Confluently Persistent Tries

Eric Price, MIT
joint work with
Erik Demaine, MIT
Stefan Langerman,
Université Libre de Bruxelles
Version Control

- Maintain multiple versions of a directory structure
  - CVS, Subversion, Git, Mercurial, ...
- Efficiently branch and merge in changes in a trie
  - Confluent Persistence
Persistence Spectrum

- Partial persistence (version line)
  - Read old versions
- Full persistence (version tree)
  - Branch tree
- Confluent persistence (version DAG)
  - *Meld* operation between branches
- Functional (no mutation)
General Persistence

- Partial & Full persistence
  - Constant overhead for pointer-machine

- Confluent persistence
  - Fiat & Kaplan 2003
  - space $\Omega(lg P)$, $P = \#$ paths in version DAG
  - Possibly exponential number of nodes
  - Linear bits to represent pointer
Problem

- Replace pointers by *fingers* maintained by DS
- Maintain a trie
  - Arbitrary degree
  - Relative labels
- Operations
  - Insert/delete/modify at a finger
  - Copy subtree between fingers
  - Finger movement
  - Duplicate finger
Functional Structure: Path Copying

• Basic Method: Path Copying
  – $O(d \lg \Delta)$ per operation, $d = \text{depth}$, $\Delta = \text{degree}$
  – Whenever all pointers point into subtree.

![Diagram of two trees showing path copying process.](image)
Functional Structure: Zipper

• More Advanced: Zipper
  – Used in Haskell (xmonad)
  – *Finger* to an everted element
  – $O(\lg \Delta)$ per operation including finger move
Our Results

<table>
<thead>
<tr>
<th>Method</th>
<th>Finger Movement</th>
<th>Modifications</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Space</td>
</tr>
<tr>
<td>Path Copying</td>
<td>lg Δ</td>
<td>0</td>
</tr>
<tr>
<td>Local (functional)</td>
<td>lg Δ</td>
<td>lg Δ</td>
</tr>
<tr>
<td>Global (functional)</td>
<td>lg Δ</td>
<td>0</td>
</tr>
<tr>
<td>Local (confluent)</td>
<td>lg lg Δ</td>
<td>lg lg Δ</td>
</tr>
<tr>
<td>Global (confluent)</td>
<td>lg lg Δ</td>
<td>0</td>
</tr>
</tbody>
</table>

- Fully Persistent Hash Tables: $O(lg \ lg N)$
  - Weight-balanced fully persistent hash tables
Globally Balanced Functional Trie

- **Link-Cut trees**
  - Store “heavy” paths in weight-balanced BSTs

- **Finger movement needs fast predecessor and successor**
Finger Movement

- Need fast predecessor and successor
- Finger is deque of deques of ancestors in alternating directions
  - Can find preceding ancestor
  - Use functional catenable deques with $O(1)$ operation
- Predecessor is preceding ancestor or rightmost descendant of left child
  - Nodes store deque of left/right spines
Functional vs. Confluent Persistent

- Functional structures use BSTs on the children
- Can replace with fully persistent (not confluent) hash tables
  - Improve $O(\lg \lg UM)$ arrays in [Dietz 89] for dynamic resizing
  - Rehash occasionally to keep $M$ small, avoid collisions
  - Improves $O(\lg \Delta)$ finger movement to $O(\lg \lg \Delta)$
Weight-Balanced Hash Tables

- Link-cut trees use weight-balanced BSTs
  - $\text{depth}_v = O(\lg(1 / w_v))$, $\sum w_v = 1$

- Want structure with $O(\lg \lg 1/w_v)$ lookups, $O(\lg 1/w_v)$ updates
  - Trick from BST-based working set structure
  - Triply exponentially sized hash tables (16, 65536, 115792089237316195423570985008687907853269984665640564039457584007913129639936)
  - Weight-balanced BSTs to shift between tables
## Conclusion

<table>
<thead>
<tr>
<th>Method</th>
<th>Finger Movement</th>
<th>Modifications</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Space</td>
</tr>
<tr>
<td>Path Copying</td>
<td>lg $\Delta$</td>
<td>0</td>
</tr>
<tr>
<td>Local (functional)</td>
<td>lg $\Delta$</td>
<td>lg $\Delta$</td>
</tr>
<tr>
<td>Global (functional)</td>
<td>lg $\Delta$</td>
<td>0</td>
</tr>
<tr>
<td>Local (confluent)</td>
<td>lg lg $\Delta$</td>
<td>lg lg $\Delta$</td>
</tr>
<tr>
<td>Global (confluent)</td>
<td>lg lg $\Delta$</td>
<td>0</td>
</tr>
</tbody>
</table>