A Fistful of Dollars: Formalizing Asymptotic Complexity Claims via Deductive Program Verification

Armaël Guéneau, Arthur Charguéraud, François Pottier

Inria
Motivational example: binary search

Claim: “binary search finds an element in time $O(\log n)$”

Goal: formalize this claim in Coq for a concrete implementation
let rec bsearch (a: int array) v i j = 
  if j <= i then -1 else 
    let k = i + (j - i) / 2 in 
    if v = a.(k) then k 
    else if v < a.(k) then bsearch a v i k 
    else bsearch a v (i+1) j 

- We can test this program 
- We can prove functional correctness (Why3, CFML, ...)

let rec bsearch (a: int array) v i j = 
  if j <= i then -1 else 
  let k = i + (j - i) / 2 in 
  if v = a.(k) then k 
  else if v < a.(k) then bsearch a v i k 
  else bsearch a v (i+1) j

- We can test this program
- We can prove functional correctness (Why3, CFML, ...)
Yet, there is a bug

(* search for v in the range [i, j] *)

let rec bsearch (a: int array) v i j =
    if j <= i then -1 else
    let k = i + (j - i) / 2 in
    if v = a.(k) then k
    else if v < a.(k) then bsearch a v i k
    else bsearch a v (i+1) j

Can you spot the bug?
Yet, there is a bug

(* search for v in the range [i, j) *)

let rec bsearch (a: int array) v i j =
  if j <= i then -1 else
  let k = i + (j - i) / 2 in
  if v = a.(k) then k
  else if v < a.(k) then bsearch a v i k
  else bsearch a v (i+1) j

Can you spot the complexity bug?
Yet, there is a bug

(* search for \( v \) in the range \([i, j]\) *)

\[
\text{let rec bsearch (a: int array) v i j =}
\]

\[
\begin{align*}
\text{if } j \leq i \text{ then } -1 \text{ else} \\
\text{let } k = i + \frac{j - i}{2} \text{ in} \\
\text{if } v = a.(k) \text{ then } k \\
\text{else if } v < a.(k) \text{ then bsearch a v i k} \\
\text{else bsearch a v (i+1) j}
\end{align*}
\]

buggy, should be \( k+1 \)

Can you spot the complexity bug?
In this talk

Goal: prove OCaml programs, including their asymptotic complexity expressed with $O()$ bounds

State of the art:

- Automatic inference for polynomial bounds
- Interactive proofs using *time credits*, e.g. “bsearch costs $3 \log n + 4$”

Issue: conciseness, and modularity of specifications
In this talk (2)

Solution: introduce the \( O() \) notation for conciseness and modularity

Challenges:

- How to write specifications?
- What is the meaning of \( O() \) in the multivariate case?
- How to do proofs (paper proofs are too informal)?
- How to automate the cost analysis?
Separation Logic with Time Credits
Each function call (or loop iteration) consumes $1$.

$n$ asserts the ownership of $n$ time credits.

$(n + m) = n \times m$

Credits are not duplicable: $1 \not\Rightarrow 1 \times 1$

Enables amortized analysis.

References:

- Atkey (2011): time credits in Separation Logic
- Charguéraud & Pottier (2015): practical verification framework (CFML), applied to Union-Find
Example of using time credits

A specification of the complexity of bsearch:

\[
\forall i \, j \, a \, v. \\
\{ (3 \log(j - i) + 4) \ast \ldots \} \text{ (bsearch a v i j) } \{ \ldots \}
\]
Example of using time credits

A specification of the complexity of bsearch:

\[ \forall i \ j \ a \ v. \\{(3 \log(j - i) + 4) \ast \ldots \}\ (\text{bsearch}\ a\ v\ i\ j) \ \{\ldots\} \]

- Conciseness issue: even non dominant terms must appear
Example of using time credits

A specification of the complexity of \texttt{bsearch}:

\[ \forall i \ j \ a \ v. \ \{ $(3 \log(j - i) + 4) \ast \ldots \} \text{ (bsearch a v i j) } \{ \ldots \} \]

- Conciseness issue: even non dominant terms must appear
- Modularity issue: changing (even slightly) \texttt{bsearch} requires updating the specification, and \textbf{all proofs that depend on it}.
Example of using time credits

