FORMAL VERIFICATION OF ASYMPTOTIC COMPLEXITY BOUNDS FOR OCAMML PROGRAMS

Armaël Guéneau
supervised by François Pottier & Arthur Charguéraud
November 12, 2015
Time complexity can be formalized in separation logic, thanks to *time credits*.

Example of specification:

\[
\{ \text{UF N D R} \star (3 \times (\text{alpha N}) + 6) \}
\]

union \ x \ y

\[
\{ \lambda z \Rightarrow \text{UF N D} (\text{fun w} \Rightarrow \text{If R w = R x \lor R w = R y then z else R w}) \star [z = R x \lor z = R y] \}
\]

Amortized cost for union: \(3 \times \alpha(N) + 6\).
Counting credits explicitly quickly becomes impractical, compared to using the “$O()$” notation:

- $n^2 \times m + 3nm + 3n + 6m + 5\log(n) + 2\log(m) + 5\log(n)\log(m) + 8$ instead of $O(n^2 \times m)$
- Specifications using explicit credits count are not modular
- Credits count are to be considered up to a constant factor anyway
We present “CFML+credits+big-Os”, an extension of “CFML+credits” which formalizes (in Coq) the big-O notation, to be used in program specifications.
OUTLINE OF THIS TALK

Formalizing big-Os: challenges and proposed solutions

Proof automation

Case studies
FORMALIZING BIG-OS: CHALLENGES AND PROPOSED SOLUTIONS
Recall the standard textbook definition for “$O()$”:

$$f \in O(g) \equiv \exists c, \exists n_0, \forall n \geq n_0, |f(n)| \leq c \times |g(n)|$$

Why is this not trivial to formalize?
We often informally write “$f$ is $O(n^2)$”.

However $O()$ is a relation on functions, not expressions.

$\Rightarrow$ We should write “$f$ is $O(\lambda n.n^2)$” instead.
CHALLENGE 2: GOING TO INFINITY

How do we handle cost functions with multiple parameters?

let fill_rect n m =
for j = 1 to m do
  for i = 1 to n do
    draw_pixel i j
  done
done

Concrete cost:

\[
\begin{align*}
  f(n, m) &= m \times (1 + n) + 1 \\
          &= m \times n + m + 1
\end{align*}
\]
Is \texttt{fill\_rect} $O(\lambda(n, m).m \times n)$?

- If $n$ and $m$ go to infinity, then indeed $f(n, m) \in O(\lambda(n, m).m \times n)$

What about the asymptotic cost of “\texttt{fill\_rect 0 m}”? 

- Concrete cost: $f(0, m) = m + 1$
- Clearly not $O(\lambda m.m \times 0) = O(0)$

$\Rightarrow$ We cannot reuse the previous asymptotic bound
CHALLENGE 2: GOING TO INFINITY

- Big-O bounds are proved for one given notion of “going to infinity”
- There are multiple, non-equivalent ones

⇒ Let the user choose, while keeping a lightweight notation for the common cases.
CHALLENGE 2 SOLUTION: FILTERS, A FORMAL NOTION OF “GOING TO INFINITY”

A filter on a set $A$:

- is of type $(A \rightarrow \text{Prop}) \rightarrow \text{Prop}$, named `filter A`;
- represents the set of neighborhoods of infinity;
- must satisfy additional properties, bundled in a `Filter` Coq class.

E.g. the standard filter on $\mathbb{Z}$ is:

```
Definition towards_infinity_Z : filter Z :=
  fun (P : Z \rightarrow \text{Prop}) \Rightarrow \exists x \theta, \forall x, x \theta \leq x \rightarrow P x
```
“O()” definition parameterized by a filter \textit{ultimately}:

\begin{verbatim}
Definition dominated
    (ultimately: filter A)
    (f g : A \rightarrow \mathbb{Z}) :=
    \exists c, ultimately (\lambda x. \text{norm}(f x) \leq c \times \text{norm}(g x)).
\end{verbatim}

We use Coq typeclasses to allow the filter to be inferred in standard cases.
What were the filters involved in our `fill_rect` example?

