The fixed point theory of complexity

Yiannis N. Moschovakis
UCLA and University of Athens

a commercial for
Abstract Recursion and Intrinsic Complexity
Published by CUP, ASL LNL Series # 48
Posted on my homepage

Panhellenic Logic Symposium 12, June 26 – 30, 2019, Anogeia

The fixed point theory of complexity

Yiannis N. Moschovakis
UCLA and University of Athens

a commercial for
Abstract Recursion and Intrinsic Complexity
Published by CUP, ASL LNL Series # 48
Posted on my homepage
Panhellenic Logic Symposium 12, June 26 – 30, 2019, Anogeia

Denotational semantics for programming languages

- Introduced in 1971 by Dana Scott and Christopher Strachey
- Assigns to every program E (of a well specified programming language) its denotation, the object $\mathrm{den}(E)$ computed by E, typically a function or relation of some sort
- The key mathematical tools it uses are fixed point theorems in various complete partially ordered sets (domains)
- It is important because it provides a precise, mathematical criterion of correctness for programs, which should compute what we wanted them to compute
- It has developed into a rich, intricate mathematical theory, not always easy to apply in specific cases

$\star$ $\mathrm{den}(E)$ *gives no information about the complexity of the algorithm expressed by* E

Denotational semantics for programming languages

- Introduced in 1971 by Dana Scott and Christopher Strachey
- Assigns to every program E (of a well specified programming language) its denotation, the object $\mathrm{den}(E)$ computed by E, typically a function or relation of some sort
- The key mathematical tools it uses are fixed point theorems in various complete partially ordered sets (domains)
- It is important because it provides a precise, mathematical criterion of correctness for programs, which should compute what we wanted them to compute
- It has developed into a rich, intricate mathematical theory, not always easy to apply in specific cases

$\star$ $\mathrm{den}(E)$ *gives no information about the complexity of the algorithm expressed by* E

Denotational semantics for programming languages

- Introduced in 1971 by Dana Scott and Christopher Strachey
- Assigns to every program E (of a well specified programming language) its denotation, the object $\mathrm{den}(E)$ computed by E, typically a function or relation of some sort
- The key mathematical tools it uses are fixed point theorems in various complete partially ordered sets (domains)
- It is important because it provides a precise, mathematical criterion of correctness for programs, which should compute what we wanted them to compute
- It has developed into a rich, intricate mathematical theory, not always easy to apply in specific cases

$\star$ $\mathrm{den}(E)$ *gives no information about the complexity of the algorithm expressed by* E

Denotational semantics for programming languages

- Introduced in 1971 by Dana Scott and Christopher Strachey
- Assigns to every program E (of a well specified programming language) its denotation, the object $\mathrm{den}(E)$ computed by E, typically a function or relation of some sort
- The key mathematical tools it uses are fixed point theorems in various complete partially ordered sets (domains)
- It is important because it provides a precise, mathematical criterion of correctness for programs, which should compute what we wanted them to compute
- It has developed into a rich, intricate mathematical theory, not always easy to apply in specific cases

$\star$ $\mathrm{den}(E)$ *gives no information about the complexity of the algorithm expressed by* E

Denotational semantics for programming languages

- Introduced in 1971 by Dana Scott and Christopher Strachey
- Assigns to every program E (of a well specified programming language) its denotation, the object $\mathrm{den}(E)$ computed by E, typically a function or relation of some sort
- The key mathematical tools it uses are fixed point theorems in various complete partially ordered sets (domains)
- It is important because it provides a precise, mathematical criterion of correctness for programs, which should compute what we wanted them to compute
- It has developed into a rich, intricate mathematical theory, not always easy to apply in specific cases

$\star$ $\mathrm{den}(E)$ *gives no information about the complexity of the algorithm expressed by* E

Denotational semantics for programming languages

- Introduced in 1971 by Dana Scott and Christopher Strachey
- Assigns to every program E (of a well specified programming language) its denotation, the object $\mathrm{den}(E)$ computed by E, typically a function or relation of some sort
- The key mathematical tools it uses are fixed point theorems in various complete partially ordered sets (domains)
- It is important because it provides a precise, mathematical criterion of correctness for programs, which should compute what we wanted them to compute
- It has developed into a rich, intricate mathematical theory, not always easy to apply in specific cases

$\star$ $\mathrm{den}(E)$ *gives no information about the complexity of the algorithm expressed by* E

Denotational semantics for programming languages

- Introduced in 1971 by Dana Scott and Christopher Strachey
- Assigns to every program E (of a well specified programming language) its denotation, the object $\mathrm{den}(E)$ computed by E, typically a function or relation of some sort
- The key mathematical tools it uses are fixed point theorems in various complete partially ordered sets (domains)
- It is important because it provides a precise, mathematical criterion of correctness for programs, which should compute what we wanted them to compute
- It has developed into a rich, intricate mathematical theory, not always easy to apply in specific cases

★ $\mathrm{den}(E)$ *gives no information about the complexity of the algorithm expressed by* E

The Euclidean algorithm for coprimeness

on $\mathbb{N} = \{0, 1, \ldots\}$, with division

$\gcd(x, y) =$ the greatest common divisor of x, y $\quad (x, y \geq 1)$

$$x \perp\!\!\!\perp y \iff \gcd(x, y) = 1$$

$$\mathrm{rem}(x, y) = r \iff [x = yq + r \;\&\; r < y], \quad \mathrm{eq}_w(x) \iff x = w$$

(ε) `vars` x, y

`while` $(y \neq 0)$ $[(x, y) := (y, \mathrm{rem}(x, y))]$; `return` $\mathrm{eq}_1(x)$

- *If* $x, y \geq 1$, *then* $\mathrm{den}(\varepsilon)(x, y) \iff x \perp\!\!\!\perp y$
- **Def**. $\mathrm{calls}_\varepsilon(\mathrm{rem})(x, y) =$ the number of divisions (calls to rem) used by ε to decide $x \perp\!\!\!\perp y$
- *If* $x \geq y$ *and* $y \geq 2$, *then* $\underline{\mathrm{calls}_\varepsilon(\mathrm{rem})(x, y) \leq 2 \log y}$
- *For a fixed* $\bar{r} > 0$ *and all the Fibonacci numbers* F_k *with* $k \geq 2$, $\underline{\mathrm{calls}_\varepsilon(\mathrm{rem})(F_{k+1}, F_k) = k - 1 \geq \bar{r} \log F_{k+1}}$