A specification of the complexity of bsearch:

$$\forall i \ j \ a \ v. \ \{(3 \log(j - i) + 4) \ast ...\} \ (\text{bsearch} \ a \ v \ i \ j) \ \{\ldots\}$$

- Conciseness issue: even non dominant terms must appear
- Modularity issue: changing (even slightly) bsearch requires updating the specification, and all proofs that depend on it.
- Tempting: $$\{(O(\log(j - i)) \ast ...\} \ (\text{bsearch} \ a \ v \ i \ j) \ \{\ldots\}$$
Challenges in reasoning with $O$
Informal reasoning principles can be abused

```
1 let rec bsearch a v i j =
2   if j <= i then -1 else
3     let k = i + (j - i) / 2 in
4     if v = a.(k) then k
5     else if v < a.(k) then
6        bsearch a v i k
7     else
8        bsearch a v (k+1) j
```

“Claim”:

*bsearch a v i j* costs \(O(1)\).
Informal reasoning principles can be abused

```ocaml
1 let rec bsearch a v i j =
2   if j <= i then -1 else
3     let k = i + (j - i) / 2 in
4     if v = a.(k) then k
5     else if v < a.(k) then
6       bsearch a v i k
7     else
8       bsearch a v (k+1) j
```

“Claim”:
\[ bsearch a v i j \text{ costs } O(1). \]

Proof:
Informal reasoning principles can be abused

```
let rec bsearch a v i j =
  if j <= i then -1 else
  let k = i + (j - i) / 2 in
  if v = a.(k) then k
  else if v < a.(k) then
    bsearch a v i k
  else
    bsearch a v (k+1) j
```

“Claim”:
bsearch a v i j costs $O(1)$.

Proof:
By induction on $j - i$: 
Informal reasoning principles can be abused

1 let rec bsearch a v i j =
2   if j <= i then -1 else
3     let k = i + (j - i) / 2 in
4     if v = a.(k) then k
5     else if v < a.(k) then
6       bsearch a v i k
7     else
8       bsearch a v (k+1) j

“Claim”: bsearch a v i j costs $O(1)$.

Proof:
By induction on $j - i$:
• $j - i \leq 0$: $O(1)$. OK!
Informal reasoning principles can be abused

```ml
let rec bsearch a v i j =  
  if j <= i then -1 else  
  let k = i + (j - i) / 2 in  
  if v = a.(k) then k  
  else if v < a.(k) then  
    bsearch a v i k  
  else  
    bsearch a v (k+1) j
```

“Claim”:

\[ \text{bsearch } a \text{ } v \text{ } i \text{ } j \text{ } \text{costs } O(1). \]

Proof:

By induction on \( j - i \):

- \( j - i \leq 0 \): \( O(1) \). OK!
- \( j - i > 0 \): \( O(1) + O(1) + O(1) = O(1) \). OK!
Informal reasoning principles can be abused

```
1 let rec bsearch a v i j =
2   if j <= i then -1 else
3     let k = i + (j - i) / 2 in
4     if v = a.(k) then k
5     else if v < a.(k) then
6       bsearch a v i k
7     else
8       bsearch a v (k+1) j
```

“Claim”: bsearch a v i j costs $O(1)$.

Proof:
By induction on $j - i$:

- $j - i \leq 0$: $O(1)$. OK!
- $j - i > 0$: $O(1) + O(1) + O(1) = O(1)$. OK!

Where is the mistake?
Informal reasoning principles can be abused

1 let rec bsearch a v i j =
2   if j <= i then -1 else
3     let k = i + (j - i) / 2 in
4     if v = a.(k) then k
5     else if v < a.(k) then
6       bsearch a v i k
7     else
8       bsearch a v (k+1) j

“Claim”: bsearch a v i j costs $O(1)$.

Proof:

\[ \text{By induction on } j - i: \]

...but which statement are we proving?

- $j - i \leq 0$: $O(1)$. OK!
- $j - i > 0$: $O(1) + O(1) + O(1) = O(1)$. OK!
Meaning of $O(1)$

What we just proved:

$$\forall i, j, \exists c, \text{bsearch } a \text{ v } i \text{ j runs in } c \text{ steps}$$
Meaning of $O(1)$

What we just proved:

$$\forall i, j, \exists c, \text{bsearch a v i j runs in c steps}$$

What “$O(1)$” means:

$$\exists c, \forall i, j, \text{bsearch a v i j runs in c steps}$$
Meaning of $O(\log n)$

“bsearch a v i j runs in $O(\log(j - i))$ steps.”

