

CMSC 631 Final exam Answers, Fall 2002

1. Garbage collectors:

- (a) Under what circumstances would you use a copying GC?

Answer: A copying collector is generally more efficient than a non-copying collector, and so would be used unless you are forced to use a non-copying collector. Since a copying collector updates pointers, you must be 100% in identifying all the pointers to an object before copying the object.

Most efficient Java VM's use some form of copying collector.

- (b) Under what circumstances would you use a non-copying GC?

Answer:

If you can't accurately identify all pointers, you must use a copying collector.

Non-copying collectors will sometimes work better than copying collectors when the working set is close to the size of the physical memory available. However, in either case the GC overhead will be high in that situation. In other words, a non-copying collector will have bad performance. A copying collector may have even worse performance.

Garbage collectors for unsafe languages (e.g., C and C++) are always conservative non-copying collectors. Initial implementations of Java used a non-copying collector due to the difficulty of identifying root pointers in the stack frame.

- (c) Discuss the differences in overhead and pause time between these two approaches, both in general, and what happens as more physical memory is made available (while not changing the working set of the program).

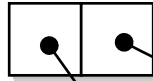
Answer: For copying collector, I'm going to assume a Cheney style collector, as opposed to a mark-and-compact collector. For the non-copying collector, I'm going to assume a standard mark-and-sweep collector.

A Cheney collector only has to visit and copy all of the live data, while a mark-and-sweep collector has to visit the entire heap. This generally leads to higher pause times and overhead for a mark-and-sweep collector.

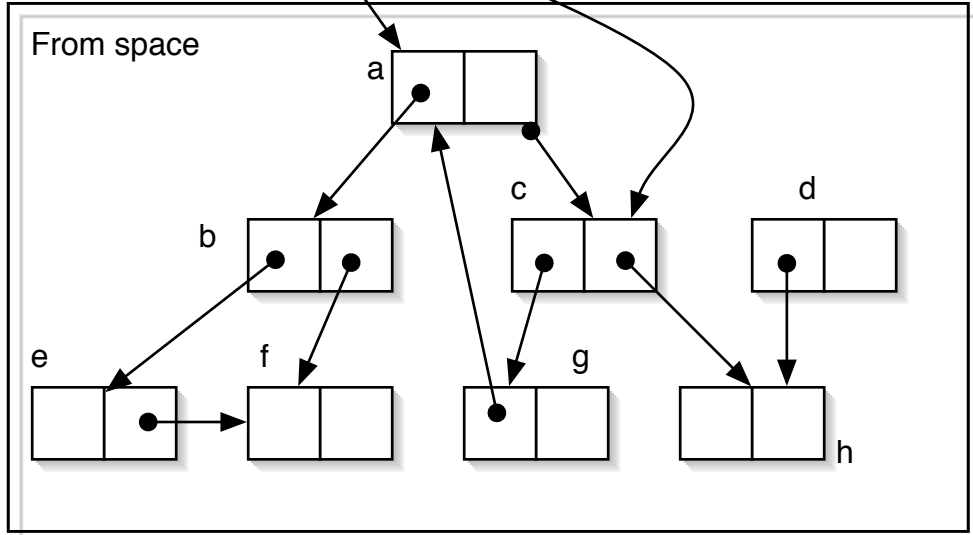
It is possible to reduce pause times by using a generational copying collector, or by performing incremental sweeping. Generational GC's are substantially more complicated (but yield a number of benefits), while incremental sweeping is relatively easy to implement.

2. Cheney garbage collector. For the following root pointers and from space, show the situation after the Cheney collector has copied 3 cons cells and after the collector has finished. For the mid-execution snapshot, show any internal pointers or state of the GC algorithm. Make sure that in the to space, cons cells are laid out in the order they are copied. Label the cells (a - h).

