Sequential Algorithms, Deterministic Parallelism, and Intensional Expressiveness

---

Stephen Brookes
Denis Dancanet

School of Computer Science
Carnegie Mellon University

ACM Symposium on Principles
of Programming Languages
San Francisco, CA
22–25 January 1995

Goal of this work

- Establish relative intensional expressiveness results for programming languages
- Definitions:
  - L_1 is *intensionally* more expressive than L_2 if there are functions computable *faster* in L_1 than in L_2
  - L_1 is *extensionally* more expressive than L_2 if there are functions computable in L_1 but not in L_2

Outline

- Previous work
- Part I: Computing minimum with sequential and parallel algorithms
- Part II: Deterministic parallelism
  - PCF
  - PCF and circuits
  - Parallel extensions of PCF
- Conclusions

Previous work: Loïc Colson

- First work on intensional expressiveness of programming languages
- Can primitive recursive ($\mathcal{PR}$) algorithms express minimum efficiently?
- Minimum of two integers in unary form:

$$min(x, 0) = 0$$
$$min(0, x) = 0$$
$$min(S(x), S(y)) = S(min(x, y))$$

- Note that min computes the minimum of x, y in time $O(min(x, y))$
- **Prop:** There is no $\mathcal{PR}$ algorithm which computes the minimum of two numbers x and y in unary representation, and is of complexity $O(min(x, y))$

Colson's work continued

- But wait! Minimum is primitive recursive:

$$pred(0) = 0$$
$$pred(S(x)) = x$$

$$sub(x, 0) = x$$
$$sub(x, S(y)) = pred(sub(x, y))$$

$$MIN(x, y) = sub(x, sub(x, y))$$

- MIN computes the minimum of x, y in time $O(max(x, y))$
- What is the problem with $\mathcal{PR}$ algorithms?
- Answer: Ultimate obstination

Part I: Sequential algorithms

- Sequential algorithms were developed by Berry and Curien and implemented as a programming language called CDS0
- CDS0 is a canonical *intensional* sequential language
- Example:

```
let LEFT_OR = algo
    request B do
      valof (B.1) is
        tt: output tt
        ff: valof (B.2) is
              tt: output tt
              ff: output ff
            end
      end
    end
end;
```

Sequential algorithms and minimum

- Can CDS0 compute minimum efficiently?
- **Prop:** There is no sequential algorithm which computes *min*
- **Prop:** CDS0 can compute minimum in $O(min)$

```
algo  left_min (n1, n2) =
        case n1 of
              0 ⇒ 0
            | S(x) ⇒ case n2 of
                          0 ⇒ 0
                        | S(y) ⇒ S(left_min(x, y))
```

Parallel algorithms and minimum

- Parallel algorithms are a deterministic parallel generalization of CDS0, developed by Brookes and Geva
- Language of parallel algorithms is called CDSP
- CDSP algorithm for parallel-or:

algo por (b1, b2) = **query** (b1, b2) **is**
$(tt, _) \Rightarrow tt$
$\mid (_, tt) \Rightarrow tt$
$\mid (ff, ff) \Rightarrow ff$

- **Prop:** CDSP can compute *min*:

algo min (n1, n2) =
query (n1, n2) **is**
(0, _) ⇒ 0
| (_, 0) ⇒ 0
| (S(x), S(y)) ⇒ S(min(x, y))

Part II: Deterministic parallelism

- Is deterministic parallelism "useful"?
- Cartwright, Curien, Felleisen: Deterministic parallelism adds expressiveness, but additional expressiveness applies only to unbounded computations, hence it is not useful
- **Prop:** CDSP is intensionally more expressive than CDS0: n-ary disjunction and n-ary addition can be computed in time $O(log\ n)$
- n-ary disjunction:

algo $\mathrm{por}_n\ (\mathrm{b}_1, \ldots, \mathrm{b}_n) =$
$\mathrm{por}\ (\mathrm{por}_{n/2}\ (\mathrm{b}_1, \ldots, \mathrm{b}_{n/2}),$
$\mathrm{por}_{n/2}\ (\mathrm{b}_{n/2+1}, \ldots, \mathrm{b}_n))$

PCF

- Canonical sequential programming language
- Types:

  $\sigma ::= o \mid \iota \mid \sigma \to \sigma$

- Syntax:

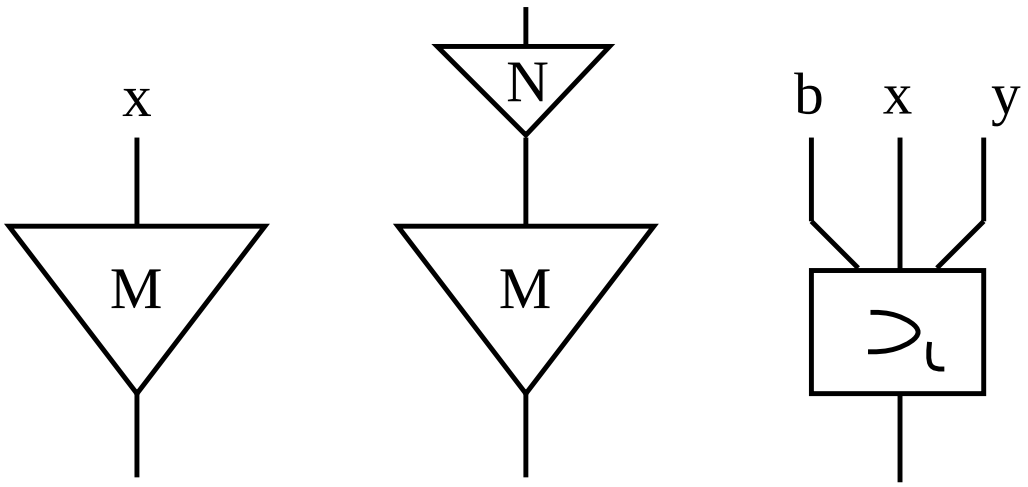
  $M ::= c \mid x \mid \lambda x.\ M \mid MM$

