Asymptotic Speedup via Effect Handlers

Daniel Hillerström

Laboratory for Foundations of Computer Science
The University of Edinburgh, Scotland, UK

December 13, 2022

Formal Analysis, Theory and Algorithms
School of Computing Science
University of Glasgow

(joint work with Sam Lindley and John Longley)

Effects for Efficiency

Asymptotic Speedup with First-Class Control

DANIEL HILLERSTRÖM, The University of Edinburgh, UK
SAM LINDLEY, The University of Edinburgh and Imperial College London and Heriot-Watt University, UK
JOHN LONGLEY, The University of Edinburgh, UK

We study the fundamental efficiency of delimited control. Specifically, we show that effect handlers enable an asymptotic improvement in runtime complexity for a certain class of functions. We consider the *generic count* problem using a pure PCF-like base language λ_b and its extension with effect handlers λ_h. We show that λ_h admits an asymptotically more efficient implementation of generic count than any λ_b implementation. We also show that this efficiency gap remains when λ_b is extended with mutable state.

To our knowledge this result is the first of its kind for control operators.

CCS Concepts: • **Theory of computation** → **Lambda calculus**; **Abstract machines**; **Control primitives**.

Additional Key Words and Phrases: effect handlers, asymptotic complexity analysis, generic search

ACM Reference Format:
Daniel Hillerström, Sam Lindley, and John Longley. 2020. Effects for Efficiency: Asymptotic Speedup with First-Class Control. *Proc. ACM Program. Lang.* 4, ICFP, Article 100 (August 2020), 29 pages. https://doi.org/10.1145/3408982

100

JFP, 52 pages, 2022.

doi:10.1017/xxxxx

Asymptotic Speedup via Effect Handlers

DANIEL HILLERSTRÖM

Laboratory for Foundations of Computer Science
The University of Edinburgh, Scotland, UK
(e-mail: daniel.hillerstrom@ed.ac.uk)

SAM LINDLEY

Laboratory for Foundations of Computer Science
The University of Edinburgh, Scotland, UK
(e-mail: sam.lindley@ed.ac.uk)

JOHN LONGLEY

Laboratory for Foundations of Computer Science
The University of Edinburgh, Scotland, UK
(e-mail: jrl@staffmail.ed.ac.uk)

Abstract

We study a fundamental efficiency benefit afforded by delimited control, showing that for certain higher-order functions, a language with advanced control features offers an asymptotic improvement in runtime over a language without them. Specifically, we consider the *generic count* problem in the context of a pure PCF-like base language λ_{b} and an extension λ_{h} with general *effect handlers*. We prove that λ_{h} admits an asymptotically more efficient implementation of generic count than any implementation in λ_{b}. We also show that this gap remains even when λ_{b} is extended to a language λ_{a} with *affine effect handlers*, which is strong enough to encode exceptions, local state, coroutines and single-shot continuations. This locates the efficiency difference in the gap between 'single-shot' and

Effect handlers primer

Effect handlers provide a request-response paradigm-style of programming

- Perform an abstract request: do $\ell\ V$ (Plotkin and Power 2003)
- Respond to requests in some computation: handle M with H (Plotkin and Pretnar 2009)

Effect handlers primer

Effect handlers provide a request-response paradigm-style of programming

- Perform an abstract request: do $\ell\ V$ (Plotkin and Power 2003)
- Respond to requests in some computation: handle M with H (Plotkin and Pretnar 2009)

Example: Count the number of true valuations.
One request operation $\mathsf{Branch} : \mathsf{Unit} \to \mathsf{Bool}$.

$$(\mathsf{do}\ \mathsf{Branch}\ \langle\rangle \ ||\ \mathsf{do}\ \mathsf{Branch}\ \langle\rangle)$$

Effect handlers primer

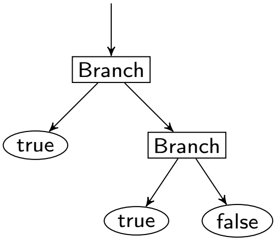
Effect handlers provide a request-response paradigm-style of programming

- Perform an abstract request: $\mathsf{do}\ \ell\ V$ (Plotkin and Power 2003)
- Respond to requests in some computation: $\mathsf{handle}\ M\ \mathsf{with}\ H$ (Plotkin and Pretnar 2009)

Example: Count the number of true valuations.
One request operation $\mathsf{Branch} : \mathsf{Unit} \to \mathsf{Bool}$.

$$(\mathsf{do}\ \mathsf{Branch}\ \langle\rangle \mid\mid \mathsf{do}\ \mathsf{Branch}\ \langle\rangle)$$

Computation tree model



Effect handlers primer

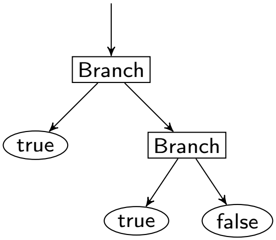
Effect handlers provide a request-response paradigm-style of programming

- Perform an abstract request: do $\ell\ V$ (Plotkin and Power 2003)
- Respond to requests in some computation: handle M with H (Plotkin and Pretnar 2009)

Example: Count the number of true valuations.
One request operation Branch : Unit $\rightarrow$ Bool.

handle (do Branch $\langle\rangle$ || do Branch $\langle\rangle$) with

Computation tree model



Effect handlers primer

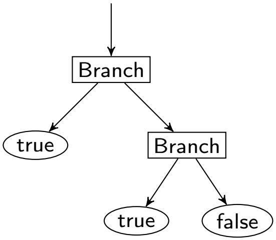
Effect handlers provide a request-response paradigm-style of programming

- Perform an abstract request: do $\ell\ V$ (Plotkin and Power 2003)
- Respond to requests in some computation: handle M with H (Plotkin and Pretnar 2009)

Example: Count the number of true valuations.
One request operation Branch : Unit $\to$ Bool.

$$\begin{array}{l} \text{handle}\ (\text{do Branch}\ \langle\rangle \mathbin{||} \text{do Branch}\ \langle\rangle)\ \text{with} \\ \quad \text{val}\ x \qquad\qquad\qquad \mapsto \text{if}\ x\ \text{then}\ 1\ \text{else}\ 0 \end{array}$$