• Meaning of $f \in O(g)$?
• How to provide a witness for $f$?
Meaning of $O(\log n)$

“bsearch a v i j runs in $O(\log(j - i))$ steps.”

**there exists** a cost function $f \in O(\log n)$ such that, for every $a, v, i, j$, bsearch $a \ v \ i \ j$ runs in $f(j - i)$ steps.”
Meaning of $O(\log n)$

"bsearch a v i j runs in $O(\log(j - i))$ steps."

"**there exists** a cost function $f \in O(\log n)$ such that, for every $a, v, i, j$, $bsearch a v i j$ runs in $f(j - i)$ steps."

"**there exists** a cost function $f \in O(\log n)$ such that, for every $a, v, i, j$, \{$f(j - i) \ast \ldots\}$ (bsearch a v i j) \{\ldots\}".
Meaning of $O(\log n)$

“bsearch a v i j runs in $O(\log(j - i))$ steps.”

“there exists a cost function $f \in O(\log n)$ such that, for every $a, v, i, j$, bsearch a v i j runs in $f(j - i)$ steps.”

“there exists a cost function $f \in O(\log n)$ such that, for every $a, v, i, j$, \{$f(j - i) \ast \ldots\}$ (bsearch a v i j) \{$\ldots\}$.”

- Meaning of “$f \in O(g)$”?
- How to provide a witness for $f$?
A generic definition of $O$
**Definition of** $O$

- **Single variable case:**
  \[ f \in O(g) \iff \exists c, \exists n_0, \forall n \geq n_0, |f(n)| \leq c |g(n)| \]
  
  with $f$ of type $\mathbb{N} \to \mathbb{Z}$

- **Multivariate case:** $f$ of type $\mathbb{N}^k \to \mathbb{Z}$

- **In our library:** $f$ of type $A \to \mathbb{Z}$, with a *filter* on type $A$
O as a relation between functions

We define $O$ as a domination pre-order between functions of $A$ to $\mathbb{Z}$:

$$f \leq_A g \equiv \exists c. \mathbb{U}_A x. |f(x)| \leq c |g(x)|$$

$A$ must be equipped with a filter $\mathbb{U}_A$.
We define $O$ as a \textit{domination} pre-order between functions of $A$ to $\mathbb{Z}$:

$$f \leq_A g \equiv \exists c. \, \mathcal{U}_A x. \, |f(x)| \leq c \, |g(x)|$$

$A$ must be equipped with a filter $\mathcal{U}_A$

- "$\mathcal{U}_A x. P$": "ultimately $P$" / "$P$ holds of every sufficiently large $x$"
- Can be thought of as a quantifier
We define $O$ as a domination pre-order between functions of $A$ to $\mathbb{Z}$:

$$f \leq_A g \equiv \exists c. \, \mathcal{U}_A x. \, |f(x)| \leq c \, |g(x)|$$

$A$ must be equipped with a filter $\mathcal{U}_A$

- "$\mathcal{U}_A x. P$": "ultimately $P$" / "$P$ holds of every sufficiently large $x$"
- Can be thought of as a quantifier
- A standard notion in math (see e.g. Bourbaki)
- We prove in our library many properties of $\leq_A$ for an arbitrary filtered type $A$
Proving specifications: automatic (guided) cost synthesis
Providing the cost function

“there exists a cost function \( f \in O(\log n) \) such that, for every \( a, v, i, j \),
\[
\{ f(j - i) \ast \ldots \} (\text{bsearch} \ a \ v \ i \ j) \{ \ldots \}
\]
becomes

\[
\exists f : \mathbb{Z} \rightarrow \mathbb{Z}.
\begin{align*}
& f \leq_{\mathbb{Z}} \lambda n. \log n \\
& \forall i j a v. \{ f(j - i) \ast \ldots \} (\text{bsearch} \ a \ v \ i \ j) \{ \ldots \}
\end{align*}
\]

- First step of the proof: exhibit a concrete cost function. Guess “\( \lambda n. 3 \log n + 4 \)” from the start?
- It seems desirable to (semi) automatically construct the witness as the proof progresses.
Our approach to this problem

- Convince Coq to postpone the moment where the concrete cost function is provided
Our approach to this problem

