Verified Algorithm Analysis: Correctness and Complexity

A Biased Survey

Tobias Nipkow

Fakultät für Informatik
TU München

Focus on algorithm analyses in ITPs

Unless otherwise noted: in Isabelle/HOL

Please let me know of missed references

Out of scope: related work on completely automatic running time analyses by Martin Hofmann, Jan Hoffmann, Madhavan & Kuncak, ...

1. Mathematical Foundations

2. Programming and Verification Frameworks

3. Algorithms

Slides and results by Manuel Eberl

Classic concepts and results

- Landau symbols
- Generating functions
- Linear recurrences (theory and solver)
- Asymptotics of $n!$, Γ, H_n, C_n, $\ldots$

Akra–Bazzi theorem

Generalisation of the
Master Theorem for divide-and-conquer recurrences

Input (simple case):

$$T(x) = g(x) + \sum_{i=1}^{k} a_i T\left(\lfloor b_i x \rceil\right) \quad \text{for } g \in \Theta(x^q \ln^r x)$$

Result:

$$T \in \Theta(x^p) \quad T \in \Theta(x^p \ln \ln x)$$
$$T \in \Theta(x^q) \quad T \in \Theta(x^p \ln^{q+1} x)$$

where p is the unique solution to $\sum a_i b_i^p = 1$

Examples for Akra–Bazzi

Algorithm	Recurrence	Solution
Binary search	$T(\lceil n/2 \rceil) + O(1)$	$O(\log n)$
Merge sort	$T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n)$	$O(n \log n)$
Karatsuba	$2T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n)$	$O(n^{\log_2 3})$
Median-of-med's	$T(\lceil 0.2n \rceil) + T(\lceil 0.7n \rceil + 6) + O(n)$	$O(n)$

All of this is (almost) automatic.

Automated asymptotics

Isabelle can automatically prove

- $f(x) \xrightarrow{x \to L} L'$
- $f \in O(g)$, $f \in o(g)$, $f \in \Theta(g)$, $f(x) \sim g(x)$
- $f(x) \leq g(x)$ for x sufficiently close to L

for a wide class of $\mathbb{R}$-valued functions/sequences.

How? Multiseries expansions

Similar to algorithms used in Mathematica/Maple

Automated asymptotics

Example from Akra–Bazzi proof:

$$\lim_{x\to\infty} \left(1 - \frac{1}{b\log^{1+\varepsilon} x}\right)^p \left(1 + \frac{1}{\log^{\varepsilon/2}\left(bx + \frac{x}{\log^{1+\varepsilon} x}\right)}\right) - \left(1 + \frac{1}{\log^{\varepsilon/2} x}\right) \;=\; 0^+$$

Can be proved automatically in 0.3 s.

1. Mathematical Foundations

2. Programming and Verification Frameworks

3. Algorithms

For programming, refinement and verification
of algorithms
in Isabelle/HOL

Functional vs Imperative

Functional algorithms are expressed as HOL functions

Imperative algorithms are expressed in

`Imperative HOL`

a monadic framework with arrays and references by Bulwahn & Co [TPHOLs 08]

Can generate code in SML, OCaml, Haskell and Scala [Haftmann, N. FLOPS 10]

A problem:

Head-on verification of efficient algorithms
is painful or impossible

The cure:

Start from an abstract functional version
and refine it to an efficient (imperative) algorithm

A second problem:

Not every algorithm is deterministic:

```
for every neighbour do ...
```

Isabelle refinement framework

Lammich [ITP 12, ITP 13, ITP 15, CPP 16]

Provides abstract programming language with

- nondeterminism
- loops (incl. `foreach`)
- general recursion
- specification statement

Isabelle refinement framework

Lammich [ITP 12, ITP 13, ITP 15, CPP 16]

Stepwise program refinement by:

- algorithm refinement
- semi-automatic data refinement
  using verified collections library
- semi-automatic refinement to `Imperative HOL`