- “Both components go to infinity”:  
  
  \[
  \text{Definition} \quad \text{towards\_infinity\_ZZ} := \text{fun}(P: \mathbb{Z}^2 \rightarrow \text{Prop}) \Rightarrow \exists P1, P2, \text{towards\_infinity\_Z} P1 \land \text{towards\_infinity\_Z} P2 \land \forall x1, x2, P1 \ x1 \rightarrow P2 \ x2 \rightarrow P(x1, x2)
  \]

- “The first component is fixed to \(x0\), the second goes to infinity”:  
  
  \[
  \text{Definition} \quad \text{towards\_infinity\_xZ} (x0: \mathbb{Z}) := \text{fun}(P: \{ p: \mathbb{Z}^2 | \text{fst} \ p = x0 \} \rightarrow \text{Prop}) \Rightarrow \text{towards\_infinity\_Z} (\text{fun} y \Rightarrow P(x0, y))
  \]
“The cost of $p$ is $O(g)$” hides an additional existential quantification.

“The cost of $p$ is $O(g)$” is in fact “there exists a cost function $f$ st. $f \in O(g)$ and running $p(n)$ takes $f(n)$ steps”.

- Convenient informal notation
- But more error prone: some incorrect proofs are harder to detect syntactically
let rec loop n =
    if n <= 0 then () else loop (n - 1)

Lemma (incorrect)

The asymptotic complexity of \texttt{loop} is $O(1)$.

Proof.

(flawed, but not so obviously). By induction on \( n \),

- \( n \leq 0 \): \texttt{loop} terminates in $O(1)$;
- \( n \geq 1 \): the cost of \texttt{loop}(n) is the cost of \texttt{loop}(n-1) plus $O(1)$. By induction, the cost of \texttt{loop}(n-1) is $O(1)$. $O(1) + O(1) = O(1) \Rightarrow$ total cost of $O(1)$. 
The mistake: an invalid quantifier permutation.

- “there exists a cost function $f$ st. for all $n$, …”, is not
- “for all $n$, there exists a cost function $f$ …”.

The explicit cost function must be instantiated before entering the induction.

Coq is able to reject this kind of incorrect reasoning; the challenge is to keep a lightweight presentation.
We define $\text{SpecO}$, in order to write specifications using big-Os:

**Definition** $\text{SpecO}(\text{ultimately: filter } A) \ (g: A \to Z)(\text{spec: } (A \to Z) \to \text{Prop})$

$$:= \exists (f: A \to Z), \text{dominated } _f g \land \text{spec } f.$$  

$$\forall n, \{\$ (3 \ast n^2 + 2 \ast n + 5) \ast H \} t(n) \{Q\}$$

becomes

$$\text{SpecO } _{(\lambda n \Rightarrow n^2)(\lambda F \Rightarrow \forall n, \{\$ F n \ast H \} t(n) \{Q\})}$$
Remark: arguments of the cost function do not have to be the arguments of the program.

**Example: specification for List.length**

\[
\forall l, \\
\text{SpecO } (\lambda n \Rightarrow n)(\lambda F \Rightarrow \\
\quad \{ F(\text{length } l) \} \text{List.length } l \{ \lambda n \Rightarrow [n = \text{length } l] \})
\]
It does not cover all usages though, e.g. quantifying over a class of filters for the same cost function.

\[ \exists (f: A \rightarrow Z), \]
\[ (\forall x\theta, \text{dominated}(\text{towards\_infinity}\_xZ \_x\theta)f \_g) \land \]
\[ \text{spec } f \]

\[ \Rightarrow \text{ More general version of SpecO parameterized by any relation on } f, g. \]
Paper proofs assume extensively that cost functions are non-decreasing.
Example:

\[ F(h) \preceq G(N) \]

\[ F \in O(\lambda h. h) \quad G \in O(\lambda N. \log(N)) \]

\[ G(N) := F(\log(N) + 1) \]

\[ \Rightarrow \text{We need to prove } F(h) \preceq G(N). \]

\[ \Rightarrow \text{We need } F \text{ to be non-decreasing.} \]
Definition SpecO (ultimately: filter A) le
  (g: A → Z)(spec: (A → Z) → Prop)
:=
  ∃(f: A → Z),
  (∀ x, 0 ≤ f x) ∧
  monotonic _ _ f ∧
  dominated _ f g ∧
  spec f.
We would like to have:

“if $f$ is $O(g)$, then $f + c$ is also $O(g)$ (with $c$ a constant)”.  

Yet, this is false for $g = 0$.  

