Lightweight Semiformal Time Complexity Analysis for Purely Functional Data Structures

Nils Anders Danielsson

Chalmers/Nottingham

Teaser

$$
\begin{aligned}
(+\!\!+) : &\ \mathit{Seq}\ a\ m \rightarrow \mathit{Seq}\ a\ n \\
&\rightarrow \mathit{Thunk}\ (1 + 2 * m)\ (\mathit{Seq}\ a\ (m + n))
\end{aligned}
$$

Focus

- Purely functional (persistent) data structures.
- Complexity results valid for arbitrary usage patterns, not just single-threaded use.
- PFDSs which are efficient for all usage patterns often make essential use of laziness (call-by-need).
- Complexity analysis becomes subtle; many details to keep track of.
- This work: Type system and library which ensure that no details are forgotten.

Focus

- Purely functional (persistent) data structures.
- Complexity results valid for arbitrary usage patterns, not just single-threaded use.
- PFDSs which are efficient for all usage patterns often make essential use of laziness (call-by-need).
- Complexity analysis becomes subtle; many details to keep track of.
- This work: Type system and library which ensure that no details are forgotten.

Focus

- Purely functional (persistent) data structures.
- Complexity results valid for arbitrary usage patterns, not just single-threaded use.
- PFDSs which are efficient for all usage patterns often make essential use of laziness (call-by-need).
- Complexity analysis becomes subtle; many details to keep track of.
- This work: Type system and library which ensure that no details are forgotten.

Library

Library

- Types keep track of time complexity:

  $f : (n : \mathbb{N}) \rightarrow \mathit{Thunk}\ (1 + n)\ \mathbb{N}$

- In *dependently* typed language (Agda).

Meaning

$e : \mathit{Thunk}\ n_1\ (\mathit{Thunk}\ n_2 \ldots (\mathit{Thunk}\ n_k\ a) \ldots)$ means that it takes at most

$$n_1 + n_2 + \ldots + n_k$$

steps amortised time to evaluate e to WHNF, if this computation terminates.

Annotations

- Library based on user-inserted annotations:

  $\checkmark : \mathit{Thunk}\ n\ a \to \mathit{Thunk}\ (1 + n)\ a$

- Every right-hand side should be ticked:

  $f\ (x :: xs) = \checkmark \ldots$

Machine assistance

- The library only checks correctness.
- Almost nothing inferred automatically.
- Recurrence equations have to be solved manually.

Example

Sequences

$$
\begin{aligned}
&\textbf{data}\ \mathit{Seq}\ (a : \star) : \mathbb{N} \to \star\ \textbf{where}\\
&\quad \mathrm{nil}\ \ : \mathit{Seq}\ a\ 0\\
&\quad (::) : a \to \mathit{Seq}\ a\ n \to \mathit{Seq}\ a\ (1 + n)
\end{aligned}
$$

Append

$$
\begin{aligned}
(+\!\!+) : \mathit{Seq}\ a\ m &\rightarrow \mathit{Seq}\ a\ n \\
&\rightarrow \mathit{Seq}\ a\ (m + n) \\
\mathsf{nil} \quad +\!\!+\ ys &= ys \\
(x :: xs) +\!\!+\ ys &= x :: (xs +\!\!+\ ys)
\end{aligned}
$$

Append

$$
\begin{aligned}
(+\!\!+) &: \mathit{Seq}\ a\ m \rightarrow \mathit{Seq}\ a\ n \\
&\quad \rightarrow \mathit{Seq}\ a\ (m + n) \\
\mathsf{nil}\ &+\!\!+\ \mathit{ys} = \checkmark\ \mathit{ys} \\
(x :: \mathit{xs})\ &+\!\!+\ \mathit{ys} = \checkmark\ x :: (\mathit{xs} +\!\!+ \mathit{ys})
\end{aligned}
$$