$\star$ *Is the Euclidean worst-case optimal from its primitives?*

The Euclidean algorithm for coprimeness

on $\mathbb{N} = \{0, 1, \ldots\}$, with division

$\gcd(x, y) =$ the greatest common divisor of x, y $\quad (x, y \geq 1)$

$$x \perp\!\!\!\perp y \iff \gcd(x, y) = 1$$

$$\mathrm{rem}(x, y) = r \iff [x = yq + r \ \&\ r < y], \quad \mathrm{eq}_w(x) \iff x = w$$

(ε) `vars` x, y

`while` $(y \neq 0)$ $[(x, y) := (y, \mathrm{rem}(x, y))]$; `return` $\mathrm{eq}_1(x)$

- *If* $x, y \geq 1$, *then* $\mathrm{den}(\varepsilon)(x, y) \iff x \perp\!\!\!\perp y$
- **Def**. $\mathrm{calls}_\varepsilon(\mathrm{rem})(x, y) =$ the number of divisions (calls to rem) used by ε to decide $x \perp\!\!\!\perp y$
- *If* $x \geq y$ *and* $y \geq 2$, *then* $\underline{\mathrm{calls}_\varepsilon(\mathrm{rem})(x, y) \leq 2 \log y}$
- *For a fixed* $\bar{r} > 0$ *and all the Fibonacci numbers* F_k *with* $k \geq 2$, $\underline{\mathrm{calls}_\varepsilon(\mathrm{rem})(F_{k+1}, F_k) = k - 1 \geq \bar{r} \log F_{k+1}}$

$\star$ *Is the Euclidean worst-case optimal from its primitives?*

The Euclidean algorithm for coprimeness

on $\mathbb{N} = \{0, 1, \ldots\}$, with division

$\gcd(x, y) =$ the greatest common divisor of x, y $\quad (x, y \geq 1)$

$$x \perp\!\!\!\perp y \iff \gcd(x, y) = 1$$

$$\mathrm{rem}(x, y) = r \iff [x = yq + r \;\&\; r < y], \quad \mathrm{eq}_w(x) \iff x = w$$

(ε) $\quad$ `vars` x, y

$\quad$ `while` $(y \neq 0)$ $[(x, y) := (y, \mathrm{rem}(x, y))]$; `return` $\mathrm{eq}_1(x)$

- *If* $x, y \geq 1$, *then* $\mathrm{den}(\varepsilon)(x, y) \iff x \perp\!\!\!\perp y$
- **Def**. $\mathrm{calls}_\varepsilon(\mathrm{rem})(x, y) =$ the number of divisions (calls to rem) used by ε to decide $x \perp\!\!\!\perp y$
- *If* $x \geq y$ *and* $y \geq 2$, *then* $\underline{\mathrm{calls}_\varepsilon(\mathrm{rem})(x, y) \leq 2 \log y}$
- *For a fixed* $\bar{r} > 0$ *and all the Fibonacci numbers* F_k *with* $k \geq 2$, $\underline{\mathrm{calls}_\varepsilon(\mathrm{rem})(F_{k+1}, F_k) = k - 1 \geq \bar{r} \log F_{k+1}}$

$\star$ *Is the Euclidean worst-case optimal from its primitives?*

The Euclidean algorithm for coprimeness

on $\mathbb{N} = \{0, 1, \ldots\}$, with division

$\gcd(x, y) =$ the greatest common divisor of x, y $\quad (x, y \geq 1)$

$$x \perp\!\!\!\perp y \iff \gcd(x, y) = 1$$

$$\mathrm{rem}(x, y) = r \iff [x = yq + r \;\&\; r < y], \quad \mathrm{eq}_w(x) \iff x = w$$

(ε) $\quad$ `vars` x, y

$\quad$ `while` $(y \neq 0)$ $[(x, y) := (y, \mathrm{rem}(x, y))]$; `return` $\mathrm{eq}_1(x)$

- *If* $x, y \geq 1$, *then* $\mathrm{den}(\varepsilon)(x, y) \iff x \perp\!\!\!\perp y$
- **Def**. $\mathrm{calls}_\varepsilon(\mathrm{rem})(x, y) =$ the number of divisions (calls to rem) used by ε to decide $x \perp\!\!\!\perp y$
- *If* $x \geq y$ *and* $y \geq 2$, *then* $\mathrm{calls}_\varepsilon(\mathrm{rem})(x, y) \leq 2 \log y$
- *For a fixed* $\bar{r} > 0$ *and all the Fibonacci numbers* F_k *with* $k \geq 2$, $\mathrm{calls}_\varepsilon(\mathrm{rem})(F_{k+1}, F_k) = k - 1 \geq \bar{r} \log F_{k+1}$

$\star$ *Is the Euclidean worst-case optimal from its primitives?*

The Euclidean algorithm for coprimeness

on $\mathbb{N} = \{0, 1, \ldots\}$, with division

$\gcd(x, y) =$ the greatest common divisor of x, y $\quad (x, y \geq 1)$

$$x \perp\!\!\!\perp y \iff \gcd(x, y) = 1$$

$$\mathrm{rem}(x, y) = r \iff [x = yq + r \;\&\; r < y], \quad \mathrm{eq}_w(x) \iff x = w$$

(ε) $\quad$ `vars` x, y

$\quad$ `while` $(y \neq 0)$ $[(x, y) := (y, \mathrm{rem}(x, y))]$; `return` $\mathrm{eq}_1(x)$

- *If* $x, y \geq 1$, *then* $\mathrm{den}(\varepsilon)(x, y) \iff x \perp\!\!\!\perp y$
- **Def**. $\mathrm{calls}_\varepsilon(\mathrm{rem})(x, y) =$ the number of divisions (calls to rem) used by ε to decide $x \perp\!\!\!\perp y$
- *If* $x \geq y$ *and* $y \geq 2$, *then* $\underline{\mathrm{calls}_\varepsilon(\mathrm{rem})(x, y) \leq 2 \log y}$
- *For a fixed* $\bar{r} > 0$ *and all the Fibonacci numbers* F_k *with* $k \geq 2$, $\underline{\mathrm{calls}_\varepsilon(\mathrm{rem})(F_{k+1}, F_k) = k - 1 \geq \bar{r} \log F_{k+1}}$

