Asymptotic Improvement through Delimited Control

Fast Generic Search with Effect Handlers

Daniel Hillerström

Laboratory for Foundations of Computer Science
School of Informatics
The University of Edinburgh, UK

March 28, 2019

Shonan Seminar 146, Japan

(Joint work with Sam Lindley and John Longley)

The Fundamental Efficiency of Effect Handlers

We consider whether a language with effect handlers admit essential expressiveness differences over a "pure" language.

The question

Let $\mathcal{L}_{\mathit{eff}}$ a language with effect handlers, and $\mathcal{L} \subset \mathcal{L}_{\mathit{eff}}$ the fragment modulo effect handlers. Does $\mathcal{L}_{\mathit{eff}}$ admit asymptotically more efficient programs than $\mathcal{L}$?

The Fundamental Efficiency of Effect Handlers

We consider whether a language with effect handlers admit essential expressiveness differences over a "pure" language.

The question

Let $\mathcal{L}_{\mathit{eff}}$ a language with effect handlers, and $\mathcal{L} \subset \mathcal{L}_{\mathit{eff}}$ the fragment modulo effect handlers. Does $\mathcal{L}_{\mathit{eff}}$ admit asymptotically more efficient programs than $\mathcal{L}$?

Spoiler alert: the answer is **YES**. Specifically $\mathcal{O}(2^n)$ vs $\Omega(n2^n)$.

To answer positively, it suffices to find one such program. We shall use *generic search* as our program.

The Fundamental Efficiency of Effect Handlers

We consider whether a language with effect handlers admit essential expressiveness differences over a "pure" language.

The question

Let $\mathcal{L}_{\mathit{eff}}$ a language with effect handlers, and $\mathcal{L} \subset \mathcal{L}_{\mathit{eff}}$ the fragment modulo effect handlers. Does $\mathcal{L}_{\mathit{eff}}$ admit asymptotically more efficient programs than $\mathcal{L}$?

Spoiler alert: the answer is **YES**. Specifically $\mathcal{O}(2^n)$ vs $\Omega(n2^n)$.

To answer positively, it suffices to find one such program. We shall use *generic search* as our program.

Take $\mathcal{L}$ to be cbv PCF and endow it with effect handlers to obtain $\mathcal{L}_{\mathit{eff}}$.

The Generic Search Problem

Problem: Given a boolean-valued predicate P on a space $\mathbb{B}^n$ of boolean vectors of length n (for some fixed $n \in \mathbb{N}$), return the number of such vectors p for which $P(p) = \mathrm{true}$. Thus for each n, we ask for an implementation of

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

There is but one rule:

No change of types is allowed! (Longley and Normann 2015)

This rules out tricks such as

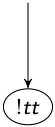
- CPS conversion
- Implementing an interpreter for $\mathcal{L}_{eff}$ in $\mathcal{L}$

Example predicates and their models I

A boring constant predicate

$$
\begin{aligned}
tt_0 &: (\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool} \\
tt_0 &\doteq \lambda p.\mathsf{true}
\end{aligned}
$$

Admits a with no queries model

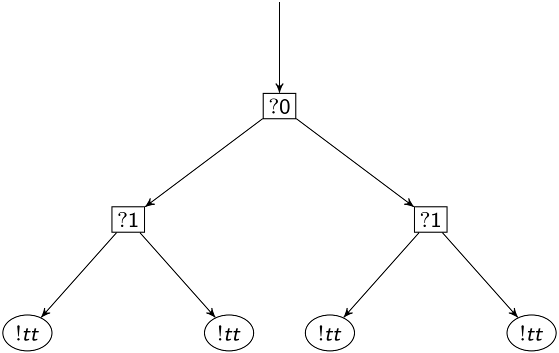


Example predicates and their models II

A slightly more interesting constant predicate

$$
\begin{aligned}
tt_2 &: (\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool} \\
tt_2 &\doteq \lambda p.p\,0; p\,1; \mathsf{true}
\end{aligned}
$$