$$\checkmark : \mathit{Thunk}\ n\ a \rightarrow \mathit{Thunk}\ (1 + n)\ a$$

Append

$$
\begin{aligned}
(+\!\!+) &: Seq\ a\ m \to Seq\ a\ n \\
&\quad \to Thunk\ (1 + m)\ (Seq\ a\ (m + n)) \\
\text{nil} \quad &+\!\!+\ ys = {\checkmark}\ ys \\
(x :: xs)\ &+\!\!+\ ys = {\checkmark}\ x :: (xs +\!\!+ ys)
\end{aligned}
$$

$$
\checkmark : Thunk\ n\ a \to Thunk\ (1 + n)\ a
$$

Append

$$
\begin{aligned}
&(+\!\!+) : \mathit{Seq}\ a\ m \to \mathit{Seq}\ a\ n \\
&\qquad\quad \to \mathit{Thunk}\ (1 + m)\ (\mathit{Seq}\ a\ (m + n)) \\
&\text{nil} \qquad\quad +\!\!+\ ys = {}^{\checkmark}\ \mathit{return}\ ys \\
&(x :: xs) +\!\!+\ ys = {}^{\checkmark} \\
&\quad xs +\!\!+\ ys \ \gg\!= \lambda zs \to \\
&\quad \mathit{return}\ (x :: zs)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{return} : a \to \mathit{Thunk}\ 0\ a \\
&(\gg\!=)\ : \mathit{Thunk}\ m\ a \to (a \to \mathit{Thunk}\ n\ b) \\
&\qquad\qquad \to \mathit{Thunk}\ (m + n)\ b
\end{aligned}
$$

Append

$$\begin{aligned}
&(+\!\!+) : \mathit{Seq}\ a\ m \to \mathit{Seq}\ a\ n \\
&\qquad\quad \to \mathit{Thunk}\ (1 + 2 * m)\ (\mathit{Seq}\ a\ (m + n)) \\
&\text{nil} \qquad\quad +\!\!+\ ys = {}^{\checkmark}\ \mathit{return}\ ys \\
&(x :: xs) +\!\!+\ ys = {}^{\checkmark} \\
&\quad xs +\!\!+\ ys \ \ \gg\!= \lambda zs \to {}^{\checkmark} \\
&\quad \mathit{return}\ (x :: zs)
\end{aligned}$$

$$\begin{aligned}
&\mathit{return} : a \to \mathit{Thunk}\ 0\ a \\
&(\gg\!=)\ \ : \mathit{Thunk}\ m\ a \to (a \to \mathit{Thunk}\ n\ b) \\
&\qquad\qquad \to \mathit{Thunk}\ (m + n)\ b
\end{aligned}$$

Append

- Linear time to evaluate to WHNF?

$$(+\!\!+) : \ldots \to \mathit{Thunk}\ (1 + 2 * m)\ (\mathit{Seq}\ a\ (m + n))$$

- *Seq* does not contain embedded *Thunk*s.
- Non-strict sequences also possible:

$$\begin{aligned}&\textbf{data}\ S\ (a : \star)\ (c : \mathbb{N}) : \mathbb{N} \to \star\ \textbf{where}\\ &\quad [\,] \quad : S\ a\ c\ 0\\ &\quad (::)\ : a \to \mathit{Thunk}\ c\ (S\ a\ c\ n) \to S\ a\ c\ (1 + n)\end{aligned}$$

$$\begin{aligned}(+\!\!+) &: S\ a\ c\ m \to S\ a\ c\ n\\ &\quad \to \mathit{Thunk}\ 2\ (S\ a\ (3 + c)\ (m + n))\end{aligned}$$

Append

- Linear time to evaluate to WHNF?

$$(+\!\!+) : \ldots \to \mathit{Thunk}\ (1 + 2 * m)\ (\mathit{Seq}\ a\ (m + n))$$

- *Seq* does not contain embedded *Thunk*s.
- Non-strict sequences also possible:

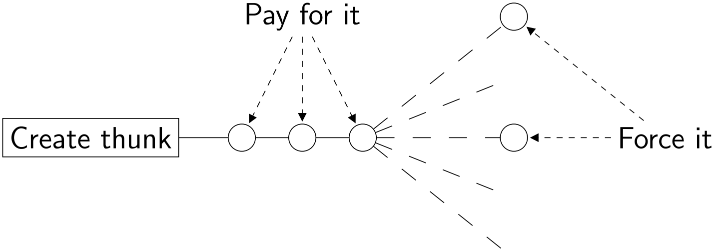
$$\begin{array}{l} \textbf{data}\ S\ (a : \star)\ (c : \mathbb{N}) : \mathbb{N} \to \star\ \textbf{where} \\ \quad [\,] \quad : S\ a\ c\ 0 \\ \quad (::) \ : a \to \mathit{Thunk}\ c\ (S\ a\ c\ n) \to S\ a\ c\ (1 + n) \end{array}$$

$$\begin{array}{l} (+\!\!+) : S\ a\ c\ m \to S\ a\ c\ n \\ \qquad \to \mathit{Thunk}\ 2\ (S\ a\ (3 + c)\ (m + n)) \end{array}$$

Essential laziness

Pay now, use later

- How can one take advantage of laziness (memoisation)?
- Let earlier operations pay for thunks which are forced later (perhaps several times):



Pay now, use later

- How can one take advantage of laziness (memoisation)?
- Let earlier operations pay for thunks which are forced later (perhaps several times):

$$
\begin{aligned}
&\mathit{pay} : (m : \mathbb{N}) \to \mathit{Thunk}\ n\ a \\
&\qquad \to \mathit{Thunk}\ m\ (\mathit{Thunk}\ (n - m)\ a)
\end{aligned}
$$

Summary

Library summary

$$\begin{aligned}
\mathit{Thunk} &: \mathbb{N} \to \star \to \star \\
\\
\checkmark &: \mathit{Thunk}\ n\ a \to \mathit{Thunk}\ (1 + n)\ a \\
\mathit{return} &: a \to \mathit{Thunk}\ 0\ a \\
(\ggg\!=) &: \mathit{Thunk}\ m\ a \to (a \to \mathit{Thunk}\ n\ b) \\
&\quad \to \mathit{Thunk}\ (m + n)\ b \\
\mathit{pay} &: (m : \mathbb{N}) \to \mathit{Thunk}\ n\ a \\
&\quad \to \mathit{Thunk}\ m\ (\mathit{Thunk}\ (n - m)\ a)
\end{aligned}$$

Correctness

- Type system proved correct with respect to annotated big-step semantics (for toy language).
- Proof developed and checked using the Agda proof assistant.

Conclusions

- Simple library/type system for analysing time complexity of lazy functional programs.
- Well-defined semantics.
- Proved correct.
- Limitations:
  - Unstable type signatures: *Thunk* $(2 + 5 * n)$ *a*.
  - Little support for aliasing.
- Applied to real-world examples.

Extra slides

Equality proofs

$$
\begin{aligned}
&(+\!\!+) : S\ a\ m \to S\ a\ n \\
&\qquad\quad \to Thunk\ (1 + 2 * m)\ (S\ a\ (m + n)) \\
&\text{nil} \qquad\ \ +\!\!+\ ys =^{\checkmark} return\ ys \\
&x ::_m xs +\!\!+ ys =^{\checkmark} \\
&\quad cast\ (lemma\ m) \\
&\qquad (xs +\!\!+ ys \gg\!= \lambda zs \to^{\checkmark} \\
&\qquad\ return\ (x :: zs))
\end{aligned}
$$