$\star$ *Is the Euclidean worst-case optimal from its primitives?*

The Euclidean algorithm for coprimeness

on $\mathbb{N} = \{0, 1, \ldots\}$, with division

$\gcd(x, y) =$ the greatest common divisor of x, y $\quad (x, y \geq 1)$

$$x \perp\!\!\!\perp y \iff \gcd(x, y) = 1$$

$$\mathrm{rem}(x, y) = r \iff [x = yq + r \;\&\; r < y], \quad \mathrm{eq}_w(x) \iff x = w$$

(ε) $\quad$ `vars` x, y

$\quad$ `while` $(y \neq 0)$ $[(x, y) := (y, \mathrm{rem}(x, y))]$; `return` $\mathrm{eq}_1(x)$

- *If* $x, y \geq 1$, *then* $\mathrm{den}(\varepsilon)(x, y) \iff x \perp\!\!\!\perp y$
- **Def**. $\mathrm{calls}_\varepsilon(\mathrm{rem})(x, y) =$ the number of divisions (calls to rem) used by ε to decide $x \perp\!\!\!\perp y$
- *If* $x \geq y$ *and* $y \geq 2$, *then* $\underline{\mathrm{calls}_\varepsilon(\mathrm{rem})(x, y) \leq 2 \log y}$
- *For a fixed* $\bar{r} > 0$ *and all the Fibonacci numbers* F_k *with* $k \geq 2$, $\underline{\mathrm{calls}_\varepsilon(\mathrm{rem})(F_{k+1}, F_k) = k - 1 \geq \bar{r} \log F_{k+1}}$

★ *Is the Euclidean worst-case optimal from its primitives?*

The Euclidean algorithm for coprimeness

on $\mathbb{N} = \{0, 1, \ldots\}$, with division

$\gcd(x, y) =$ the greatest common divisor of x, y $\quad (x, y \geq 1)$

$$x \perp\!\!\!\perp y \iff \gcd(x, y) = 1$$

$$\mathrm{rem}(x, y) = r \iff [x = yq + r \ \&\ r < y], \quad \mathrm{eq}_w(x) \iff x = w$$

(ε) $\quad$ `vars` x, y

$\quad$ `while` $(y \neq 0)$ $[(x, y) := (y, \mathrm{rem}(x, y))]$; `return` $\mathrm{eq}_1(x)$

- *If* $x, y \geq 1$, *then* $\mathrm{den}(\varepsilon)(x, y) \iff x \perp\!\!\!\perp y$
- **Def**. $\mathrm{calls}_\varepsilon(\mathrm{rem})(x, y) =$ the number of divisions (calls to rem) used by ε to decide $x \perp\!\!\!\perp y$
- *If* $x \geq y$ *and* $y \geq 2$, *then* $\underline{\mathrm{calls}_\varepsilon(\mathrm{rem})(x, y) \leq 2 \log y}$
- *For a fixed* $\bar{r} > 0$ *and all the Fibonacci numbers* F_k *with* $k \geq 2$, $\underline{\mathrm{calls}_\varepsilon(\mathrm{rem})(F_{k+1}, F_k) = k - 1 \geq \bar{r} \log F_{k+1}}$

$\star$ *Is the Euclidean worst-case optimal from its primitives?*

The Euclidean algorithm for coprimeness

on $\mathbb{N} = \{0, 1, \ldots\}$, with division

$\gcd(x, y) =$ the greatest common divisor of x, y $\quad (x, y \geq 1)$

$$x \perp\!\!\!\perp y \iff \gcd(x, y) = 1$$

$$\mathrm{rem}(x, y) = r \iff [x = yq + r \;\&\; r < y], \quad \mathrm{eq}_w(x) \iff x = w$$

(ε) $\quad$ `vars` x, y

$\quad$ `while` $(y \neq 0)$ $[(x, y) := (y, \mathrm{rem}(x, y))]$; `return` $\mathrm{eq}_1(x)$

- *If* $x, y \geq 1$, *then* $\mathrm{den}(\varepsilon)(x, y) \iff x \perp\!\!\!\perp y$
- **Def**. $\mathrm{calls}_\varepsilon(\mathrm{rem})(x, y) =$ the number of divisions (calls to rem) used by ε to decide $x \perp\!\!\!\perp y$
- *If* $x \geq y$ *and* $y \geq 2$, *then* $\underline{\mathrm{calls}_\varepsilon(\mathrm{rem})(x, y) \leq 2 \log y}$
- *For a fixed* $\bar{r} > 0$ *and all the Fibonacci numbers* F_k *with* $k \geq 2$, $\underline{\mathrm{calls}_\varepsilon(\mathrm{rem})(F_{k+1}, F_k) = k - 1 \geq \bar{r} \log F_{k+1}}$

★ *Is the Euclidean worst-case optimal from its primitives?*

Intensional properties of programs

- To extract from a program E many of its intensional properties (including complexity functions) using fixed point theory you need to make two moves:

(1) *Identify the primitives that E can call*

- We consider only programs which compute (partial) functions and decide relations on some set A from primitives of the same kind, i.e., programs of a (partial) first order structure $\boxed{\mathbf{A} = (A, \mathbf{\Phi})}$ of vocabulary (characteristic, similarity type) Φ
- For the Euclidean: $\mathbf{A}_\varepsilon = (\mathbb{N}, \mathrm{eq}_0, \mathrm{eq}_1, \mathrm{rem})$

(2) *Translate E faithfully into a recursive (McCarthy) program of* $\mathbf{A}$ (using routine, well-understood methods)

$$(\varepsilon) \quad \boxed{\mathrm{eq}_1(p(x,y)) \text{ where } \left\{ p(x,y) = \text{if } \mathrm{eq}_0(y) \text{ then } x \text{ else } p(y, \mathrm{rem}(x,y)) \right\}}$$

Intensional properties of programs

- To extract from a program E many of its intensional properties (including complexity functions) using fixed point theory you need to make two moves:

(1) *Identify the primitives that E can call*

- We consider only programs which compute (partial) functions and decide relations on some set A from primitives of the same kind, i.e., programs of a (partial) first order structure $\boxed{\mathbf{A} = (A, \mathbf{\Phi})}$ of vocabulary (characteristic, similarity type) Φ
- For the Euclidean: $\mathbf{A}_\varepsilon = (\mathbb{N}, \mathrm{eq}_0, \mathrm{eq}_1, \mathrm{rem})$

(2) *Translate E faithfully into a recursive (McCarthy) program of* $\mathbf{A}$ (using routine, well-understood methods)

(ε) $\boxed{\mathrm{eq}_1(p(x,y)) \text{ where } \left\{ p(x,y) = \text{if } \mathrm{eq}_0(y) \text{ then } x \text{ else } p(y, \mathrm{rem}(x,y)) \right\}}$

