# On the (Relational) Algebra of Algorithmic Complexity

IFIP WG2.1 meeting #84, Vihula (Estonia), 30th April 2026

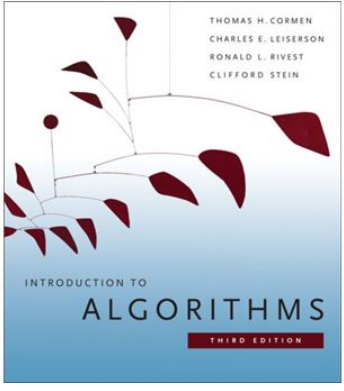
J.N. Oliveira (joint work with C. Ribeiro)


U. Minho & INESC TEC

# Part I — Big-O notation “refactored”

# Back to an old topic

- An old, respectable topic
- Much relevant to computer science
- The subject of well-known "bibles"
- Yet still hard to grasp by students (as can be seen in e.g. internet fora)



# History

*"(...) Although there have been papers and books written on the history of algorithms, the history of* **algorithmic complexity** *has not been given nearly as much attention (...).*

*This paper will address this need (...) beginning with the text of Ibn al-majdi, a fourteenth century Egyptian astronomer, through the 21st century.*

*In addition, this paper highlights the confusion surrounding* **big-O** *notation as well as the [work] of a group of mathematicians [which] was critical to the development of the field (...)"*

(Nasar, 2016) "The history of Algorithmic complexity"

# History

*"In order to compare the efficiencies of competing algorithms for a given problem, it is necessary to consider the number of operations performed by each algorithm for large inputs. This is done by classifying and comparing the growth rates of each algorithm's complexity function.*

*(...)* **Big-O notation**, *which was introduced by the German mathematician Paul Bachmann in 1894, is used extensively in the analysis of algorithms to describe the order of growth of a complexity function."*

(Nasar, 2016) 'The history of Algorithmic complexity'

Donald Knuth et al: $\mathcal{O}(f)$, $\Theta(f)$, $\Omega(f)$.

# Warnings

*(...) There are two* **dangerous bends** *one has to navigate with asymptotic notation. Firstly, the equality sign in* $f = \Theta(g)$ *is* **not true equality** *with all the attendant properties that equality entails.*

*(. . . ) Another way to define* $\Theta$ *notation is to say that* $\Theta(g)$ *denotes the* **set** *of all functions* $f$ *with the stated property, and to write "*$f \in \Theta(g)$*" instead of "*$f = \Theta(g)$*".*

(Bird and Gibbons, 2020) – 'Algorithm Design with Haskell'

# Criticism

*(...) we show that assertions of the form "this code has asymptotic complexity $\mathcal{O}(n)$" are too* **informal***: they do not have suffiently precise meaning, and can be easily abused to produce* **flawed** *paper* **proofs***.*

(Gueneau, 2020) – 'Mechanized verification of the correctness and asymptotic complexity (...) '

# Abstract interpretation

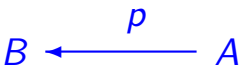
*(...) [**Complexity**] **Analysis** is often done by obtaining an abstract version of the program by relying on **abstract interpretation** (...) the abstraction consist[ing] in inferring (...) size **relations** (...) usually expressed by means of linear constraints.*

(Albert et al., 2008) — 'COSTA: Design and Imp. of a Cost and Termination Analyzer(...)'

# Cost analysis

**Program** $p$ accepts input $a$ and yields output $b$.

$$b = p\,(a)$$



# Cost analysis

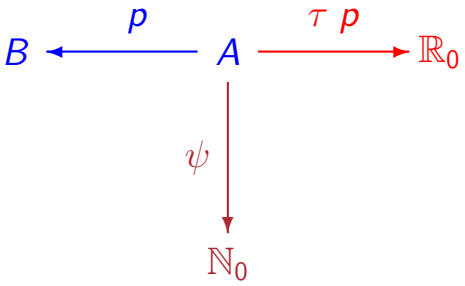
**Cost** $y$ of program $p$ receiving input $a$ and producing output $b$:

$$y = \tau\ p\ (a)$$

# Cost analysis

**Size** $n$ of input $a$ from which program $p$ produces $b$, costing $y$

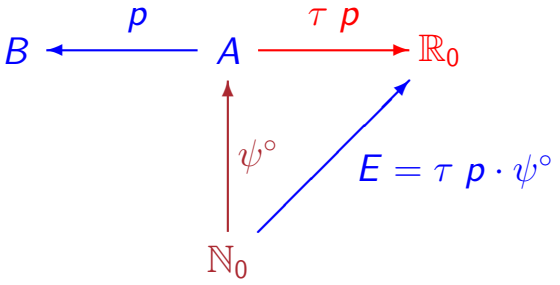
$n = \psi(a)$



# Cost analysis

**Abstract relation** between cost $y$ of $p$ handling some input of size $n$

$y\ (E\ p)\ n$



# Cost analysis

Cost relation

$$E\ p = \tau\ p \cdot \psi^{\circ} \tag{1}$$

is an **abstract interpretation** of $p$ under **abstraction function** $\psi$, targeting **measure** $\tau$.

In general, it is a **relation**, not a function:

$$y\ (E\ p)\ n \quad \Leftrightarrow \quad \langle \exists\ a\ :\ \psi\ a = n :\ y = \tau\ p\ a \rangle$$

Questions:

- So we have **cost relations**, not **cost functions** ...?
- A more basic one: what is a "measure"?

# Cost analysis

Cost relation

$$E\ p = \tau\ p \cdot \psi^{\circ} \tag{1}$$

is an **abstract interpretation** of $p$ under **abstraction function** $\psi$, targeting **measure** $\tau$.

In general, it is a **relation**, not a function:

$$y\ (E\ p)\ n \quad \Leftrightarrow \quad \langle \exists\ a\ :\ \psi\ a = n :\ y = \tau\ p\ a \rangle$$

Questions:

- So we have **cost relations**, not **cost functions** ...?
- A more basic one: what is a "measure"?

# Measures

# Two notations placed in parallel

In calculus, the **product rule** (or **Leibniz rule**[2] or **Leibniz product rule**) is a formula used to find the derivatives of products of two or more functions. For two functions, it may be stated in Lagrange's notation as

$$(u \cdot v)' = u' \cdot v + u \cdot v'$$

or in Leibniz's notation as

$$\frac{d}{dx}(u \cdot v) = \frac{du}{dx} \cdot v + u \cdot \frac{dv}{dx}.$$

The rule may be extended or generalized to products of three or more functions, to a rule for higher-order derivatives of a product, and to other contexts.

Differential calculus

**The coproduct functor**

We also obtain a bifunctor $+$ whose definition on arrows is

$$f + g \quad = \quad [inl \cdot f, inr \cdot g].$$

The composition and fusion laws

$$\begin{aligned} (f + g) \cdot (h + k) &= f \cdot h + g \cdot k \\ [f, g] \cdot (h + k) &= [f \cdot h, g \cdot k] \end{aligned}$$

follow at once by duality, though one can also give a direct proof.

Algebra of Programming

Can we have both at the same time?

# Measures

Let a **measure** $f$ of type $A$ be any function $f : A \to \mathbb{K}$ where $\mathbb{K}$ is an **ordered field**.

This turns the exponential type $\mathbb{K}^A$ into a **vector space** under the usual **lifted** operators, namely vector **addition**

$$(f + g)\, x = f\, x + g\, x \tag{2}$$

and **scalar product** ($C \in \mathbb{K}$)

$$(C\, f)\, x = C\, (f\, x) \tag{3}$$

# Measures

One also has the **Hadamard product**,

$$(f \times g)\, x = (f\, x)\, (g\, x) \tag{4}$$

where $f \times g$ is usually abbreviated to $f\ g$.

