Garbage Collection

- Compacting precise garbage collector
- Decent performance
- Straight forward extension to generational GC. (If you use tagbits for tag words).

Integer Allocation

- As high as 95% of Sudoku2 are integer allocations.
- 32-bits versus :
  - 64-bits for pointer
  - 64-bits for tag word
  - 64-bits for integer value
- Garbage Collection is slow?

Host Language Interaction

```
case ARRAY_INS :
    val init = pop from operand stack
    val length = pop from operand stack
    val array = halloc(8 + 8 + 8 * length.value)
                    // Tag + Len + Slots ..
    array[0] = ARRAY_TAG
    array[1] = length.value
    array[2 to 2 + length.value] = init
    push array to operand stack
```

Bytecode vs AST Interpreter

```
case SET_EXP:{
  SetExp* e = (SetExp*)exp;
  EvalObj* v = eval_exp(genv, env, e->exp);
  EnvEntry* var = lookup_var(env, e->name);
  var->value = v;
  return nullobj;
}
```

Long Living Objects

- Allocate very large array at the beginning of the program.
- Array is live for duration of program.

Primitive Arrays

- Large array of 64-bit pointers to 128-bit heap structs.
- Must scan the array.

Linked Lists

- Many long linked lists for duration of program.
- Your garbage collector does a breadth-first traversal through object graph.
- Linked lists are interleaved with each other.
- There are depth-first GC algorithms.

Performance Complexity

- Scavenging garbage collection is time is proportional to number of live objects.
- Not true for all garbage collection techniques:
  - Reference counting overhead generally proportional to number of allocations and assignments.
  - Naive Mark/Sweep generally proportional to size of heap.

Foreign Function Interface

- C holds onto a pointer to a Feeny object.
- Two problems:
  - How do you know whether that pointer is a root?
  - How do you relocate that pointer?

Tagged Primitives

- Way less memory allocated in total.
- Arithmetic operators are also much less expensive (though currently still dominated by lookup).

Multiply

$$f(x \cdot y) = f(x) \cdot f^{-1}(f(y))$$

$$f(x \cdot y) = f^{-1}(f(x) \cdot f(y))$$

Comparison Operator

$$g(x < y) = 1 \text{ if } x < y$$
$$0 \text{ otherwise}$$

$$f(g(x < y)) = 0 \text{ if } x < y$$
$$2 \text{ otherwise}$$

$$f(g(x < y)) = (g(x < y) \text{ xor } 1) * 2$$

Comparison Operator: Taking the Dual

$g(x < y) = 1$ if $x < y$

0 otherwise

$f(g(x < y)) = 0$ if $x < y$

2 otherwise

$f(g(x < y)) = 2 * g(x >= y)$