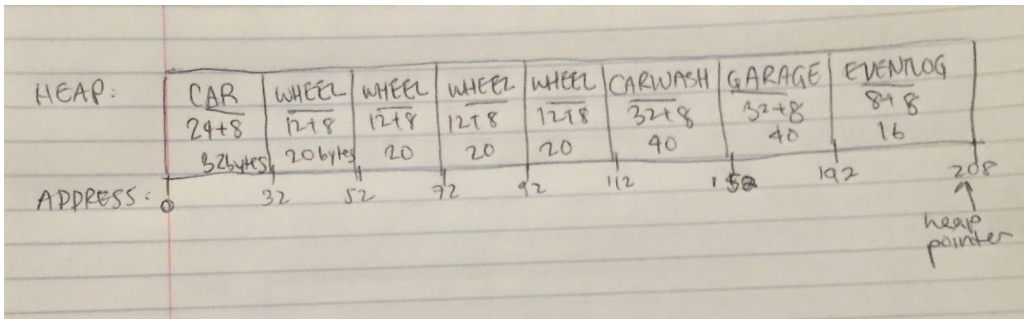
Kaitlin Gu
Cory Plock
Programming Languages

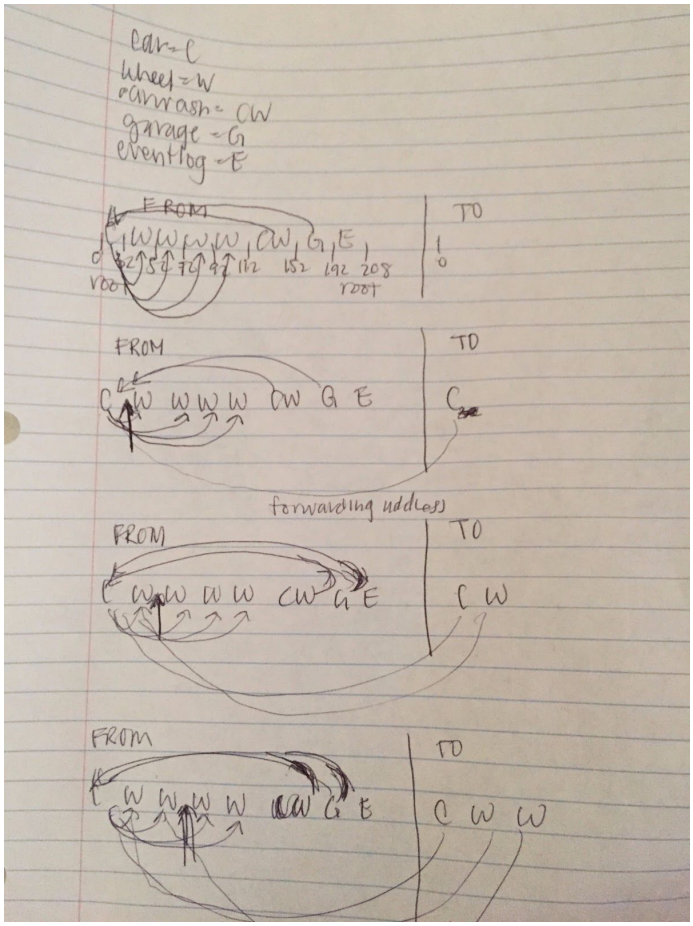
Homework #4

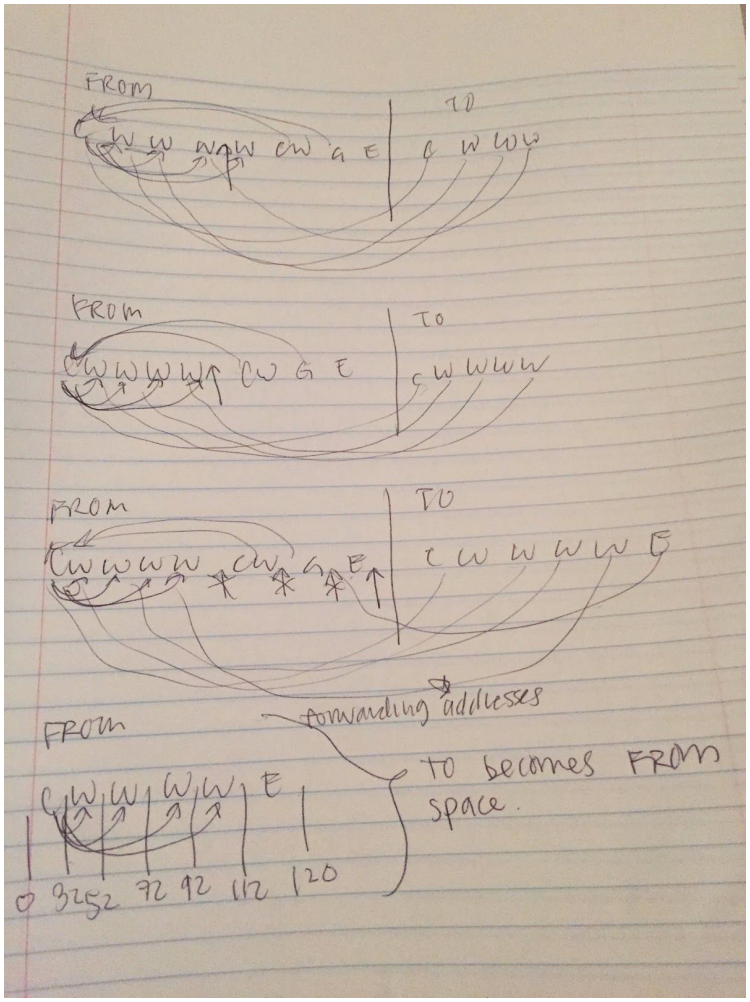
1. Garbage Collection

1.



