

# Meaningful Type Names as a Basis for Object Lifetime Prediction

Jeremy Singer   Mikel Luján   Ian Watson

School of Computer Science, University of Manchester

# Natural Language Information

- Not normally used in program analysis
  - ▶ imprecision
  - ▶ inaccuracy
  - ▶ incomprehensibility

# Java Type Names

- Follow a standard form
- clear word separation
- (*adjective*)\* (*noun*)+
- Clues to object lifetimes
- hypothesis: *type name suffix is an indicator of object lifetime*

# Our aim

- predict object lifetimes ahead-of-time, from Java type names
- (particularly *suffixes*)
- use predictions to optimization garbage collection

# What is Garbage Collection?

- automatic heap-memory management
- many different algorithms, parameters, etc
- We focus on generational GC
  - ▶ hypothesis
  - ▶ fridge analogy

# Generational GC

- optimized GC for *short-lived* objects
- split heap into several *spaces*
- initially allocate all objects in *nursery*
  - ▶ small, frequently collected
- promote survivors to *mature*
  - ▶ large, infrequently collected

# Optimizations to Generational GC

- optimized GC for *short-lived* and *long-lived* objects
- predict object lifetimes at or before their allocation
- allocate long-lived objects directly into mature space
- mark short-lived objects and defer promotion from nursery

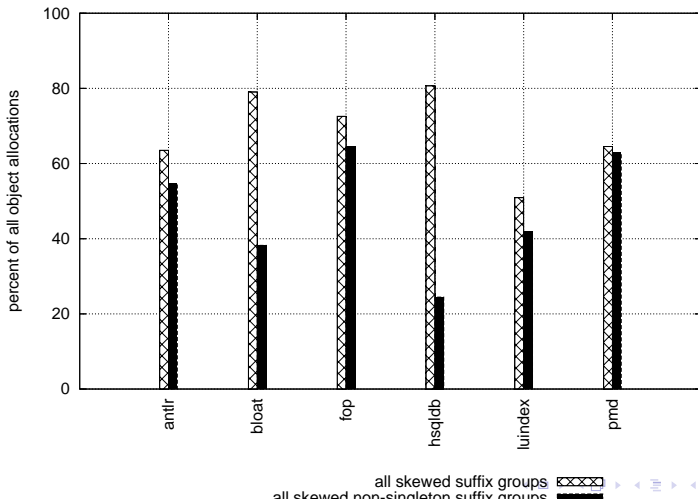
# Object Lifetime Prediction

- offline—simply look at profile run of program
- online—for previously unseen programs
  - ▶ use metrics/patterns to identify typical object ‘shapes’
  - ▶ analyse *types* of objects

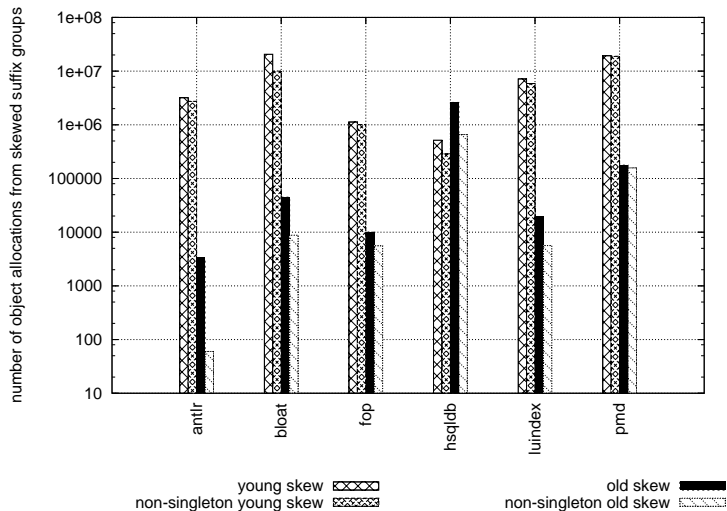
# Some Terminology

- suffix group: all objects with same type name suffix
- skewed suffix group: 95% of objects have same lifetime
- young-skew: 95% of objects are short-lived
- old-skew: 95% of objects are long-lived