Intensional properties of programs

- To extract from a program E many of its intensional properties (including complexity functions) using fixed point theory you need to make two moves:

(1) *Identify the primitives that E can call*

- We consider only programs which compute (partial) functions and decide relations on some set A from primitives of the same kind, i.e., programs of a (partial) first order structure $\boxed{\mathbf{A} = (A, \Phi)}$ of vocabulary (characteristic, similarity type) Φ
- For the Euclidean: $\mathbf{A}_\varepsilon = (\mathbb{N}, \mathrm{eq}_0, \mathrm{eq}_1, \mathrm{rem})$

(2) *Translate E faithfully into a recursive (McCarthy) program of* $\mathbf{A}$ (using routine, well-understood methods)

(ε) $\boxed{\mathrm{eq}_1(p(x,y)) \text{ where } \left\{ p(x,y) = \text{if } \mathrm{eq}_0(y) \text{ then } x \text{ else } p(y, \mathrm{rem}(x,y)) \right\}}$

Intensional properties of programs

- To extract from a program E many of its intensional properties (including complexity functions) using fixed point theory you need to make two moves:

(1) *Identify the primitives that E can call*

- We consider only programs which compute (partial) functions and decide relations on some set A from primitives of the same kind, i.e., programs of a (partial) first order structure $\boxed{\mathbf{A} = (A, \mathbf{\Phi})}$ of vocabulary (characteristic, similarity type) Φ
- For the Euclidean: $\mathbf{A}_\varepsilon = (\mathbb{N}, \mathrm{eq}_0, \mathrm{eq}_1, \mathrm{rem})$

(2) *Translate E faithfully into a recursive (McCarthy) program of $\mathbf{A}$* (using routine, well-understood methods)

$$(\varepsilon) \quad \boxed{\mathrm{eq}_1(p(x,y)) \text{ where } \left\{ p(x,y) = \text{if } \mathrm{eq}_0(y) \text{ then } x \text{ else } p(y, \mathrm{rem}(x,y)) \right\}}$$

Intensional properties of programs

- To extract from a program E many of its intensional properties (including complexity functions) using fixed point theory you need to make two moves:

(1) *Identify the primitives that E can call*

- We consider only programs which compute (partial) functions and decide relations on some set A from primitives of the same kind, i.e., programs of a (partial) first order structure $\boxed{\mathbf{A} = (A, \mathbf{\Phi})}$ of vocabulary (characteristic, similarity type) Φ
- For the Euclidean: $\mathbf{A}_\varepsilon = (\mathbb{N}, \mathsf{eq}_0, \mathsf{eq}_1, \mathsf{rem})$

(2) *Translate E faithfully into a recursive (McCarthy) program of $\mathbf{A}$* (using routine, well-understood methods)

$$(\varepsilon) \quad \boxed{\mathsf{eq}_1(p(x,y)) \text{ where } \left\{ p(x,y) = \text{if } \mathsf{eq}_0(y) \text{ then } x \text{ else } p(y, \mathsf{rem}(x,y)) \right\}}$$

Intensional properties of programs

- To extract from a program E many of its intensional properties (including complexity functions) using fixed point theory you need to make two moves:

(1) *Identify the primitives that E can call*

- We consider only programs which compute (partial) functions and decide relations on some set A from primitives of the same kind, i.e., programs of a (partial) first order structure $\boxed{\mathbf{A} = (A, \mathbf{\Phi})}$ of vocabulary (characteristic, similarity type) Φ
- For the Euclidean: $\mathbf{A}_\varepsilon = (\mathbb{N}, \mathrm{eq}_0, \mathrm{eq}_1, \mathrm{rem})$

(2) *Translate E faithfully into a recursive (McCarthy) program of* $\mathbf{A}$ (using routine, well-understood methods)

$$(\varepsilon) \quad \mathrm{eq}_1(p(x,y)) \text{ where } \left\{ p(x,y) = \text{if } \mathrm{eq}_0(y) \text{ then } x \text{ else } p(y, \mathrm{rem}(x,y)) \right\}$$

Intensional properties of programs

- To extract from a program E many of its intensional properties (including complexity functions) using fixed point theory you need to make two moves:

(1) *Identify the primitives that E can call*

- We consider only programs which compute (partial) functions and decide relations on some set A from primitives of the same kind, i.e., programs of a (partial) first order structure $\boxed{\mathbf{A} = (A, \mathbf{\Phi})}$ of vocabulary (characteristic, similarity type) Φ
- For the Euclidean: $\mathbf{A}_\varepsilon = (\mathbb{N}, \mathrm{eq}_0, \mathrm{eq}_1, \mathrm{rem})$

(2) *Translate E faithfully into a recursive (McCarthy) program of* $\mathbf{A}$ (using routine, well-understood methods)

(ε) $\boxed{\mathrm{eq}_1(p(x,y)) \text{ where } \left\{ p(x,y) = \text{if } \mathrm{eq}_0(y) \text{ then } x \text{ else } p(y, \mathrm{rem}(x,y)) \right\}}$

Recursive programs in the vocabulary Φ

- Φ-terms (pure, explicit, of boolean or ind sort, with rec variables):

$$E :\equiv \mathrm{tt} \mid \mathrm{ff} \mid \mathrm{v}_i \mid \phi(E_1, \ldots, E_{n_\phi}) \mid \text{if } E_0 \text{ then } E_1 \text{ else } E_2 \mid \mathrm{q}_i^{s,n}(E_1, \ldots, E_n)$$

  where $\mathrm{v}_0, \mathrm{v}_1, \ldots$ are formal individual variables over A; each $\mathrm{q}_i^{s,n}$ is a formal variable over partial functions on A, of arity n and (boolean or individual) sort s;

- Recursive Φ-programs:

$$E \equiv \left[E_0(\vec{\mathrm{x}}) \text{ where } \left\{ \mathrm{p}_1(\vec{\mathrm{x}}_1) = E_1(\vec{\mathrm{x}}_1), \ldots, \mathrm{p}_K(\vec{\mathrm{x}}_K) = E_K(\vec{\mathrm{x}}_K) \right\} \right]$$

  where each E_i is a Φ-term whose individual variables are in the list $\vec{\mathrm{x}}_i$ (with $\vec{\mathrm{x}}_0 = \vec{\mathrm{x}}$) and its recursive variables are among $p_1, \ldots, p_K, \ldots$

