Project 1: Garbage Collection

Design Check: March 5 - March 9, 2020
Due: 11:59pm Thursday, March 12, 2020

Project Partners

You will be working with a partner. You can find your partner assignment here. You can generally contact your partner by emailing them at <login>@cs.brown.edu

Assignment Specifications

Formal methods tools are useful as instruments of persuasion. Whether or not algorithms, protocols, or data structures obey desirable mathematical properties can be powerful evidence for introducing new technical designs, or renovating old ones.

For this assignment, you will be modeling the garbage collection algorithms **Mark and Sweep** and **Stop and Copy** with the goal of showing that they're both **sound** and **complete**.

Up until now, we've done most of the work designing models, dictating what the sigs, relations, and predicates are. In this assignment, you will take on that job! There is no stencil code. You should hand in 2 Forge files, one for each algorithm.

We highly recommend looking over Lab 2: Memory Management to review the concepts of soundness and completeness, and to borrow heavily from the model design there. Design decisions you will need to make include:

- How will we use sigs to represent units of memory?
- How will reachability and whether memory is allocated be represented?
- What will each state represent? How many states will there be?
- Besides predicates for safety, cleanness, soundness and completeness, what other predicates are needed?
- What sanity checks will ensure the model is consistent to the algorithm specification? This may not only be checking that invalid states or transitions occur, but also that all valid states and transitions can occur.

You are free to design the model as you wish, with only these rigid specifications:

1. Do not use the traces and transition predicate syntax; instead, explicitly constrain transitions between your states like we did in the lab.
2. Comment each sig and pred to briefly explain their purpose. For trickier logic, please include inline comments so that we can follow your thought process.
3. Include a check for both soundness and completeness for both algorithms like we did in the lab.

Mark and Sweep

Like in Lab 2, the Mark and Sweep algorithm works on a heap composed of HeapCells, with one of the acting as the 'Root' (equivalent to the Stack in the lab) that serves to determine reachability of memory. The idea is to distinguish live, reachable memory from garbage memory (the 'mark' phase), and then clear the garbage memory (the 'sweep' phase).
The algorithm works by passing over all objects in the Heap twice:

1. On the first pass, traverse the Heap starting from the Root, marking all reachable memory.
2. On the second pass, traverse the entirety of the Heap, garbage-collecting all memory that was not marked.

We recommend taking a look at Wikipedia's explanation of the naive mark-and-sweep method.

Stop and Copy

Stop and Copy is a GC algorithm that works by partitioning the heap into at least two (possibly non-consecutive) semispaces - one 'active' and the other 'inactive'. At any point in time, all allocated HeapCells are in the active semispace, and all memory in the inactive semispace is unallocated. Whenever the active space fills up (or at a regular interval), the garbage collector copies every live cell from the active space into the inactive space. Here, 'live' generally means reachable from the stack. Then, the roles of the active space and inactive space are flipped - the previously inactive space becomes the new active space, and contains all memory reachable from the stack, and the previous active space becomes inactive, and all its HeapCells are considered garbage.

Here's a good explanation of the stop and copy algorithm:
https://book.huihoo.com/data-structures-and-algorithms-with-object-oriented-design-patterns-in-java/html/page426.html

Here's a resource that discusses both algorithms:
http://cs.brown.edu/courses/cs195y/2020/pages/pdf/gcsurvey.pdf
You can find the explanation of Mark and Sweep starting at the bottom of page 7 (section 2.2), and Stop and Copy on page 9 (section 2.4).

Testing

We want you to prove to us that your model works. To this extent, it is **crucial** that you include sanity-checking preds (predicates that check for implicit consequences of your other preds) as well as instance tests to ensure your model is neither under- nor overconstrained. The better the tests, the more confidence we will be that your spec works. If you do not include demonstrative tests, your grade **will** be affected.

README

For this project, we ask that you submit a README that explains your implementation. At a minimum, you should address each of the following points:

- Give an overview of your model design choices, what checks or run statements you wrote, and what we should expect to see from an instance produced by Sterling. How should we look at and interpret an instance created by your spec?
- At a high level, what do each of your sigs and preds represent in the context of the model? Justify the purpose for their existence and how they fit together.

Design Check

There will be one mandatory design check with a TA. To make sure you're on the right track, we ask that you have the following code completed:

- Memory and State sigs
- Some way of representing reachability and allocation status of memory

You do not have to stick with whatever you show us, but you must demonstrate that you have at least one clear idea of how to accomplish these representations.

Extra Credit

Your group may earn extra credit by completing or showing significant progress toward a 'reach goal' - in particular, these should be goals that are nontrivial modeling challenges that show something new beyond soundness and completeness in mark and sweep/stop and copy. This can be whatever you want - in the past, some successful reach goals have been:

- Modeling a third GC algorithm that has some advantages over one of these two, and showing in your model how those advantages manifest.
- Modeling memory fragmentation in mark and sweep/stop and copy, and showing how stop and copy helps to defragment the heap.

You don't need to fully complete a reach goal to earn some extra credit - in general, we appreciate seeing some ambition in your work and understand if it is not fully feasible given the time constraints. As long as you've put good faith effort into implementing something interesting, we will recognize that.

Your design check is a great time to consult with a TA about your reach goal if you have questions about its feasibility or whether it is sufficient to count as extra credit.

Collaboration Policy

Please remember to review our collaboration policy for project assignments. In addition to high-level discussions, you may also receive debugging help from others (while their code is closed), but you must list the people you received assistance from in your `collaborators` file.

Handing In

For this assignment, you should hand in two Forge files (`mark_and_sweep.rkt` and `stop_and_copy.rkt`), as well as your `README` (either as a txt file or a pdf - something that we can open on department machines). Please also submit a file named `collaborators` containing the logins of the people you received debugging help from (one on each line). Please **do not** include any identifying information for you or your partner in any of your files.

To handin, run `cs1950y_handin midterm1` from a directory containing these files. You should receive an email confirming that your handin has worked. **Only one partner needs to hand in the project.**