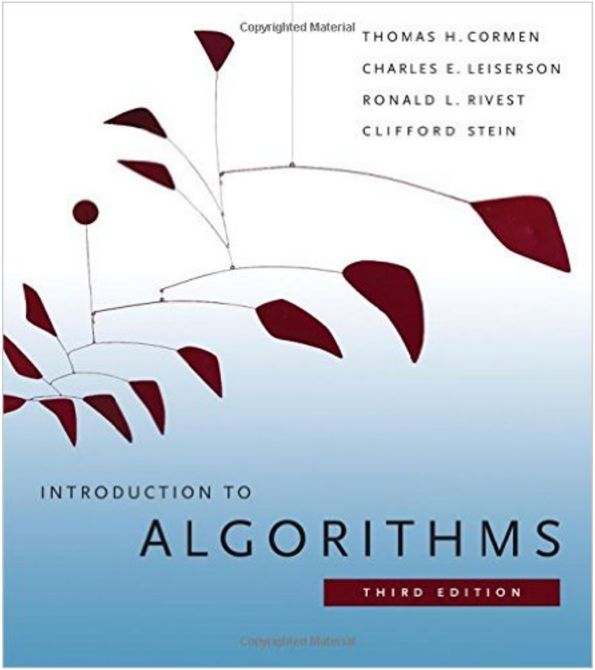
Almost all referenced Isabelle proofs can be found in the

Archive of Formal Proofs (AFP)

1. Mathematical Foundations

2. Programming and Verification Frameworks

3. Algorithms


Copyrighted Material
THOMAS H. CORMEN
CHARLES E. LEISERSON
RONALD L. RIVEST
CLIFFORD STEIN
INTRODUCTION TO
ALGORITHMS
THIRD EDITION
Copyrighted Material

3 Algorithms

Sorting & Order statistics

Search trees

Advanced Design and Analysis Techniques

Dynamic Programming

Advanced Data Structures

Graph Algorithms

Randomized Algorithms

Sorting

TIMsort: `java.util.Arrays.sort`

- A complex combination of mergesort and insertion sort on arrays
- De Gouw & Co [CAV 15] discover bug and suggest fixes
- De Gouw & Co [JAR 17] verify termination and exception freedom using the KeY system. Meanwhile: verification of functional correctness

k-th smallest element via median of medians

```
select k xs =
  let x = select ... (map median5 (chop 5 xs));
      (ls, es, gs) = partition3 x xs
  in if ... then select k ls
     else ... select ... gs
```

- Functional version by Eberl [AFP 17]
- Imperative refinement (incl linear time proof) by Zhan & Haslbeck [IJCAR 18] using Akra-Bazzi

3 Algorithms

- Sorting & Order statistics
- Search trees
- Advanced Design and Analysis Techniques
- Dynamic Programming
- Advanced Data Structures
- Graph Algorithms
- Randomized Algorithms

Popular case study for ITPs because nicely functional.

AVL and Red-Black trees:

- Filliâtre & Letouzey [ESOP 04] (in Coq)
- N. & Pusch [AFP 04]
- Krauss & Reiter [08]
- Charguéraud [10] (in Coq)
- Appel [11] (in Coq)
- Dross & Moy [14] (in SPARK)
- ...

Functional correctness

- Functional correctness obvious to humans
  but until recently more or less verbose in ITPs
- Most verifications based on some variant of
  $bst\langle l, a, r\rangle \leftrightarrow$
  $bst\ l \wedge bst\ r \wedge (\forall x \in l.\ x < a) \wedge (\forall x \in r.\ a < x)$
- Correctness proofs can be automated if $bst(t)$ is replaced by N. $sorted(inorder\ t)$ [N. ITP 16]
- Works for AVL, RBT, 2-3, 2-3-4, AA, splay and other search trees covered in this talk
- Not automated: balance invariants

Some more search trees
Not in CLRS

Weight-Balanced Trees

Nievergelt & Reingold [72,73]

- Parameter: balance factor $0 < \alpha \leq 0.5$
- Every subtree must be balanced:

$$\alpha \leq \frac{\text{size of smaller child}}{\text{size of whole subtree}}$$