- Constants:

  $tt : o$
  $ff : o$
  $n : \iota$ (the integers, $n \geq 0$)
  $isZero? : \iota \to o$
  $+1 : \iota \to \iota$
  $-1 : \iota \to \iota$
  $\supset_o : o \to o \to o \to o$ (boolean conditional)
  $\supset_\iota : o \to \iota \to \iota \to \iota$ (integer conditional)
  $Y_\sigma : (\sigma \to \sigma) \to \sigma$ (one for each σ)
  $= : \iota \to \iota \to o$ (integer equality)

PCF and circuits 1

- We view PCF programs as circuits:



(a) $\lambda x.\ M$, (b) $(\lambda x.\ M)N$, (c) $\supset_\iota\ b\ x\ y$

- Semantics of circuits based on PCF's operational semantics

PCF and circuits 2

- A circuit is *static* if it represents a non-recursive PCF program
- A circuit is *dynamic* if it represents a recursive PCF program
- The *depth* of a static circuit is equal to the height of the underlying tree
- A circuit is *constant-depth* if it is either static, or a dynamic circuit which does not expand more than a fixed constant number of times (independent of the inputs)
- If two functions can be implemented in terms of each other with constant-depth circuits, we say that the two functions are *intensionally equivalent*

Parallel extensions of PCF

- Parallel-or:
  $por : o \to o \to o$
  $por\ tt\ \bot = tt$
  $por\ \bot\ tt = tt$
  $por\ ff\ ff = ff$
- Parallel conditionals:
  $pif_\sigma : o \to \sigma \to \sigma \to \sigma$
  $pif_\sigma\ \bot\ x\ x = x,$
  $pif_\sigma\ tt\ x\ \bot = x,$
  $pif_\sigma\ ff\ \bot\ x = x,$ for $\sigma = \iota, o$
- It is well known that por, pif_o, pif_ι are extensionally equivalent [Stoughton, Curien]

Intensional equivalence

- **Prop:** por and pif_o are intensionally equivalent
- Proof:

$$
\begin{aligned}
&por = \lambda xy.\ pif_o\ x\ tt\ y, \\
&pif_o = \lambda bxy.\ por_3\ (pand\ b\ x) \\
&\qquad\qquad\qquad\quad (pand\ (not\ b)\ y) \\
&\qquad\qquad\qquad\quad (pand\ x\ y)
\end{aligned}
$$

- Parallel conjunction:

$$pand = \lambda xy.\ not\ (por\ (not\ x)\ (not\ y))$$

Known implementations

- **Prop:** pif_ι can implement pif_o in constant-depth:

  $pif_o = \lambda bxy.\ (= 1\ (pif_\iota\ b\ (\supset_\iota\ x\ 1\ 0)\ (\supset_\iota\ y\ 1\ 0)))$

- Well known implementation of pif_ι using por:

  $pif_\iota = Y F\, 0,$ where
  $F = \lambda fnbxy.\ \supset_\iota\ (por\ (pand\ (= x\ n)\ (= y\ n))$
  $(pand\ b\ (= x\ n))$
  $(pand\ (not\ b)\ (= y\ n)))$
  n
  $(f\ (+1\ n)\ b\ x\ y)$

- This is not constant-depth

Main result

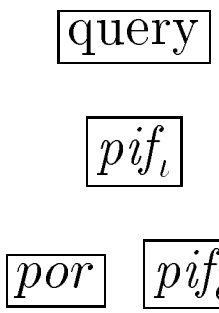
- **Lemma 1:** It is not possible to write a program in PCF + *por*, which computes $pif_\iota\ b\ x\ y$ and is of the form $\supset_\iota\ B\ x\ y$, where B is a static circuit yielding a boolean
- **Lemma 2:** It is not possible to write a program in PCF + *por*, which computes $pif_\iota\ b\ x\ y$ and is of the form $\supset_\iota\ B\ N_1\ N_2$, where B, N_1, N_2 are static circuits yielding a boolean and two integers respectively
- **Prop:** PCF + *por* cannot implement pif_ι with a constant-depth circuit

Query is more powerful

- Assume existence of binary addition operation in PCF
- **Prop:** PCF + pif_t cannot implement n-ary addition with a circuit of depth $log\ n$
- **Corollary:** PCF + por cannot implement n-ary addition with a circuit of depth $log\ n$

Conclusions

- Sequential algorithms are intensionally more expressive than primitive recursive algorithms
- CDSP is intensionally more expressive than CDS0
- Parallel PCF is intensionally more expressive than PCF
- Hierarchy of intensional expressiveness for deterministic parallel constructs:



- Exhibited languages extensionally but not intensionally equivalent