$f + g$, $f\ g$ and $C\ f$ can also be defined *pointfree*,

$$f + g = \widehat{(+)} \cdot \langle f\ ,\ g \rangle \tag{5}$$

$$f\ g = \widehat{(\times)} \cdot \langle f\ ,\ g \rangle \tag{6}$$

$$C\ f = (C \times) \cdot f = \underline{C}\ f \tag{7}$$

where $\langle f\ ,\ g \rangle\ x = (f\ x, g\ x)$ is the **pairing** combinator. (7) uses constant function $\underline{C}$ and the Hadamard product.

# Measures

The **unit elements** are constant functions:

$$f \times \underline{1} = f \tag{8}$$
$$f + \underline{0} = f \tag{9}$$

**Linearity**:

$$(f + g) \cdot h = f \cdot h + g \cdot h \tag{10}$$
$$(C + D)\, f = C\, f + D\, f \tag{11}$$

Moreover,

$$(f\ g) \cdot k = (f \cdot k)\, (g \cdot k) \tag{12}$$

# Measures

## Affine transformation

$$\underline{C} + f = (C+) \cdot f = (\underline{C} + id) \cdot f \tag{13}$$

## Scalar product

$$C\ (f\ g) = (C\ f)\ g = f\ (C\ g) \tag{14}$$

$$C\ (f \cdot g) = (C\ f) \cdot g \tag{15}$$

# Comparing measures

**Pointwise comparison**:

$$f \mathrel{\dot{\leqslant}} g \quad \Leftrightarrow \quad f \subseteq (\leqslant) \cdot g \quad \Leftrightarrow \quad \langle \forall\ n\ ::\ f\,(n) \leqslant g\,(n) \rangle \tag{16}$$

(16) is the Reynolds square:

$$f \mathrel{\dot{\leqslant}} g \quad \Leftrightarrow \quad \begin{array}{ccc} A & \xleftarrow{id} & A \\ {\scriptstyle f}\downarrow & \subseteq & \downarrow{\scriptstyle g} \\ B & \xleftarrow[(\leqslant)]{} & B \end{array} \tag{17}$$

**(+)-shunting**:

$$f + g \mathrel{\dot{\leqslant}} h \quad \Leftrightarrow \quad f \mathrel{\dot{\leqslant}} h - g \tag{18}$$

# Scaling

**Scaled order**

$$(\leqslant_C) = (\leqslant) \cdot (C\times) \tag{19}$$

i.e. $x \leqslant_C y \quad \Leftrightarrow \quad x \leqslant C\, y.$

Clearly:

$$(\leqslant_{DC}) = (\leqslant_D) \cdot (C\times) \tag{20}$$

Also useful ($C \geqslant 0$):

$$(\leqslant_C) \cdot (\leqslant_D) \subseteq (\leqslant_{CD}) \tag{21}$$

# Comparing measures

Generalization of (18), for $C > 0$:

$$f + g \;\dot{\leqslant}_C\; h \;\Leftrightarrow\; f \;\dot{\leqslant}_C\; h - \frac{1}{C}\, g \tag{22}$$

For nonzero $g$ (i.e. $g \cap \underline{0} = \bot$):

$$f\, g \;\dot{\leqslant}_C\; h \;\Leftrightarrow\; f \;\dot{\leqslant}_C\; \frac{h}{g} \tag{23}$$

**Scaled pointwise comparison**

$$f \;\dot{\leqslant}_C\; g \;=\; \langle \forall\, n \;::\; f\,(n) \leqslant C\,(g\,(n)) \rangle \tag{24}$$

# Comparing measures

**Maxima** (lubs etc)

$$f \sqcup g \dot{\leqslant} h \Leftrightarrow \begin{cases} f \dot{\leqslant} h \\ g \dot{\leqslant} h \end{cases} \tag{25}$$

$$(f \sqcup g) \cdot h = f \cdot h \sqcup g \cdot h \tag{26}$$

where

$$(f \sqcup g)\,(x) = max\,(f\,(x), g\,(x)) \tag{27}$$

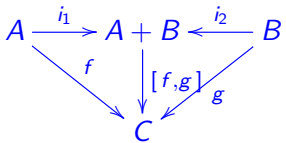
For non-negative $f$, $g$, we have $f \dot{\leqslant} f + g$ and $g \dot{\leqslant} f + g$, and therefore

$$f \sqcup g \dot{\leqslant} f + g \tag{28}$$

# Coproducts of measures

Recalling

$$k = [\, f, g \,] \Leftrightarrow \begin{cases} k \cdot i_1 = f \\ k \cdot i_2 = g \end{cases}$$




one has:

$$[f, g] + [h, k] = [f + h, g + k] \tag{29}$$

$$[f, g]\,[h, k] = [f\ h, g\ k] \tag{30}$$

$$C\,[f, g] = [C\ f, C\ g] \tag{31}$$

$$[f, g] + \underline{C} = [f + \underline{C}, g + \underline{C}] \tag{32}$$

# Coproducts of measures

These hold thanks to the **exchange law**

$$[\langle f\,,\,g\rangle, \langle h\,,\,j\rangle] = \langle [f,h]\,,\,[g,j]\rangle \tag{33}$$

and

$$[\,\underline{C}, \underline{C}\,] = \underline{C} \tag{34}$$

etc.

Some of the above laws extend to if-then-else, for instance

$$\underline{C} + (p \to f, g) = p \to \underline{C} + f, \underline{C} + g$$
$$C\ (p \to f, g) = p \to C\ f, C\ g$$

etc.

# Notation clash

Since $f + g$ and $f \times g$ (i.e. $f\ g$) are already taken, for the usual bifunctors we borrow notation and terminology from linear algebra:

**Direct sum**

$$f \oplus g = [\, i_1 \cdot f, i_2 \cdot g \,] \tag{35}$$

**Kronecker product**

$$(f \otimes g)\ (a, b) = (f\ a, g\ b) \tag{36}$$

# Asymptotic notation

# Asymptotic notation

We already know how to compare (functional) measures, be these simple pointwise comparison

$$f \dot{\leqslant} g \Leftrightarrow \langle \forall\ n\ ::\ f\,(n) \leqslant g\,(n) \rangle$$

or its scaling

$$f \dot{\leqslant}_C g = \langle \forall\ n\ ::\ f\,(n) \leqslant C\,(g\,(n)) \rangle$$

**Asymptotic notation** evolves from these, for measures of type $\mathbb{N}_0$ (size) to $\mathbb{K}$ (cost).

# Asymptotic notation

*"Let $f$ and $g$ be two functions taking natural numbers as arguments and returning nonnegative results, not necessarily integers. We say that (...) $f$ is of order* **at most** *$g$ and write $f = O(g)$ if there is a positive constant $C$ and a natural number $n_0$ such that $f(n) \leqslant C\, g(n)$ for all $n > n_0$."*

(Bird and Gibbons, 2020) – 'Algorithm Design with Haskell'

# Asymptotic notation

Written in symbols only, where $f, g \in \mathbb{K}^{\mathbb{N}_0}$:

$$f = \mathcal{O}(g) \quad \Leftrightarrow \quad \langle \exists\ n_0, C\ ::\ \langle \forall\ n\ :\ n > n_0 :\ f(n)\ \leqslant_C\ g(n) \rangle \rangle$$

**Remark**: $(= \mathcal{O})$ is an unfortunate choice of notation, for it clashes with the usual notation $y = f(x)$ for **functions**.

What we have is a higher-order **relation** $f\ \mathcal{O}\ g$. Thus we are faced with

a "typically relational" topic!