Computation tree model



Effect handlers primer

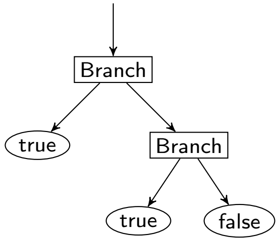
Effect handlers provide a request-response paradigm-style of programming

- Perform an abstract request: do $\ell\ V$ (Plotkin and Power 2003)
- Respond to requests in some computation: handle M with H (Plotkin and Pretnar 2009)

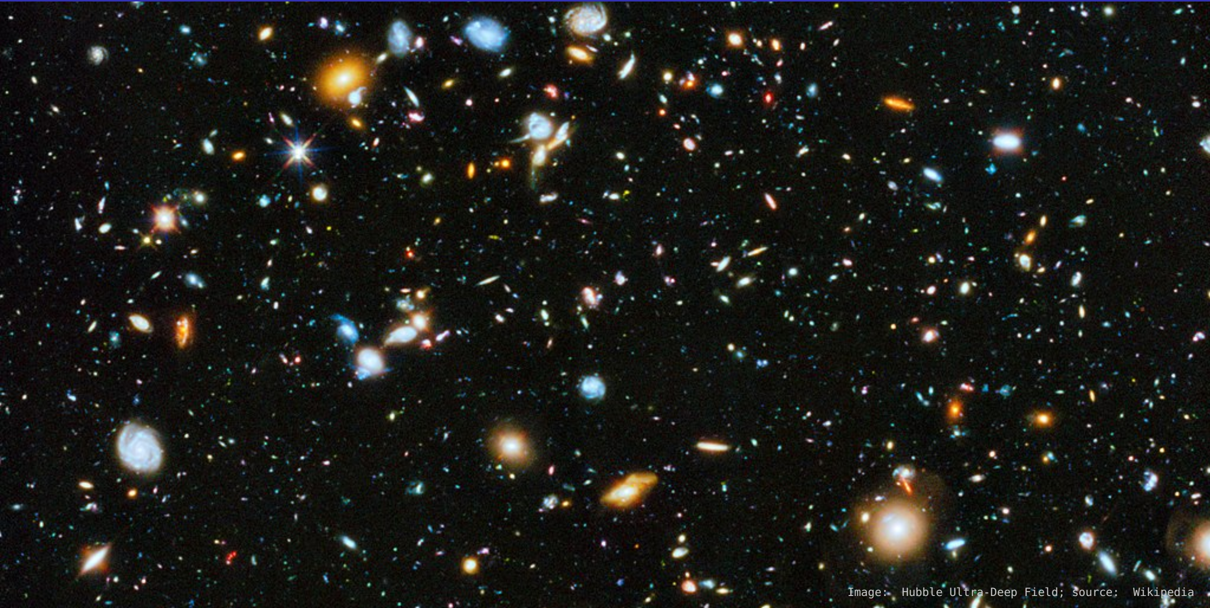
Example: Count the number of true valuations.
One request operation Branch : Unit $\to$ Bool.

$$\begin{array}{l}
\mathsf{handle}\ (\mathsf{do\ Branch}\ \langle\rangle \mid\mid \mathsf{do\ Branch}\ \langle\rangle)\ \mathsf{with} \\
\quad \begin{array}{lcl}
\mathsf{val}\ x & \mapsto & \mathsf{if}\ x\ \mathsf{then}\ 1\ \mathsf{else}\ 0 \\
\mathsf{Branch}\ \langle\rangle\ \mathit{resume} & \mapsto & \mathit{resume}\ \mathsf{true} + \mathit{resume}\ \mathsf{false}
\end{array}
\end{array}$$

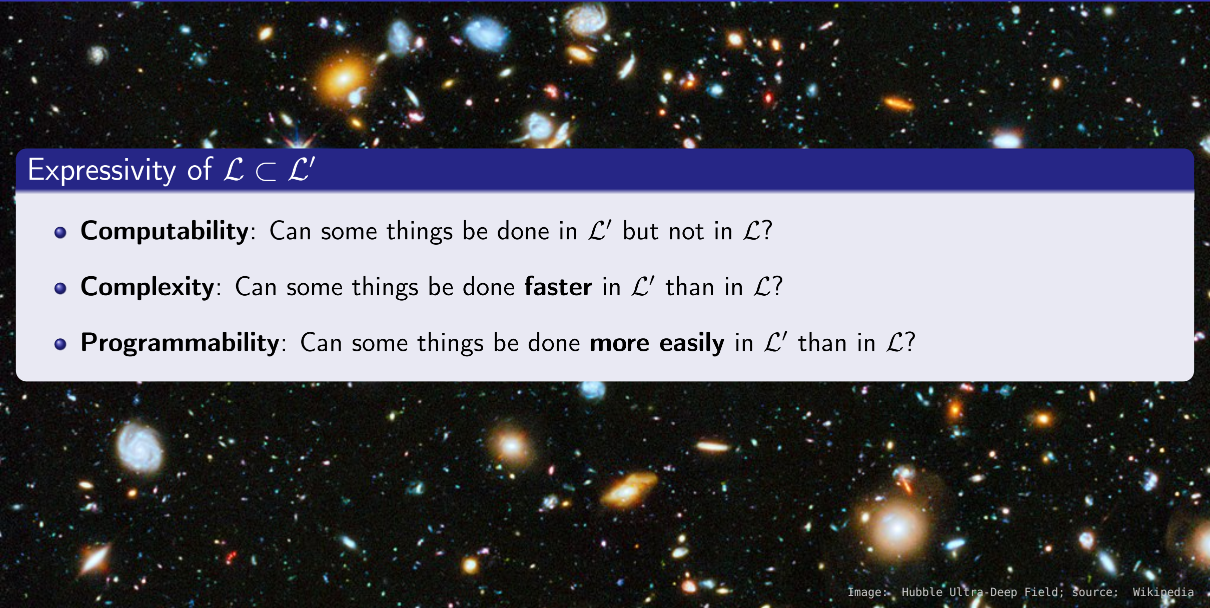
Computation tree model



Motivation: space exploration



Motivation: space exploration



Expressivity of $\mathcal{L} \subset \mathcal{L}'$

- **Computability**: Can some things be done in $\mathcal{L}'$ but not in $\mathcal{L}$?
- **Complexity**: Can some things be done **faster** in $\mathcal{L}'$ than in $\mathcal{L}$?
- **Programmability**: Can some things be done **more easily** in $\mathcal{L}'$ than in $\mathcal{L}$?