Root pointers

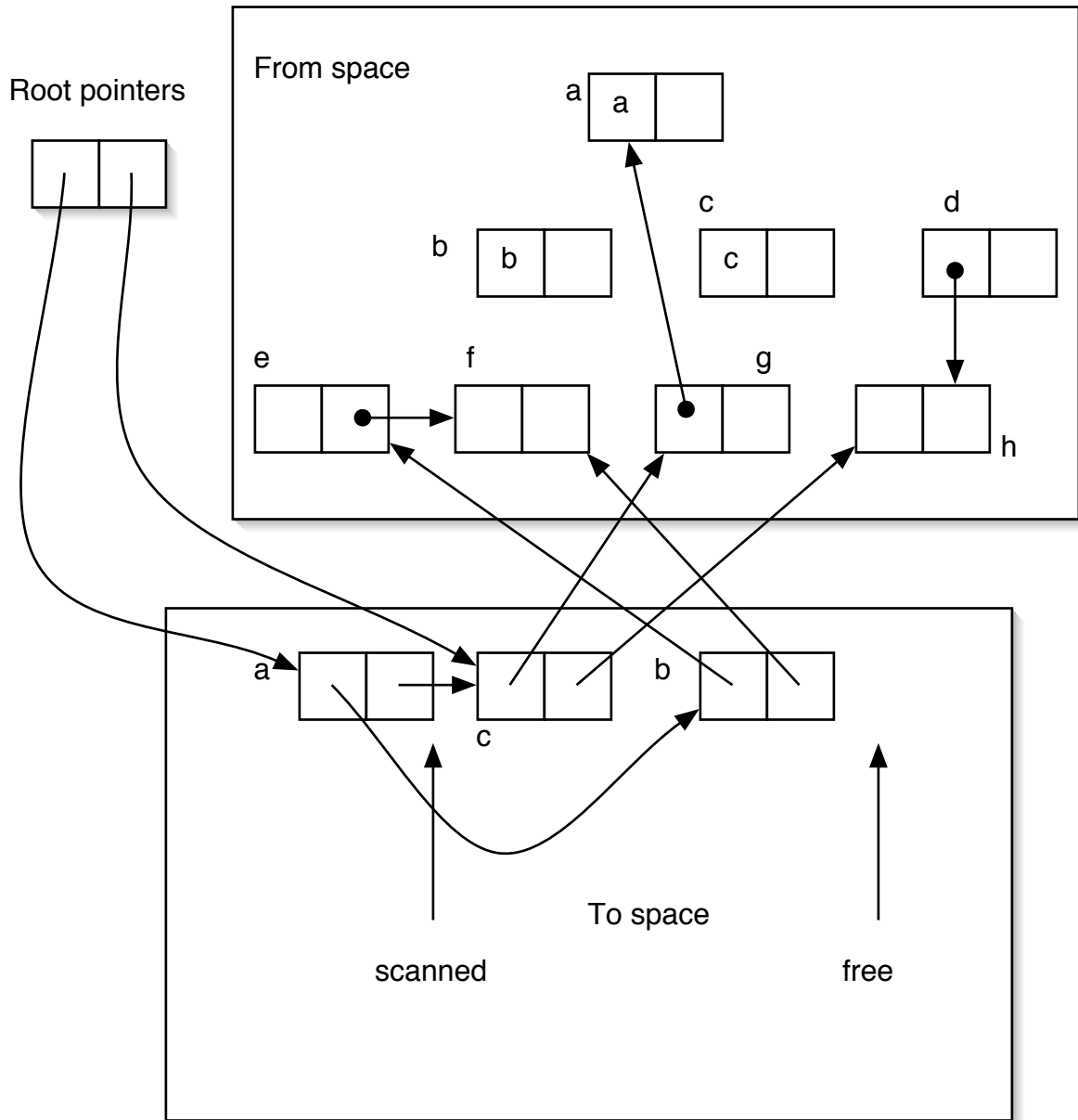


From space

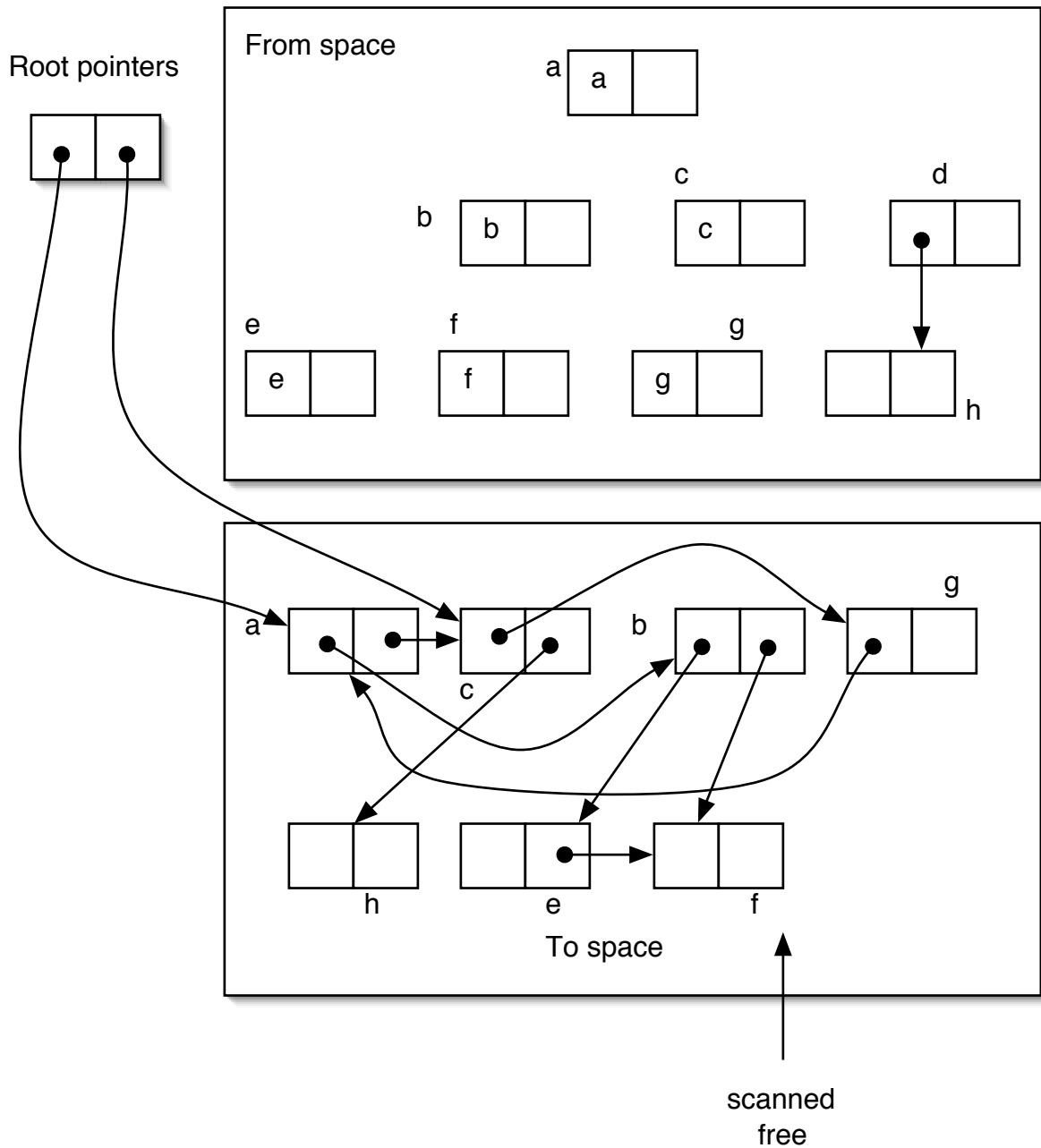


Answer: To reduce the number of lines in the graph, I use labels in the from space to show the forwarding pointers for cells that have been relocated to the to space.

After three cells have been moved:



After all cells have been moved:



3. CTL Model checking of a Microwave oven model.

For each of the following formulas, give an English description of the formula, and the states of the following model where the formula is true.