# Asymptotic notation

Our approach is to make such relations **parametric** — $\mathcal{O}_C^{n_0}$ — as follows:

$$f\ \mathcal{O}_C^{n_0}\ g \quad \Leftrightarrow \quad \langle \forall\ n\ :\ n \geqslant n_0 :\ f\ (n)\ \leqslant_C\ g\ (n) \rangle \tag{37}$$

Using relational algebra, (37) converts to

$$f\ \mathcal{O}_C^{n_0}\ g \quad \Leftrightarrow \quad f \cdot (\geqslant n_0)? \subseteq (\leqslant_C) \cdot g \tag{38}$$

where $p?$ is the **coreflexive** binary relation:

$$b\ p?\ a \quad \Leftrightarrow \quad p\ (a) \wedge b = a \tag{39}$$

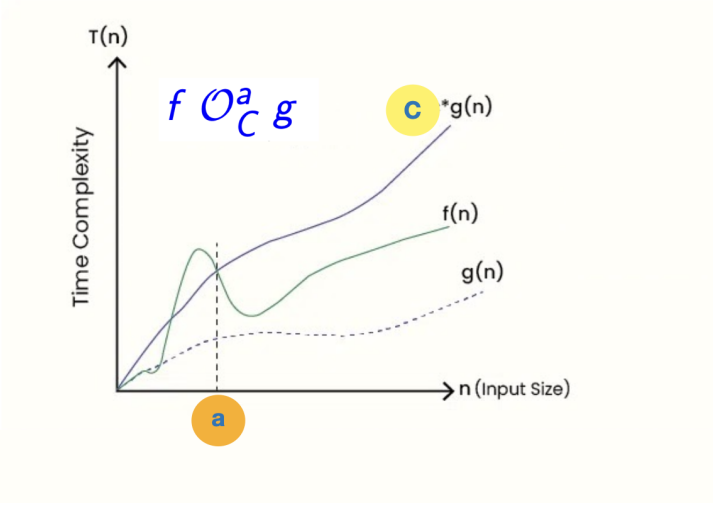
# Asymptotic notation

Thus big-$\mathcal{O}$ facts are **Reynolds squares**

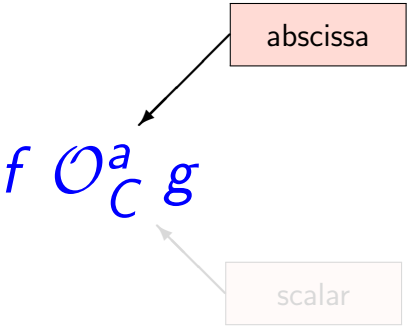
$$f\ \mathcal{O}_C^{n_0}\ g \quad \Leftrightarrow \quad \begin{array}{ccc} \mathbb{N}_0 & \xleftarrow{(\geqslant n_0)?} & \mathbb{N}_0 \\ {\scriptstyle f}\downarrow & \subseteq & \downarrow{\scriptstyle g} \\ \mathbb{K} & \xleftarrow[(\leqslant c)]{} & \mathbb{K} \end{array}$$

$$\langle \forall\ n\ :\ n \geqslant n_0 :\ f\ (n)\ \leqslant_C\ g\ (n) \rangle$$

# Some terminology



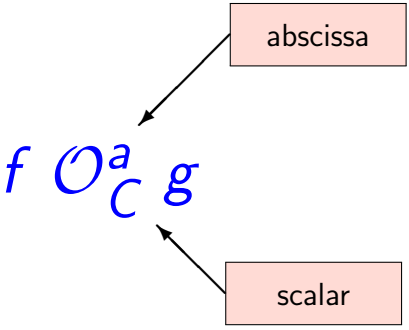
# Some terminology




The **abscissa** (from the Latin *ubi linea abscissa* = 'where the line is cut off')

The **scalar** (from the Latin *scalaris* = 'the scale')

# Some terminology




The **abscissa** (from the Latin *ubi linea abscissa* = 'where the line is cut off')

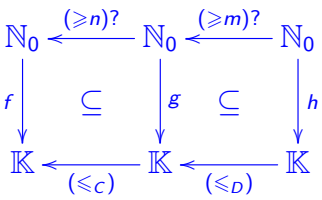
The **scalar** (from the Latin *scalaris* = 'the scale')

# Asymptotic notation

## Transitivity

$$\mathcal{O}_C^n \cdot \mathcal{O}_D^m \subseteq \mathcal{O}_{CD}^{max\,(m,n)} \tag{40}$$

holds by Reynolds square **horizontal composition**:




(Recall $(\leqslant_C) \cdot (\leqslant_D) \subseteq (\leqslant_{CD})$ and maxima.)

# Asymptotic notation

## Reflexivity

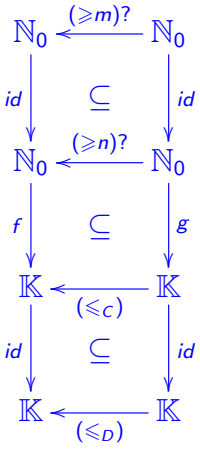
$$id \subseteq \mathcal{O}^n_C \quad \Leftrightarrow \quad \begin{array}{ccc} \mathbb{N}_0 & \xleftarrow{(\geqslant n)?} & \mathbb{N}_0 \\ {\scriptstyle f}\downarrow & \subseteq & \downarrow{\scriptstyle f} \\ \mathbb{K} & \xleftarrow[(\leqslant C)]{} & \mathbb{K} \end{array}$$

for scalar $C \geqslant 1$, since $f : (\geqslant n)? \to (\leqslant C)$ etc.

So, **big-$\mathcal{O}$ notation** amounts to an indexed collection of (higher-order) **preorders**.

# Asymptotic notation

**$\mathcal{O}$-weakening**:




$$\mathcal{O}^n_C \subseteq \mathcal{O}^m_D \Leftarrow \begin{cases} n \leqslant m \\ C \leqslant D \end{cases} \tag{41}$$

(abscissas and scalars can always grow larger)

# Least $\mathcal{O}$-preorder

Since

$$(\dot{\leqslant}) = \mathcal{O}_1^0$$

we have that $(\dot{\leqslant})$ is the least of all $\mathcal{O}$-preorders, that is

$$f \dot{\leqslant} g \;\Rightarrow\; f\ \mathcal{O}_C^n\ g$$

for $C \geqslant 1$.

---

**Shorthand**: *$C = 1$ and $n = 0$ are often omitted, so e.g. we write $\mathcal{O}_C$ instead of $\mathcal{O}_C^0$ and $\mathcal{O}^n$ for $\mathcal{O}_1^n$. Trivially,*

$$\underline{0}\ \mathcal{O}\ f$$

*holds for any non-negative $f$.*

---

# Improving (38)

It can be observed that $\mathbb{N}_0 \xrightarrow{(\geqslant n_0)?} \mathbb{N}_0$ is the range

$$(n_0+) \cdot (n_0+)^\circ \qquad (42)$$

of $(n_0+)$ — see aside.

Then (next slide):

$$
\begin{array}{ll}
 & y\ ((n_0+) \cdot (n_0+)^\circ)\ x \\
\Leftrightarrow & \quad \{\ z \geqslant 0 \text{ for all } z \in \mathbb{N}_0\ \} \\
 & \langle \exists\ z\ :\ z \geqslant 0 :\ y = n_0 + z \wedge x = n_0 + z \rangle \\
\Leftrightarrow & \quad \{\ \exists\text{-trading}\ \} \\
 & \langle \exists\ z\ :\ z = y - n_0 :\ z \geqslant 0 \wedge x = n_0 + z \rangle \\
\Leftrightarrow & \quad \{\ \exists\text{-one point}\ \} \\
 & y - n_0 \geqslant 0 \wedge x = n_0 + (y - n_0) \\
\Leftrightarrow & \quad \{\ \text{trivia}\ \} \\
 & y \geqslant n_0 \wedge x = y \\
\Leftrightarrow & \quad \{\ \text{coreflexive (39)}\ \} \\
 & y\ (\geqslant n_0)?\ x
\end{array}
$$

# Improving (38)

It can be observed that $\mathbb{N}_0 \xrightarrow{(\geqslant n_0)?} \mathbb{N}_0$ is the range

$$(n_0+)\cdot(n_0+)^\circ \qquad (42)$$

of $(n_0+)$ — see aside.