Image: Hubble Ultra-Deep Field; source: Wikipedia

Motivation: space exploration

Asymptotic speedup with effect handlers

The **generic count** problem

$$\mathrm{count}_n : ((\mathrm{Nat}_n \to \mathrm{Bool}) \to \mathrm{Bool}) \to \mathrm{Nat}$$

Image: Hubble Ultra-Deep Field; source: Wikipedia

Motivation: space exploration

Asymptotic speedup with effect handlers

The **generic count** problem

$$\mathsf{count}_n : \underbrace{((\mathsf{Nat}_n \to \mathsf{Bool})}_{\mathsf{point}} \to \mathsf{Bool}) \to \mathsf{Nat}$$

- point: boolean-valued vector of size n

Image: Hubble Ultra-Deep Field; source: Wikipedia

Motivation: space exploration

Asymptotic speedup with effect handlers

The **generic count** problem

$$\mathsf{count}_n : \underbrace{(\underbrace{(\mathsf{Nat}_n \to \mathsf{Bool})}_{\text{point}} \to \mathsf{Bool})}_{\text{predicate}} \to \mathsf{Nat}$$

- point: boolean-valued vector of size n
- predicate: encodes some search problem (e.g. n-Queens)

Image: Hubble Ultra-Deep Field; source: Wikipedia

Motivation: space exploration

Asymptotic speedup with effect handlers

The **generic count** problem

$$\mathsf{count}_n : \underbrace{(\underbrace{(\mathsf{Nat}_n \to \mathsf{Bool})}_{\text{point}} \to \mathsf{Bool})}_{\text{predicate}} \to \mathsf{Nat}$$

- point: boolean-valued vector of size n
- predicate: encodes some search problem (e.g. n-Queens)
- $\mathsf{count}_n\ P$: returns number of n-points satisfying P

Motivation: space exploration

Asymptotic speedup with effect handlers

The **generic count** problem

$$\mathrm{count}_n : \underbrace{(\underbrace{(\mathrm{Nat}_n \to \mathrm{Bool})}_{\text{point}} \to \mathrm{Bool})}_{\text{predicate}} \to \mathrm{Nat}$$

- point: boolean-valued vector of size n
- predicate: encodes some search problem (e.g. n-Queens)
- count_n P: returns number of n-points satisfying P

Fix $\mathcal{L} := \mathrm{PCF}$

Image: Hubble Ultra-Deep Field; source: Wikipedia

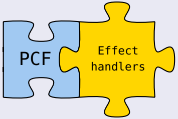
Motivation: space exploration

Asymptotic speedup with effect handlers

The **generic count** problem

$$\mathsf{count}_n : (\underbrace{\underbrace{(\mathsf{Nat}_n \to \mathsf{Bool})}_{\text{point}} \to \mathsf{Bool})}_{\text{predicate}} \to \mathsf{Nat}$$

- point: boolean-valued vector of size n
- predicate: encodes some search problem (e.g. n-Queens)
- $\mathsf{count}_n\ P$: returns number of n-points satisfying P



Fix $\mathcal{L} := \mathrm{PCF}$ and $\mathcal{L}' := \mathrm{PCF}_\mathrm{h}$ with effect handlers

Image: Hubble Ultra-Deep Field; source: Wikipedia

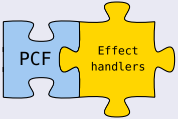
Motivation: space exploration

Asymptotic speedup with effect handlers

The **generic count** problem

$$\mathrm{count}_n : \underbrace{(\underbrace{(\mathrm{Nat}_n \to \mathrm{Bool})}_{\text{point}} \to \mathrm{Bool})}_{\text{predicate}} \to \mathrm{Nat}$$

- point: boolean-valued vector of size n
- predicate: encodes some search problem (e.g. n-Queens)
- $\mathrm{count}_n\ P$: returns number of n-points satisfying P



Fix $\mathcal{L} := \mathrm{PCF}$ and $\mathcal{L}' := \mathrm{PCF}_\mathrm{h}$ with effect handlers

1. There **exists** an implementation, effcount $\in \mathrm{PCF}_\mathrm{h}$, of generic count such that effcount $\in \mathcal{O}(2^n)$

Image: Hubble Ultra-Deep Field; source: Wikipedia

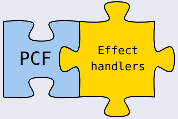
Motivation: space exploration

Asymptotic speedup with effect handlers

The **generic count** problem

$$\mathsf{count}_n : \underbrace{(\underbrace{(\mathsf{Nat}_n \to \mathsf{Bool})}_{\mathsf{point}} \to \mathsf{Bool})}_{\mathsf{predicate}} \to \mathsf{Nat}$$

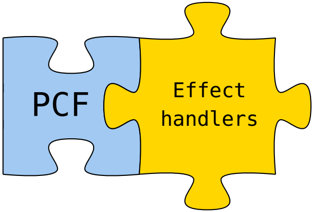
- point: boolean-valued vector of size n
- predicate: encodes some search problem (e.g. n-Queens)
- $\mathsf{count}_n\ P$: returns number of n-points satisfying P



Fix $\mathcal{L} := \mathrm{PCF}$ and $\mathcal{L}' := \mathrm{PCF}_\mathrm{h}$ with effect handlers

1. There **exists** an implementation, $\mathsf{effcount} \in \mathrm{PCF}_\mathrm{h}$, of generic count such that $\mathsf{effcount} \in \mathcal{O}(2^n)$
2. **For all** implementations, $\mathsf{count} \in \mathrm{PCF}$, of generic count it holds that $\mathsf{count} \in \Omega(n2^n)$

Methodology



One ground rule:

No change of type signatures is allowed!