$$lemma : (m : \mathbb{N}) \to (1 + 2 * m) + 1 \equiv 2 * (1 + m)$$

Library summary

$$Thunk : \mathbb{N} \to \star \to \star$$

$$\begin{aligned}
\checkmark \quad &: Thunk\ n\ a \to Thunk\ (1+n)\ a \\
return \quad &: a \to Thunk\ 0\ a \\
(\gg=) \quad &: Thunk\ m\ a \to (a \to Thunk\ n\ b) \\
&\quad \to Thunk\ (m+n)\ b \\
pay \quad &: (m : \mathbb{N}) \to Thunk\ n\ a \\
&\quad \to Thunk\ m\ (Thunk\ (n-m)\ a)
\end{aligned}$$

$$force : Thunk\ n\ a \to a$$

Library implementation

$$Thunk\ n\ a = a$$

$$\begin{aligned}
{}^{\checkmark}x &= x \\
return\ x &= x \\
x \gg= f &= f\ x
\end{aligned}$$

$$pay\ \_\ x = x$$

$$force\ x = x$$

Essential laziness

$$\begin{array}{l}
\textbf{data}\ \mathit{Queue}\ (a : \star) : \star\ \textbf{where} \\
\quad \mathrm{empty} : \qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{Queue}\ a \\
\quad \mathrm{cons}_{10} : a \rightarrow \mathit{Queue}\ (a \times a) \qquad\quad\ \rightarrow \mathit{Queue}\ a \\
\quad \mathrm{cons}_{11} : a \rightarrow \mathit{Queue}\ (a \times a) \rightarrow a \rightarrow \mathit{Queue}\ a
\end{array}$$

$$\begin{array}{l}
\mathit{snoc} : \mathit{Queue}\ a \rightarrow a \rightarrow \mathit{Queue}\ a \\
\mathit{snoc}\ \mathrm{empty} \qquad\qquad\quad\ x_1 = \mathrm{cons}_{10}\ x_1\ \mathrm{empty} \\
\mathit{snoc}\ (\mathrm{cons}_{10}\ x_1\ xs_2) \qquad x_3 = \mathrm{cons}_{11}\ x_1\ xs_2\ x_3 \\
\mathit{snoc}\ (\mathrm{cons}_{11}\ x_1\ xs_2\ x_3)\ x_4 = \\
\quad \mathrm{cons}_{10}\ x_1\ (\mathit{snoc}\ xs_2\ (x_3, x_4))
\end{array}$$

Essential laziness

$$\begin{aligned}
&\textbf{data}\ \mathit{Queue}\ (a : \star) : \star\ \textbf{where} \\
&\quad \mathrm{cons}_{10} : a \longrightarrow \mathit{Queue}\ (a \times a) \longrightarrow \mathit{Queue}\ a
\end{aligned}$$

$$\begin{aligned}
&\mathit{snoc} : \mathit{Queue}\ a \longrightarrow a \longrightarrow \color{red}{\mathit{Thunk}\ ?}\ (\mathit{Queue}\ a) \\
&\mathit{snoc}\ \mathrm{empty} \qquad\qquad\qquad\quad x_1 = {}^{\color{red}{\checkmark}}\mathrm{cons}_{10}\ x_1\ \mathrm{empty} \\
&\mathit{snoc}\ (\mathrm{cons}_{10}\ x_1\ \mathit{xs}_2) \qquad x_3 = {}^{\color{red}{\checkmark}}\mathrm{cons}_{11}\ x_1\ \mathit{xs}_2\ x_3 \\
&\mathit{snoc}\ (\mathrm{cons}_{11}\ x_1\ \mathit{xs}_2\ x_3)\ x_4 = {}^{\color{red}{\checkmark}} \\
&\quad \mathrm{cons}_{10}\ x_1\ (\mathit{snoc}\ \mathit{xs}_2\ (x_3, x_4))
\end{aligned}$$