Admits a finite model with no repeated queries

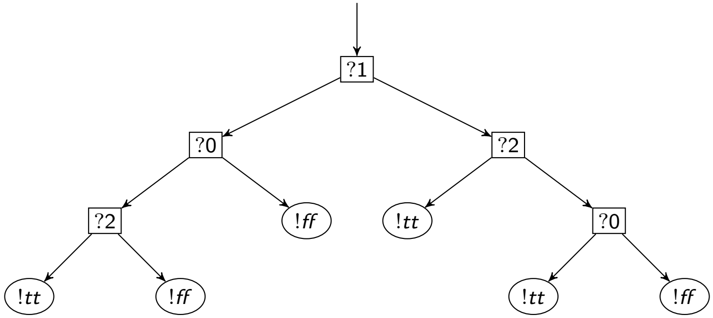


Example predicates and their models III

A non-constant predicate

$$
\begin{aligned}
tf_3 &\;:\; (\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool} \\
tf_3 &\doteq \lambda p.\textbf{if } p\ 1 \\
&\qquad\quad \textbf{then if } p\ 0 \textbf{ then } p\ 2 \textbf{ else } \mathsf{false} \\
&\qquad\quad \textbf{else if } p\ 2 \textbf{ then } \mathsf{true} \textbf{ else } p\ 0
\end{aligned}
$$

Admits a finite model with no repeated queries

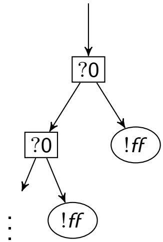


Example predicates and their models IV

Possibly divergent predicate

$$
\begin{aligned}
\mathit{div}_0 &: (\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool} \\
\mathit{div}_0 &\doteq \mathbf{rec}\ \mathit{div}_0\ p.\mathbf{if}\ p\,0\ \mathbf{then}\ \mathit{div}_0\ p\ \mathbf{else}\ \mathsf{false}
\end{aligned}
$$

Admits an infinite model with repeated queries



Restriction to n-standard predicates

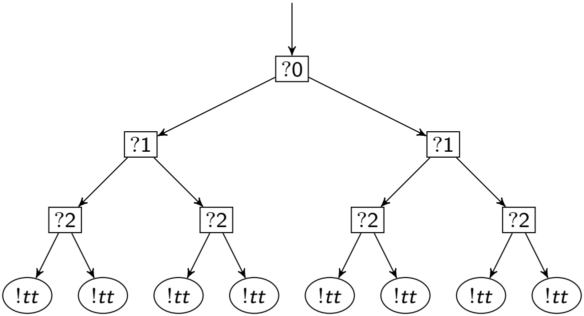
We restrict our analysis to predicates whose models are "n-standard"; informally

- A perfect binary tree of height $n > 0$, whose interior nodes are queries and leaves are answers.
- Contains every query $?j$ for $j \in \{0, \ldots, n-1\}$.
- No repeated queries along any path in the model.

For example

$$tt_3 \doteq \lambda p.p\ 0; p\ 1; p\ 2; \mathsf{true}$$

is 3-standard because its model is 3-standard



A pure generic search procedure

A possible implementation of generic search in $\mathcal{L}$

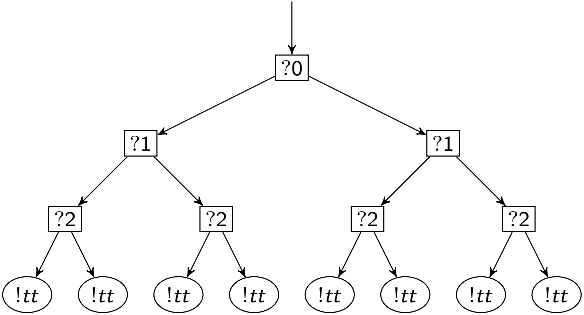
$$
\begin{array}{l}
\mathit{count}_n : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
\mathit{count}_n \doteq \lambda \mathit{pred}.\mathit{count}'\, n\, (\lambda i.\bot) \\
\qquad\qquad \textbf{where} \\
\qquad\qquad\qquad \begin{array}{lcl}
\mathit{count}'\, 0 \quad\quad\; p & \doteq & \textbf{if } \mathit{pred}\, p \textbf{ then } 1 \textbf{ else } 0 \\
\mathit{count}'\, (1+n)\; p & \doteq & \quad \mathit{count}'\, n\, (\lambda i.\textbf{if } i = n \textbf{ then } \mathsf{true} \textbf{ else } p\, i) \\
 & & + \; \mathit{count}'\, n\, (\lambda i.\textbf{if } i = n \textbf{ then } \mathsf{false} \textbf{ else } p\, i)
\end{array}
\end{array}
$$