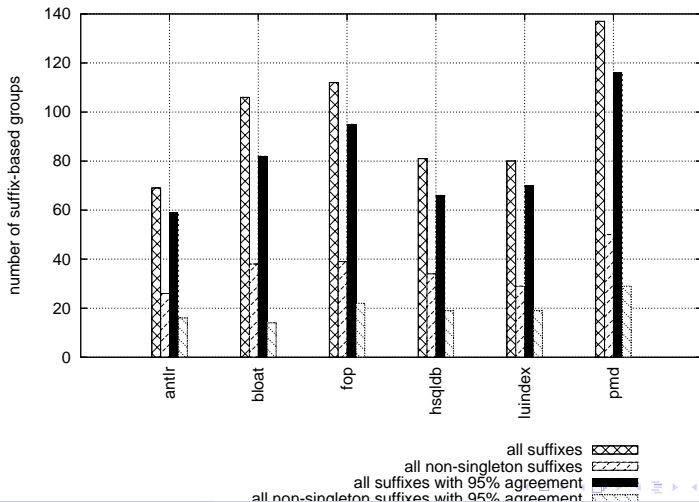
# Are skewed suffix groups common? (yes)



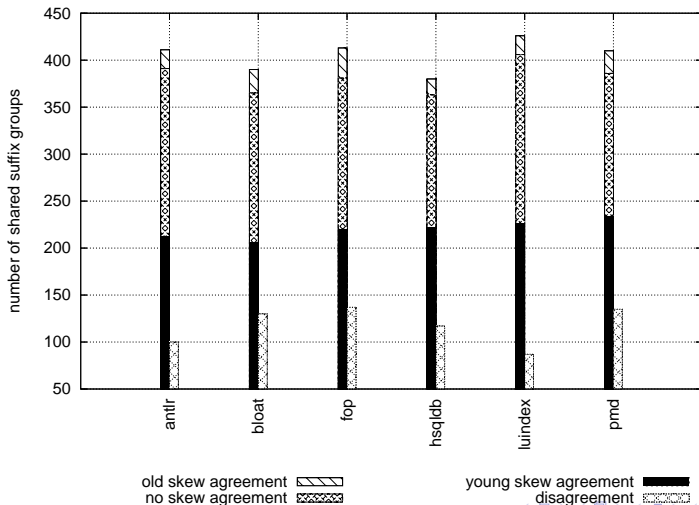
# Young or old more prolific? (young)



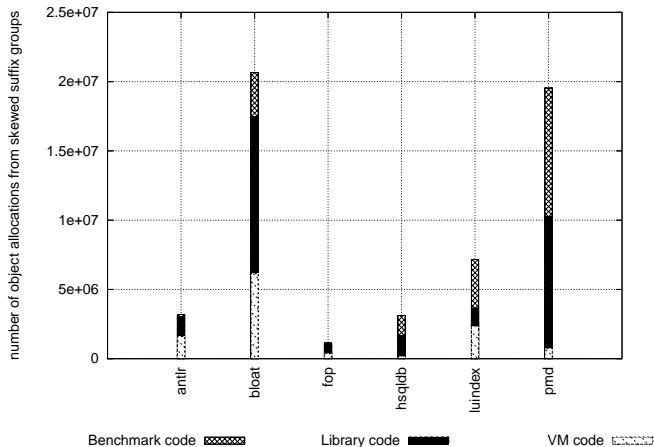
# Are classes in skew group consistent? (mostly)



# Do suffix group trends carry across programs? (yes)



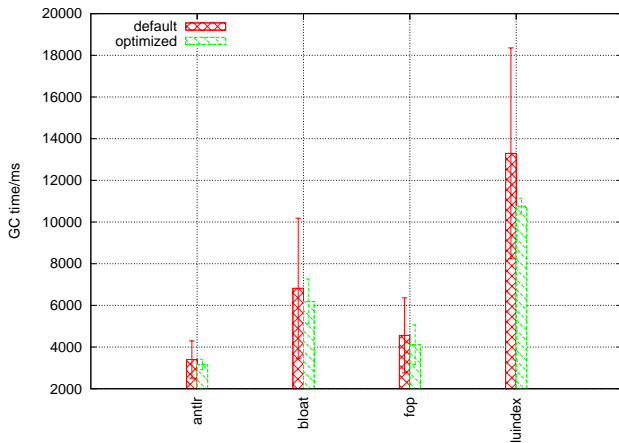
# Where do these suffixes originate? (everywhere!)