- Insertion and deletion: single and double rotations depending on subtle numeric conditions
- Nievergelt and Reingold deletion incorrect
- Mistake discovered and corrected by
  Blum & Mehlhorn [80]
  and Hirai & Yamamoto [JFP 11] (in Coq)

Scapegoat trees

Anderson [89,99], Igal & Rivest [93]

Central idea:

Don't rebalance every time,
Rebuild when the tree gets "too unbalanced"

- Tricky: amortized logarithmic complexity analysis
- Recently verified [N. APLAS 17]

Functional finger tree

Hinze & Paterson [06]

Tree representation of sequences with

- access time to both ends in amortized $O(1)$
- concatenation and splitting in $O(\log n)$

General purpose data structure for implementing *sequences*, *priority queues*, *search trees*, ...

Verifications:

- Functional correctness:
  - Sozeau [ICFP 07] (in Coq)
  - Nordhoff, Körner, Lammich [AFP 10]
- Amortized complexity:
  - Danielsson [POPL 08] (in Agda)

❸ Algorithms

Sorting & Order statistics
Search trees
Advanced Design and Analysis Techniques
Dynamic Programming
Advanced Data Structures
Graph Algorithms
Randomized Algorithms

Huffman's algorithm

Huffman [52]

- Purpose: lossless text compression,
  eg Unix `zip` command
- Input: frequency table for all characters
- Output:
  variable length *binary code* for each character
  that minimizes the length of the encoded text
  $\Rightarrow$ short codes for frequent characters
- Functional correctness proof: Blanchette [JAR 09]

3 Algorithms

- Sorting & Order statistics
- Search trees
- Advanced Design and Analysis Techniques
- Dynamic Programming
- Advanced Data Structures
- Graph Algorithms
- Randomized Algorithms

The functional approach

Wimmer & Co [ITP 18]

Write recursive program

```
fib(n) = fib(n-1) + fib(n-2)
```

Crank the handle and obtain monadic memoized version

```
fib' n := do { a ← fib'(n-1);
               b ← fib'(n-2);
               return (a+b) }
```

with correctness theorem

```
snd (runstate (fib' n) empty) = fib n
```

where `f x := rhs` abbreviates

```
f x = do a ← lookup x;
         case a of
           Some r ⇒ return r |
           None ⇒ do r ← rhs;
                     update x r;
                     return r
```

Automation

- Automatic definition
  of monadic memoized function
- Automatic correctness proof
  via parametricity reasoning

How is the state $(=$ memory$)$ realized?

Two state monads

- Purely functional state monad
  based on some search tree
- State monad of `Imperative HOL` using arrays
  Same $O(.)$ running time
  as standard imperative programs

Applications

- Bellman-Ford (SSSP)
- CYK (Context-free parsing)
- Minimum Edit Distance
- Optimal Binary Search Tree
- ...

Including correctness proofs

But without complexity analysis (yet)

Optimal Binary Search Tree

Input:

- set of keys $k_1, \dots, k_n$
- access frequencies $b_1, \dots, b_n$ (hits):
  $b_i =$ number of searches for k_i
- and $a_0, \dots, a_n$ (misses):
  $a_i =$ number of searches in (k_i, k_{i+1})

Algorithms for building optimal search tree:

- Straightforward recursive cubic algorithm
- Knuth [71]: a quadratic optimization
- Yao [80]: simpler proof
- N. & Somogyi [AFP 18]

3. Algorithms
   - Sorting & Order statistics
   - Search trees
   - Advanced Design and Analysis Techniques
   - Dynamic Programming
   - **Advanced Data Structures**
   - Graph Algorithms
   - Randomized Algorithms

B-trees

Functional verification:

- Malecha & Co [POPL 10] (in Coq + Ynot)
- Ernst & Co [SSM 15] (in KIV)

Priority queues

Verification of functional implementations:

- Leftist heap
- Braun tree [N. AFP 14]
- Amortized analysis of
  Skew heap, Splay heap, Pairing heap
  N. [ITP 16], N. & Brinkop [JAR 18]
- Binomial heap and Skew binomial heap
  Meis, Nielsen, Lammich [AFP 10]

None of the above provide `decrease-key` ...
Challenge!

