such that $o \prec b\}$*

A collector that always threatens an ideal of the pre-order will be interpreting the pre-order as meaning that if $a \prec b$ and b is threatened, then a will be threatened as well. In this way, the pre-order restricts which subsets of the objects can be threatened.

2.4 Generational Collection

Time-based generational garbage collectors take advantage of some observed statistics to collect a large fraction of the garbage but only examine a small number of objects to do so [Han77, LH83, Moo84, Ung84]. Most objects are garbage very soon after they are created and conversely, an object that has survived a fairly long time is very unlikely to be collected in the future.

Time-based generational collectors segregate objects by their creation time and only attempt to collect those objects that are relatively new. The use of a shibboleth to allow a collector to search for garbage only among a subset of the objects can be formalized by the introduction of *threatened sets,* and *immune sets.*

Definition 8 *A* threatened set $T \subseteq O$ *of storage state* $\langle AS, P\rangle$ *induces a garbage collection* $\mathcal{G}_{T \triangleright \langle AS,P\rangle}$ *where* $T \triangleright \langle AS, P\rangle$ *is the pointer augmentation* $\langle AS, P \cup \{\langle r, o\rangle | o \in AS - T\}\rangle$ *caused by adding links from the root to the non-threatened objects. The dual of the threatened set, the* immune set, *is* $AS - T$.

In the augmentation introduced by the threatened set all the objects in the immune set are made reachable and so cannot be garbage. Thus all the garbage that is in the allocated set must be in the threatened set.

Theorem 8 $AS \cap \mathcal{G}_{T \triangleright \langle AS,P\rangle} \subseteq T$.

> If $a \in (AS - T)$ then $\langle r, a\rangle \in T \triangleright \langle AS, P\rangle$ by Definition 8 and so $a \notin \mathcal{G}_{T \triangleright \langle AS,P\rangle}$ by Definition 3. But if $(AS - T) \cap \mathcal{G}_{T \triangleright \langle AS,P\rangle}$ is empty, then $AS \cap \mathcal{G}_{T \triangleright \langle AS,P\rangle} \subseteq T$.

Performing a full tracing garbage collection requires the examination of the full graph of interconnections among live objects, and so a full garbage collection requires at least examination of every live object. By using pointer augmentations characterized by threatened sets a collection must establish reachability only for those objects in the threatened set and not the immune objects. If the

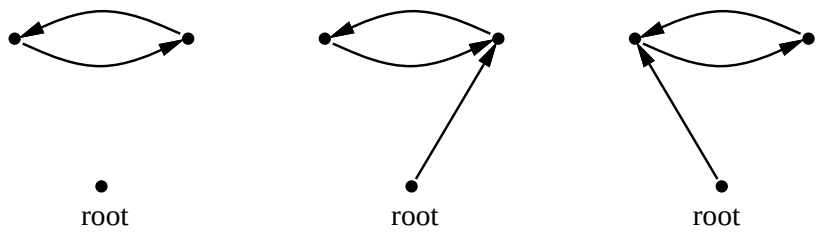

Figure 1: A Storage State and Two Pointer Augmentations

Definition 5 *A* partial order *on a set is a reflexive, transitive, anti-symmetric relation on that set.*

Definition 6 *Define a partial order* $\leq$ *on the pointer augmentations of a storage state such that* $\langle AS, Q\rangle \leq \langle AS, R\rangle$ *if* $Q \supseteq R$.

Adding more pointers leads to a smaller set of garbage, by Theorem 3, and a less precise collection. Since the partial order is defined in terms of subsets of the points-to relation, not only is is there a partial order but the pointer augmentations form a lattice in a natural way.

Definition 7 *The least upper bound of two pointer augmentations* $\langle AS, P\rangle \vee \langle AS, Q\rangle$ *is* $\langle AS, P \cap Q\rangle$.

A collector using the least upper bound of two pointer augmentations can be thought of as looking at pointers that appear in both pointer augmentations. This will give it a better approximation of the points-to relation than would using either augmentation alone.

Theorem 7 *For any two storage states* $\langle AS, Q\rangle$ *and* $\langle AS, R\rangle$, $\mathcal{G}_{\langle AS,Q\rangle \vee \langle AS,R\rangle} \supseteq \mathcal{G}_{\langle AS,Q\rangle} \cup \mathcal{G}_{\langle AS,R\rangle}$.

> If $a \in \mathcal{G}_{\langle AS,Q\rangle}$, then $a \notin Q^*$. But then $a \notin (Q \cap R)^*$, and so $a \in \mathcal{G}_{\langle AS,Q\rangle \vee \langle AS,R\rangle}$ by definition 7. The same argument applies when $a \in \mathcal{G}_{\langle AS,R\rangle}$.