- Semantics: The body of E defines a system of recursive equations

$$\left[p_1(\vec{x}_1) = f_1(\vec{x}_1, \vec{p}), \ldots p_K(\vec{x}_K) = f_K(\vec{x}_K, \vec{p}_K) \right];$$

  this has least solutions $\overline{p}_1, \ldots, \overline{p}_K$; and $\mathrm{den}(E)$ is the partial function defined by plugging these solutions in the head $E_0(\vec{x})$

Recursive programs in the vocabulary Φ

- Φ-terms (pure, explicit, of boolean or ind sort, with rec variables):

$$E :\equiv \mathrm{tt} \mid \mathrm{ff} \mid \mathrm{v}_i \mid \phi(E_1, \ldots, E_{n_\phi}) \mid \text{if } E_0 \text{ then } E_1 \text{ else } E_2 \mid \mathrm{q}_i^{s,n}(E_1, \ldots, E_n)$$

where $\mathrm{v}_0, \mathrm{v}_1, \ldots$ are formal individual variables over A; each $\mathrm{q}_i^{s,n}$ is a formal variable over partial functions on A, of arity n and (boolean or individual) sort s;

- Recursive Φ-programs:

$$E \equiv \boxed{E_0(\vec{\mathrm{x}}) \text{ where } \left\{ \mathrm{p}_1(\vec{\mathrm{x}}_1) = E_1(\vec{\mathrm{x}}_1), \ldots, \mathrm{p}_K(\vec{\mathrm{x}}_K) = E_K(\vec{\mathrm{x}}_K) \right\}}$$

where each E_i is a Φ-term whose individual variables are in the list $\vec{\mathrm{x}}_i$ (with $\vec{\mathrm{x}}_0 = \vec{\mathrm{x}}$) and its recursive variables are among $p_1, \ldots, p_K, \ldots$

- Semantics: The body of E defines a system of recursive equations

$$\boxed{p_1(\vec{x}_1) = f_1(\vec{x}_1, \vec{p}), \ldots p_K(\vec{x}_K) = f_K(\vec{x}_K, \vec{p}_K)} \ ;$$

this has least solutions $\overline{p}_1, \ldots, \overline{p}_K$; and $\mathrm{den}(E)$ is the partial function defined by plugging these solutions in the head $E_0(\vec{x})$

Recursive programs in the vocabulary Φ

- Φ-terms (pure, explicit, of boolean or ind sort, with rec variables):

$$E :\equiv \mathrm{tt} \mid \mathrm{ff} \mid \mathrm{v}_i \mid \phi(E_1, \ldots, E_{n_\phi}) \mid \text{if } E_0 \text{ then } E_1 \text{ else } E_2 \mid \mathrm{q}_i^{s,n}(E_1, \ldots, E_n)$$

  where $\mathrm{v}_0, \mathrm{v}_1, \ldots$ are formal individual variables over A; each $\mathrm{q}_i^{s,n}$ is a formal variable over partial functions on A, of arity n and (boolean or individual) sort s;

- Recursive Φ-programs:

$$E \equiv \left[E_0(\vec{\mathrm{x}}) \text{ where } \left\{ \mathrm{p}_1(\vec{\mathrm{x}}_1) = E_1(\vec{\mathrm{x}}_1), \ldots, \mathrm{p}_K(\vec{\mathrm{x}}_K) = E_K(\vec{\mathrm{x}}_K) \right\} \right]$$

  where each E_i is a Φ-term whose individual variables are in the list $\vec{\mathrm{x}}_i$ (with $\vec{\mathrm{x}}_0 = \vec{\mathrm{x}}$) and its recursive variables are among $p_1, \ldots, p_K, \ldots$

- Semantics: The body of E defines a system of recursive equations

$$\left[p_1(\vec{x}_1) = f_1(\vec{x}_1, \vec{p}), \ldots p_K(\vec{x}_K) = f_K(\vec{x}_K, \vec{p}_K) \right];$$

  this has least solutions $\overline{p}_1, \ldots, \overline{p}_K$; and $\mathrm{den}(E)$ is the partial function defined by plugging these solutions in the head $E_0(\vec{x})$

Recursive programs in the vocabulary Φ

- Φ-terms (pure, explicit, of boolean or ind sort, with rec variables):

$$E :\equiv \mathrm{tt} \mid \mathrm{ff} \mid \mathrm{v}_i \mid \phi(E_1, \ldots, E_{n_\phi}) \mid \text{if } E_0 \text{ then } E_1 \text{ else } E_2 \mid \mathrm{q}_i^{s,n}(E_1, \ldots, E_n)$$

  where $\mathrm{v}_0, \mathrm{v}_1, \ldots$ are formal individual variables over A; each $\mathrm{q}_i^{s,n}$ is a formal variable over partial functions on A, of arity n and (boolean or individual) sort s;

- Recursive Φ-programs:

$$E \equiv \boxed{E_0(\vec{\mathrm{x}}) \text{ where } \left\{\mathrm{p}_1(\vec{\mathrm{x}}_1) = E_1(\vec{\mathrm{x}}_1), \ldots, \mathrm{p}_K(\vec{\mathrm{x}}_K) = E_K(\vec{\mathrm{x}}_K)\right\}}$$

  where each E_i is a Φ-term whose individual variables are in the list $\vec{\mathrm{x}}_i$ (with $\vec{\mathrm{x}}_0 = \vec{\mathrm{x}}$) and its recursive variables are among $p_1, \ldots, p_K$, ..

- Semantics: The body of E defines a system of recursive equations

$$p_1(\vec{x}_1) = f_1(\vec{x}_1, \vec{p}), \ldots p_K(\vec{x}_K) = f_K(\vec{x}_K, \vec{p}_K) \; ;$$

  this has least solutions $\overline{p}_1, \ldots, \overline{p}_K$; and $\mathrm{den}(E)$ is the partial function defined by plugging these solutions in the head $E_0(\vec{x})$

Recursive programs in the vocabulary Φ

- Φ-terms (pure, explicit, of boolean or ind sort, with rec variables):

$$E :\equiv \mathrm{tt} \mid \mathrm{ff} \mid \mathrm{v}_i \mid \phi(E_1, \ldots, E_{n_\phi}) \mid \text{if } E_0 \text{ then } E_1 \text{ else } E_2 \mid \mathrm{q}_i^{s,n}(E_1, \ldots, E_n)$$

  where $\mathrm{v}_0, \mathrm{v}_1, \ldots$ are formal individual variables over A; each $\mathrm{q}_i^{s,n}$ is a formal variable over partial functions on A, of arity n and (boolean or individual) sort s;