- Convince Coq to postpone the moment where the concrete cost function is provided
- Progressively synthesize the cost function while applying the reasoning rules from separation logic
Our approach to this problem

- Convince Coq to postpone the moment where the concrete cost function is provided

- Progressively synthesize the cost function while applying the reasoning rules from separation logic

- The synthesized function has the same structure as the code
Our approach to this problem

- Convince Coq to postpone the moment where the concrete cost function is provided
- Progressively synthesize the cost function while applying the reasoning rules from separation logic
- The synthesized function has the same structure as the code
- Afterwards, prove a $O()$ bound for the cost function
let rec bsearch a v i j =
  if j <= i then -1 else
  let k = i + (j - i) / 2 in
  if v = Array.get a k then k
  else if v < Array.get a k then
    bsearch a v i k
  else
    bsearch a v (k+1) j

f n := 1 + (if n <= 0 then 0 else
  0 + 1 + max 0 (1 + max (f (n/2))
    (f (n - n/2 - 1))
  )
)
where n = j-i
if $j \leq i$ then -1 else
let $k = i + (j - i) / 2$ in
if $v = Array.get a k$ then $k$
else if $v < Array.get a k$ then
    bsearch $a v i k$
else
    bsearch $a v (k+1) j$

\[
f(j-i) := 1 + \ldots\]

A hole ("\ldots") is implemented as an evar in Coq.
if $j \leq i$ then -1 else
  let $k = i + (j - i) / 2$ in
  if $v = \text{Array}.get\ a\ k$ then $k$
  else if $v < \text{Array}.get\ a\ k$ then
    bsearch $a\ v\ i\ k$
  else
    bsearch $a\ v\ (k+1)\ j$

$f\ (j-i) := 1 + (\text{if } j \leq i\ \text{then } \ldots\ \text{else } \ldots)$
if \( j \leq i \) then -1 else
let \( k = i + (j - i) / 2 \) in
if \( v = \text{Array.get} a k \) then \( k \)
else if \( v < \text{Array.get} a k \) then
    \text{bsearch} a v i k
else
    \text{bsearch} a v (k+1) j

\[
\text{f} (j-i) := 1 + (\text{if } j \leq i \text{ then } \ldots \text{ else } \ldots)
\]
if \( j \leq i \) then -1 else
let \( k = i + (j - i) / 2 \) in
if \( v = \text{Array}.\text{get} \ a \ k \) then \( k \)
else if \( v < \text{Array}.\text{get} \ a \ k \) then
    \text{bsearch} \ a \ v \ i \ k
else
    \text{bsearch} \ a \ v \ (k+1) \ j

f (j-i) := 1 + (if (j-i) <= 0 then ... else ...
if \( j \leq i \) then -1 else

let \( k = i + (j - i) / 2 \) in

if \( v = \text{Array.get} a k \) then \( k \)
else if \( v < \text{Array.get} a k \) then

\text{bsearch} a \ v \ i \ k

else

\text{bsearch} a \ v \ (k+1) \ j

\hline

\( f (j-i) := 1 + (\text{if} \ (j-i) \leq 0 \ \text{then} \ 0 \ \text{else} \ ...) \)
if \( j \leq i \) then -1 else

let \( k = i + (j - i) / 2 \) in

if \( v = \text{Array.get\ a\ k} \) then \( k \)
else if \( v < \text{Array.get\ a\ k} \) then
    \text{bsearch\ a\ v\ i\ k}
else
    \text{bsearch\ a\ v\ (k+1)\ j}

f (j-i) := 1 + (\n    if (j-i) \leq 0 then 0 else
    0 + ...
)
if \( j \leq i \) then -1 else
let \( k = i + (j - i) / 2 \) in
if \( v = \text{Array.get a k} \) then \( k \)
else if \( v < \text{Array.get a k} \) then
  \text{bsearch a v i k}
else
  \text{bsearch a v (k+1) j}

\[
f(j-i) := 1 + (\begin{array}{c}
  \text{if } (j-i) \leq 0 \text{ then } 0 \\
  0 + 1 + \ldots
\end{array})
\]
if j <= i then -1 else
let k = i + (j - i) / 2 in
if v = Array.get a k then k
else if v < Array.get a k then
  bsearch a v i k
else
  bsearch a v (k+1) j