A pure generic search procedure

A possible implementation of generic search in $\mathcal{L}$

$$
\begin{array}{l}
\mathit{count}_n : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
\mathit{count}_n \doteq \lambda \mathit{pred}.\mathit{count}'\, n\, (\lambda i.\bot) \\
\qquad\qquad \textbf{where} \\
\qquad\qquad\qquad \begin{array}{lll}
\mathit{count}'\, 0 & p \doteq & \textbf{if } \mathit{pred}\ p \textbf{ then } 1 \textbf{ else } 0 \\
\mathit{count}'\, (1+n) & p \doteq & \quad \mathit{count}'\, n\, (\lambda i.\textbf{if } i = n \textbf{ then } \mathsf{true} \textbf{ else } p\, i) \\
 & & + \ \mathit{count}'\, n\, (\lambda i.\textbf{if } i = n \textbf{ then } \mathsf{false} \textbf{ else } p\, i)
\end{array}
\end{array}
$$

Example $\mathit{count}_3\ \mathit{tt}_3$:

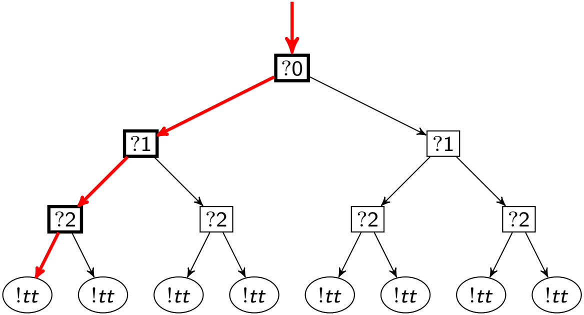


A pure generic search procedure

A possible implementation of generic search in $\mathcal{L}$

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

Example $\mathit{count}_3\ \mathit{tt}_3$: reaches the first leaf

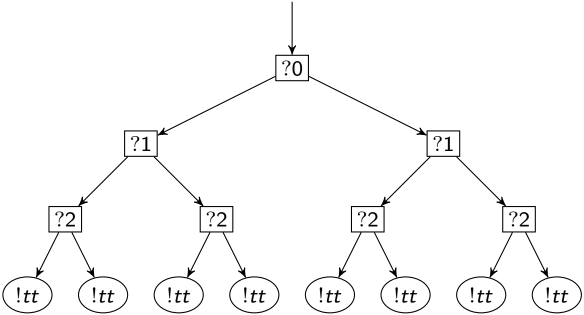


A pure generic search procedure

A possible implementation of generic search in $\mathcal{L}$

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

Example $\mathit{count}_3\, \mathit{tt}_3$: computation restarts

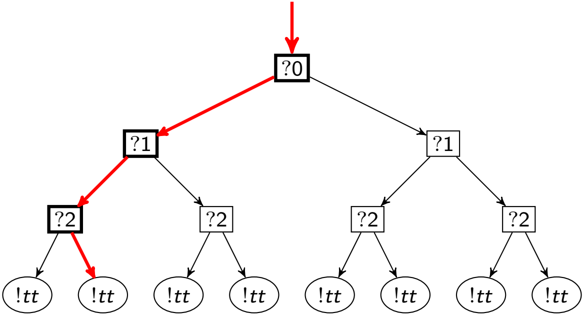


A pure generic search procedure

A possible implementation of generic search in $\mathcal{L}$

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

Example $count_3\, tt_3$: reaches the second leaf

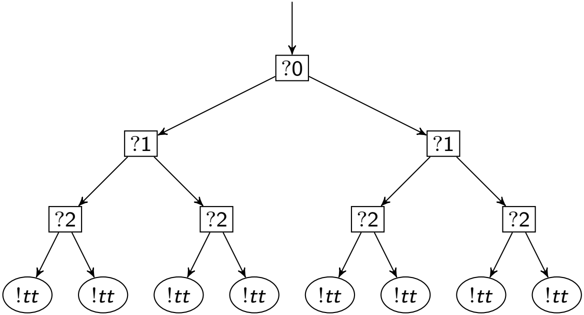


A pure generic search procedure

A possible implementation of generic search in $\mathcal{L}$

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

Example $\mathit{count}_3\, \mathit{tt}_3$: computation restarts

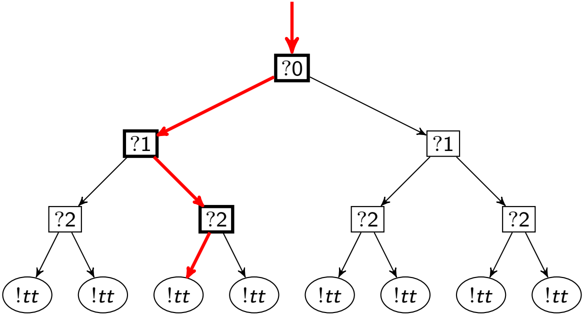


A pure generic search procedure

A possible implementation of generic search in $\mathcal{L}$

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

Example $\mathit{count}_3\ \mathit{tt}_3$: reaches the third leaf, etc. . .