Then (next slide):

$$
\begin{array}{ll}
 & y\ ((n_0+)\cdot(n_0+)^\circ)\ x \\
\Leftrightarrow & \quad \{\ z \geqslant 0 \text{ for all } z \in \mathbb{N}_0\ \} \\
 & \langle \exists\ z\ :\ z \geqslant 0 :\ y = n_0 + z \wedge x = n_0 + z \rangle \\
\Leftrightarrow & \quad \{\ \exists\text{-trading}\ \} \\
 & \langle \exists\ z\ :\ z = y - n_0 :\ z \geqslant 0 \wedge x = n_0 + z \rangle \\
\Leftrightarrow & \quad \{\ \exists\text{-one point}\ \} \\
 & y - n_0 \geqslant 0 \wedge x = n_0 + (y - n_0) \\
\Leftrightarrow & \quad \{\ \text{trivia}\ \} \\
 & y \geqslant n_0 \wedge x = y \\
\Leftrightarrow & \quad \{\ \text{coreflexive (39)}\ \} \\
 & y\ (\geqslant n_0)?\ x
\end{array}
$$

# Improving (38)

$$f\ \mathcal{O}_C^{n_0}\ g \quad \Leftrightarrow \quad f \cdot (n_0+) \mathrel{\dot{\leqslant}} Cg \cdot (n_0+) \tag{43}$$

Calculation:

$$\begin{array}{ll} & f\ \mathcal{O}_C^{n_0}\ g \\ \Leftrightarrow & \quad \{\ (38);\ \text{coreflexive}\ (\geqslant n_0)? = (n_0+) \cdot (n_0+)^\circ\ \} \\ & f \cdot (n_0+) \cdot (n_0+)^\circ \subseteq (\leqslant_C) \cdot g \\ \Leftrightarrow & \quad \{\ \text{shunting}\ \} \\ & f \cdot (n_0+) \subseteq (\leqslant_C) \cdot g \cdot (n_0+) \\ \Leftrightarrow & \quad \{\ (19);\ (13);\ (16)\ \} \\ & f \cdot (n_0+) \mathrel{\dot{\leqslant}} Cg \cdot (n_0+) \end{array}$$

# Asymptotic notation

## Constant time

$f$ is constant time — vulg. $\mathcal{O}(1)$ — wherever $f\ \mathcal{O}^n_C\ \underline{1}$ holds.

Clearly:

$$\underline{C}\ \mathcal{O}_C\ \underline{1} \tag{44}$$

## Linear time

$f$ is linear — vulg. $\mathcal{O}(n)$ — wherever $f\ \mathcal{O}^k_C\ id$ holds. (But mind the abuse of the type of "$id$"...)

Clearly:

$$(C\times)\ \mathcal{O}_C\ id \tag{45}$$

# Asymptotic notation

**Constant factor rule**:

$$f\ \mathcal{O}^n_D\ g \quad \Leftrightarrow \quad (C\ f)\ \mathcal{O}^n_{CD}\ g$$

that is,

$$\mathcal{O}^n_D \ = \ (C\times)^\circ \cdot \mathcal{O}^n_{CD}$$

Thus

$$(C\ f)\ \mathcal{O}_C\ f$$

etc.

# Asymptotic notation

Exercise 2.2 of (Bird and Gibbons, 2020):

*Are the following two assertions true?*

$$f\ (n) \times \mathcal{O}\ (g\ (n)) = \mathcal{O}\ (f\ (n) \times g\ (n))$$
$$f\ (n) + \mathcal{O}\ (g\ (n)) = \mathcal{O}\ (f\ (n) + g\ (n))$$

□

Written pointfree, the first assertion becomes

$$f\ (\mathcal{O}\ g) = \mathcal{O}\ (f\ g)$$

the "spirit" of which is:

$$h = \mathcal{O}\ g \ \Rightarrow\ f\ h = \mathcal{O}\ (f\ g) \tag{46}$$

— just replace $h$ and simplify.

# Asymptotic notation

Exercise 2.2 of (Bird and Gibbons, 2020):

*Are the following two assertions true?*

$$f\ (n) \times \mathcal{O}\ (g\ (n)) = \mathcal{O}\ (f\ (n) \times g\ (n))$$
$$f\ (n) + \mathcal{O}\ (g\ (n)) = \mathcal{O}\ (f\ (n) + g\ (n))$$

□

Written pointfree, the first assertion becomes

$$f\ (\mathcal{O}\ g) = \mathcal{O}\ (f\ g)$$

the "spirit" of which is:

$$h = \mathcal{O}\ g\ \Rightarrow\ f\ h = \mathcal{O}\ (f\ g) \tag{46}$$

— just replace $h$ and simplify.

# Asymptotic notation

But the actual meaning of (46) is:

$$h \in \Lambda\mathcal{O}\ g \ \Rightarrow\ f\ h \in \Lambda\mathcal{O}\ (f\ g)$$

where $\Lambda R\ x = \{y \mid y\ R\ x\}$. Then:

$$
\begin{aligned}
& h \in \Lambda\mathcal{O}\ g \Rightarrow f\ h \in \Lambda\mathcal{O}\ (f\ g) \\
\Leftrightarrow & \quad \{\ \text{power transpose cancellation}\ \} \\
& h\ \mathcal{O}\ g \Rightarrow (f\ h)\ \mathcal{O}\ (f\ g) \\
\Leftrightarrow & \quad \{\ \text{go pointfree (higher order)}\ \} \\
& \mathcal{O}\ \subseteq\ (f\times)^{\circ} \cdot \mathcal{O} \cdot (f\times) \\
\Leftrightarrow & \quad \{\ \text{shunting; add parameters}\ \} \\
& (f\times) \cdot \mathcal{O}^{n}_{C} \subseteq \mathcal{O}^{n}_{C} \cdot (f\times)
\end{aligned}
$$

# Asymptotic notation

But the actual meaning of (46) is:

$$h \in \Lambda\mathcal{O}\ g \ \Rightarrow\ f\ h \in \Lambda\mathcal{O}\ (f\ g)$$

where $\Lambda R\ x = \{y \mid y\ R\ x\}$. Then:

$$
\begin{aligned}
& h \in \Lambda\mathcal{O}\ g \Rightarrow f\ h \in \Lambda\mathcal{O}\ (f\ g) \\
\Leftrightarrow & \quad \{\ \text{power transpose cancellation}\ \} \\
& h\ \mathcal{O}\ g \Rightarrow (f\ h)\ \mathcal{O}\ (f\ g) \\
\Leftrightarrow & \quad \{\ \text{go pointfree (higher order)}\ \} \\
& \mathcal{O}\ \subseteq\ (f\times)^{\circ} \cdot \mathcal{O} \cdot (f\times) \\
\Leftrightarrow & \quad \{\ \text{shunting; add parameters}\ \} \\
& (f\times) \cdot \mathcal{O}^{n}_{C} \subseteq \mathcal{O}^{n}_{C} \cdot (f\times)
\end{aligned}
$$

