Reasoning about B+ Trees with Operational Semantics and Separation Logic

Hayo Thielecke
(joint work with Alan Sexton)

University of Birmingham, UK
Introduction

B+ tree

- Balanced ordered search tree, $O(\log n)$ operations
- Widely used for efficient search in databases

Operational semantics: Abstract Machine

- Formalizes operations on B+ trees, abstractly
- In the style of SECD machine or Krivine’s machine

Separation logic

- Very expressive for inductive data structures with pointers
- Here: ordinary predicate logic extended with $\ast$ and $\mapsto$
- No Hoare logic $\{P\} \ c \ \{Q\}$
B+ tree visually

\[ L(e_1 \ldots e_n ; f) \]

\[ I(d_1 \ldots d_{n-1} ; q_1 \ldots q_n) \]
Operational semantics: abstract machine

Landin’s SECD machine (1964)

Configurations: Stack, Environment, Control, Dump

\[ \langle s, e, c, d \rangle \leadsto \langle s', e', c', d' \rangle \]

Transitions by pattern matching (e.g., stack or control empty).

Our machine: insertion transitions

\[ \langle C, r, \pi, \sigma \rangle \leadsto \ldots \]

C: command, e.g., Insert \( (e) \), S, D \( (k, q) \)

r: root of current subtree

\( \pi \): stack, used for tree traversal

\( \sigma \): store, holds nodes of the tree, \( \approx \) heap in Sep Logic
Separation logic connectives

\[ \sigma \models p \rightarrow N \text{ iff } \sigma = \{ p \rightarrow N \} \text{ for a node } N \text{ (leaf or internal)} \]

\[ \sigma \models Q_0 \ast Q_1 \text{ iff } \sigma = \sigma_0 \cup \sigma_1 \text{ where } \sigma_0 \cap \sigma_1 = \emptyset \]

\[ \text{and } \sigma_0 \models Q_0 \text{ and } \sigma_1 \models Q_1 \]

\[ \sigma \models Q_0 \land Q_1 \text{ iff } \sigma \models Q_0 \text{ and } \sigma \models Q_1 \]

\[ \sigma \models Q_0 \lor Q_1 \text{ iff } \sigma \models Q_0 \text{ or } \sigma \models Q_1 \]

\[ \sigma \models \text{true} \text{ iff } \sigma \text{ is any store} \]

\[ \sigma \models \text{emp} \text{ iff } \text{dom}(\sigma) = \emptyset \]

Local reasoning: footprint and frame:

\[
\begin{array}{c}
\{ P \} \ c \ \{ Q \} \\
\{ P \ast R \} \ c \ \{ Q \ast R \}
\end{array}
\]
B+ trees in Separation Logic

Predicate describes data structure
\[ \sigma \models Btree_h(r, S, a, z, n) \]
- height \( h \)
- \( r \) points to root node of tree
- set of entries \( S \)
- first leaf at address \( a \), last leaf node points to \( z \)
- size of root is at least \( n \)

Induction on height: Leaf case

\[ Btree_1(r, S, a, z, n) \iff \exists e_1, \ldots, e_n. n \leq MaxN \]
\[ \land \quad r \mapsto L(e_1 \ldots e_n ; z) \]
\[ \land \quad S = \{e_1, \ldots, e_n\} \land a = r \land e_1 \sqsubseteq \cdots \sqsubseteq e_n \]
\(B^+\) tree continued: internal nodes