The effectful generic search procedure

For the efficient implementation of generic search in $\mathcal{L}_{\mathit{eff}}$, we require one operation; fix $\Sigma \doteq \{\mathsf{Branch} : \langle\rangle \to \mathsf{Bool}\}$

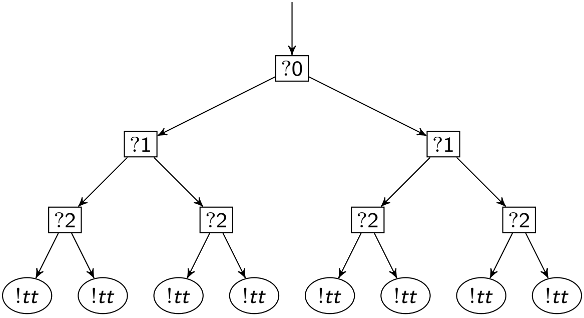
$$
\begin{aligned}
&\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
&\mathsf{count} \doteq \lambda \mathit{pred}.\textbf{handle}\ \mathit{pred}\ (\lambda n.\textbf{do}\ \mathsf{Branch})\ \textbf{with} \\
&\qquad\qquad\qquad \textbf{val}\ x \mapsto \textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ 0 \\
&\qquad\qquad\qquad \mathsf{Branch}\ \langle\rangle\ r \mapsto r\ \mathsf{true} + r\ \mathsf{false}
\end{aligned}
$$

The effectful generic search procedure

For the efficient implementation of generic search in $\mathcal{L}_{eff}$, we require one operation; fix $\Sigma \doteq \{\mathsf{Branch} : \langle\rangle \to \mathsf{Bool}\}$

$$\begin{aligned}
&\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat}\\
&\mathsf{count} \doteq \lambda \mathit{pred}.\mathbf{handle}\ \mathit{pred}\ (\lambda n.\mathbf{do}\ \mathsf{Branch})\ \mathbf{with}\\
&\qquad\qquad\quad \mathbf{val}\ x \mapsto \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ 0\\
&\qquad\qquad\quad \mathsf{Branch}\ \langle\rangle\ r \mapsto r\,\mathsf{true} + r\,\mathsf{false}
\end{aligned}$$

Example $\mathsf{count}\ tt_3$:

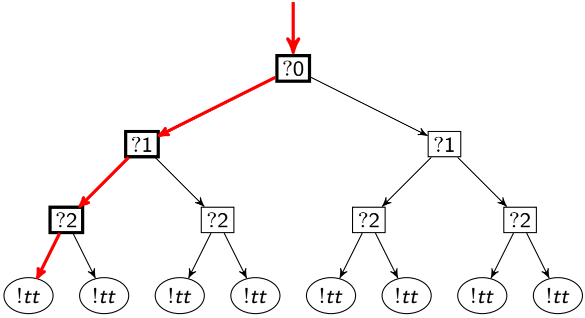


The effectful generic search procedure

For the efficient implementation of generic search in $\mathcal{L}_{eff}$, we require one operation; fix $\Sigma \doteq \{\mathsf{Branch} : \langle\rangle \to \mathsf{Bool}\}$

$$
\begin{aligned}
&\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
&\mathsf{count} \doteq \lambda \mathit{pred}.\mathbf{handle}\ \mathit{pred}\ (\lambda n.\mathbf{do}\ \mathsf{Branch})\ \mathbf{with} \\
&\qquad\qquad\qquad \mathbf{val}\ x \mapsto \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \\
&\qquad\qquad\qquad \mathsf{Branch}\ \langle\rangle\ r \mapsto r\,\mathsf{true} + r\,\mathsf{false}
\end{aligned}
$$