# Asymptotic notation

The meaning of (46) is thus the Reynolds square

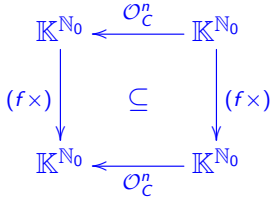
$$\begin{array}{ccc} \mathbb{K}^{\mathbb{N}_0} & \xleftarrow{\mathcal{O}_C^n} & \mathbb{K}^{\mathbb{N}_0} \\ \downarrow{\scriptstyle (f\times)} & \subseteq & \downarrow{\scriptstyle (f\times)} \\ \mathbb{K}^{\mathbb{N}_0} & \xleftarrow[\mathcal{O}_C^n]{} & \mathbb{K}^{\mathbb{N}_0} \end{array} \tag{47}$$

i.e. $(f\times)$ "preserves" $\mathcal{O}_C^n$, cf. **relational type** $(f\times) : \mathcal{O}_C^n \to \mathcal{O}_C^n$

Let us calculate (47), introducing variables $h$ and $g$:

# Asymptotic notation

The meaning of (46) is thus the Reynolds square




(47)

i.e. $(f\times)$ "preserves" $\mathcal{O}_C^n$, cf. **relational type** $(f\times) : \mathcal{O}_C^n \to \mathcal{O}_C^n$

Let us calculate (47), introducing variables $h$ and $g$:

# Asymptotic notation

$$
\begin{aligned}
& f\ h\ \mathcal{O}^n_C\ f\ g \\
\Leftrightarrow \quad & \{\ \text{big-}\mathcal{O}\ \text{in version (43) )}\ \} \\
& f\ h \cdot (n+)\ \dot{\leqslant}\ C\ (f\ g) \cdot (n+) \\
\Leftrightarrow \quad & \{\ \text{distribution (12) and scaling (15)}\ \} \\
& (f \cdot (n+))\ (h \cdot (n+))\ \dot{\leqslant}\ (f \cdot (n+))\ (C\ g \cdot (n+)) \\
\Leftarrow \quad & \{\ \times\text{-monotonicity, assuming } f \text{ non-negative}\ \} \\
& \begin{cases} f \cdot (n+) \dot{\leqslant} f \cdot (n+) \\ h \cdot (n+) \dot{\leqslant} (C\ g) \cdot (n+) \end{cases} \\
\Leftrightarrow \quad & \{\ (\dot{\leqslant})\text{-reflexivity; } \mathcal{O}\ (43)\ \} \\
& h\ \mathcal{O}^n_C\ g
\end{aligned}
$$

# Product and sum rules

**Product rule**: by transitivity (40), (47) generalizes to:

$$\begin{cases} f\ \mathcal{O}_C^m\ h \\ g\ \mathcal{O}_D^n\ k \end{cases} \Rightarrow f\ g\ \mathcal{O}_{C\ D}^{max\ (m,n)}\ h\ k \tag{48}$$

**Sum rule**

$$\mathcal{O}\ (f) + \mathcal{O}\ (g) = \mathcal{O}\ (f \sqcup g)$$

that is, if "$h = \mathcal{O}\ (f)$" and "$k = \mathcal{O}\ (g)$" then "$h + k = \mathcal{O}\ (f \sqcup g)$"

and more formally:

$$\begin{cases} h\ \mathcal{O}\ f \\ k\ \mathcal{O}\ g \end{cases} \Rightarrow (h + k)\ \mathcal{O}\ (f \sqcup g)$$

# Asymptotic notation

In detail:

$$\begin{cases} h\ \mathcal{O}_C^m\ f \\ k\ \mathcal{O}_D^n\ g \end{cases} \Rightarrow (h+k)\ \mathcal{O}_{C+D}^{max\,(m,n)}\ (f \sqcup g) \tag{49}$$

Reynolds square:

$$\begin{array}{ccc} \mathbb{K}^{n_0} \times \mathbb{K}^{n_0} & \xleftarrow{\mathcal{O}_C^m \times \mathcal{O}_D^m} & \mathbb{K}^{n_0} \times \mathbb{K}^{n_0} \\ \downarrow{\scriptstyle (+)} & \subseteq & \downarrow{\scriptstyle (\cup)} \\ \mathbb{K}^{n_0} & \xleftarrow[\mathcal{O}_{C+D}^{max\,(m,n)}]{} & \mathbb{K}^{n_0} \end{array} \qquad (+) \cdot (\mathcal{O}_C^m \times \mathcal{O}_D^n) \subseteq \mathcal{O}_{C+D}^{max\,(m,n)} \cdot (\cup) \tag{50}$$

# Calculation of (49) — ie. (50)

Let us abbreviate $max\ (m, n)$ to $p$, that is,

$$\begin{cases} m \leqslant p \\ n \leqslant p \end{cases} \qquad (51)$$

hold. Then:

$$\begin{array}{ll} & (h+k)\ \mathcal{O}^{p}_{C+D}\ (f \sqcup g) \\ \Leftrightarrow & \quad \{\ \text{definition (43)}\ \} \\ & (h+k) \cdot (p+) \mathrel{\dot{\leqslant}} (C+D)\ (f \sqcup g) \cdot (p+) \\ \Leftrightarrow & \quad \{\ \text{distributions (10), (11), (26)}\ \} \\ & h \cdot (p+) + k \cdot (p+) \mathrel{\dot{\leqslant}} C\ (f \sqcup g) \cdot (p+) + D\ (f \sqcup g) \cdot (p+) \\ \Leftarrow & \quad \{\ (+) \text{ is } \dot{\leqslant}\text{-monotonic}\ \} \\ & \begin{cases} h \cdot (p+) \mathrel{\dot{\leqslant}} C\ (f \sqcup g) \cdot (p+) \\ k \cdot (p+) \mathrel{\dot{\leqslant}} D\ (f \sqcup g) \cdot (p+) \end{cases} \\ \Leftarrow & \quad \{\ \mathcal{O}\text{-weakening (41) by (51); then lower the upper sides}\ \} \\ & \begin{cases} h\ \mathcal{O}^{m}_{C}\ f \\ k\ \mathcal{O}^{n}_{D}\ g \end{cases} \end{array}$$

# Asymptotic notation

**Thumb rule** Corollary

$$f \; \mathcal{O}_C^m \; g \;\Rightarrow\; (f + g) \; \mathcal{O}_C^m \; g \tag{52}$$

immediately follows from (50) since $g \sqcup g = g$ and $g \stackrel{\cdot}{\leqslant} g$.

Thus the thumb rule:

---

*"If $f\ (x)$ is a sum of several terms, if there is one with largest growth rate, it can be kept, and all others omitted".*

---

# Linear functions

**Example**: An **affine** function $f\ (n) = A + B\ (n)$ is expected to be $\mathcal{O}$-linear.

---

*Which $C$ and $n$ in $\mathcal{O}_C^n$ best tell that such $f$ is linear?*

---

Since $(\underline{A}\ \mathcal{O}_A^0\ \underline{1})$ and $(B\times)\ \mathcal{O}_B^0\ id$, by the sum rule (50):

$$\underline{A} + (B\times)\ \mathcal{O}_{A+B}^0\ (\underline{1} \sqcup id)$$

It is easy to calculate $(\underline{1} \sqcup id)\ \mathcal{O}_1^1\ id$. Then, by **transitivity**,

$$\underline{A} + (B\times)\ \mathcal{O}_{A+B}^1\ id$$

as expected.

# Big-O and function composition

Another question: (Discussion at math.stackexchange.com)

> *"On the last page of this document, a property of big-O operations is listed which says that if*
>
> *$f1(n)=O(g1(n))$, and $f2(n)=O(g2(n))$*
>
> *Then $f1 \circ f2 = O(g1 \circ g2)$*
>
> *Why is that?"*

Really?

# Big-O and function composition

Another question: (Discussion at math.stackexchange.com)

> *"On the last page of this document, a property of big-O operations is listed which says that if*
>
> *$f1(n)=O(g1(n))$, and $f2(n)=O(g2(n))$*
>
> *Then $f1 \circ f2 = O(g1 \circ g2)$*
>
> *Why is that?"*

Really?

