Jones and Kelly

Sample C code with Jones and Kelly method

\[(lo, hi) = \text{lookup}(p)\]
assert \((lo < p < hi)\)

\[(lo, hi) = \text{lookup}(p)\]
assert\((lo \leq p + r \leq hi)\)

\[q = p + r\]

Evaluation Criteria for Jones and Kelly Method

1. Speed Overhead: How much slower will a C program by Jones and Kelly Method run? – 10x Slower
2. Memory Overhead: Usually not a problem
3. Compilation Speed: Not important
4. Effectiveness:
   a. Temporal Memory Safety Issues
   ```c
   int *p = malloc(sizeof(int));
   r = p;
   free(p);
   q = malloc(sizeof(int));
   *r = 5;
   ```
   b. Intra Structure Overflows
struct foo
{
  int a;
  float b;
}
func1()
{
  struct foo *p = malloc(sizeof(int) + sizeof(float))
  q = p -> a
  q = q + 1
  *q = 0
}

5. False Positives :
int A[10]
insert(A,10)
int * p = A
for (i = 0; i < 10; i++)
{
  (lo,hi) = lookup(p);
  assert(lo <= p <= hi);
  *p = 0;
  (lo,hi) = lookup(p);
  assert(lo <= p+1 <= hi);
  p = p + 1;
}
6. Separate Compilation Support - Jones and Kelly works great

7. Linking Transformed and Untransformed Code

8. Issuing recording allocations in untransformed code

9. No Code change - Jones and Kelly requires no code changes

Point 1, 4, 5, 6, 8, 9 are the major design requirements.

**Baggy Bounds Checking**
- Similar to Jones and Kelly
  a. Bounds kept for objects
  b. Efficient Lookups
  c. Efficient Checks
  d. Other Optimizations

Main trick is enable lookups via Hashing

Bounds = BNDS \[2^{\log_2 (\text{Size of memory allocated around } p)}\] bytes = 256 Mbs

\[\log_2 (\text{Size of memory allocated around } p)\]

Bounds = BNDS \[| P / 16|\] bytes
- Memory all of form \(2^x\)
- Also aligned to \(2^x\)

\[
\text{low} = p \& ( \left( 1 << \text{BNDS} \mid p \ggg 4 \right) - 1 )
\]