Example $\mathsf{count}\ tt_3$: reaches the first leaf

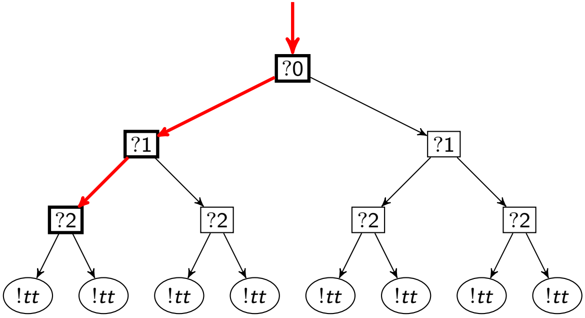


The effectful generic search procedure

For the efficient implementation of generic search in $\mathcal{L}_{\mathit{eff}}$, we require one operation; fix $\Sigma \doteq \{\mathsf{Branch} : \langle\rangle \to \mathsf{Bool}\}$

$$\begin{aligned}
&\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
&\mathsf{count} \doteq \lambda \mathit{pred}.\textbf{handle}\ \mathit{pred}\ (\lambda n.\textbf{do}\ \mathsf{Branch})\ \textbf{with} \\
&\qquad\qquad\qquad \textbf{val}\ x \mapsto \textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ 0 \\
&\qquad\qquad\qquad \mathsf{Branch}\ \langle\rangle\ r \mapsto r\,\mathsf{true} + r\,\mathsf{false}
\end{aligned}$$

Example $\mathsf{count}\ tt_3$: computation backtracks

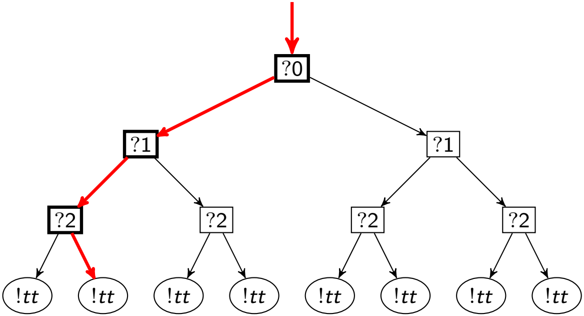


The effectful generic search procedure

For the efficient implementation of generic search in $\mathcal{L}_{\mathit{eff}}$, we require one operation; fix $\Sigma \doteq \{\mathsf{Branch} : \langle\rangle \to \mathsf{Bool}\}$

$$
\begin{aligned}
&\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
&\mathsf{count} \doteq \lambda \mathit{pred}.\mathbf{handle}\ \mathit{pred}\ (\lambda n.\mathbf{do}\ \mathsf{Branch})\ \mathbf{with} \\
&\qquad\qquad\qquad \mathbf{val}\ x \mapsto \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \\
&\qquad\qquad\qquad \mathsf{Branch}\ \langle\rangle\ r \mapsto r\,\mathsf{true} + r\,\mathsf{false}
\end{aligned}
$$

Example $\mathsf{count}\ \mathit{tt}_3$: reaches the second leaf

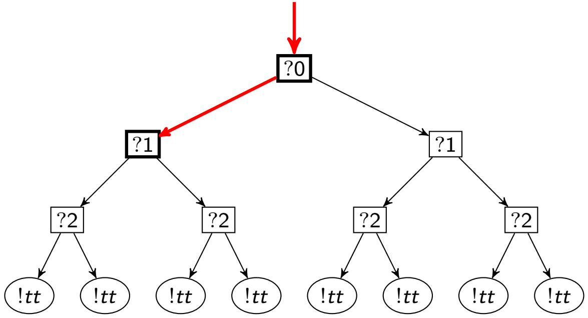


The effectful generic search procedure

For the efficient implementation of generic search in $\mathcal{L}_{eff}$, we require one operation; fix $\Sigma \doteq \{\mathsf{Branch} : \langle\rangle \to \mathsf{Bool}\}$

$$\begin{aligned}
&\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
&\mathsf{count} \doteq \lambda \mathit{pred}.\textbf{handle}\ \mathit{pred}\ (\lambda n.\textbf{do}\ \mathsf{Branch})\ \textbf{with} \\
&\qquad\qquad\quad \textbf{val}\ x \mapsto \textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ 0 \\
&\qquad\qquad\quad \mathsf{Branch}\ \langle\rangle\ r \mapsto r\,\mathsf{true} + r\,\mathsf{false}
\end{aligned}$$