# Big-O and function composition

**Question** *"On the last page of this document, a property of big-O operations is listed which says that if*

$f1(n) = O(g1(n))$, *and*
$f2(n) = O(g2(n))$

*Then* $f1 \circ f2 = O(g1 \circ g2)$

*Why is that?"*

**Answer** *"This is false unless you make extra assumptions. Assume that* $f2(x) = 2x$ , $g2(x) = x$. *Then* $f2(x) = O(g2(x))$. *Now take* $f1(x) = g1(x) = e^x$. *Then you get* $f1(f2(x)) = e^{2x}$ *and* $g1(g2(x)) = e^x$, *and* $e^{2x} \neq O(e^x)$ *so there's something wrong."*

---

*Let us calculate the right answer.*

---

## Composition Rule

$$(f \cdot g)\ \mathcal{O}^p_C\ (h \cdot k) \Leftarrow \begin{cases} f\ \mathcal{O}_C\ h \\ g\ \mathcal{O}^p\ k \\ h \text{ is monotone} \end{cases}$$

$$\begin{array}{ll} & (f \cdot g)\ \mathcal{O}^p_C\ (h \cdot k) \\ \Leftrightarrow & \quad \{\ \text{relational unfolding of } \mathcal{O}^p_C\ \} \\ & f \cdot g \cdot (p+) \subseteq (\leqslant) \cdot C\ (h \cdot k) \cdot (p+) \\ \Leftarrow & \quad \{\ (\leqslant)\text{-transitivity}\ \} \\ & f \cdot g \cdot (p+) \subseteq (\leqslant) \cdot (\leqslant) \cdot C\ h \cdot k \cdot (p+) \\ \Leftarrow & \quad \{\ \text{assume } \leqslant\text{-monotone } h\ \} \\ & f \cdot g \cdot (p+) \subseteq (\leqslant) \cdot C\ h \cdot (\leqslant) \cdot k \cdot (p+) \\ \Leftarrow & \quad \{\ \text{relation composition is monotone}\ \} \\ & \begin{cases} f \subseteq (\leqslant) \cdot C\ h \\ g \cdot (p+) \subseteq (\leqslant) \cdot k \cdot (p+) \end{cases} \\ \Leftrightarrow & \quad \{\ \text{definitions}\ \} \\ & \begin{cases} f\ \mathcal{O}^0_C\ h \\ g\ \mathcal{O}^p_1\ k \end{cases} \end{array}$$

# Asymptotic notation

**Question** *"On the last page of this document, a property of big-O operations is listed which says that if*

$f1(n) = O(g1(n))$, *and*
$f2(n) = O(g2(n))$

*Then f1∘f2=O(g1∘g2)*

*Why is that?"*

**Answer** *"This is false unless you make extra assumptions. Assume that* $f2(x) = 2x$ , $g2(x) = x$. *Then* $f2(x) = O(g2(x))$. *Now take* $f1(x) = g1(x) = e^x$. *Then you get* $f1(f2(x)) = e^{2x}$ *and* $g1(g2(x)) = e^x$, *and* $e^{2x} \neq O(e^x)$ *so there's something wrong."*

---

$f_1 \ \mathcal{O} \ g_1$ — *ok:* $(e^x \ \mathcal{O} \ e^x)$
$f_2 \ \mathcal{O} \ g_2$ — *no:* $(2\ x) \ \mathcal{O}_2 \ x$ *fails the composition rule since* $2 > 1$.

---

# Structural Cost Analysis

# Cost analysis

Cost analysis of **functional programs** is quite an old topic, carried out over e.g. **LISP**:

> *This paper (...) describes a system, Metric, which is able to analyze simple Lisp programs and produce, for example, closed-form expressions for their running time expressed in terms of size of input.*
>
> Wegbreit (1975) 'Mechanical program analysis'

> *This paper describes a method to construct time bound programs for programs in a first-order [L]isp subset. (...) All the algorithms described in this paper have been realised in Franz Lisp (...)*
>
> (Rosendahl, 1989), 'Automatic Complexity Analysis'

# Cost analysis

Also based on Backus' **FP**:

> *Our system, called ACE (for Automatic Complexity Evaluator), uses FP with a non-strict semantics as its source language (BACKUS 78). (...) FP axioms are then used to perform simplifications and program transformations; the final step is the application of the recursion induction principle in order to get a non-recursive function (MACCARTHY 63) (...)*
>
> (Métayer, 1985) 'Mechanical analysis of program complexity'

# Cost analysis

As already meantioned in the meeting, the following gives a comprehensive calculus for cost-equation derivation for a higher-order functional language:

> *(...) The derivation yields a functional program, is mechanisable, and thus provides a concise calculus for reasoning about the time complexity of higher-order functions (...)*
>
> (Sands, 1990) 'Calculi for Time Analysis of Functional Programs'

# Cost analysis

of these definitions. The syntax of expressions in this language is defined in figure 3.1

| | | | |
|---|---|---|---|
| $e$ | $::=$ | $e(e_1, \ldots, e_j)$ | (application) |
| | $\mid$ | $f_i$ | (function) |
| | $\mid$ | $p$ | (primitive function) |
| | $\mid$ | $x$ | (identifier) |
| | $\mid$ | $c$ | (constants) |
| | $\mid$ | **if** $e_1$ **then** $e_2$ **else** $e_3$ | (conditional) |

Figure 3.1: Expression Syntax

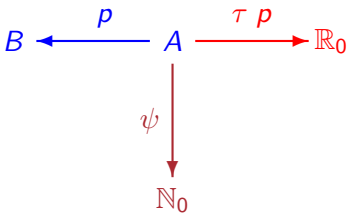
(Sands, 1990) 'Calculi for Time Analysis of Functional Programs'

# This approach

- Unfold structural cost analysis across standard **Algebra of Programming** (AoP) combinators, from basic to e.g. catamorphisms and anamorphisms.
- Instead of recursion induction principles, use **relation algebra** to solve the relational **recurrence equations** that arise in the analysis.
- In particular, starts by checking which AoP **laws** are "preserved" by cost-analysis.

# Cost analysis

Recall:




The core definition is how $\tau$ handles **function composition**,

$$\tau\ (f \cdot g)\ x = (\tau\ f)\ (g\ x) + \tau\ g\ x$$

i.e., in the point-free style:

$$\tau\ (f \cdot g) = \tau\ f \cdot g + \tau\ g \tag{53}$$

Note that, as a unary operator, $\tau$ takes precedence over "dot":

$$\tau\ f \cdot g = (\tau\ f) \cdot g$$

# Analogy

**Cost** (time)

$$\tau\,(f \cdot g) = \tau\, f \cdot g + \tau\, g$$

**Derivative** (change)

$$(f \cdot g)' = (f' \cdot g) \;\times\; g'$$

Cost is *additive*, change is *multiplicative*.

# Pairing

Recall the **pairing** combinator $\langle f\ ,\ g\rangle\ x = (f\ x, g\ x)$.

Costs add up when pairing computations,

$$\tau\ \langle f\ ,\ g\rangle = \tau\ f + \tau\ g \tag{54}$$

assuming an **eager** evaluation semantics with neither **parallelism** nor **concurrency**.

Otherwise, e.g. in a parallel environment, one would have

$$\tau\ \langle f\ ,\ g\rangle = \tau\ f \sqcup \tau\ g$$

instead of (54).

# Copairing

Alternative costs:

$$\tau\,[f, g] = [\tau\, f, \tau\, g] \tag{55}$$
$$\tau\,(f \oplus g) = [\tau\, f, \tau\, g] \tag{56}$$

**Conditionals**

$$\tau\,(p \to f, g) = (p \to \tau\, f, \tau\, g) + \tau\, p \tag{57}$$

follows from (55).