- Recursive Φ-programs:

$$E \equiv \boxed{E_0(\vec{\mathrm{x}}) \text{ where } \left\{\mathrm{p}_1(\vec{\mathrm{x}}_1) = E_1(\vec{\mathrm{x}}_1), \ldots, \mathrm{p}_K(\vec{\mathrm{x}}_K) = E_K(\vec{\mathrm{x}}_K)\right\}}$$

  where each E_i is a Φ-term whose individual variables are in the list $\vec{\mathrm{x}}_i$ (with $\vec{\mathrm{x}}_0 = \vec{\mathrm{x}}$) and its recursive variables are among $p_1, \ldots, p_K, \ldots$

- Semantics: The body of E defines a system of recursive equations

$$\boxed{p_1(\vec{x}_1) = f_1(\vec{x}_1, \vec{p}), \ldots p_K(\vec{x}_K) = f_K(\vec{x}_K, \vec{p}_K)} \ ;$$

  this has least solutions $\overline{p}_1, \ldots, \overline{p}_K$; and $\mathrm{den}(E)$ is the partial function defined by plugging these solutions in the head $E_0(\vec{x})$

⋆ The tree-depth complexity in $\mathbf{A} = (A, \mathbf{\Phi})$

$$E \equiv \boxed{E_0(\vec{\mathrm{x}}) \text{ where } \left\{ \mathrm{p}_1(\vec{\mathrm{x}}_1) = E_1(\vec{\mathrm{x}}_1), \ldots, \mathrm{p}_K(\vec{\mathrm{x}}_K) = E_K(\vec{\mathrm{x}}_K) \right\}}$$

- The convergent parts: $M \in \boxed{\mathrm{Conv}(\mathbf{A}, E)}$ if $M \equiv N\{\vec{y} :\equiv \vec{y}\})$ where

N is a subterm of E, $\vec{y} \in A^m$ and $\overline{M} = \mathrm{den}((\mathbf{A}, \overline{p}_1, \ldots, \overline{p}_K), M)\downarrow$

- *There is exactly one function* $D : \mathrm{Conv}(\mathbf{A}, E) \to \mathbb{N}$ *such that*

(D1) $D(\mathrm{tt}) = D(\mathrm{ff}) = D(x) = D(\phi) = 0$ (if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$)

(D2) $D(\phi(M_1, \ldots, M_m)) = \max\{D(M_1), \ldots, D(M_m)\} + 1$

(D3) If $M \equiv$ if M_0 then M_1 else M_2, then

$$D(M) = \begin{cases} \max\{D(M_0), D(M_1)\} + 1, & \text{if } \overline{M}_0 = \mathrm{tt}, \\ \max\{D(M_0), D(M_2)\} + 1, & \text{if } \overline{M}_0 = \mathrm{ff} \end{cases}$$

(D4) If p_i is a recursive variable of E of arity m, then

$$D(\mathrm{p}_i(M_1, \ldots, M_m)) = \max\{D(M_1), \ldots, D(M_m), D(E_i(\overline{M}_1, \ldots, \overline{M}_m))\} + 1$$

Proved by analyzing the construction of the least solutions $\overline{p}_1, \ldots, \overline{p}_K$

⋆ The tree-depth complexity in $\mathbf{A} = (A, \mathbf{\Phi})$

$$E \equiv \boxed{E_0(\vec{\mathrm{x}}) \text{ where } \left\{ \mathrm{p}_1(\vec{\mathrm{x}}_1) = E_1(\vec{\mathrm{x}}_1), \ldots, \mathrm{p}_K(\vec{\mathrm{x}}_K) = E_K(\vec{\mathrm{x}}_K) \right\}}$$

- The convergent parts: $M \in \boxed{\mathrm{Conv}(\mathbf{A}, E)}$ if $M \equiv N\{\vec{y} :\equiv \vec{y}\})$ where

N is a subterm of E, $\vec{y} \in A^m$ and $\overline{M} = \mathrm{den}((\mathbf{A}, \overline{p}_1, \ldots, \overline{p}_K), M)\downarrow$

- *There is exactly one function* $D : \mathrm{Conv}(\mathbf{A}, E) \to \mathbb{N}$ *such that*

(D1) $D(\mathrm{tt}) = D(\mathrm{ff}) = D(x) = D(\phi) = 0$ (if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$)

(D2) $D(\phi(M_1, \ldots, M_m)) = \max\{D(M_1), \ldots, D(M_m)\} + 1$

(D3) If $M \equiv$ if M_0 then M_1 else M_2, then

$$D(M) = \begin{cases} \max\{D(M_0), D(M_1)\} + 1, & \text{if } \overline{M}_0 = \mathrm{tt}, \\ \max\{D(M_0), D(M_2)\} + 1, & \text{if } \overline{M}_0 = \mathrm{ff} \end{cases}$$

(D4) If p_i is a recursive variable of E of arity m, then

$$D(\mathrm{p}_i(M_1, \ldots, M_m)) = \max\{D(M_1), \ldots, D(M_m), D(E_i(\overline{M}_1, \ldots, \overline{M}_m))\} + 1$$

Proved by analyzing the construction of the least solutions $\overline{p}_1, \ldots, \overline{p}_K$

⋆ The tree-depth complexity in $\mathbf{A} = (A, \mathbf{\Phi})$

$$E \equiv \boxed{E_0(\vec{\mathrm{x}}) \text{ where } \left\{ \mathrm{p}_1(\vec{\mathrm{x}}_1) = E_1(\vec{\mathrm{x}}_1), \ldots, \mathrm{p}_K(\vec{\mathrm{x}}_K) = E_K(\vec{\mathrm{x}}_K) \right\}}$$

- The convergent parts: $M \in \boxed{\mathrm{Conv}(\mathbf{A}, E)}$ if $M \equiv N\{\vec{y} :\equiv \vec{y}\})$ where

$\boxed{N \text{ is a subterm of } E,\ \vec{y} \in A^m \text{ and } \overline{M} = \mathrm{den}((\mathbf{A}, \overline{p}_1, \ldots, \overline{p}_K), M)\downarrow}$

- *There is exactly one function* $D : \mathrm{Conv}(\mathbf{A}, E) \to \mathbb{N}$ *such that*

(D1) $D(\mathrm{tt}) = D(\mathrm{ff}) = D(x) = D(\phi) = 0$ (if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$)