Example $\mathsf{count}\ tt_3$: computation backtracks

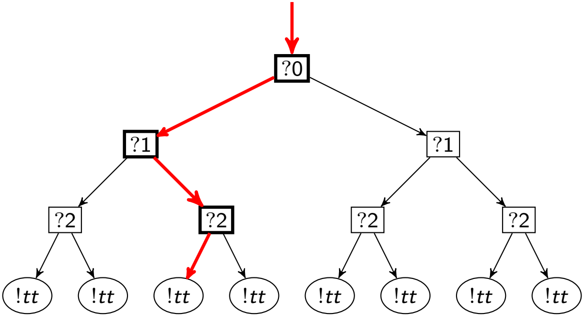


The effectful generic search procedure

For the efficient implementation of generic search in $\mathcal{L}_{\mathit{eff}}$, we require one operation; fix $\Sigma \doteq \{\mathsf{Branch} : \langle\rangle \to \mathsf{Bool}\}$

$$
\begin{array}{l}
\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
\mathsf{count} \doteq \lambda \mathit{pred}.\mathbf{handle}\ \mathit{pred}\ (\lambda n.\mathbf{do}\ \mathsf{Branch})\ \mathbf{with} \\
\qquad\qquad\qquad \mathbf{val}\ x \mapsto \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \\
\qquad\qquad\qquad \mathsf{Branch}\ \langle\rangle\ r \mapsto r\,\mathsf{true} + r\,\mathsf{false}
\end{array}
$$

Example $\mathsf{count}\ tt_3$: reaches the third leaf, etc...



Main theorem

Theorem

1. *For every n-standard predicate pred, the generic counting procedure has at most time complexity*

$$\mathrm{DTime}(\mathrm{count}\ \mathit{pred}) = \sum_{bs \in \mathbb{B}^*, |bs| \leq n} \mathrm{steps}(t)(bs) + \mathcal{O}(2^n)$$

2. *Every generic counting function count $\in \mathcal{L}$ has for every n-standard predicate pred at least time complexity*

$$\mathrm{DTime}(\mathit{count}\ \mathit{pred}) = \sum_{bs \in \mathbb{B}^*, |bs| \leq n} 2^{n-|bs|}\mathrm{steps}(t)(bs) + \mathcal{O}(n2^n)$$

Here t denotes the model of pred, and $\mathrm{steps}(t)(bs)$ *computes the number of reduction steps used to arrive at the query or answer node determined by bs.*

Proving the positive result

Define suitable machine configuration computing functions

$$\mathrm{arrive}, \mathrm{depart} : \mathbb{B}^* \times \mathrm{Model} \rightharpoonup \mathrm{Conf}$$

Lemma

Suppose t is a model of a n-standard predicate, then for every boolean list $bs \in \mathbb{B}^*$

$$\mathrm{arrive}(bs, t) \longrightarrow^{\sum_{|bs| \leq n} \mathsf{steps}(t)(bs) + 2^{n-|bs|}} \mathrm{depart}(bs, t)$$

Proof.

Proof by downward induction on the list of booleans bs. □

Proving the negative result

Suppose that we have an arbitrary implementation of generic search $\textit{count} \in \mathcal{L}$. Pick any n-standard predicate *pred* and look at the computation arising from *count pred*. Now we need to show that

Lemma (Every leaf is visited (A))

The computation (count pred) visits every leaf in the model of pred.

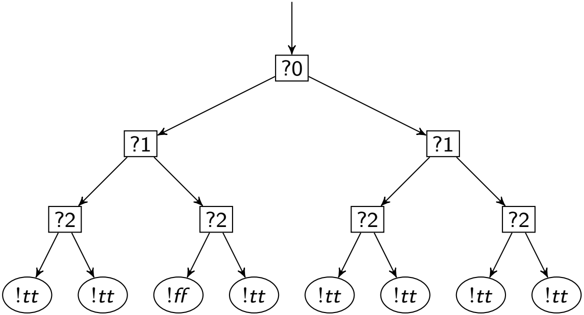
Lemma (No shared computation (B))

If p and p' are distinct points then their subcomputations are disjoint.

Since each subcomputation has length at least $\Omega(n)$ the entire computation must have at least length $\Omega(n2^n)$.