\[
B_{h+1}(r, S, a, z, n) \iff \exists d_1, \ldots, d_{n-1}, q_1 \ldots q_n, m_1, \ldots, m_n \\
\hspace{1cm} \land (r \mapsto I(d_1 \ldots d_{n-1}; q_1 \ldots q_n) \\
\hspace{1cm} \land \ast B_{h}(q_1, S_1, a_1, a_2, m_1) \\
\hspace{1cm} \ast B_{h}(q_2, S_2, a_2, a_3, m_2) \\
\hspace{1cm} \ast \ldots \\
\hspace{1cm} \ast B_{h}(q_n, S_n, a_n, a_{n+1}, m_n)) \\
\hspace{1cm} \land a_1 = a \land a_{n+1} = z \\
\hspace{1cm} \land S = S_1 \cup \cdots \cup S_n \\
\hspace{1cm} \land (\forall j. 1 < j < n - 1 \Rightarrow d_j \sqsubseteq S_j \sqsubset d_{j+1}) \\
\hspace{1cm} \land (\forall j. 1 < j \leq n \Rightarrow \lceil \text{MaxN}/2 \rceil \leq m_j) \\
\hspace{1cm} \land (S_1 \sqsubset d_1) \\
\hspace{1cm} \land (d_{n-1} \sqsubseteq S_n)
\]
**B+ tree insertion**

- **Invariant:** balanced
- **Insertion** may split a node.
- **Parent node** may then split as well.
- Splitting ripples up the tree recursively.
- Insertion produces either a subtree or two subtrees.
- In the operational semantics: “single” or “double” commands

\[
\langle S, r, \pi, \sigma \rangle
\]

\[
\langle D(k, q), r, \pi, \sigma \rangle
\]
Insert Rules — Descending the tree and splitting leaf

\[
\langle \text{Insert}(a), r, \pi, \sigma \rangle \leadsto \langle \text{Insert}(a), p_i, (r, i) :: \pi, \sigma \rangle
\]

if \( \sigma(r) = I(d; p) \)

where \( i = \text{first}(d, \lambda x. x > \text{key}(a)) \)

\[
\langle \text{Insert}(a), r, \pi, \sigma \rangle \leadsto \langle \text{D}(k, q), r, \pi, \sigma \begin{bmatrix}
  r & \mapsto & \text{L}(e'; q), \\
  q & \mapsto & \text{L}(e''; f)
\end{bmatrix} \rangle
\]

where \( \sigma(r) = \text{L}(e; f) \)

and \( i = \text{first}(e, \lambda x. \text{key}(x) \geq \text{key}(a)) \)

and \( \langle e', k, e'' \rangle = \text{splitL}(i, a, e) \)

and \( q \notin \text{dom}(\sigma) \)
Insert Rules — Returning up the tree with S or D(k, q)

\[\langle S, r, (t, i) :: \pi, \sigma \rangle \rightsquigarrow \langle S, t, \pi, \sigma \rangle\]

\[\langle D(k, q), r, (t, i) :: \pi, \sigma \rangle \rightsquigarrow \langle S, t, \pi, \sigma[t \mapsto I(d'; p')] \rangle\]
if \(|p| < \text{MaxN}\)

where \(\sigma(t) = I(d; p)\)

and \(d' = \text{ins}(k, i, d)\)

and \(p' = \text{ins}(q, i + 1, p)\)

\[\langle D(k, q), r, (t, i) :: \pi, \sigma \rangle \rightsquigarrow \langle D(k', q'), t, \pi, \sigma \left[\begin{array}{l}
    t \mapsto I(d'; p'), \\
    q' \mapsto I(d''; p'')
\end{array}\right]\rangle\]

where \(\sigma(t) = I(d; p)\)

and \(\langle d', p', k', d'', p'' \rangle = \text{splitI}(i, k, q, d, p)\)

and \(q' \notin \text{dom}(\sigma)\)
Correctness of insertion Idea: *footprint frame* for the same stack $\pi$.

**Lemma**

Assume $\sigma \models \text{Btree}_h(r, S, a, z, n) \ast R$. Then either:

1. $\langle \text{Insert} (e), r, \pi, \sigma \rangle \leadsto \cdots \leadsto \langle S, r, \pi, \sigma' \rangle$ and 

   $\sigma' \models \text{Btree}_h(r, S + e, a, z, n) \ast R$

2. $\langle \text{Insert} (e), r, \pi, \sigma \rangle \leadsto \cdots \leadsto \langle \text{D} (k, q), r, \pi, \sigma' \rangle$ and 

   $\sigma' \models \text{Btree}_h(r, S_r, a, b, n') \ast \text{Btree}_h(q, S_q, b, z, n'') \ast R$

