Recycling Garbage Theory

Christian Grothoff
Department of Computer Sciences, Purdue University
grothoff@cs.purdue.edu

March 18, 2004

1 Introduction

In [1], a general collection theory (CT) that encompasses generational and conservative collection was introduced. The authors then use this theory as the basis for an entire class of collectors, which we will call CGCGC for Conservative Generational Connectivity-based Garbage Collection, short CGCGC. While the implementation only discussed two and n-generational collectors, the theoretical foundations presented were intended to be broader, though not explored in the paper itself. Recently, [2] introduced the idea of connectivity-based garbage collection (CBGC). In this paper, we show that the CT used for CGCGC is powerful enough to fully model CBGC. Furthermore, we show that the CGCGC collectors can also be viewed as variants of CBGC. The resulting constructions demonstrate that the two different collector models are in fact equivalent modulo the specific style how the designed collectors model the points-to graph.

The merrit of this approach is twofold. First, it gives a theoretical foundation to CBGC-based collectors. Second, it yields a natural path towards combining connectivity-based and conservative collection (in the same way that CGCGC combined generational and conservative collection). This combination can be useful to apply CBGC to problems where a static analysis of the code is not able to obtain a safe approximation of the points-to graph. The resulting conservative CBGC would use selective write-barriers at run-time to lift the optimistic result from the static analysis to a safe, conservative approximation which would then allow for incremental collection in the style of CBGC.

2 Review of Collector Theory (CT)

This section reviews and slightly reformulates the collector theory presented in section II of [1]. Note that portions are reproduced almost verbatim. The reformulations try to simplify the theory a bit and improve the presentation. A few restrictions of the original formulation are lifted and proofs are detailed. The core of the theory is not changed.

Let O be a countable set of objects, $r \in O$ the root of the object graph and $P \subseteq O \times O$ a reflexive binary points-to relation on O.

2.1 Basic definitions

The following basic definitions are used to model the heap and describe what a correct garbage collector is allowed to do. They also characterize the behavior of any correct conservative collector.

Definition 1 (Valid allocation) *An allocation $AS \subseteq O$ is* valid *with respect to a root r and points-to relation $P \subseteq O \times O$ if and only if:*

$$r \in AS \tag{1}$$

$$(a, b) \in P \quad \wedge \quad a \in AS \Rightarrow b \in AS \tag{2}$$

Axiom 2 (Mutator) *All changes to the points-to relation P performed by the mutator preserve validity.*

Definition 3 (Garbage Collection) *A* garbage collection *of a state $\langle AS, P, r\rangle$ is a state $\langle AS', P, r\rangle$ with $AS' \subseteq AS$. A garbage collection is* valid *if AS' is valid with respect to r and P.*

Lemma 4 (Precise collection) *For a given state $\langle AS, P, r\rangle$ there exists a minimal set $AS^*(P, r)$ such that $AS^*(P, r) \subseteq AS'$ for any valid garbage collection $\langle AS', P, r\rangle$ of $\langle AS, P, r\rangle$.*

Proof: Let $T_i := T_{i-1} \cup \{y \in O | x \in T_{i-1} \wedge (x, y) \in P\}$ with $T_0 = \{r\}$ be monotonically increasing sequence of sets. Since O is countable, there exists a set T_∞ such that $T_n \subseteq T_\infty$ for all n. We now show that $AS^*(P, r) = T_\infty$. Note that $\langle T_\infty, P, r\rangle$ is valid by definition. Let $\langle S, P, r\rangle$ be another valid state. If $T_\infty \nsubseteq S$ there exists a $z \in T_\infty - S$. Let n be the smallest n such that $z \in T_n$. By construction of T_n, there must then exist a sequence $(r = z_0, \ldots, z_n = z)$ such that $(z_i, z_{i+1}) \in P$. If $\langle S, P, r\rangle$ is valid, S must contain z_i for $i \in [0, n]$ (by induction on the definition). But $z \in S$ contradicts $z \in T_\infty - S$. $\square$

Definition 5 (Pointer augmentation) *Q is a pointer augmentation of P if $P \subseteq Q$.*

Lemma 6 *$C \subseteq O$ is a valid allocation with respect to r and P if and only if there exists a pointer augmentation Q of P such that $C = AS^*(Q, r)$.*

Proof: If Q is a pointer augmentation of P, then $AS^*(Q, r) \supseteq AS^*(P, r)$ since $Q \supseteq P$. Since $r \in AS^*(P, r)$ this implies $r \in AS^*(Q, r)$. Furthermore, (2) is satisfied since AS^* is defined as the minimal set for which (2) holds. Thus if there exists a pointer augmentation Q of P the subset $C = AS^*(Q, r)$ is a valid allocation.