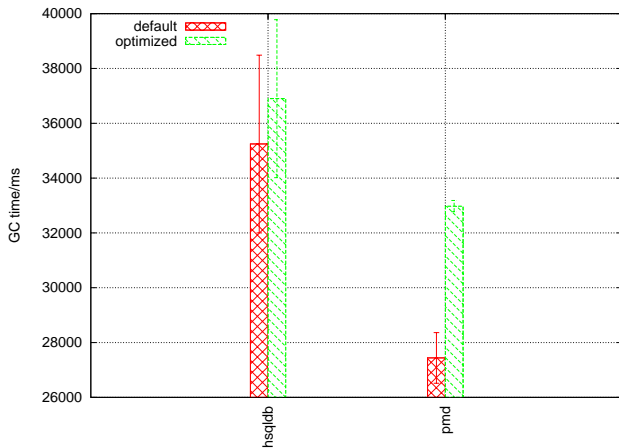
# Evaluation

- separate short-lived and long-lived optimizations
- self-prediction versus true-prediction
- GC time versus total time
- inherent noise in system

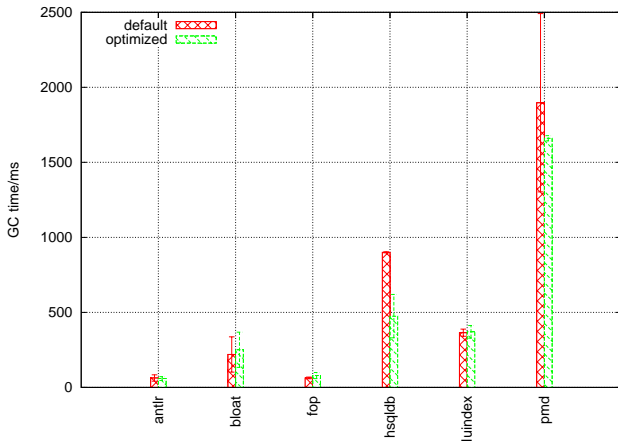
# Short-lived, self-prediction, GC times



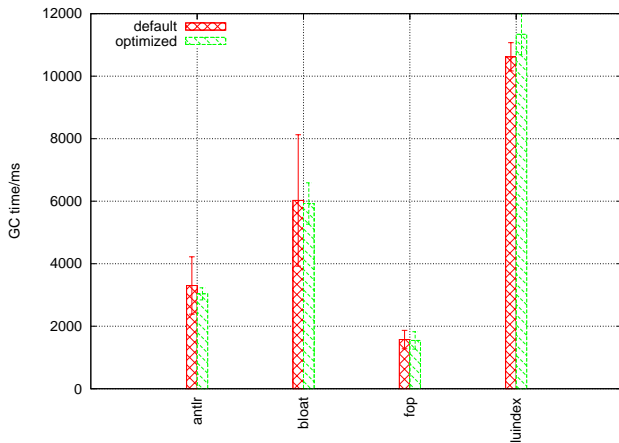
# Short-lived, self-prediction, GC times



# Long-lived, self-prediction, GC times



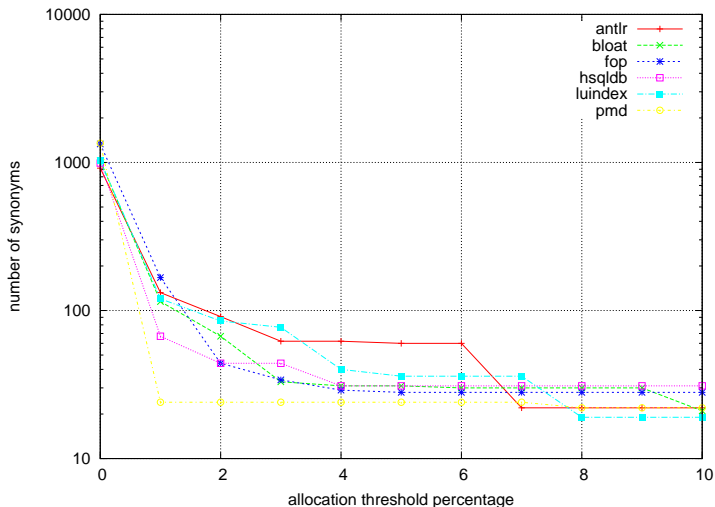
# Short-lived, true-prediction, GC times