Moreover, $S_r \cup S_q = S + e$ and $S_r \sqsupset k \sqsubset S_q$. 
Proving Correctness of Insertion

$B \ast R$
Proving Correctness of Insertion

\[ r \mapsto I(d \ ; \ p) \ * \]
\[ B_1 * \cdots * B_{i-1} * \]
\[ B_i * \]
\[ B_{i+1} * \cdots * B_n * \]
\[ R \]
Proving Correctness of Insertion

\[ r \mapsto \mathbb{I}(d ; p) \ast \]
\[ B_1 \ast \cdots \ast B_{i-1} \ast \]
\[ B_i \ast \]
\[ B_{i+1} \ast \cdots \ast B_n \ast \]
\[ R \]
Proving Correctness of Insertion

\[ B_i \ast R' \]
Proving Correctness of Insertion

$B'_i \ast B''_i \ast R'$
Proving Correctness of Insertion

\[ r \mapsto I(d; p) \star \]
\[ B_1 \star \cdots \star B_{i-1} \star \]
\[ B_i' \star B_i'' \star \]
\[ B_{i+1} \star \cdots \star B_n \star \]
\[ R \]
Finding a list of entries for a range query

- Find the list of entries with key $\geq k$.
- Need to find the first such element at the leaf level
- Use fringe links of the leaf nodes
- Machine descends the tree (no need to keep a stack).

$$\langle \text{Find}(k), r, \sigma \rangle \leadsto \langle \text{Find}(k), p_i, \sigma \rangle$$

if $\sigma(r) = I(d; p)$

where $i = \text{first}(d, \lambda x. x > k)$

$$\langle \text{Find}(k), r, \sigma \rangle \leadsto \langle \text{Ret}(r, i), \sigma \rangle$$

where $\sigma(r) = L(e; f)$

and $i = \text{first}(e, \lambda x. \text{key}(x) \geq k)$
Correctness of find

- Start from a B+ tree with its implicit fringe list
- Result of find is covered by a list predicate
- Rest of the tree is swept into the catch-all predicate true.
- Gerrymander the existing store to reveal the result list.

Theorem

If $\sigma \models \exists h, a.\text{Btree}_h(r, S, a, \text{null}, n)$, then

\[
\langle \text{Find}(k), r, \sigma \rangle \leadsto \cdots \leadsto \langle \text{Ret}(q, i), \sigma \rangle
\]

for some $q$ and $i$ with

$\sigma \models \text{FList}(q, i, S \uparrow k)$

* true

list of results

everything else
Correctness of find: Appending the fringe list onto an accumulator.

Lemma
Suppose $\sigma \models \text{Btree}_h(r, S_r, a, z, n) \ast \text{FList}(z, 1, S_z)$

Then $\sigma \models \text{FList}(a, 1, S_r \cup S_z) \ast \text{true}$. 
Correctness of find: induction
A technical lemma combines:

- current subtree
- partial result list
- everything else, by way of \textit{true}

\textbf{Lemma}

\textit{Let}

\[ \sigma \models \text{Btree}_h(p, S_p, a, z, n) \ast \text{FList}(z, 1, S_z) \ast \text{true} \]

\textit{Then we have a sequence of} \( h \) \textit{transitions}

\[ \langle \text{Find}(k), r, \sigma \rangle \rightsquigarrow \cdots \rightsquigarrow \langle \text{Ret}(q, i), \sigma \rangle \]

\textit{for some} \( q \) \textit{and} \( i \) \textit{with} \( \sigma \models \text{FList}(q, i, (S_r \uparrow k) \cup S_z) \ast \text{true} \).
Conclusions

Summary

- Programming semantics technology to address B+ trees
- Separation logic $\rightarrow$ and $\ast$ for invariants and local reasoning
- Abstract machines from functional programming
- Translation to CAML prototype is straightforward

Further work

- Deletion will be covered in the full version of the paper
- High-level primitives, like Gardner’s context logic for XML
- Trees with “holes”, holey Brick and BV tree: a job for $\rightarrow\ast$?
- Concurrent B trees
The End

Thank you for your attention.