Suppose $C \subseteq O$ is a valid allocation with respect to r and P. Define Q to be $Q := P \cup \{(r,e)|e \in C\}$. Then by construction $C \subseteq AS^*(Q,r)$. Since $AS^*(Q,r)$ is by construction the minimal set containing C which is a valid allocation $C = AS^*(Q,r)$ follows. □

2.2 Posets and Embeddings

The following text gives the basic definitions for posets and embeddings. A poset, short for partially-ordered set, is another term for a directed, acyclic graph which abstracts points-to relationships. An embedding is a pair of functions that map an object to two, not necessarily distinct elements in the poset. The two functions can be thought of abstracting which other objects an object can (transitively) point to (function F for *f*ollowers) and all objects that an object can be reached from (function A for *a*ncestors). In other words, the embedding (F, A) describes that an object o can transitively refer to all objects p for which $F(o) \geq A(p)$.

Definition 7 (Poset) *A* pointed partially ordered set *(poset) is a triple $\langle D, \geq, \perp\rangle$ where D is a set, $\geq$ is a reflexive, transitive and antisymmetric relation on D and $\perp \in D$ such that $\forall x \in D \quad x \geq \perp$.*

Definition 8 (Embedding) *Let $\mathcal{D} = \langle D, \geq, \perp\rangle$ be a poset and $AS \subseteq O$ a valid allocation with respect to r and P. An embedding of S into $\mathcal{D}$ is a pair (F, A) of functions from O to D such that:*

$$\forall (a,b) \in P \Rightarrow F(a) \geq A(b). \tag{3}$$

An embedding (F, A) determines an induced points-to relation $P_{F,A}$ on O by:

$$(a,b) \in P_{F,A} \Leftrightarrow F(a) \geq A(b). \tag{4}$$

Definition 9 (Lossless Embedding) *An embedding (F, A) of P is said to be lossless if and only if $P_{F,A} = P$.*

Lemma 10 *Let $AS \subseteq O$ be a valid allocation with respect to r and P and let (F, A) be an embedding of AS in the poset $\mathcal{D} = \langle D, \geq, \perp\rangle$. Then the induced points-to relation $Q := P_{F,A}$ is a pointer augmentation of P.*

Note that (F, A) is a lossless embedding of Q in $\mathcal{D}$ by definition.
Proof: If (F, A) is an embedding of AS in $\mathcal{D}$ then for each element $(a, b) \in P$ the relation $F(a) \geq A(b)$ must hold. But then $(a, b) \in Q$ by definition of $P_{F,A}$ and thus $Q \supseteq P$ (Q is a pointer augmentation of P). □

Lemma 11 *Let $AS \subseteq O$ be a valid allocation with respect to r and P and let Q be a pointer augmentation of P. Then there exists a poset $\mathcal{D} = \langle D, \geq, \perp\rangle$ and an embedding (F, A) of AS in $\mathcal{D}$ such that $P_{F,A} = Q$.*

Proof: Let $D = \{\perp\} \cup ((\{\perp\} \cup O) \times (\{\perp\} \cup O))$ with $\perp = (\perp, \perp)$. Define the poset operation $\geq$ to be

$$(a,b) \geq (c,d) \quad \Leftrightarrow \quad (a,d) \in Q \tag{5}$$

$$\texttt{and} \quad \forall_{(e,f)\in D} \quad (e,f) \geq \perp. \tag{6}$$

Define $F(o) = (o, \perp)$ and $A(o) = (\perp, o)$. Then (F, A) is a lossless embedding of AS in $\mathcal{D} = \langle D, \geq, \perp \rangle$ by construction. □

Definition 12 (Poset-homomorphism) *Let $\mathcal{D} = \langle D, \geq_D, \perp_D \rangle$ and $\mathcal{E} = \langle E, \geq_E, \perp_E \rangle$ be posets. A function h from $\mathcal{D}$ to $\mathcal{E}$ is a* homomorphism *from $\mathcal{D}$ to $\mathcal{E}$ if it is strict and monotonic, i.e.̃for all $a, b \in D$ with $a \geq_D b$ the application of h must result in $h(a) \geq_E h(b)$ with $h(a) = h(b)$ only if $a = b$.*