Essential laziness

$$\begin{array}{l}
\textbf{data}\ \mathit{Queue}\ (a : \star) : \star\ \textbf{where} \\
\quad \mathrm{cons}_{10} : a \longrightarrow \mathit{Queue}\ (a \times a) \longrightarrow \mathit{Queue}\ a
\end{array}$$

$$\begin{array}{ll}
\multicolumn{2}{l}{\mathit{snoc} : \mathit{Queue}\ a \longrightarrow a \longrightarrow \color{red}{\mathit{Thunk}\ ?}\ (\mathit{Queue}\ a)} \\
\mathit{snoc}\ \mathrm{empty} & x_1 = ^{\color{red}\checkmark} \color{red}{\mathit{return}}\ (\mathrm{cons}_{10}\ x_1\ \mathrm{empty}) \\
\mathit{snoc}\ (\mathrm{cons}_{10}\ x_1\ \mathit{xs}_2) & x_3 = ^{\color{red}\checkmark} \color{red}{\mathit{return}}\ (\mathrm{cons}_{11}\ x_1\ \mathit{xs}_2\ x_3) \\
\mathit{snoc}\ (\mathrm{cons}_{11}\ x_1\ \mathit{xs}_2\ x_3) & x_4 = ^{\color{red}\checkmark} \\
\quad \color{red}{\mathit{return}}\ (\mathrm{cons}_{10}\ x_1\ (\mathit{snoc}\ \mathit{xs}_2\ (x_3, x_4))) &
\end{array}$$

Essential laziness

$$
\begin{array}{l}
\textbf{data}\ \mathit{Queue}\ (a : \star) : \star\ \textbf{where} \\
\quad \mathrm{cons}_{10} : a \to \mathit{Thunk}\ ?\ (\mathit{Queue}\ (a \times a)) \to \mathit{Queue}\ a
\end{array}
$$

$$
\begin{array}{l}
\mathit{snoc} : \mathit{Queue}\ a \to a \to \mathit{Thunk}\ ?\ (\mathit{Queue}\ a) \\
\mathit{snoc}\ \mathrm{empty} \qquad\qquad\qquad\quad x_1 = \checkmark \\
\quad \mathit{return}\ (\mathrm{cons}_{10}\ x_1\ (\mathit{return}\ \mathrm{empty})) \\
\mathit{snoc}\ (\mathrm{cons}_{10}\ x_1\ xs_2) \qquad x_3 = \checkmark\ \mathit{return}\ (\mathrm{cons}_{11}\ x_1\ xs_2\ x_3) \\
\mathit{snoc}\ (\mathrm{cons}_{11}\ x_1\ xs_2\ x_3)\ x_4 = \checkmark \\
\quad \mathit{return}\ (\mathrm{cons}_{10}\ x_1\ (\mathit{snoc}\ xs_2\ (x_3, x_4)))
\end{array}
$$

Essential laziness

$$
\begin{array}{l}
\textbf{data}\ \mathit{Queue}\ (a : \star) : \star\ \textbf{where} \\
\quad \mathrm{cons}_{10} : a \to \color{red}{\mathit{Thunk}\ ?}\ (\mathit{Queue}\ (a \times a)) \to \mathit{Queue}\ a
\end{array}
$$

$$
\begin{array}{l}
\mathit{snoc} : \mathit{Queue}\ a \to a \to \color{red}{\mathit{Thunk}\ ?}\ (\mathit{Queue}\ a) \\
\mathit{snoc}\ \mathrm{empty} \qquad\qquad\qquad x_1 = \color{red}{\checkmark} \\
\quad \color{red}{\mathit{return}}\ (\mathrm{cons}_{10}\ x_1\ (\color{red}{\mathit{return}}\ \mathrm{empty})) \\
\mathit{snoc}\ (\mathrm{cons}_{10}\ x_1\ xs_2) \quad x_3 = \color{red}{\checkmark} \\
\quad xs_2\ \color{red}{\ggg\!=\ \lambda xs_2' \to \checkmark} \\
\quad \color{red}{\mathit{return}}\ (\mathrm{cons}_{11}\ x_1\ \color{red}{xs_2'}\ x_3) \\
\mathit{snoc}\ (\mathrm{cons}_{11}\ x_1\ xs_2\ x_3)\ x_4 = \color{red}{\checkmark} \\
\quad \color{red}{\mathit{return}}\ (\mathrm{cons}_{10}\ x_1\ (\mathit{snoc}\ xs_2\ (x_3, x_4)))
\end{array}
$$