It is somewhat surprising that, in fact, the sum can be greater than the parts and the coordinated use of two pointer augmentations can give a more precise collection than the use of both individually. For example, Figure 1 shows a storage state and two of its augmentations, where the nodes represent objects in AS and the arrows represent the points-to relation. In both augmentations there is no garbage, but their least upper bound, which is the intersection of the relations, is the same as the original storage state where only the root is reachable.

This example also illustrates the value of defining operations on pointer augmentations rather than trying to define operations directly on the garbage discovered using the augmentation. Both augmentations in Figure 1 make all three objects reachable, and none collectable. If the information in the augmentations were not preserved then there would be no way to discover that two objects can be collected.

There are many reasons to use an augmented state rather than the true state. First, it may be that the relation P is not available to the collector, but some superset of it is. This happens when the collector cannot tell the difference between pointers and other data item [BW88]. Second, the designer of a garbage collector may wish to trade quality of collection, in terms of objects recovered, for efficiency. This is the case in generational garbage collection. The true points-to relationship is available but heuristics to make the collection more efficient call for making some objects immune from collection.

Lemma 5 *For any storage state $\langle AS, P\rangle$, and any $I \subseteq AS$ such that $\langle I, P\rangle$ is a storage state, there is a pointer augmentation $\langle AS, Q\rangle$ of $\langle AS, P\rangle$ such that $\mathcal{G}_{\langle AS,Q\rangle}$ is exactly $\mathcal{G}_{\langle AS,P\rangle} - I$.*

> A pointer augmentation that will make all objects in I immune from collection is $Q = P \cup \{\langle r, o\rangle | o \in I\}$. bott@pyramid.pyramid.com Any reachable object of $\langle AS, Q\rangle$ is either reachable in $\langle AS, P\rangle$ or contained in I. To see that there are no objects outside of I that are made reachable by the addition of edges from r to all the objects in I, note that $\langle I, P\rangle$ is a storage state, and Invariant 2 guarantees that no object in I points to an object not in I. The unreachable objects of $\langle AS, Q\rangle$, then, are both garbage in $\langle AS, P\rangle$ and not in I, so the garbage in $\langle AS, Q\rangle$ is just $\mathcal{G}_{\langle AS,P\rangle} - I$.

Lemmata 4 and 5 show how pointer augmentations and garbage collection interact. A collection over any augmented storage state finds a subset of the garbage that can be safely collected, and any subset that can be deleted from the allocated set to produce a new storage state is given by a garbage collection of a pointer augmentation.

Theorem 6 *For any storage state $\langle AS, P\rangle$, $\langle AS - T, P\rangle$ is a storage state if and only if $T = \mathcal{G}_{\langle AS,Q\rangle}$ and $\langle AS, Q\rangle$ is a pointer augmentation of $\langle AS, P\rangle$.*

> If $\langle AS - T, P\rangle$ is a storage state, then Lemma 5 applies, and so there is a pointer augmentation of $\langle AS, P\rangle$ in which T is garbage.
>
> If $\langle AS, Q\rangle$ is a pointer augmentation of $\langle AS, P\rangle$ and $T = \mathcal{G}_{\langle AS,Q\rangle}$ then by Lemma 4 $\langle AS - T, P\rangle$ is also a storage state.

Theorem 6 proves the relationship between pointer augmentations and subsets of the garbage of a storage state that need not be collected. If a collector chooses not to collect all the garbage in a storage state, it cannot choose to ignore an arbitrary subset, but only those subsets that arise from pointer augmentations of the storage state. Further, it is free to choose any pointer augmentation and the subset of the garbage that will be collected will always leave a storage state.

2.3 Ordering Augmentations

The edges added to the graph $\langle O, P\rangle$ by a pointer augmentation indicate pointers that a collector is assuming to be present, even if they are not. A collector that assumes no extra pointers will find all the garbage but must examine the full graph, while a collector that assumes extra pointers may not find all the garbage but need not examine the whole graph. Effectiveness can be traded for efficiency. Some augmentations are more effective than others and the options available can be arranged in a partial order.

2.2 Pointer Augmentations

Definition 4 *If $\langle AS, P\rangle$ and $\langle AS, Q\rangle$ are storage states, then $\langle AS, Q\rangle$ is a* pointer augmentation *of $\langle AS, P\rangle$ if $Q \supseteq P$.*

Not every $Q \supseteq P$ will give a pointer augmentation, since $\langle AS, Q\rangle$ must be a storage state. In particular, no pointers may be introduced that will violate Invariant 2. This is no great restriction, and all the pointer augmentations in this paper will only introduce new edges between objects in AS. This ensures that the chosen supersets of P will not violate the invariants, and the result will always be a storage state, and thus a pointer augmentation.