2.





**Heap pointer is now at addr. 120

2. Prolog (See .pl files)

3. Prolog performances

1. Reordered facts:

```
foo(roberta).
foo(ashwin).

hello(roberta).
hello(brock).
hello(john).

world(roberta).
world(ashwin).
```

2. The original ordering of the facts would cause the interpreter to backtrace from the goal(X) query. This is because the subgoals are assessed in order so although roberta satisfies both subgoals it is not addressed first. Therefore, by allowing roberta to be assessed first the subgoals would all be satisfied without a backtrace. Because a backtrace is prevented, there are fewer steps to the goal and thus, the execution time is faster. This is shown by the traces:

   Original trace:

```
goal(X).
Call: (7) goal(_G2816) ? creep
Call: (8) sub1(_G2816) ? creep
Call: (9) foo(_G2816) ? creep
Exit: (9) foo(ashwin) ? creep
Exit: (8) sub1(ashwin) ? creep
Call: (8) sub2(ashwin) ? creep
Call: (9) hello(ashwin) ? creep
Fail: (9) hello(ashwin) ? creep
Fail: (8) sub2(ashwin) ? creep
Redo: (9) foo(_G2816) ? creep
Exit: (9) foo(roberta) ? creep
Exit: (8) sub1(roberta) ? creep
Call: (8) sub2(roberta) ? creep
Call: (9) hello(roberta) ? creep
Exit: (9) hello(roberta) ? creep
Call: (9) world(roberta) ? creep
Exit: (9) world(roberta) ? creep
Exit: (8) sub2(roberta) ? creep
Exit: (7) goal(roberta) ? creep
X = roberta.
```

   Modified trace:

```
goal(X).
Call: (7) goal(_G12556) ? creep
Call: (8) sub1(_G12556) ? creep
Call: (9) foo(_G12556) ? creep
Exit: (9) foo(roberta) ? creep
Exit: (8) sub1(roberta) ? creep
Call: (8) sub2(roberta) ? creep
Call: (9) hello(roberta) ? creep
Exit: (9) hello(roberta) ? creep
Call: (9) world(roberta) ? creep
```

```
Exit: (9) world(roberta) ? creep
Exit: (8) sub2(roberta) ? creep
Exit: (7) goal(roberta) ? creep
X = roberta
```

** General explanation for 3 and 4: The (!) cut operator is used to prevent backtracking so all unifications up to the point of the ! operator are set in stone. So any subgoal to the left of the ! operator will never be evaluated more than once.

3. After the interpreter tries hello(ashwin) it fails, which means that the sub2(X) also fails. However, since the interpreter can not try and go back to try unifying sub1(X) with roberta due to the (!) cut operator, the subgoal sub1(X) can never be satisfied causing goal(X) to fail. This behavior is demonstrated by the trace:

```
Call: (7) goal(_G5249177) ? creep
Call: (8) sub1(_G5249177) ? creep
Call: (9) foo(_G5249177) ? creep
Exit: (9) foo(ashwin) ? creep
Exit: (8) sub1(ashwin) ? creep
Call: (8) sub2(ashwin) ? creep
Call: (9) hello(ashwin) ? creep
Fail: (9) hello(ashwin) ? creep
Fail: (8) sub2(ashwin) ? creep
Fail: (7) goal(_G5249177) ? creep
False.
```

4. The unification of X to ashwin fails for hello(X) causing the program to backtrack. Namely, since the unification to roberta is the only one that satisfies hello(X) and foo(X), roberta will already be unified to X by the time the interpreter reaches the cut operator. This behavior is demonstrated by the trace:

```
Call: (7) goal(_G905) ? creep
Call: (8) sub1(_G905) ? creep
Call: (9) foo(_G905) ? creep
Exit: (9) foo(ashwin) ? creep
Exit: (8) sub1(ashwin) ? creep
Call: (8) sub2(ashwin) ? creep
Call: (9) hello(ashwin) ? creep
Fail: (9) hello(ashwin) ? creep
Fail: (8) sub2(ashwin) ? creep
Redo: (9) foo(_G905) ? creep
Exit: (9) foo(roberta) ? creep
```

```
Exit: (8) sub1(roberta) ? creep
Call: (8) sub2(roberta) ? creep
Call: (9) hello(roberta) ? creep
Exit: (9) hello(roberta) ? creep
Call: (9) world(roberta) ? creep
Exit: (9) world(roberta) ? creep
Exit: (8) sub2(roberta) ? creep
Exit: (7) goal(roberta) ? creep
X = roberta.
```

4. Unification

1. Unification isn’t possible: d and c are different functors
2. X = 4,
   Y = b(3,1,Y) unless caught by occurs check (infinite recursion)
3. Unification is possible:
   X = 4
   A = 2
   B = 7
   D = C
4. Unification is not possible: A is bound to 2 then A is bound to 7 and 7 and 2 do not unify. A cannot be bound to different values.
5. Unification is not possible: 2 and 8 do not unify.
6. Unification is possible:
   X = e(f(6, 2), g(8, 1)).
7. Unification is possible:
   X = f(6,2)
   8 = 8
8. Unification is not possible because the functors have different arity / number of arguments.
9. Unification is possible:
   X = 1
   Y = 2