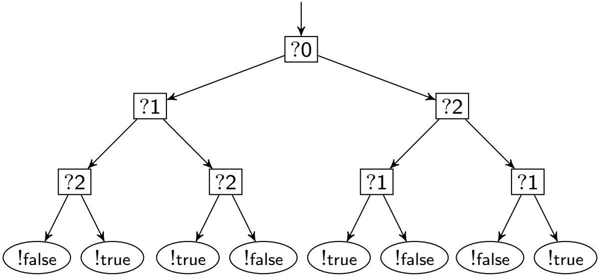
- Fixed signature $\mathsf{count}_n : ((\mathsf{Nat}_n \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat}$
- Prohibits translation of PCF_h into PCF (interpreter / CPS)
- Programming against a fixed interface

(PCF due to Plotkin (1977))

A predicate and its model

$$\begin{aligned}
&\mathsf{ex} : (\mathsf{Nat}_3 \to \mathsf{Bool}) \to \mathsf{Bool} \\
&\mathsf{ex} \doteq \lambda p.\ \mathsf{if}\ p\,0\ \mathsf{then}\ p\,1\ \mathsf{xor}\ p\,2 \\
&\qquad\qquad\ \mathsf{else}\ \mathsf{not}\,(p\,2\ \mathsf{xor}\ p\,1)
\end{aligned}$$

Behaviour of ex $(\lambda j.\mathsf{nth}\,[\mathsf{true}, \mathsf{false}, \mathsf{true}]\,j)$:

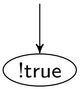


More predicates, more models (1)

Consider a constant predicate, e.g.

$$\mathsf{T}_0 \doteq \lambda q.\mathsf{true}$$

whose model is



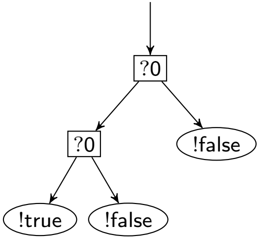
Potential problem: the runtime of the predicate doesn't depend on the input q.

More predicates, more models (2)

Consider the following identity predicate over $\mathbb{B}^1 \to \mathbb{B}$

$$\mathsf{I}_2 \doteq \lambda q.(q\,0)\,\&\&\,(q\,0)$$

whose model is



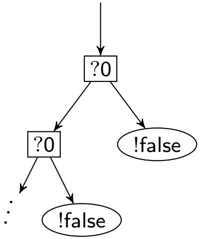
Potential problem: Repeated queries may yield imperfect binary tree models.

More predicates, more models (3)

Consider the following false-only yielding predicate

$$\infty \doteq \text{rec } P\,q.\text{if } q\,0 \text{ then } P\,q \text{ else false}$$

whose model is infinite



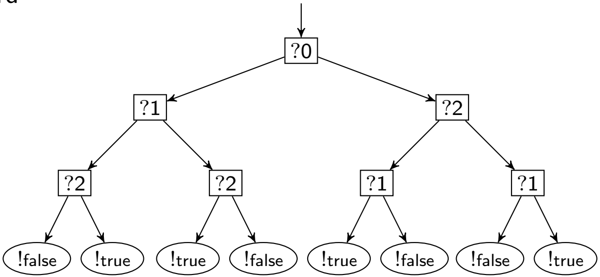
Potential problem: Possibly infinite runtime.

Restriction to n-standard predicates

Properties of an n-standard model

- Perfect binary tree of height $n > 0$
- Contains every query $?j$ for $j \in \{0, \ldots, n-1\}$
- No repeated queries along any path

Example: ex is 3-standard



Definition (untimed decision tree)

1. The address set Addr is simply the set $\mathbb{B}^*$ of finite lists of booleans. If $bs, bs' \in \text{Addr}$, we write $bs \sqsubseteq bs'$ (resp. $bs \sqsubset bs'$) to mean that bs is a prefix (resp. proper prefix) of bs'.
2. The label set Lab consists of **queries** parameterised by a natural number and **answers** parameterised by a boolean:

$$\text{Lab} \doteq \{?k \mid k \in \mathbb{N}\} \cup \{!b \mid b \in \mathbb{B}\}$$

3. An (untimed) decision tree is a partial function $\tau : \text{Addr} \rightharpoonup \text{Lab}$ such that:
   - The domain of τ (written $dom(\tau)$) is prefix closed.
   - Answer nodes are always leaves: if $\tau(bs) = !b$ then $\tau(bs')$ is undefined whenever $bs \sqsubset bs'$.

Definition (timed decision tree)

A timed decision tree is a partial function $\tau : \text{Addr} \rightharpoonup \text{Lab} \times \mathbb{N}$ such that its first projection $bs \mapsto \tau(bs).1$ is a decision tree. We write $\text{labs}(\tau)$ for the first projection $(bs \mapsto \tau(bs).1)$ and $\text{steps}(\tau)$ for the second projection $(bs \mapsto \tau(bs).2)$ of a timed decision tree.

Restriction to n-standard trees and predicates, formally

Definition (n-standard trees and predicates)

An n-predicate tree τ is said to be ***n*-standard** if the following hold:

- The domain of τ is precisely Addr_n, the set of bit vectors of length $\leq n$.
- There are no repeated queries along any path in τ:

$$\forall bs, bs' \in dom(\tau),\ k \in \mathbb{N}_n.\ bs \sqsubseteq bs' \wedge \tau(bs) = \tau(bs') = ?k \Rightarrow bs = bs'$$

A timed decision tree τ is n-standard if its underlying untimed decision tree $(bs \mapsto \tau(bs).1)$ is so. An n-predicate P is n-standard if its model is n-standard.