Union-Find

Charguéraud, Pottier, Guéneau [ITP 15, JAR 17, ESOP 18]

Framework ("Characteristic Formula"):

- Translates OCaml program into a logical formula that captures the program behaviour, including effects and running time.
- Import into Coq as axiom
- Verify program in Coq

Verified amortized complexity $O(\alpha(n))$ of each call
(Following Alstrup & Co [JA 14])

Strongly connected components

- Tarjan [72],
  verified by Schimpf & Smaus [ICLA 2015]
- Gabow [IPL 00],
  verified by Lammich [ITP 14]

Used in verified model checker CAVA

Dijkstra (SSSP)

Dijkstra [59]

Functional correctness verified:

- Nordhoff & Lammich [AFP 12]:
  purely functionally with finger trees
- Lammich [CPP 16]:
  imperative with arrays

Floyd-Warshall (APSP)

Functional correctness verified by Wimmer & Lammich [AFP 17]:

- Functional implementation
- Refined to imperative algorithm on an array
- Main complication: destructive update
- All related verifications make simplifying assumptions — also in CLRS

Maximum network flow

- Edmonds-Karp:
  Lammich & Sefidgar [ITP 16]
  Imperative, running time $O(|V||E|^2)$
- Push-Relabel (2 variants):
  Lammich & Sefidgar [JAR 17]
  Imperative, running time $O(|V|^2|E|)$

Competitive with a Java implementation

3 Algorithms

Sorting & Order statistics

Search trees

Advanced Design and Analysis Techniques

Dynamic Programming

Advanced Data Structures

Graph Algorithms

Randomized Algorithms

Randomized algorithms formalized

Purely functionally via the Giry monad

Example:

$\texttt{do}\ \{\ \texttt{a} \leftarrow \textit{some distribution};$
$\quad\quad \texttt{b} \leftarrow \textit{some other distribution}\ (\texttt{a});$
$\quad\quad \texttt{return (a+b)}\ \}$

Quicksort

- van der Weegen & McKinna [ITP 08] (in Coq)
  Proved expected running time of randomized and deterministic quicksort $\leq 2n\lceil\log_2 n\rceil$
- Eberl & Co [ITP 18]
  Proved closed form $2(n+1)H_n - 4n$
  and asymptotics $\sim 2n\ln n$
- Tassarotti & Harper [ITP 18] (in Coq)
  Formalized and extended cookbook method for tail bounds [Karp JACM 94]
  Applied it to quicksort:
  $\Pr[T(n) > (c+1)n\log_{4/3} n + 1] \leq \frac{1}{n^{c-1}}$

Analysis of random BSTs

Eberl & Co [ITP 18]

"Random BST" means

BST generated from a random permutation of keys

Thm Expected height of random BST
$\leq \ldots \sim 3 \log_2 n$

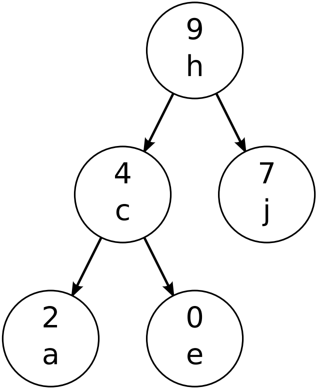
Thm Distribution of internal path lengths
$=$ distribution of running times of quicksort

Treaps

Aragon & Seidel [89, 96]

Random BSTs are pretty good,
but keys are typically not random

Treaps: combine each key with a random *priority*



treap = tree + heap

Treaps verified

Eberl & Co [ITP 18]

- Functional correctness straightforward
- Treaps need a *continuous* distribution of priorities to avoid duplicates (with probability 1)
- Reasoning about continuous distributions is hard because of measurability proofs
- **Thm** Distribution of treaps
  = distribution of random BSTs (modulo priorities)

Conclusion: Comparison with CLRS

The first 750 pages (parts I–VI, the "core")

- Much of the basic material has been verified
- Major omissions (afaik):
  - Hashing incl. probabilities
  - Fibonacci heaps
  - van Emde Boas trees