Threads and sections

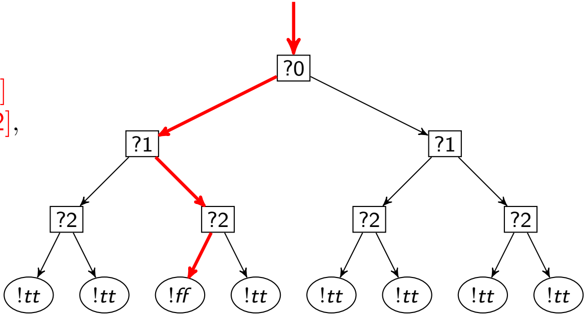
Consider a 3-standard predicate *seven* (has seven true leaves)



Threads and sections

Consider a 3-standard predicate *seven* (has seven true leaves)

$$\mathit{Thread} \doteq \{\ \mathit{pred}\ p \rightsquigarrow^* \mathcal{E}_0[p\,0],\ \mathcal{E}_0[\mathsf{true}] \rightsquigarrow^* \mathcal{E}_1[p\,1]\ \mathcal{E}_1[\mathsf{false}] \rightsquigarrow^* \mathcal{E}_2[p\,2],\ \mathcal{E}_2[\mathsf{true}] \rightarrow \mathsf{false}\ \}$$

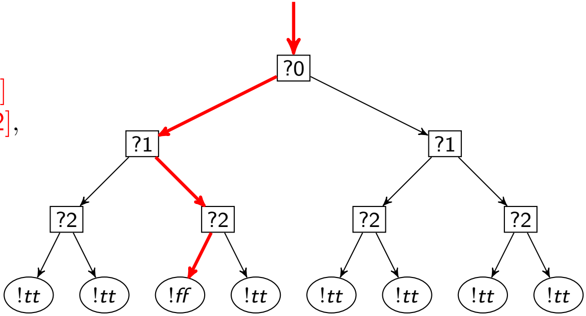


Any n-standard predicate has 2^n threads, and every thread consists of $n+1$ sections.

Threads and sections

Consider a 3-standard predicate *seven* (has seven true leaves)

$$\mathit{Thread} \doteq \{\ \mathit{pred}\ p \rightsquigarrow^* \mathcal{E}_0[p\,0],\ \mathcal{E}_0[\mathsf{true}] \rightsquigarrow^* \mathcal{E}_1[p\,1]\ \mathcal{E}_1[\mathsf{false}] \rightsquigarrow^* \mathcal{E}_2[p\,2],\ \mathcal{E}_2[\mathsf{true}] \rightarrow \mathsf{false}\ \}$$



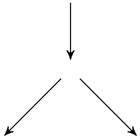
Any n-standard predicate has 2^n threads, and every thread consists of $n+1$ sections.

Proof of Lemma A.

By contradiction: pick a leaf that has no thread; negate the value at the leaf; tweak the predicate accordingly; observe a wrong result. $\square$

No shared computation

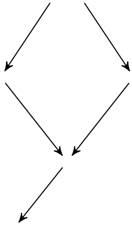
Every section has a unique successor



Proof.

Follows by definition of section and the semantics being deterministic. □

Every section has a single predecessor



Proof.

By direct calculation on the reduction sequence induced by a section. □

Summary and future work

In summary

- We have defined two languages $\mathcal{L}$ and $\mathcal{L}_{\mathit{eff}}$
- We have demonstrated that $\mathcal{L}_{\mathit{eff}}$ provides strictly more efficient implementations of generic search than $\mathcal{L}$ ($\mathcal{O}(2^n)$ vs $\Omega(n2^n)$)
- ...which establish a new complexity result for control operators
- Intuition: control operators build in support for backtracking.

Future considerations

- Perform empirical experiments to observe the result in practice (Daniels 2016)
- Study the robustness of the result, i.e. what feature(s) can we add to $\mathcal{L}$ whilst retaining an efficiency gap between $\mathcal{L}$ and $\mathcal{L}_{\mathit{eff}}$?
- Generalise the result to all conceivable effective models of computations

References I

Daniels, Robbie (Aug. 2016). "Efficient Generic Searches and Programming Language Expressivity". MA thesis. Scotland: School of Informatics, the University of Edinburgh.

Longley, John and Dag Normann (2015). *Higher-Order Computability*. Theory and Applications of Computability. Springer.