---

f (j-i) := 1 + (  
  if (j-i) <= 0 then 0 else
  0 + 1 + max ... ... 
)
if \( j \leq i \) then -1 else

let \( k = i + (j - i) / 2 \) in

if \( v = Array\text{.get} \ a \ k \) then \( k \)
else if \( v < Array\text{.get} \ a \ k \) then

bsearch \( a \ v \ i \ k \)

else

bsearch \( a \ v \ (k+1) \ j \)

---

\[
f (j-i) := 1 + (\text{if} \ (j-i) \leq 0 \ \text{then} \ 0 \ \text{else} \ 0 + 1 + \max \ 0 \ldots)
\]
if $j \leq i$ then $-1$ else
let $k = i + (j - i) / 2$ in
if $v = \text{Array.get} a k$ then $k$ else
  if $v < \text{Array.get} a k$ then
    $\text{bsearch} a v i k$
  else
    $\text{bsearch} a v (k+1) j$

\[
f (j-i) := 1 + \left( \begin{array}{l}
  \text{if} (j-i) \leq 0 \text{ then } 0 \\
  \text{else} \\
  0 + 1 + \max 0 (1 + \ldots)
  \end{array} \right)
\]
if $j \leq i$ then $-1$ else

let $k = i + (j - i) / 2$ in

if $v = \text{Array.get a } k$ then $k$
else if $v < \text{Array.get a } k$ then

bsearch a $v$ i $k$
else

bsearch a $v$ $(k+1)$ j

\[
f (j-i) := 1 + (\text{if } (j-i) \leq 0 \text{ then } 0 \text{ else } 0 + 1 + \max 0 (1 + \max \ldots \ldots))\]
if \( j \leq i \) then -1 else

let \( k = i + (j - i) / 2 \) in

if \( v = \text{Array.get a} k \) then \( k \)
else if \( v < \text{Array.get a} k \) then

bsearch a v i k

else

bsearch a v (k+1) j

\[
f(j-i) := 1 + \begin{cases} 
0 & \text{if } (j-i) \leq 0 \\
0 + 1 + \max \ 0 (1 + \max \ (f (\lfloor (j-i)/2 \rfloor)) ... 
\end{cases}
\]
if \( j \leq i \) then -1 else 
let \( k = i + \frac{(j - i)}{2} \) in 
if \( v = \text{Array.get a k} \) then \( k \)
else if \( v < \text{Array.get a k} \) then 
bsearch a v i k
else 
bsearch a v (k+1) j

\[
f(j-i) := 1 + (\ 
\begin{cases} 
0 &\text{if } (j-i) \leq 0 \\
0 + 1 + \max \ 0 (\ 
1 + \max (f((j-i)/2)) 
(f((j-i) - (j-i)/2 - 1)) 
\end{cases} 
) 
\]
\]
if \( j \leq i \) then -1 else \\
let \( k = i + (j - i) / 2 \) in \\
if \( v = \text{Array}.\text{get a} \ k \) then \( k \) \\
else if \( v < \text{Array}.\text{get a} \ k \) then \\
\text{bsearch a} \ v \ i \ k \\
else \\
\text{bsearch a} \ v \ (k+1) \ j \\
\hline

\[
\text{f n} \quad := 1 + ( \\
\text{if n} \leq 0 \text{ then 0 else} \\
0 + 1 + \max 0 ( \\
1 + \max 0 ( \text{f} (\text{n}/2)) \\
(\text{f} (\text{n} - \text{n}/2 - 1)) \\
) \\
) \\
) 
\]
Our cost synthesis achieves the following objectives:

- The user inspects the code only once
- The user can guide the synthesis of the cost function
Summary

This work

Modular specifications using $O()$ and filters

Better automation for proofs with time credits: cost function synthesis
Closely related work


- In Isabelle/HOL: **Zhan & Haslbeck** (2018) implement the same formal framework, with strong focus on automation but no “cost function synthesis”. They build on **Eberl**’s (2017) impressive formalization of the Akra-Bazzi theorem.

More in the paper:

- Details about side-conditions for cost functions: monotonic and non-negative
- Clear up some confusion about multivariate $O()$
- Variable substitution in multivariate specifications
- Other case studies: selection sort, Bellman-Ford, Union-Find

http://gallium.inria.fr/~agueneau/big0

Challenging case studies in the works!