Canonical n-predicates

Definition (canonical n-standard predicates)

Given an n-standard tree τ, we may associate to each address $bs \in dom(\tau)$ a λ_b term $T(\tau, bs)$ (with free variable $q : (\mathrm{Nat}_n \to \mathrm{Bool})$) by reverse induction on the length of bs:

$$\begin{array}{lcll} T(\tau, bs) & \doteq & b & \text{if } \tau(bs) = !b \\ T(\tau, bs) & \doteq & \text{if } q(k) \text{ then } T(\tau, bs +\!\!+ [\text{true}]) \text{ else } T(\tau, bs +\!\!+ [\text{false}]) & \text{if } \tau(bs) = ?k \end{array}$$

We then define

$$P(\tau) \doteq \lambda q.\ T(\tau, [])$$

such that model of $P(\tau)$ is τ, and call $P(\tau)$ the **canonical n-standard predicate** for τ.

Example: Canonicalising ex

$$\begin{aligned}
&\mathsf{ex} : (\mathsf{Nat}_3 \to \mathsf{Bool}) \to \mathsf{Bool} \\
&\mathsf{ex} \doteq \lambda p.\ \mathsf{if}\ p\,0\ \mathsf{then}\ p\,1\ \mathsf{xor}\ p\,2 \\
&\qquad\qquad\ \mathsf{else}\ \mathsf{not}\,(p\,2\ \mathsf{xor}\ p\,1)
\end{aligned}
\quad = \quad
\begin{aligned}
&\mathsf{ex} : (\mathsf{Nat}_3 \to \mathsf{Bool}) \to \mathsf{Bool} \\
&\mathsf{ex} \doteq \lambda p.\ \mathsf{if}\ p\,0\ \mathsf{then} \\
&\qquad\qquad\quad \mathsf{if}\ p\,1\ \mathsf{then} \\
&\qquad\qquad\qquad \mathsf{if}\ p\,2\ \mathsf{then}\ \mathsf{true}\ \mathsf{xor}\ \mathsf{true} \\
&\qquad\qquad\qquad \mathsf{else}\ \mathsf{true}\ \mathsf{xor}\ \mathsf{false} \\
&\qquad\qquad\quad \mathsf{else} \\
&\qquad\qquad\qquad \mathsf{if}\ p\,2\ \mathsf{then}\ \mathsf{false}\ \mathsf{xor}\ \mathsf{true} \\
&\qquad\qquad\qquad \mathsf{else}\ \mathsf{false}\ \mathsf{xor}\ \mathsf{false} \\
&\qquad\qquad \mathsf{else} \\
&\qquad\qquad\quad \mathsf{if}\ p\,2\ \mathsf{then} \\
&\qquad\qquad\qquad \mathsf{if}\ p\,1\ \mathsf{then}\ \mathsf{not}\,(\mathsf{true}\ \mathsf{xor}\ \mathsf{true}) \\
&\qquad\qquad\qquad \mathsf{else}\ \mathsf{not}\,(\mathsf{true}\ \mathsf{xor}\ \mathsf{false}) \\
&\qquad\qquad\quad \mathsf{else} \\
&\qquad\qquad\qquad \mathsf{if}\ p\,1\ \mathsf{then}\ \mathsf{not}\,(\mathsf{false}\ \mathsf{xor}\ \mathsf{true}) \\
&\qquad\qquad\qquad \mathsf{else}\ \mathsf{not}\,(\mathsf{false}\ \mathsf{xor}\ \mathsf{false})
\end{aligned}$$

Specification of generic counting

Definition (n-points)

A closed value $Q : (\mathsf{Nat}_n \to \mathsf{Bool})$ is said to be a **syntactic n-point** if:

$$\forall k \in \mathbb{N}_n. \exists b \in \mathbb{B}.\ Q\ k \rightsquigarrow^* b$$

A **semantic n-point** π is a mathematical function $\pi : \mathbb{N}_n \to \mathbb{B}$. Any syntactic n-point Q is said to **denote** the semantic n-point $\mathbb{P}[\![Q]\!]$ given by:

$$\forall k \in \mathbb{N}_n,\ b \in \mathbb{B}.\ \mathbb{P}[\![Q]\!](k) = b \Leftrightarrow\ Q\ k \rightsquigarrow^* b$$

Any two syntactic n-points Q and Q' are said to be **distinct** if $\mathbb{P}[\![Q]\!] \neq \mathbb{P}[\![Q']\!]$.

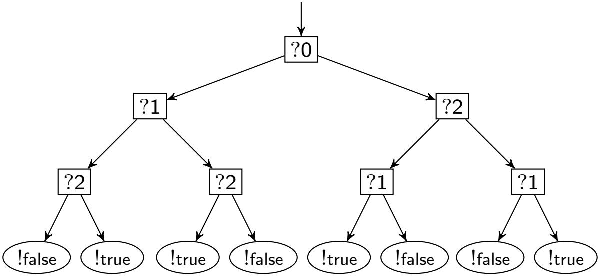
Definition (Generic count specification)

1. The **count** of a semantic n-predicate Π, written $\sharp\Pi$, is simply the number of semantic n-points $\pi \in \mathbb{B}^n$ for which $\Pi(\pi) = \mathsf{true}$.
2. If P is any n-predicate, we say that K **correctly counts** P if $K\ P \rightsquigarrow^* m$, where $m = \sharp\mathbb{P}[\![P]\!]$.

Efficient generic count with effect handlers

$$\mathsf{effcount} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat}$$
$$\mathsf{effcount} \doteq$$

Behaviour of effcount ex:

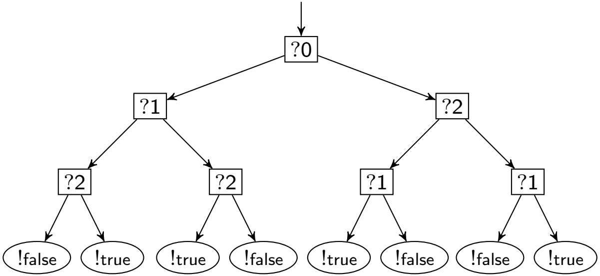


Efficient generic count with effect handlers

$$\begin{array}{l} \mathsf{effcount} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\ \mathsf{effcount} \doteq \lambda P. \qquad\qquad P\,(\lambda j.\mathsf{do}\ \mathsf{Branch}\ \langle\rangle) \end{array}$$

(where $\mathsf{Branch} : \langle\rangle \to \mathsf{Bool} \in \Sigma$)

Behaviour of effcount ex:

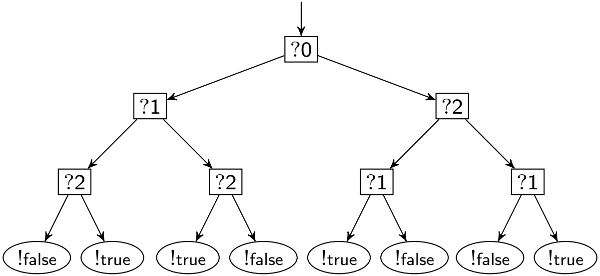


Efficient generic count with effect handlers

$$\begin{aligned}
&\mathsf{effcount} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
&\mathsf{effcount} \doteq \lambda P.\ \mathsf{handle}\ P\,(\lambda j.\mathsf{do}\ \mathsf{Branch}\ \langle\rangle)\ \mathsf{with}
\end{aligned}$$

(where $\mathsf{Branch} : \langle\rangle \to \mathsf{Bool} \in \Sigma$)

Behaviour of effcount ex:

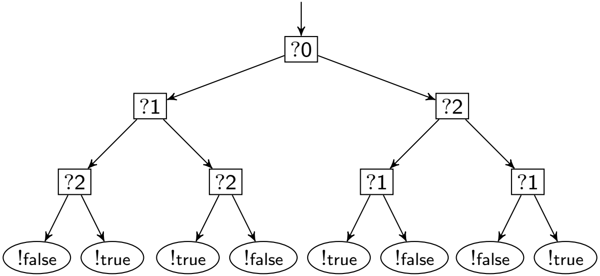


Efficient generic count with effect handlers

$$\begin{array}{l}\text{effcount} : ((\text{Nat} \to \text{Bool}) \to \text{Bool}) \to \text{Nat} \\ \text{effcount} \doteq \lambda P.\ \text{handle}\ P\,(\lambda j.\text{do Branch}\,\langle\rangle)\ \text{with} \\ \qquad\qquad\qquad \text{val}\ \mathit{ans} \qquad\qquad \mapsto \text{if}\ \mathit{ans}\ \text{then 1 else 0}\end{array}$$

(where Branch : $\langle\rangle \to \text{Bool} \in \Sigma$)

Behaviour of effcount ex:

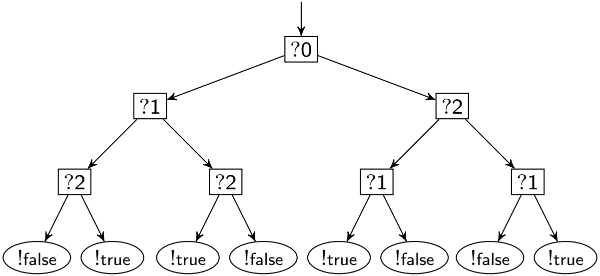


Efficient generic count with effect handlers

$$\begin{array}{l}
\mathsf{effcount} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
\mathsf{effcount} \doteq \lambda P.\ \mathsf{handle}\ P\,(\lambda j.\mathsf{do}\ \mathsf{Branch}\,\langle\rangle)\ \mathsf{with} \\
\qquad\qquad\qquad \begin{array}{ll}
\mathsf{val}\ \mathit{ans} & \mapsto \mathsf{if}\ \mathit{ans}\ \mathsf{then}\ 1\ \mathsf{else}\ 0 \\
\mathsf{Branch}\ \langle\rangle\ \mathit{resume} & \mapsto \mathit{resume}\,\mathsf{true} + \mathit{resume}\,\mathsf{false}
\end{array}
\end{array}$$

(where $\mathsf{Branch} : \langle\rangle \to \mathsf{Bool} \in \Sigma$)

Behaviour of effcount ex:

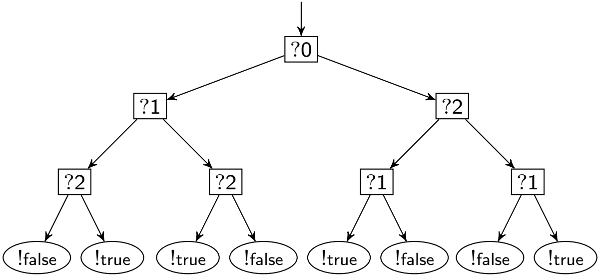


Efficient generic count with effect handlers

$$
\begin{aligned}
&\mathsf{effcount} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat}\\
&\mathsf{effcount} \doteq \lambda P.\ \mathsf{handle}\ P\,(\lambda j.\mathsf{do\ Branch}\,\langle\rangle)\ \mathsf{with}\\
&\qquad\qquad\qquad \mathsf{val}\ \mathit{ans} \qquad\qquad\quad\ \mapsto \mathsf{if}\ \mathit{ans}\ \mathsf{then}\ 1\ \mathsf{else}\ 0\\
&\qquad\qquad\qquad \mathsf{Branch}\ \langle\rangle\ \mathit{resume} \mapsto \mathit{resume}\,\mathsf{true} + \mathit{resume}\,\mathsf{false}
\end{aligned}
$$

(where $\mathsf{Branch} : \langle\rangle \to \mathsf{Bool} \in \Sigma$)

Behaviour of effcount ex:



Steps: $\mathcal{O}(2^n)$

Efficient generic count theorem

Theorem

The following hold for any $n \in \mathbb{N}$ and any n-standard predicate P of $\mathsf{PCF_h}$*:*

1. effcount *correctly counts* P*.*
2. *The number of steps required to evaluate* effcount P *is*

$$\left(\sum_{bs \in \mathsf{Addr}_n} \mathsf{steps}(\mathcal{T}(P))(bs) \right) + \mathcal{O}(2^n)$$

Proof.

