# Progress Toward Fast Finite Sets and Maps in ACL2

Grant Jurgensen

Kestrel Institute

May, 2025

# Outline

# Introduction

# Goals

- Want finite sets and maps that are:
  - Fast.
  - Functional and persistent.
  - Verified (no new raw lisp code, trust tags).
  - Logically convenient.

# Existing Libraries

- `osets` and `omaps`
  - **Pros**:
    - Functional and persistent.
    - Unique representation means set equivalence is regular equality.
    - Generally doesn't expose internal implementation (users don't have to deal with `<<`).
  - **Cons**:
    - Inefficient.
- What about fast `alists`, `stobj` hash tables, or arrays?
  - Generally limited in usage (e.g. the "fast alist disipline", `stobj` single-threadeness).
  - Overhead of maintaining logical twin to raw lisp component.
  - May require global object or name.
- What about `bitsets`?
  - Limited to small sets of natural numbers.

# Attempt #1: Ordered Sets with Treaps

# Binary Search Trees

- What about binary search trees (BSTs)?
  - A natural evolution from `osets`.
  - **Challenge**: achieving a unique representation (without degrading performance).
    - Self-balancing BSTs (e.g. red-black trees, AVL trees) are generally sensitive to the order of insertion/deletion.

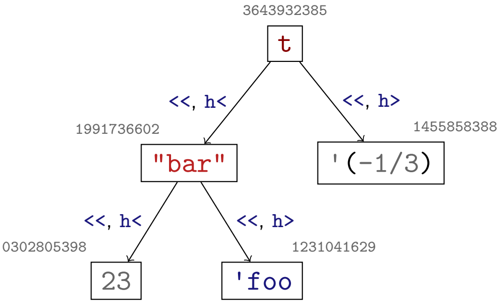
# Treaps

- A treap (= "tree" + "heap") is a BST with an additional max heap property.
- If the BST and heap orders are total, the tree must have a unique representation.
- If the BST and heap orders are generally uncorrelated, the tree will be practically balanced.
- The BST order can use `<<`. The heap order can use a new order, `h<`, based on hash values.

# Hash Function

- Based on `check-sum-obj`.
- Implementations: FNV-1a, Jenkins one-at-a-time.
- Likely could be optimized much further
- `h<` compares two objects' hash values. If they are the same, it falls back to `<<`.

# treesets



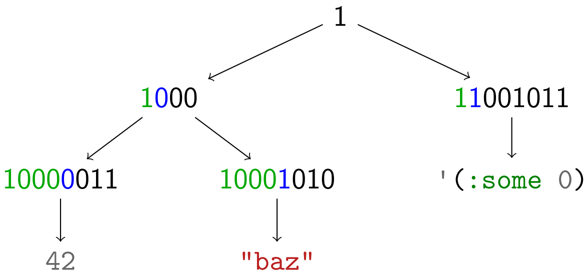

- Implemented in `books/kestrel/treeset`.
- **But wait!** There are faster data structures!

# Attempt #2: Unordered Sets with Little-Endian Patricia Trees

# Patricia Trees

- Hash values are 32-bit unsigned `fixnums`.
  - Natural numbers may be viewed as bit strings. Therefore, we can consider the use of tries.
- Patricia trees (AKA binary radix trees), are a compressed form of trie.
  - They can be very fast on fixed-size integers due to "bit-twiddling" tricks (see Okasaki and Gill, *"Fast Mergeable Integer Maps"*).
  - The non-leaf structure is naturally unique.
  - No rebalancing or tree rotations necessary.
- Hash array mapped tries (HAMTs) are faster, but require arrays.

# Patricia Tree Example




- Example with 8-bit hashes; previous prefix in green, branching bit in blue.
- Leaf buckets are implemented as `osets`.
- Verification is a WIP.

# Conclusion

# Initial Experiments

Tests on sets of size 100,000, run 10,000 times (on SBCL).

**Random Elements**

| | osets | treaps | patricia trees |
|---|---|---|---|
| Membership | 1.92 | 0.02 | 0.03 |
| Insertion | 6.80 | 0.40 | 0.03 |
| Deletion | 25.11 | 0.07 | 0.03 |

**Consecutive Naturals**

| | osets | treaps | patricia trees |
|---|---|---|---|
| Membership | 1.92 | 0.02 | 0.02 |
| Insertion | 12.04 | 0.07 | 0.02 |
| Deletion | 16.52 | 0.02 | 0.03 |