Some pointer augmentations will have no effect on the garbage collected. For instance, connecting objects directly that are already connected through a finite chain of other objects will not change the transitive closure of the pointers. However, adding pointers to a storage state never increases the amount of garbage in the storage state.

Theorem 3 *If $\langle AS, Q\rangle$ is a pointer augmentation of storage state $\langle AS, P\rangle$, then $\mathcal{G}_{\langle AS,Q\rangle} \subseteq \mathcal{G}_{\langle AS,P\rangle}$.*

> If $a \in \mathcal{G}_{\langle AS,Q\rangle}$ then by Definition 3 $\langle r, a\rangle \notin Q^*$. But $Q^* \supseteq P^*$ since $Q \supseteq P$. Thus $\langle r, a\rangle \notin P^*$, and $a \in \mathcal{G}_{\langle AS,P\rangle}$.

A pointer augmentation $\langle AS, Q\rangle$ of a storage state $\langle AS, P\rangle$ always has a subset of the original garbage, and if the objects conatined in the garbage after an augmentation are removed from AS the result is a storage state not only under the augmented points-to relation, but under the original relation as well.

Lemma 4 *For any storage state $\langle AS, P\rangle$, and any pointer augmentation $\langle AS, Q\rangle$ of that storage state, $\langle AS, P\rangle$, $\left\langle AS - \mathcal{G}_{\langle AS,Q\rangle}, P\right\rangle$ is a storage state.*

> In order for $\left\langle AS - \mathcal{G}_{\langle AS,Q\rangle}, P\right\rangle$ to be a storage state, it must satisfy the two invariants in Definition 1.
>
> I_1: Since $\langle AS, P\rangle$ is a storage state, $r \in AS$. From Definitions 3, $r \notin \mathcal{G}_{\langle AS,P\rangle}$, and by Theorem 3 $\mathcal{G}_{\langle AS,Q\rangle} \subseteq \mathcal{G}_{\langle AS,P\rangle}$ so $r \in AS - \mathcal{G}_{\langle AS,Q\rangle}$.
>
> I_2: Assume $\langle a, b\rangle \in P \wedge a \in AS - \mathcal{G}_{\langle AS,Q\rangle}$ and try to prove $b \in AS - \mathcal{G}_{\langle AS,Q\rangle}$. Since $\langle AS, P\rangle$ is a storage state, Invariant 2 applied to a and b shows that $b \in AS$. But $a \notin \mathcal{G}_{\langle AS,Q\rangle}$, so $\langle r, a\rangle \in Q^*$. Since $\langle a, b\rangle \in P$, and $\langle AS, Q\rangle$ is a pointer augmentation $\langle a, b\rangle \in Q$. But if $\langle r, a\rangle \in Q^*$ and $\langle a, b\rangle \in Q$ then $\langle r, b\rangle \in Q^*$ and so $b \notin \mathcal{G}_{\langle AS,Q\rangle}$.

In addition, there are no other subsets that can be removed from the allocated set of a storage state other than those that are the garbage under an augmentation.

Considering $\langle O, P \rangle$ as a graph, there are some objects that are reachable from the root object and others that are not. The goal of garbage collection is to find those allocated objects that are not reachable.

Definition 2 *An object O is* reachable *in a storage state $\langle AS, P \rangle$ if $\langle r, o \rangle \in P^*$, the transitive closure of P.*

Only objects connected to the root in a finite number of steps are reachable since the transitive closure is only over finite chains. This may lead to odd situations when infinite chains exist, but since such objects could neither be created nor seen in a finite time by a mutator process, it is justified to call them "unreachable."

Invariant 2 in Definition 1 keeps objects in the allocated set from pointing to objects not in the allocated set guaranteeing that all the reachable objects are in the allocated set.

Theorem 1 *Every reachable object in a storage state $\langle AS, P \rangle$ is in AS.*

> If object o is reachable, then there is a finite set of objects $o_1, \ldots, o_n$ such that $\langle r, o_1 \rangle \in P, \langle o_1, o_2 \rangle \in P, \ldots, \langle o_{n-1}, o_n \rangle \in P, \langle o_n, o \rangle \in P$. By Invariant 1, $r \in AS$, and $n + 1$ applications of Invariant 2 shows that $o \in AS$.

The objects that are not reachable from the root are the objects a collector may move out of the allocated set, and every object that is not in the allocated set is part of the garbage.