By labourious backwards induction on bs □

Naïve count

The naïve approach applies P to all 2^n possible points.

$$
\begin{aligned}
\mathsf{naivecount}_n &: ((\mathsf{Nat}_n \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
\mathsf{naivecount}_n &\doteq \lambda P.\mathsf{count}\, n\, (\lambda i.\bot) \\
&\quad \text{where } \mathsf{count}\, 0 \quad\quad\; p \doteq \mathsf{if}\ P\ p\ \mathsf{then}\ 1\ \mathsf{else}\ 0 \\
&\quad\quad\quad\;\; \mathsf{count}\, (1+n)\ p \doteq \quad \mathsf{count}\, n\, (\lambda i.\mathsf{if}\ i = n\ \mathsf{then\ true\ else}\ p\, i) \\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad\;\; + \mathsf{count}\, n\, (\lambda i.\mathsf{if}\ i = n\ \mathsf{then\ false\ else}\ p\, i)
\end{aligned}
$$

Here $(\lambda i.\bot)$ is the divergent point.

Naïve count

The naïve approach applies P to all 2^n possible points.

$$
\begin{aligned}
\mathsf{naivecount}_n &: ((\mathsf{Nat}_n \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
\mathsf{naivecount}_n &\doteq \lambda P.\mathsf{count}\, n\, (\lambda i.\bot) \\
&\quad \text{where } \mathsf{count}\, 0 \qquad\quad p \doteq \text{if } P\, p \text{ then } 1 \text{ else } 0 \\
&\qquad\quad \mathsf{count}\, (1+n)\;\, p \doteq \quad \mathsf{count}\, n\, (\lambda i.\text{if } i = n \text{ then true else } p\, i) \\
&\qquad\qquad\qquad\qquad\qquad + \mathsf{count}\, n\, (\lambda i.\text{if } i = n \text{ then false else } p\, i)
\end{aligned}
$$

Here $(\lambda i.\bot)$ is the divergent point.

Iteration suffices to implement the naïve approach.

$$
\begin{aligned}
\mathsf{while}_A &: (A \to \mathsf{Bool}) \to A \to (A \to A) \to A \\
\mathsf{while}_A\; \mathit{test}\; x\; f &\doteq \text{if } \mathit{test}\, x \text{ then } \mathsf{while}_A\; \mathit{test}\; (f\; x)\; f \text{ else } x
\end{aligned}
$$

Berger count

Counter-intuitively, nested calls to a given predicate, P, can vastly improve the performance.

$$\mathsf{bestshot}_n \ : ((\mathsf{Nat}_n \to \mathsf{Bool}) \to \mathsf{Bool}) \to (\mathsf{Nat}_n \to \mathsf{Bool})$$

Returns a point Q such that $P\, Q$ evaluates to true.

Berger count

Counter-intuitively, nested calls to a given predicate, P, can vastly improve the performance.

$$\mathsf{bestshot}_n \ : ((\mathsf{Nat}_n \to \mathsf{Bool}) \to \mathsf{Bool}) \to (\mathsf{Nat}_n \to \mathsf{Bool})$$

Returns a point Q such that $P\,Q$ evaluates to true.

For example, we can implement a 'fail-fast' variation of naivecount.

$$\mathsf{lazycount}_n \doteq \lambda P.\ \mathsf{if}\ P\ (\mathsf{bestshot}_n\ P)\ \mathsf{then}\ \mathsf{naivecount}_n\ P\ \mathsf{else}\ 0$$

Berger count

Counter-intuitively, nested calls to a given predicate, P, can vastly improve the performance.

$$\mathsf{bestshot}_n \; : ((\mathsf{Nat}_n \to \mathsf{Bool}) \to \mathsf{Bool}) \to (\mathsf{Nat}_n \to \mathsf{Bool})$$

Returns a point Q such that $P\,Q$ evaluates to true.

For example, we can implement a 'fail-fast' variation of naivecount.

$$\mathsf{lazycount}_n \doteq \lambda P.\ \mathsf{if}\ P\ (\mathsf{bestshot}_n\ P)\ \mathsf{then}\ \mathsf{naivecount}_n\ P\ \mathsf{else}\ 0$$

One can take this idea further and do better than naivecount to implement Bergercount (Berger 1990).

$$\mathsf{Bergercount} : ((\mathsf{Nat}_n \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat}$$

(see Escardó (2007) for mind-boggling uses of this trick)

Pruned count

We can do better!
Idea: remember which components of the point a given predicate inspects (Longley 1999).

$$\begin{aligned}
&\mathsf{modulus} : ((\mathsf{Nat}_n \to \mathsf{Bool}) \to \mathsf{Bool}) \to (\mathsf{Nat}_n \to \mathsf{Bool}) \to (\mathsf{Bool} \times \mathsf{List}_{\mathsf{Nat}})\\
&\mathsf{modulus}\ P\ q \doteq \mathsf{let}\ log \leftarrow \mathsf{ref}([] : \mathsf{List}_{\mathsf{Nat}})\ \mathsf{in}\\
&\qquad\qquad\qquad \mathsf{let}\ wrap \leftarrow \lambda i.(log := i :: !log;\ q\, i)\ \mathsf{in}\\
&\qquad\qquad\qquad \mathsf{let}\ b \leftarrow P\, wrap\ \mathsf{in}\\
&\qquad\qquad\qquad \langle b, !log\rangle
\end{aligned}$$

If modulus $P\ q = \langle b, xs\rangle$, then $P\, q' = b$ for every q' that agrees component-wise with q at xs.

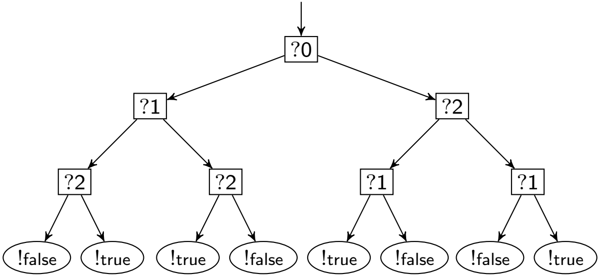
We can use this to effectively 'prune' the search space to either 'fail-fast' or 'succeed-fast'.

$$\mathsf{prunedcount}_n\ : ((\mathsf{Nat}_n \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat}$$

Generic count without effect handlers

$$\mathsf{count}_n : ((\mathsf{Nat}_n \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat}$$