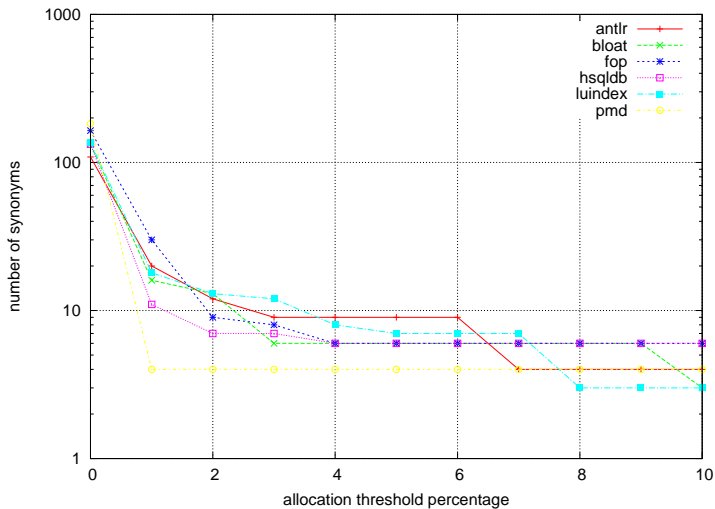
# Tackling Natural Language Problems

- synonymy: different words, same meaning
- homonymy: same word, different meanings
- analysis using WordNet

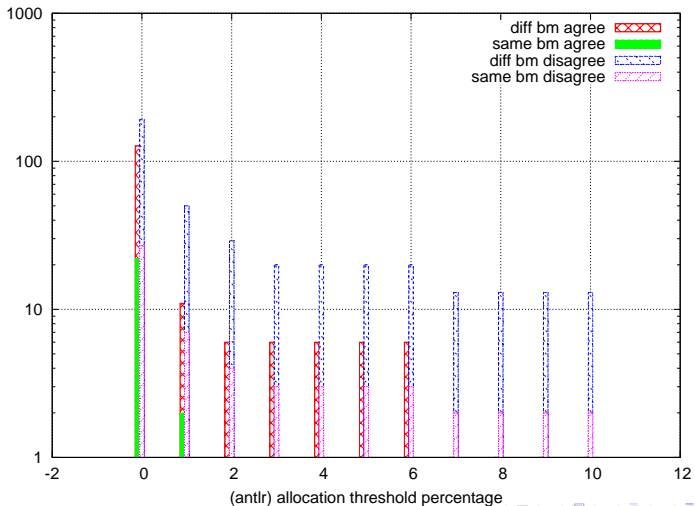
# Upper bound on synonyms



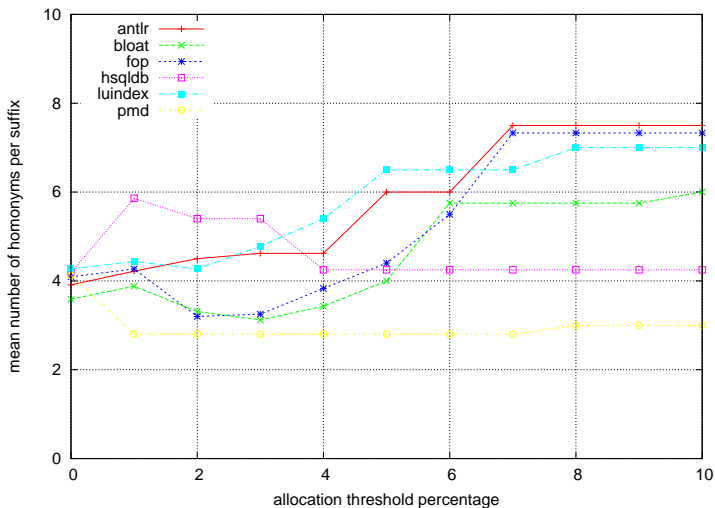
# Actual synonyms



# Do synonyms mean missed optimizations



# Potential Homonyms



# Are syn / hom problems serious?

- WordNet is *conservative*
- Actual synonymy / homonymy much less
- Are programming languages better than natural language?

# Limitations

- non-standard coding conventions
- random variable names (girlfriends!)
- international variable names (Klingon!)
- VM and libraries should be standard

# Conclusions

- GC is ideal area for Machine Learning / Data Mining / Information Retrieval
- Plenty of low-hanging fruit
- efficient runtime techniques required
- type name suffix analysis is a promising starter...