Lemma 13 *Let $\mathcal{D} = \langle D, \geq_D, \perp_D \rangle$ and $\mathcal{E} = \langle E, \geq_E, \perp_E \rangle$ be posets and let h be a homomorphism from $\mathcal{D}$ to $\mathcal{E}$. Let $AS \subseteq O$ be a valid allocation with respect to r and P and (F, A) an embedding of AS in $\mathcal{D}$. Then $(h \circ F, h \circ A)$ is an embedding of AS in $\mathcal{E}$ with $P_{h\circ F, h\circ A} \supseteq P_{F,A} \supseteq P$.*

Proof: For all $(a, b) \in P$ we have $F(a) \geq A(b)$ since (F, A) is an embedding of AS in $\mathcal{D}$. Since h is a poset-homomorphism $s \geq t$ implies that $h(s) \geq h(t)$ and thus $h(F(a)) \geq h(A(b))$. Thus $(h \circ F, h \circ A)$ is an embedding of AS in $\mathcal{E}$.
Since for all $(a, b) \in P$ the fact that (F, A) is an embedding guarantees that $F(a) \geq A(b)$ it must hold that $(a, b) \in P_{F,A}$ by definition of $P_{F,A}$, showing that $P_{F,A} \supseteq P$. Similarly, if $(a, b) \in P_{F,A}$ then $F(a) \geq A(b)$ and again $h(F(a)) \geq h(A(b))$ and thereby $(a, b) \in P_{h\circ F, h\circ A}$. □

Lemma 14 (Canonical posets and embeddings) *Let $AS \subseteq O$ be a valid allocation with respect to r and P. There exists a poset $\mathcal{D} = \langle D, \geq_D, \perp_D \rangle$ called a* canonical *poset for AS and a lossless embedding (F, A) of AS in $\mathcal{D}$ called a* canonical *embedding for AS with the property that for any pointer augmentation Q of P there exists a poset $\mathcal{E} = \langle E, \geq_E, \perp_E \rangle$ and a homomorphism h from $\mathcal{D}$ to $\mathcal{E}$ such that $(h \circ F, h \circ A)$ is a lossless embedding of AS in $\mathcal{E}$.*

Proof: According to lemma 11 it is possible to construct a poset $\mathcal{D}$ and lossless embedding (F, A) for any pointer augmentation of P. Use P for the pointer augmentation to construct the canonical poset and embedding according to lemma 11.
Given any pointer augmentation Q of P, construct the poset $\mathcal{E}$ for AS again according to lemma 11. The homomorphism h between $\mathcal{D}$ and $\mathcal{E}$ is the pairwise identity function. This definition of h results in an homomorphism since $Q \supseteq P$ guarantees that for all pairs $(a, b) \geq_D (c, d)$ also $(a, b) \geq_E (c, d)$ holds. □

The theory developed so far can help building efficient collectors. Suppose (F, A) is a lossless embedding in some poset for a valid allocation AS with respect to r and P. Consider the set $A^{-1}(\perp)$. All objects $o \in A^{-1}(\perp)$ will never be collected since r is life and $F(r) \geq A(o) = \perp$. In other words, the objects $A^{-1}(\perp)$ are not threatened by the collector. Now consider the set $F^{-1}(\perp)$. Any object $a \in F^{-1}(\perp)$ can only have pointers into $A^{-1}(\perp)$ since the existence of a reference from a to b in P requires that $(a, b) \in P$ which implies that $\perp = F(a) \geq A(b) \geq \perp$ and thereby $A(b) = \perp$. Thus a collector does not have to trace the *bystander set* $F^{-1}(\perp)$ or sweep $A^{-1}(\perp)$.

2.3 Combining collectors: Ideals

The general theory described so far can be used to combine different collectors that can be expressed with the definitions given above. For combining different collectors, we first define a product combinator on posets and then derive functions that can be used to specify the collector.

Definition 15 (Poset-ideal) *An* ideal *in a poset $D = \langle D, \geq, \perp \rangle$ is a nonempty subset I of D such that:*

$$\forall_{i \in I}\ i \geq j \quad \Rightarrow \quad j \in I. \tag{7}$$

Definition 16 (Cartesian product of posets) *Let $\mathcal{D}_1 = \langle D_1, \geq_1, \perp_1 \rangle$ and $\mathcal{D}_2 = \langle D_2, \geq_2, \perp_2 \rangle$ be posets. The Cartesian product $\mathcal{D}_1 \times \mathcal{D}_2$ is the poset $\mathcal{D} = \langle D, \geq, \perp \rangle$ where:*

$$D = \{\perp\} \cup \{(x_1, x_2) | x_1 \in D_1 - \{\perp_1\} \wedge x_2 \in D_2 - \{\perp_2\}\} \tag{8}$$

$$\perp \geq \perp \tag{9}$$

$$(x_1, x_2) \geq \perp \tag{10}$$

$$(x_1, x_2) \geq (y_1, y_2) \quad \Leftrightarrow \quad x_1 \geq_1 y_1 \wedge x_2 \geq_2 y_2 \tag{11}$$