(D2) $D(\phi(M_1, \ldots, M_m)) = \max\{D(M_1), \ldots, D(M_m)\} + 1$

(D3) If $M \equiv$ if M_0 then M_1 else M_2, then

$$D(M) = \begin{cases} \max\{D(M_0), D(M_1)\} + 1, & \text{if } \overline{M}_0 = \mathrm{tt}, \\ \max\{D(M_0), D(M_2)\} + 1, & \text{if } \overline{M}_0 = \mathrm{ff} \end{cases}$$

(D4) If p_i is a recursive variable of E of arity m, then

$$D(\mathrm{p}_i(M_1, \ldots, M_m)) = \max\{D(M_1), \ldots, D(M_m), D(E_i(\overline{M}_1, \ldots, \overline{M}_m))\} + 1$$

Proved by analyzing the construction of the least solutions $\overline{p}_1, \ldots, \overline{p}_K$

⋆ The tree-depth complexity in $\mathbf{A} = (A, \mathbf{\Phi})$

$$E \equiv \boxed{E_0(\vec{\mathrm{x}}) \text{ where } \left\{ \mathrm{p}_1(\vec{\mathrm{x}}_1) = E_1(\vec{\mathrm{x}}_1), \ldots, \mathrm{p}_K(\vec{\mathrm{x}}_K) = E_K(\vec{\mathrm{x}}_K) \right\}}$$

- The convergent parts: $M \in \boxed{\mathrm{Conv}(\mathbf{A}, E)}$ if $M \equiv N\{\vec{y} :\equiv \vec{y}\})$ where

$\boxed{N \text{ is a subterm of } E,\ \vec{y} \in A^m \text{ and } \overline{M} = \mathrm{den}((\mathbf{A}, \overline{p}_1, \ldots, \overline{p}_K), M)\downarrow}$

- *There is exactly one function* $D : \mathrm{Conv}(\mathbf{A}, E) \to \mathbb{N}$ *such that*

(D1) $D(\mathrm{tt}) = D(\mathrm{ff}) = D(x) = D(\phi) = 0$ (if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$)

(D2) $D(\phi(M_1, \ldots, M_m)) = \max\{D(M_1), \ldots, D(M_m)\} + 1$

(D3) If $M \equiv$ if M_0 then M_1 else M_2, then

$$D(M) = \begin{cases} \max\{D(M_0), D(M_1)\} + 1, & \text{if } \overline{M}_0 = \mathrm{tt}, \\ \max\{D(M_0), D(M_2)\} + 1, & \text{if } \overline{M}_0 = \mathrm{ff} \end{cases}$$

(D4) If p_i is a recursive variable of E of arity m, then

$$D(\mathrm{p}_i(M_1, \ldots, M_m)) = \max\{D(M_1), \ldots, D(M_m), D(E_i(\overline{M}_1, \ldots, \overline{M}_m))\} + 1$$

Proved by analyzing the construction of the least solutions $\overline{p}_1, \ldots, \overline{p}_K$

⋆ The tree-depth complexity in $\mathbf{A} = (A, \mathbf{\Phi})$

$$E \equiv \boxed{E_0(\vec{\mathrm{x}}) \text{ where } \left\{ \mathrm{p}_1(\vec{\mathrm{x}}_1) = E_1(\vec{\mathrm{x}}_1), \ldots, \mathrm{p}_K(\vec{\mathrm{x}}_K) = E_K(\vec{\mathrm{x}}_K) \right\}}$$

- The convergent parts: $M \in \boxed{\mathrm{Conv}(\mathbf{A}, E)}$ if $M \equiv N\{\vec{y} :\equiv \vec{y}\})$ where

$\boxed{N \text{ is a subterm of } E,\ \vec{y} \in A^m \text{ and } \overline{M} = \mathrm{den}((\mathbf{A}, \overline{p}_1, \ldots, \overline{p}_K), M)\downarrow}$

- *There is exactly one function* $D : \mathrm{Conv}(\mathbf{A}, E) \to \mathbb{N}$ *such that*

(D1) $D(\mathrm{tt}) = D(\mathrm{ff}) = D(x) = D(\phi) = 0$ (if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$)

(D2) $D(\phi(M_1, \ldots, M_m)) = \max\{D(M_1), \ldots, D(M_m)\} + 1$

(D3) If $M \equiv \text{if } M_0 \text{ then } M_1 \text{ else } M_2$, then

$$D(M) = \begin{cases} \max\{D(M_0), D(M_1)\} + 1, & \text{if } \overline{M}_0 = \mathrm{tt}, \\ \max\{D(M_0), D(M_2)\} + 1, & \text{if } \overline{M}_0 = \mathrm{ff} \end{cases}$$

(D4) If p_i is a recursive variable of E of arity m, then

$$D(\mathrm{p}_i(M_1, \ldots, M_m)) = \max\{D(M_1), \ldots, D(M_m), D(E_i(\overline{M}_1, \ldots, \overline{M}_m))\} + 1$$

Proved by analyzing the construction of the least solutions $\overline{p}_1, \ldots, \overline{p}_K$

⋆ The tree-depth complexity in $\mathbf{A} = (A, \mathbf{\Phi})$

$$E \equiv \boxed{E_0(\vec{\mathrm{x}}) \text{ where } \left\{ \mathrm{p}_1(\vec{\mathrm{x}}_1) = E_1(\vec{\mathrm{x}}_1), \ldots, \mathrm{p}_K(\vec{\mathrm{x}}_K) = E_K(\vec{\mathrm{x}}_K) \right\}}$$

- The convergent parts: $M \in \boxed{\mathrm{Conv}(\mathbf{A}, E)}$ if $M \equiv N\{\vec{y} :\equiv \vec{y}\})$ where

$\boxed{N \text{ is a subterm of } E,\ \vec{y} \in A^m \text{ and } \overline{M} = \mathrm{den}((\mathbf{A}, \overline{p}_1, \ldots, \overline{p}_K), M)\downarrow}$

- *There is exactly one function* $D : \mathrm{Conv}(\mathbf{A}, E) \to \mathbb{N}$ *such that*

(D1) $D(\mathrm{tt}) = D(\mathrm{ff}) = D(x) = D(\phi) = 0$ (if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$)

(D2) $D(\phi(M_1, \ldots, M_m)) = \max\{D(M_1), \ldots, D(M_m)\} + 1$

(D3) If $M \equiv$ if M_0 then M_1 else M_2, then

$$D(M) = \begin{cases} \max\{D(M_0), D(M_1)\} + 1, & \text{if } \overline{M}_0 = \mathrm{tt}, \\ \max\{D(M_0), D(M_2)\} + 1, & \text{if } \overline{M}_0 = \mathrm{ff} \end{cases}$$