(a) **AF** *Heat*

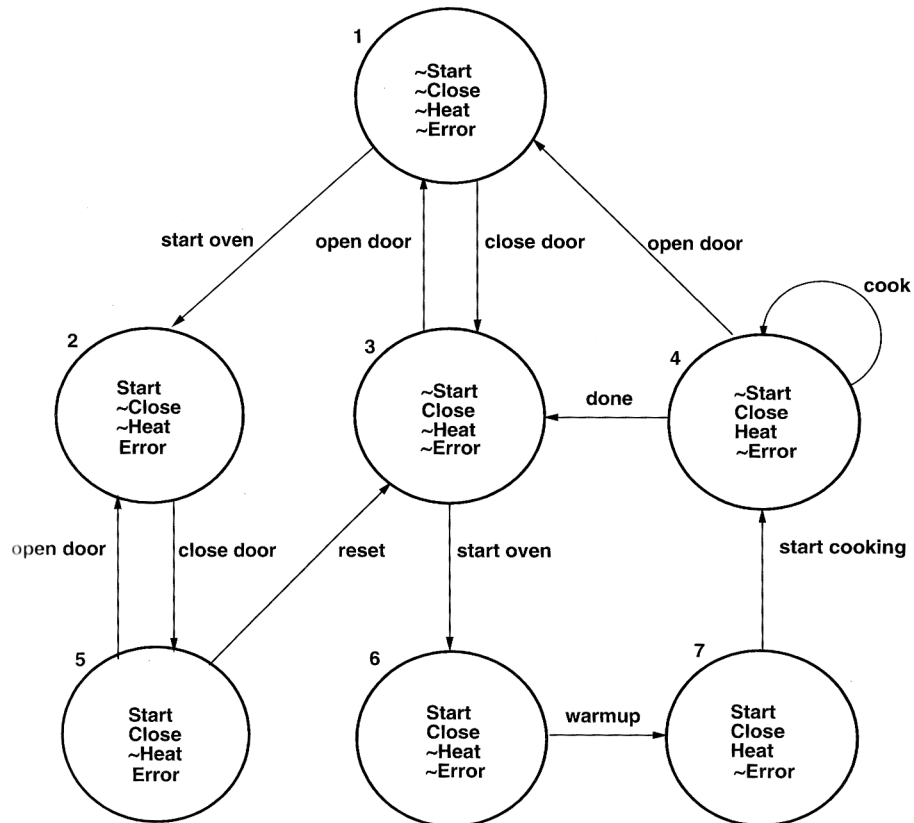
Answer: It is always the case that we reach a state where *Heat* is true. States: 6, 7 and 4.

(b) $Start \rightarrow \mathbf{AF} Heat$

Answer: If $Start$ is true, we eventually reach a state where $Heat$ is true. States: 1, 3, 4, 6, 7.

(c) $\mathbf{AG} (Start \rightarrow \mathbf{AF} Heat)$

Answer: It is forever more the case that if $Start$ is true, we eventually reach a state where $Heat$ is true. States: nowhere



4. Points to analysis:

(a) Apply Steenegaard's, Andersen's and Das's points-to analysis to the following set of statements:

```

p = &a
q = &b
a = &x
a = &y
b = &y
b = &z
  
```

For each form of analysis, give the points-to relations (e.g., what variables does the analysis think can point to what other variables).

Answer:

Analysis	p	q	a	b
Steenegaard	a	b	x,y,z	x,y,z
Andersen	a	b	x,y	y,z
Das	a	b	x,y	y,z

(b) Give the changes for each analysis from adding the statement:

`q = p`

Answer:

Analysis	p	q	a	b
Steenegaard	a,b	a,b	x,y,z	x,y,z
Andersen	a	a,b	x,y	y,z
Das	a	a,b	x,y,z	x,y,z

5. Class Hierarchy Analysis (CHA) and Rapid Type Analysis (RTA):

(a) Briefly explain CHA and RTA. Which is more precise?

Answer: They are both forms of analysis used in OO programs to figure out things such as which method implementation might be invoked by a call to `x.foo()`.

CHA just uses the declared subtype hierarchy (e.g., class B extends class A) and the types of the variables (e.g., x is a reference of to an object that is of type A).

RTA analysis also takes into account which objects have been created.

(b) Given an example of a situation where the two analysis techniques differ.

Answer: For the following program, RTA will determine that invoking the main method of class B will print 42, while CHA will be unable to statically resolve what value is printed.

```
class A {
    int f() {
return 42;
}
}
class B extends A {
    int f() {
return 17;
}
    public static main(String args[]) {
A a = new A();
System.out.println(a.f());
}
}
```

(c) What are some possible applications of the information generated?

Answer: Resolving method dispatch, which allows more efficient method invocation or inlining. Also, by determining which methods can't possibly be invoked you can perform application extraction.

(d) What are some of the reasons why or circumstances under which the less precise analysis would be preferred or required.

Answer: RTA requires knowledge of the entry point of the application and of all method invocations. If you don't know the entry point, or methods might be invoked through reflection in a way that the compiler cannot track, CHA should be used instead.

6. SSA form

Give an example of a code fragment with one assignment to x (plus an initial value for x) such that converting the code fragment to SSA introduces 3 phi nodes for x. Show both the original code fragment and the code fragment in SSA form. You may give the code fragments in either textual form or as a control flow graph.

Answer: Before:

```
x = 0
if (...)
  if (...)
    if (...)
      x = 1
```

After:

```
x0 = 0
if (...) {
  if (...) {
    if (...)
      x1 = 1
    x2 = phi(x1, x0)
  }
  x3 = phi(x2, x0)
}
x4 = phi(x3, x0)
```