We would like to have:

“if $f$ is $O(g)$, then $\lambda n. \log(f(n))$ is $O(\lambda n. \log(g(n)))$”.

Yet, this is false for $g = 1$ and $f \geq 2$. 
Alternative notion of $O()$: $\text{idominated}$.

- Matches $\text{dominated}$ on the interesting cases: when costs functions go to infinity;
- Handles more pathological cases.

**Definition** idominated

$$(\text{ultimately} : \text{filter } A)(\text{leA} : A \to A \to \text{Prop})$$

$$(\text{fg} : A \to Z)$$

$$:=$$

$$\text{ultimately } (\text{monotonic_after } \text{leA} \text{ leZ } \text{fg}) \land$$

$$((\text{bounded } _f \land \text{bounded } _g) \lor \text{dominated } _f \text{fg}).$$
The following lemmas are now true:

\[
\text{idominated} \quad f \quad g \rightarrow \\
\text{idominated} \quad (\text{fun} \quad n \rightarrow c + f \quad n) \quad g
\]

\[
\text{idominated} \quad f \quad g \rightarrow \\
\text{idominated} \quad (\text{fun} \quad x \rightarrow Z.\log2 (f \quad x))
\]

\[
(fun \quad x \rightarrow Z.\log2 (g \quad x))
\]

We also adapt Spec0 to use \text{idominated} in place of \text{dominated}.
Goal-directed tactics to solve / simplify \textit{idominated}, monotonic, \textit{monotonic\_after} goals.

Able to prove or simplify automatically goals involving $+ , \times , \log , ^\wedge$.

\textbf{Goal} \textit{idominated \_\_}

\begin{verbatim}
  (fun n => 5 * Z.log2 (3 * n + 2) + 8) Z.log2.
\end{verbatim}

\textbf{Proof}. \textit{idominated\_Z\_auto}; \texttt{math. Qed}.
Auxiliary tactics to deal with $\eta$-equivalence for $n$-ary functions (still imperfect).

- We have to reason modulo $\eta$-equivalence.
  - $O(\log) \text{ vs } O(\lambda n. \log(n))$
  - $f \in O(h) \Rightarrow g \in O(h) \Rightarrow \lambda n. f(n) + g(n) \in O(h)$

- Not automatic on $n$-ary (uncurried) functions.
  - They are of the form $\lambda p. \text{let } (n, m) = p \text{ in } \ldots$
  - $f \in O(h) \Rightarrow g \in O(h) \Rightarrow$
    $\lambda p. (\text{let } (n, m) = p \text{ in } f((n, m)) + g((n, m))) \in O(h)$
WIP: a set of tactics to elaborate the cost function through the proof.

---

SpecO _ _ (\(\lambda n \Rightarrow n\)) (\(\lambda F \Rightarrow \text{spec } F\))

\text{xfcO} (\text{fun } n \Rightarrow 3 \times n + 12). 

[...]

\text{add_credits} (\lambda n \Rightarrow 1). 

[...]

\text{add_credits} (\lambda n \Rightarrow 2 \times n). 

[...]
CASE STUDIES
We used the resulting library to formalize two non-trivial data structures:

- Dynamic Arrays, an imperative structure with amortized $O(1)$ costs;
- Binary Random Access Lists, a purely functional data structure with $O(\log n)$ costs, parameter transformation and filters on $\mathbb{Z}^2$. 
Why a parameter transformation and filters on $\mathbb{Z}^2$?

Figure 1: Induction for lookup and update
CONCLUSION: SOME NUMBERS

- Binary Random Access Lists:
  - Code: 80 lines, proof: 630 lines
  - Whole complexity analysis (credits + big-Os): $\approx 40\%$
  - Reasoning on big-Os: $\approx 25\%$

- Dynamic Arrays:
  - Code: 95 lines, proof: 520 lines
  - Whole complexity analysis (credits + big-Os): $\approx 50\%$
  - Reasoning on big-Os: $\approx 6\%$

- Size of the library: $\approx 2300$ lines of Coq
  - $\text{dominated, idominated}$ (definition, lemmas, tactics): 1260 lines
  - Filters (definitions, instances): 730 lines
  - Monotonicity (tactics): 250 lines
  - $\text{SpecO}$ (definition, lemmas, tactics): 70 lines