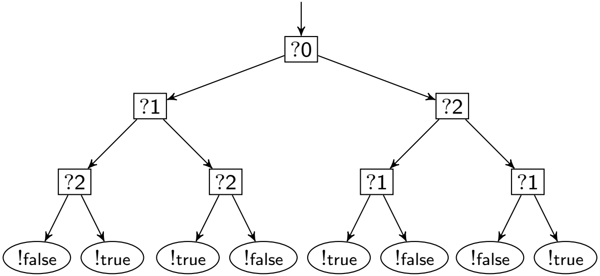
Every $\mathsf{count}_n \in \mathsf{PCF}$ **must restart** computation for every point, e.g. count_n ex:



Generic count without effect handlers

$$\mathsf{count}_n : ((\mathsf{Nat}_n \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat}$$

Every $\mathsf{count}_n \in$ PCF **must restart** computation for every point, e.g. count_n ex:



Steps: $\Omega(n2^n)$

Lower bound theorem

Theorem

If K is a PCF *program that correctly counts all canonical n-standard* PCF *predicates, and P is any canonical n-standard* PCF *predicate, then the evaluation of $K\ P$ must take time $\Omega(n2^n)$.*

Proof.

The proof involves tracking of reduction sequences and setting things up such that one can appeal to Milner (1977)'s Context Lemma. □

Experiments

The efficiency gap can be observed in practice.

Benchmarks

- Queens: enumerating solutions to the *n*-Queens problem
- Integration: exact real integration (Simpson 1998)

Methodology

- Implementations: naïve, Berger, pruned, effectful, and bespoke
- Implemented in OCaml 5 using the `multicont` package
- Ran each program 11 times. Given 3 minutes to complete
- Reporting the median speedup (or slowdown) of the effectful implementation

The source code and data are available via

`https://github.com/dhil/asymptotic-speedup-via-effect-handlers-code-jfp`

Queens experiments

Parameter	First solution			All solutions		
	20	24	28	8	10	12
Naïve	–	–	–	365.76	6633.47	–
Berger	13.89	21.72	31.83	3.91	3.51	3.18
Pruned	3.75	4.90	5.86	1.75	1.99	1.97
Bespoke	0.24	0.28	0.30	0.24	0.21	0.22

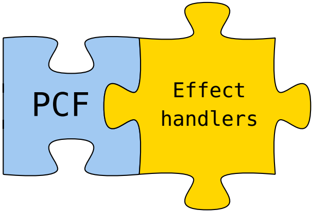
Table: Runtime of the *n*-Queens procedures relative to the effectful implementation

Integration experiments

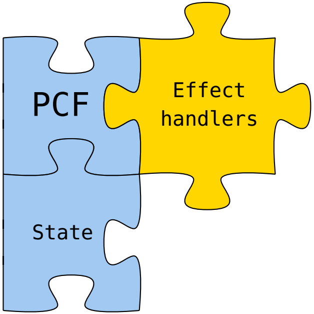
	Id	**Squaring**			**Logistic**				
Parameter	20	14	17	20	1	2	3	4	5
Naïve	6.58	18.22	22.38	27.28	23.44	63.75	36.67	—	—
Berger	3.62	7.67	7.83	8.34	8.76	11.98	11.67	12.02	12.62
Pruned	1.25	1.67	1.54	1.60	1.70	2.51	2.20	3.52	3.84

Table: Runtime of exact real integration procedures relative to the effectful implementation

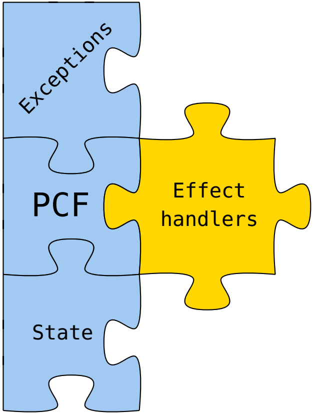
The puzzle so far



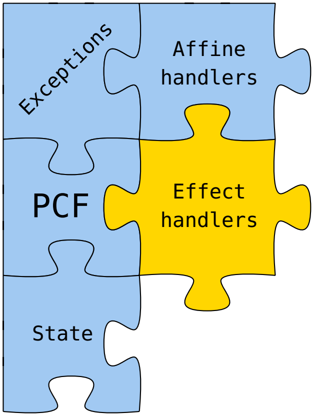
The puzzle so far



The puzzle so far



The puzzle so far



Summary and future work

Summary

- Take away: effect handlers admit asymptotically more efficient implementations
- Intuition: effect handlers enable computation to be shared via backtracking
- See the papers for rigorous mathematical analyses of this phenomenon

Future work

- What about the expressive power relative to McCarthy's amb operator?
- What about the asymptotic space characteristics of effect handlers?

References I

Berger, Ulrich (1990). "Totale Objekte und Mengen in der Bereichstheorie". PhD thesis. Munich: Ludwig Maximillians-Universtität.

Escardó, Martín Hötzel (2007). "Infinite sets that admit fast exhaustive search". In: *LICS*. IEEE Computer Society, pp. 443–452.

Hillerström, Daniel, Sam Lindley, and John Longley (2020). *Effects for Efficiency: Asymptotic Speedup with First-Class Control (extended version)*. arXiv: `2007.00605 [cs.PL]`.

Longley, John (1999). "When is a functional program not a functional program?" In: *ICFP*. ACM, pp. 1–7.

Milner, Robin (1977). "Fully Abstract Models of Typed λ-Calculi". In: *Theor. Comput. Sci.* 4.1, pp. 1–22.

Plotkin, Gordon (1977). "LCF considered as a programming language". In: *Theor. Comput. Sci.* 5.3, pp. 223–255.

Plotkin, Gordon D. and John Power (2003). "Algebraic Operations and Generic Effects". In: *Applied Categorical Structures* 11.1, pp. 69–94.

Plotkin, Gordon D. and Matija Pretnar (2009). "Handlers of Algebraic Effects". In: *ESOP*. Vol. 5502. Lecture Notes in Computer Science. Springer, pp. 80–94.

References II

Simpson, Alex K. (1998). "Lazy Functional Algorithms for Exact Real Functionals". In: *MFCS*. Vol. 1450. Lecture Notes in Computer Science. Springer, pp. 456–464.