Lemma 17 *The function $M_a(b) := (a, b)$ defines a poset-homomorphism from $\mathcal{D}_2$ to $\mathcal{D}$. The function $N_b(a) := (a, b)$ defines a poset-homomorphism from $\mathcal{D}_1$ to $\mathcal{D}$.*

Proof: Suppose $x \geq_2 y$. Then $M_a(x) \geq M_a(y)$ since $a \geq_1 a$ for all $a \in D_1$ and thus M_a is defines a poset-homomorphism from $\mathcal{D}_2$ to $\mathcal{D}$. Suppose $x \geq_1 y$. Then $N_b(x) \geq N_b(y)$ since $b \geq_2 b$ for all $b \in D_2$ and thus N_b is defines a poset-homomorphism from $\mathcal{D}_2$ to $\mathcal{D}$. □

Definition 18 ($\otimes$ operator) *Let $f : S \to D$ and $g : S \to E$ be functions. Define $f \otimes g : S \to D \times E$ by:*

$$(f \otimes g)(x) := (f(x), g(x)) \qquad \texttt{for } x \in S \texttt{ with } f(x) \neq \perp \wedge g(x) \neq \perp. \tag{12}$$

If $f(x) = \perp$ or $g(x) = \perp$ then $(f \otimes g)(x) := \perp$.

Remark 19 *Note that*

$$(f \otimes g)^{-1}(\perp) = f^{-1}(\perp) \cup g^{-1}(\perp). \tag{13}$$

Lemma 20 *Let $AS \subseteq O$ be a valid allocation with respect to r and P. Let (F_1, A_1) and (F_2, A_2) be embeddings of AS into $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively. Then $(F_1 \otimes F_2, A_1 \otimes A_2)$ is an embedding of AS into $\mathcal{D}_1 \times \mathcal{D}_2$ with*

$$P_{F_1 \otimes F_2, A_1 \otimes A_2} = (P_{F_1,A_1} \cap P_{F_2,A_2}) \cup \{(a,b) \in AS \times AS | (A_1 \otimes A_2)(b) = \perp\}.$$

Proof: In order to show that $(F_1 \otimes F_2, A_1 \otimes A_2)$ is an embedding of AS we need to show that equation (3) holds for all $(a,b) \in P$.
If $(a,b) \in P$ then $F_1(a) \geq_1 A_1(b)$ and $F_2(a) \geq_2 A_2(b)$ since (F_1, A_1) and (F_2, A_2) are embeddings. But $(F_1 \otimes F_2)(a) = (F_1(a), F_2(a))$ and $(A_1 \otimes A_2)(b) = (A_1(a), A_2(a))$ and since $F_1(a) \geq_1 A_1(a)$ and $F_2(a) \geq_2 A_2(a)$ equation (11) yields as desired $(F_1 \otimes F_2)(a) \geq (A_1 \otimes A_2)(b)$. Thus $(F_1 \otimes F_2, A_1 \otimes A_2)$ is an embedding of AS into $\mathcal{D}_1 \times \mathcal{D}_2$.
Now suppose $(a,b) \in P_{F_1 \otimes F_2, A_1 \otimes A_2}$. Then $(F_1(a), F_2(a)) \geq (A_1(b), A_2(b))$ by equation (4) and the definition of the $\otimes$ operator. Definition 16 lists three conditions under which the $\geq$ operator holds.

(11) This case implies that $F_1(a) \geq A_1(b)$ and $F_2(a) \geq A_2(b)$, which is equivalent to $(a,b) \in (P_{F_1,A_1} \cap P_{F_2,A_2})$, yielding the first part of the equation.

(10) In order for equation (10) to be applicable, $\perp = (A_1(b), A_2(b)) = (A_1 \otimes A_2)(b)$ must hold, which is the second part of the equation.

(9) Equation (9) only places additional constraints on (a,b) over (10) and thus does not contribute further elements to $P_{F_1 \otimes F_2, A_1 \otimes A_2}$.

□

3 CBGC and CGCGC

This section briefly introduces the ideas behind CBGC and then shows the equivalence between CBGC and CGCGC. Finally, it discusses how CBGC and CGCGC could be combined to a conservative, connectivity-based garbage collector.