# Basic cost analysis

## Basic

$$\tau\ id = \underline{0} \tag{58}$$
$$\tau\ i_1 = \tau\ i_2 = \underline{0} \tag{59}$$
$$\tau\ \underline{k} = \underline{C} \quad \text{for some scalar } C \tag{60}$$

**Isomorphism** - for $\alpha$ an isomorphism and $f := g \cdot \alpha$:

$$(\tau\ f) \cdot \alpha^\circ \ = \ \tau\ g + (\tau\ \alpha) \cdot \alpha^\circ \tag{61}$$

(Calculation in the annex, slide 93.)

# Structural costs

Functor $\mathbb{F}$:

$$\tau\ (\mathbb{F}\ f) = \epsilon_{\mathbb{F}} \cdot \mathbb{F}\ (\tau\ f) \tag{62}$$

where **structural summation** $\epsilon_{\mathbb{F}} : \mathbb{F}\ \mathbb{K} \to \mathbb{K}$ is defined by induction on the stucture of $\mathbb{F}$, e.g. polynomial:

$$\begin{aligned}
&\epsilon_K = \underline{0} \\
&\epsilon_I = id \\
&\epsilon_{(\mathbb{T}_1 \times \mathbb{T}_2)} = \epsilon_{\mathbb{T}_1} \cdot \pi_1 + \epsilon_{\mathbb{T}_2} \cdot \pi_2 \\
&\epsilon_{(\mathbb{T}_1 + \mathbb{T}_2)} = [\,\epsilon_{\mathbb{T}_1}, \epsilon_{\mathbb{T}_2}\,] \\
&\epsilon_{(\mathbb{T}_1 \cdot \mathbb{T}_2)} = \epsilon_{\mathbb{T}_1} \cdot \mathbb{T}_1\ (\epsilon_{\mathbb{T}_2})
\end{aligned} \tag{63}$$

# Cost analysis of AoP Laws

**Extensional equality** does not imply equal computation time:

$$f = g \not\Rightarrow \tau\, f = \tau\, g$$

By contrast, when a function is declared (i.e. implemented) by another, it shares its time behavior:

$$f := g \Rightarrow \tau\, f = \tau\, g$$

Thus one needs to check which **laws** of the Algebra of Programming are **preserved** by $\tau$.

Equal computation time **equivalence relation**:

$$f \simeq g \Leftrightarrow \tau\, f = \tau\, g \tag{64}$$

# Cost analysis of AoP Laws

As expected:

$$f \cdot id \simeq f \simeq id \cdot f$$
$$(f \cdot g) \cdot h \simeq f \cdot (g \cdot h)$$

**Coproduct fusion** is preserved:

$$f \cdot [h, k] \simeq [f \cdot h, f \cdot k] \tag{65}$$

# Absoption + exchange laws

More AoP laws preserved:

$$(i \times j) \cdot \langle g\ ,\ h \rangle \simeq \langle i \cdot g\ ,\ j \cdot h \rangle \tag{66}$$
$$[\,f, g\,] \cdot (h \oplus k) \simeq [\,f \cdot h, g \cdot k\,] \tag{67}$$
$$[\,\langle f\ ,\ g \rangle, \langle h\ ,\ j \rangle\,] \simeq \langle [\,f, h\,]\ ,\ [\,g, j\,] \rangle \tag{68}$$

However, **product fusion** is not respected:

$$\tau\ (\langle g\ ,\ h \rangle \cdot f) \leqslant \tau\ \langle g \cdot f\ ,\ h \cdot f \rangle \tag{69}$$

# Pairing cancellation

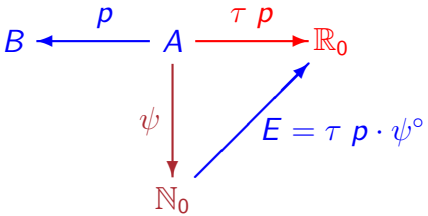
The well-known AoP equalities

$$\begin{cases} \pi_1 \cdot \langle f\,,\, g\rangle = f \\ \pi_2 \cdot \langle f\,,\, g\rangle = g \end{cases} \tag{70}$$

are illustrative about alternative **evaluation** strategies. By the composition rule (53) and assuming $\tau\ \pi_1$ constant, e.g.:

$$\begin{aligned} \tau\ (\pi_1 \cdot \langle f\,,\, g\rangle) &= \underline{\tau_{\pi_1}} + \tau\ \langle f\,,\, g\rangle \\ &= \begin{cases} \underline{\tau_{\pi_1}} + \tau\ f + \tau\ g & \text{— eager evaluation, no parallelism} \\ \underline{\tau_{\pi_1}} + (\tau\ f \sqcup \tau\ g) & \text{— eager evaluation + parallelism} \\ \underline{\tau_{\pi_1}} + \tau\ f & \text{— lazy evaluation} \end{cases} \end{aligned}$$

# Handling cost relations




Recall that in cost analysis $E$ is a relation between sizes and costs.

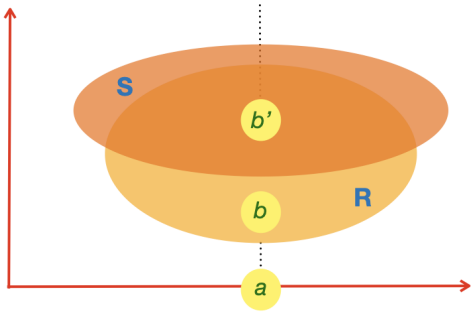
How does what we have seen above generalize to relations?

# Comparing relational measures

$$R \dot{\leqslant} S \;\Leftrightarrow\; R \subseteq (\leqslant) \cdot S \tag{71}$$

meaning:

$$\langle \forall\ b, a\ :\ b\ R\ a :\ \langle \exists\ b'\ :\ b \leqslant b' :\ b'\ S\ a \rangle \rangle$$



# Relational measures

We need to our basic measure operation definitions (2) to relations:

$$R + S = \widehat{(+)} \cdot \langle R\,,\, S \rangle \tag{72}$$
$$R\ S = \widehat{(\times)} \cdot \langle R\,,\, S \rangle \tag{73}$$
$$C\ R = (C\times) \cdot R \tag{74}$$

respectively meaning:

$$k\ (R + S)\ a \Leftrightarrow \langle \exists\ x, y\ :\ x\ R\ a \wedge y\ S\ a :\ k = x + y \rangle$$
$$k\ (R\ S)\ a \Leftrightarrow \langle \exists\ x, y\ :\ x\ R\ a \wedge y\ S\ a :\ k = x * y \rangle$$
$$k\ (C\ R)\ a \Leftrightarrow \langle \exists\ x\ :\ x\ R\ a :\ k = C\ a \rangle$$

# Relational measures

However, distribution equalities such as e.g. (10) become inequalities, e.g.

$$(S + Q) \cdot R \subseteq S \cdot R + Q \cdot R \tag{75}$$

Under some conditions, one may still have the equalities we had before, e.g.

$$(f + \underline{C}) \cdot R = f \cdot R + \underline{C} \quad \Leftarrow R \textit{ entire}$$

# Big-$\mathcal{O}$ on relational measures

Recall (38)

$$f \; \mathcal{O}_C^{n_0} \; g \quad \Leftrightarrow \quad f \cdot (\geqslant n_0)? \subseteq (\leqslant_C) \cdot g$$

and extend it to relational measure $\mathbb{N}_0 \xrightarrow{E} \mathbb{K}$:

$$E \; \mathcal{O}_C^{n_0} \; g \quad \Leftrightarrow \quad E \cdot (\geqslant n_0)? \subseteq (\leqslant_C) \cdot g$$

What is the impact of this?