Essential laziness

$$
\begin{array}{l}
\textbf{data}\ \mathit{Queue}\ (a : \star) : \star\ \textbf{where} \\
\quad \mathrm{cons}_{10} : a \longrightarrow \mathit{Thunk}\ ?\ (\mathit{Queue}\ (a \times a)) \longrightarrow \mathit{Queue}\ a \\
\\
\mathit{snoc} : \mathit{Queue}\ a \longrightarrow a \longrightarrow \mathit{Thunk}\ ?\ (\mathit{Queue}\ a) \\
\mathit{snoc}\ \mathrm{empty} \qquad\qquad\qquad x_1 = \checkmark \\
\quad \mathit{return}\ (\mathrm{cons}_{10}\ x_1\ (\mathit{return}\ \mathrm{empty})) \\
\mathit{snoc}\ (\mathrm{cons}_{10}\ x_1\ \mathit{xs}_2) \quad x_3 = \checkmark \\
\quad \mathit{xs}_2 \ggg \lambda \mathit{xs}'_2 \rightarrow \checkmark \\
\quad \mathit{return}\ (\mathrm{cons}_{11}\ x_1\ \mathit{xs}'_2\ x_3) \\
\mathit{snoc}\ (\mathrm{cons}_{11}\ x_1\ \mathit{xs}_2\ x_3)\ x_4 = \checkmark \\
\quad \mathit{pay}\ ?\ (\mathit{snoc}\ \mathit{xs}_2\ (x_3, x_4)) \ggg \lambda \mathit{xs}_{234} \rightarrow \checkmark \\
\quad \mathit{return}\ (\mathrm{cons}_{10}\ x_1\ \mathit{xs}_{234})
\end{array}
$$

Essential laziness

$$\mathbf{data}\ \mathit{Queue}\ (a : \star) : \star\ \mathbf{where}$$
$$\quad \mathrm{cons}_{10} : a \to \mathit{Thunk}\ 2\ (\mathit{Queue}\ (a \times a)) \to \mathit{Queue}\ a$$

$$\mathit{snoc} : \mathit{Queue}\ a \to a \to \mathit{Thunk}\ 4\ (\mathit{Queue}\ a)$$
$$\mathit{snoc}\ \mathrm{empty}\qquad x_1 = \checkmark\checkmark\checkmark\checkmark$$
$$\quad \mathit{return}\ (\mathrm{cons}_{10}\ x_1\ (\checkmark\checkmark\ \mathit{return}\ \mathrm{empty}))$$
$$\mathit{snoc}\ (\mathrm{cons}_{10}\ x_1\ \mathit{xs}_2)\qquad x_3 = \checkmark$$
$$\quad \mathit{xs}_2 \ggg \lambda \mathit{xs}'_2 \to \checkmark$$
$$\quad \mathit{return}\ (\mathrm{cons}_{11}\ x_1\ \mathit{xs}'_2\ x_3)$$
$$\mathit{snoc}\ (\mathrm{cons}_{11}\ x_1\ \mathit{xs}_2\ x_3)\ x_4 = \checkmark$$
$$\quad \mathit{pay}\ 2\ (\mathit{snoc}\ \mathit{xs}_2\ (x_3, x_4)) \ggg \lambda \mathit{xs}_{234} \to \checkmark$$
$$\quad \mathit{return}\ (\mathrm{cons}_{10}\ x_1\ \mathit{xs}_{234})$$