Definition 3 *Define the* garbage *of a storage state $\langle AS, P \rangle$ to be $\mathcal{G}_{\langle AS,P \rangle} = \{o \in O | \langle r, o \rangle \notin P^*\}$*

Theorem 2 *For any storage state $\langle AS, P \rangle$, $O - AS \subseteq \mathcal{G}_{\langle AS,P \rangle}$.*

> If $a \notin AS$ then by Theorem 1 a is not reachable and so $A \in \mathcal{G}_{\langle AS,P \rangle}$ by Definition 3.

Many modern garbage collectors do not free all the unreachable storage but only a subset. For example, a collector might wish to free a subset if freeing all unreachable objects would require access to some storage not in main memory. The cost of changing the secondary storage to free the objects might be greater than the benefit.

Not every subset can be safely chosen, however, since there is a danger that Invariant 2 might be violated and objects in AS, albeit unreachable, might have pointers to objects that have been moved outside AS. There are subsets of the unreachable objects that can always be safely freed, and they are easily characterized in terms of pointer augmentations.

Likewise there have been improvements to tracing collectors. Most modern tracing collectors only try to collect the relatively new objects, relying on the statistical observation that most of the garbage will be in the new objects. Age-based discrimination lowers the overhead of tracing significantly [Han77, LH83, Moo84, Ung84] and in addition, real-time tracing collectors have been proposed and implemented [Ste75, Bak78, AEL88, DWH+90].

Many modern systems programming languages and environments have come to rely on garbage collection [Rov85, CDG+88], and collection is moving away from being a language-by-language service and becoming a part of an integrated operating system [WDH89]. Language-independent collection has lead to collectors that try to ignore the language-dependent aspects of pointers, and use conservative pointer-finding. Every word of an object must be treated as a pointer if it is possible that it might be a pointer [BW88]. Conservative collection places further restrictions on a collector, usually allowing but sometimes precluding copying of objects during collection [Bar88].

All these improvements to the efficiency and applicability of garbage collection are independent, but the effects of combining one feature with another are unclear. The goal of this paper is to shed some light on interactions of garbage collection features and outline a theory whereby the effects of various combinations of features can be analyzed.

Section 2 gives the theory of garbage collection in terms of pointer augmentations. Section 2.4 generalizes the concepts of aged-based generational collection to sets of objects determined by orderings other than time. Several examples are given in Section 2.5, including time-based generational, independent heaps, and even reference counted collection. A particular promising special case of pointer augmentations, A/F Collection, is detailed in Section 2.7.

2 Collector Theory

2.1 Storage Model

There is a set of objects O with a distinguished *root object* $r \in O$.

Definition 1 *A* storage state *is a pair* $\langle AS, P \rangle$ *where* AS*, the* allocated set*, is a subset of* O *and* $P \subseteq O \times O$*, the* points-to relation*, is a binary relation on* O*.*

In addition, a storage state must satisfy these invariants:

I_1*:* $r \in AS$*.*

I_2*:* $\langle a, b \rangle \in P \wedge a \in AS \Rightarrow b \in AS$*.*

Simply stated, allocated objects are special in that any pointer in an allocated object must point to another allocated object. A garbage collector must preserve this property and can do nothing that would create a pointer from an allocated object to an unallocated object.

A Theoretical Basis for Garbage Collection *

Barry Hayes†
Stanford University

November 20, 1990

A garbage collector that sometimes elects to collect only a subset of the free objects is equivalent to a garbage collector that collects all of the free objects that exists if extra pointers are added between objects. This is true either if the partial collection arises from basic style of the collector, as with reference counting collection, or if the partial collection comes from an efficency consideration, as with time-based generational collection, or if the partial collection is the result of inaccurate pointer filtering, as with conservative collection. The extra pointers added to change a collector's behavior can be used to classify all collectors into a lattice. The least upper bound of any set of collectors combines the abilities for partial collections from all.

1 Introduction

Opponents of garbage collection cite the added overhead of collection and the lack of real-time guarantees that can be made when so much processing is taking place under the table. Supporters of garbage collection point out that automatic collection liberates the programmer from a task that is often difficult to program correctly, guarantees that there will be no dangling reference problems from objects freed too early, and may have a lower overhead than explicit deallocation.

There are two main collection strategies, reference-counting and tracing. Naive implimentations of garbage collectors of both sorts have been known for quite a while [**?**], but often have problems with time and space efficieny. Improvements to reference-counting collectors include delayed counting [DB76], lazy counting [Bak78, GT87], and weighted pointers [Bev87, WW87]. These improvements have made reference counting a viable method of garbage collection for many applications.

*This is just a prelimiary draft, and not meant for publication yet. As a courtesy to the author, not to mention the public, please don't copy or circulate this draft.

†This work was funded by Xerox and the Northern California Chapter of ARCS Foundation, Inc.