# Big-$\mathcal{O}$ on relational measures

We calculate:

$$
\begin{array}{ll}
 & E\ \mathcal{O}_C^{n_0}\ g \\
\Leftrightarrow & \quad \{\ \text{above}\ \} \\
 & E \cdot (\geqslant n_0)? \subseteq (\leqslant_C) \cdot g \\
\Leftrightarrow & \quad \{\ \text{residuate it}\ \} \\
 & (\geqslant n_0)? \subseteq E \setminus ((\leqslant_C) \cdot g) \\
\Leftrightarrow & \quad \{\ \text{go pointwise}\ \} \\
 & \langle \forall\ n\ :\ n \geqslant n_0 :\ \langle \forall\ y\ :\ y\ E\ n :\ y\ \leqslant_C\ (g\ n) \rangle \rangle
\end{array}
$$

# Big-$\mathcal{O}$ on relational measures

That is:

$$\begin{array}{ll} & \langle \forall\ n\ :\ n \geqslant n_0 :\ \langle \forall\ y\ :\ y\ E\ n :\ y\ \leqslant_C\ (g\ n) \rangle \rangle \\ \Leftrightarrow & \quad \{\ \text{lubs}\ \} \\ & \langle \forall\ n\ :\ n \geqslant n_0 :\ \langle Max\ y\ :\ y\ E\ n :\ y \rangle \rangle\ \leqslant_C\ (g\ n)) \end{array}$$

Thus relational

$$E\ \mathcal{O}_C^{n_0}\ g$$

will perform **worst case analysis** by definition.

# Abstract (relational) cost analysis

Recall cost of **composition** (53):

$$\tau\ (f \cdot g) = \tau\ f \cdot g + \tau\ g$$

Let us generalize from **functional** $\tau$ to **relational** $E$:

$$\begin{array}{ll} & E\ (f \cdot g) \\ = & \quad \{\ (1,\ 53)\ \} \\ & (\tau\ f \cdot g + \tau\ g)\ \psi^{\circ} \\ \subseteq & \quad \{\ (75)\ \} \\ & \tau\ f \cdot g \cdot \psi^{\circ} + E\ g \\ \subseteq & \quad \{\ \text{allow for different structure with same size}\ \} \\ & \tau\ f \cdot \psi^{\circ} \cdot \psi \cdot g \cdot \psi^{\circ} + E\ g \\ = & \quad \{\ \text{definition (1)}\ \} \\ & E\ f \cdot (\psi \cdot g \cdot \psi^{\circ}) + E\ g \end{array}$$

# Abstract (relational) cost analysis

Define "$\psi$-abstracted" $g$ as

$$[g]^{\psi} = \psi \cdot g \cdot \psi^{\circ} \tag{76}$$

that is,

$$n\ [g]^{\psi}\ m \quad \Leftrightarrow \quad \langle \exists\ x\ :\ n = \psi\ (g\ x) :\ \psi\ x = m \rangle$$

That is, $[g]^{\psi}$ gives us an abstract view of $g$ as a "size transformer".

Then:

$$E\ (f \cdot g)\ \subseteq\ (E\ f) \cdot [g]^{\psi} + E\ g \tag{77}$$

# Relational cost laws

Inequation (77) has an impact on how $\tau$-laws generalize to $E$-laws: while e.g.

$$E\ (f \cdot id) = E\ f = E\ (id \cdot f)$$

and

$$E\ [\,f, g\,] = [\,E \cdot f, E \cdot g\,]$$

composition associativity is no longer an equality.

But let us first show how to handle **recursion**.

# Part II — Timing induction

This second part was left for a future meeting.

# Experimental work

- Carlos Ribeiro is implementing a **generic** AoP-based **complexity analyser** in Haskell based on this approach.

- Relations modelled via the **powerset monad**.

- From powersets it evolves towards the **distribution monad**.

- Both the work presented at the meeting and this implementation is work in progress.

- **Comments + collaboration are welcome**.

# Annex

# Proof of (61)

Since $f := g \cdot \alpha$:

$$
\begin{aligned}
& \tau\ f = \tau\ (g \cdot \alpha) \\
\Leftrightarrow & \quad \{\ \text{cost of composition (53)}\ \} \\
& \tau\ f = (\tau\ g \cdot \alpha) + \tau\ \alpha \\
\Leftrightarrow & \quad \{\ \text{subtraction}\ \} \\
& \tau\ f - \tau\ \alpha = \tau\ g \cdot \alpha \\
\Leftrightarrow & \quad \{\ \alpha \text{ is an isomorphism}\ \} \\
& (\tau\ f - \tau\ \alpha) \cdot \alpha^{\circ} = \tau\ g \\
\Leftrightarrow & \quad \{\ (10)\ ;\ \text{addition}\ \} \\
& \tau\ f \cdot \alpha^{\circ} = \tau\ g + (\tau\ \alpha) \cdot \alpha^{\circ}
\end{aligned}
$$

Proof of (57):

$$
\begin{aligned}
& \tau\ (p \to f, g) \\
\Leftrightarrow \quad & \{\ \text{definition}\ \} \\
& \tau\ ([\, f, g\,] \cdot p^{?}) \\
\Leftrightarrow \quad & \{\ \text{composition} + \text{either TBC}\ \} \\
& [\, \tau\ f, \tau\ g\,] \cdot p^{?} + \tau\ p^{?} \\
\Leftrightarrow \quad & \{\ \text{definitions TBC}\ \} \\
& (p \to \tau\ f, \tau\ g) + \tau\ (\alpha \cdot \langle p\,,\ id \rangle) \\
\Leftrightarrow \quad & \{\ \text{isomorphism } \alpha,\ \tau\ \alpha = \underline{0}\ \} \\
& (p \to \tau\ f, \tau\ g) + \tau\ p + \tau\ id \\
\Leftrightarrow \quad & \{\ \text{definitions}\ \} \\
& (p \to \tau\ f, \tau\ g) + \tau\ p
\end{aligned}
$$

E. Albert, P. Arenas, S. Genaim, G. Puebla, and D. Zanardini. Costa: Design and implementation of a cost and termination analyzer for Java bytecode. In F. de Boer, M. Bonsangue, S. Graf, and W-P. de Roever, editors, *Formal Methods for Components and Objects*, pages 113–132. Springer, 2008.

R. Bird and J. Gibbons. *Algorithm Design with Haskell*. Cambridge University Press, 2020.

Armaël Gueneau. *Mechanized verification of the correctness and asymptotic complexity of programs : the right answer at the right time*. PhD thesis, Université Paris Cité, October 2020. NNT : 2019UNIP7110.

Daniel Le Métayer. Mechanical analysis of program complexity. In Teri F. Payton, L. Peter Deutsch, and James Purtilo, editors, *Proceedings of the ACM SIGPLAN 85 Symposium on Language Issues in Programming Environments, SLIPE 1985, Seattle, Washington, USA, June 25-28, 1985*, pages 69–73. ACM, 1985. doi: 10.1145/800225.806828. URL `https://doi.org/10.1145/800225.806828`.

Audrey A. Nasar. The history of algorithmic complexity. 13(3):217–242, 2016. doi: https://doi.org/10.54870/1551-3440.1375.

Mads Rosendahl. Automatic complexity analysis. In *FPCA*, FPCA'89, page 144–156, New York, NY, USA, 1989. ACM. ISBN 0897913280. doi: 10.1145/99370.99381. URL https://doi.org/10.1145/99370.99381.

David Sands. *Calculi for Time Analysis of Functional Programs*. PhD thesis, University of London, September 1990.

Ben Wegbreit. Mechanical program analysis. *Commun. ACM*, 18(9):528–539, September 1975. ISSN 0001-0782. doi: 10.1145/361002.361016. URL https://doi.org/10.1145/361002.361016.