3.1 Connectivity-based Garbage Collection (CBGC)

CBGC divides the set of heap objects into disjoint partitions. The partitions are nodes in the partition graph. A possible reference between two objects on the heap results in a directed edge between the corresponding partitions. Partitions that are part of a cycle in the graph are joined such that the graph forms a directed, acyclic graph (DAG).

In this DAG, the GC selects a subset of the partitions that is closed under the predecessor relation and collects the corresponding subset of the heap. Since the objects in each closed subset of the partitions in the DAG can be collected independently of the rest of the heap, CBGC allows GC to happen in smaller increments that are more tailored to the run-time behavior of the program than traditional generational collections. In some sense, where generational collectors exploit the dynamic, temporal property that objects that have not been modified after the last collection cycle can not point to new objects, CBGC uses static analysis to derive a static approximation of a superset of the points-to graph.

3.2 CGCGC can be expressed using CBGC

The proposed CBGC collector uses static analysis of the mutator to determine a partitioning of the heap. The theoretic model presented in [1] assumes the existance of some conservative approximation of the points-to graph of all objects in the heap. The implementation of the combined conservative and generational collector uses time (or pre-existence) as their primary way of approximating the points-to graph. They build their system on the fundamental property that an object that has been modified no later than time t_M can not directly point-to objects that have been allocated at time t_A where $t_M < t_A$. While this approach towards building the points-to graph is not a static analysis, it still induces a partitioning of the heap into Posets which could the consequently be considered a DAG for collection with CBGC. The only major difference between the two designs is thus that CBGC uses a *static* partitioning of the heap derived from *static analysis* whereas CGCGC allows a *dynamic*ally changing set of partitions based on run-time observations made with the help of *write-barriers*. Note that this introduces a write-barrier into the CBGC collector. The original CBGC model replaced the write-barrier with static analysis. Given a static analysis that yields an approximation of the relative creation and modification times of objects, the write-barrier would again not be required. Except for differences in the partitioning induced by the different style of how the points-to graph is approximated, the collectors are identical.

In other words, the DAG used by CBGC is equivalent to a Poset in CT. The partitions are the set elements and the links induce a partial ordering. Similarly, any partially ordered set can be seen as a DAG.

3.3 CBGC can be described using CT

An optimistic[1] static analysis that approximates the dynamic points-to graph, results in an embedding (F_{SA}, A_{SA}) that maps each allocation site (or whatever the granularity of the static analysis may be) to a node in the DAG. Note that $F_{SA}(o) = A_{SA}(o)$ and that F and A are *constant* over time.

[1] The static analysis is optimistic since conservative collection allows for an uncooperative setting where complete points-to information is unavailable.

3.4 Combining CBGC and CGCGC

Note that since the static analysis is optimistic, the embedding must be lifted using a second embedding (F_{CO}, A_{CO}) that dynamically changes depending on run-time behavior collected by write-barriers in the style of the CGCGC collector. The resulting embedding induced by the product of the posets corresponding to the SA and CO embeddings can then be savely used to collect the heap.

One primary question is if this combined collector would do any better than just a conservative CGCGC collector. The question needs to be raised in particular since the optimistic static analysis can only yield less precise points-to information compared to only using the conservative CGCGC. The answer lies in the inherent complexity of building partitions (to use the CBGC terminology) that go beyond the generations used in the age-DAGs used in the original CGCGC implementation. Using general DAGs would drive up the cost of the write-barrier in CGCGC since it would now have to frequently join arbitrary partitions. A good pre-partitioning of the heap with the help of static analysis could make changes to the DAG by the write-barrier infrequent to the point where they do not happen in practice.

4 Conclusion

The next step towards putting this theoretical result into practice would be to implement a variant of CBGC that relies on static analysis to obtain a first approximation of the points-to graph of the heap and then combines this with write-barriers for changes to the heap by code that fall out of the scope of the analysis. This type of analysis would make CBGC useful for mixed-language applications where only a subset of the code can be effectively subjected to static analysis. An example would be any application written in Java that makes use of reflection and native calls[2].

Acknowledgements

I thank Martin Hirzel for comments on an earlier draft of this paper.

References

[1] Alan Demers, Mark Weiser, Barry Hayes, Hans Boehm, Daniel Bobrow, and Scott Shenker. Combining generational and conservative garbage collection: framework and implementations. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 261–269. ACM Press, 1990.

[2] Martin Hirzel, Amer Diwan, and Matthew Hertz. Connectivity-Based Garbage Collection. In *Proceedings of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'2003)*, Anaheim, California, USA, October 2003.

[2] Assuming the native calls are not guarded with a JNI-like API for efficiency.