(D4) If p_i is a recursive variable of E of arity m, then

$$D(\mathrm{p}_i(M_1, \ldots, M_m)) = \max\{D(M_1), \ldots, D(M_m), D(E_i(\overline{M}_1, \ldots, \overline{M}_m))\} + 1$$

Proved by analyzing the construction of the least solutions $\overline{p}_1, \ldots, \overline{p}_K$

⋆ The tree-depth complexity in $\mathbf{A} = (A, \mathbf{\Phi})$

$$E \equiv \boxed{E_0(\vec{x}) \text{ where } \left\{ \mathrm{p}_1(\vec{x}_1) = E_1(\vec{x}_1), \ldots, \mathrm{p}_K(\vec{x}_K) = E_K(\vec{x}_K) \right\}}$$

- The convergent parts: $M \in \boxed{\mathrm{Conv}(\mathbf{A}, E)}$ if $M \equiv N\{\vec{y} :\equiv \vec{y}\})$ where

$\boxed{N \text{ is a subterm of } E,\ \vec{y} \in A^m \text{ and } \overline{M} = \mathrm{den}((\mathbf{A}, \overline{p}_1, \ldots, \overline{p}_K), M)\downarrow}$

- *There is exactly one function* $D : \mathrm{Conv}(\mathbf{A}, E) \to \mathbb{N}$ *such that*

(D1) $D(\mathrm{tt}) = D(\mathrm{ff}) = D(x) = D(\phi) = 0$ (if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$)

(D2) $D(\phi(M_1, \ldots, M_m)) = \max\{D(M_1), \ldots, D(M_m)\} + 1$

(D3) If $M \equiv$ if M_0 then M_1 else M_2, then

$$D(M) = \begin{cases} \max\{D(M_0), D(M_1)\} + 1, & \text{if } \overline{M}_0 = \mathrm{tt}, \\ \max\{D(M_0), D(M_2)\} + 1, & \text{if } \overline{M}_0 = \mathrm{ff} \end{cases}$$

(D4) If p_i is a recursive variable of E of arity m, then

$$D(\mathrm{p}_i(M_1, \ldots, M_m)) = \max\{D(M_1), \ldots, D(M_m), D(E_i(\overline{M}_1, \ldots, \overline{M}_m))\}+1$$

Proved by analyzing the construction of the least solutions $\overline{p}_1, \ldots, \overline{p}_K$

⋆ The tree-depth complexity in $\mathbf{A} = (A, \mathbf{\Phi})$

$$E \equiv \boxed{E_0(\vec{\mathrm{x}}) \text{ where } \left\{ \mathrm{p}_1(\vec{\mathrm{x}}_1) = E_1(\vec{\mathrm{x}}_1), \ldots, \mathrm{p}_K(\vec{\mathrm{x}}_K) = E_K(\vec{\mathrm{x}}_K) \right\}}$$

- The convergent parts: $M \in \boxed{\mathrm{Conv}(\mathbf{A}, E)}$ if $M \equiv N\{\vec{y} :\equiv \vec{y}\})$ where

$\boxed{N \text{ is a subterm of } E,\ \vec{y} \in A^m \text{ and } \overline{M} = \mathrm{den}((\mathbf{A}, \overline{p}_1, \ldots, \overline{p}_K), M)\downarrow}$

- *There is exactly one function* $D : \mathrm{Conv}(\mathbf{A}, E) \to \mathbb{N}$ *such that*

(D1) $D(\mathrm{tt}) = D(\mathrm{ff}) = D(x) = D(\phi) = 0$ (if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$)

(D2) $D(\phi(M_1, \ldots, M_m)) = \max\{D(M_1), \ldots, D(M_m)\} + 1$

(D3) If $M \equiv$ if M_0 then M_1 else M_2, then

$$D(M) = \begin{cases} \max\{D(M_0), D(M_1)\} + 1, & \text{if } \overline{M}_0 = \mathrm{tt}, \\ \max\{D(M_0), D(M_2)\} + 1, & \text{if } \overline{M}_0 = \mathrm{ff} \end{cases}$$

(D4) If p_i is a recursive variable of E of arity m, then

$$D(\mathrm{p}_i(M_1, \ldots, M_m)) = \max\{D(M_1), \ldots, D(M_m), D(E_i(\overline{M}_1, \ldots, \overline{M}_m))\}+1$$

Proved by analyzing the construction of the least solutions $\overline{p}_1, \ldots, \overline{p}_K$

The sequential logical complexity $L^s(M)$ (time)

By recursion on $D(M)$:

(L^s1) $L^s(\mathrm{tt}) = L^s(\mathrm{ff}) = L^s(x) = 0$, $L^s(\phi) = 1$ if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$

(L^s2) $L^s(\phi(M_1, \ldots, M_n)) = L^s(M_1) + L^s(M_2) + \cdots + L^s(M_n) + 1$

(L^s3) If $M \equiv \text{if } M_0 \text{ then } M_1 \text{ else } M_2$, then

$$L^s(M) = \begin{cases} L^s(M_0) + L^s(M_1) + 1 & \text{if } \overline{M}_0 = \mathrm{tt}, \\ L^s(M_0) + L^s(M_2) + 1 & \text{if } \overline{M}_0 = \mathrm{ff} \end{cases}$$

(L^s4) $L^s(\mathrm{p}_i(M_1, \ldots, M_n))$

$$= L^s(M_1) + \cdots + L^s(M_n) + L^s(E_i(\overline{M}_1, \ldots, \overline{M}_n)) + 1$$

$$\boxed{\mathrm{time}_E(\vec{x}) = I^s(\mathbf{A}, E(\vec{x})) =_{\mathrm{df}} L^s(E_0(\vec{x})) \quad (\mathrm{den}^{\mathbf{A}}_E(\vec{x})\downarrow)}$$

- $\mathrm{time}_E(\vec{x})$ counts the number of steps required for the computation of $\mathrm{den}(E)(\vec{x})$ using "the algorithm expressed by" E
- *If E simulates a program E^-, then for some K, L and all $\vec{x}$,*

$$\mathrm{time}_E(\vec{x}) \leq \mathrm{TIME}(E^-, \vec{x}) \leq K\,\mathrm{time}_E(\vec{x}) + L$$

The sequential logical complexity $L^s(M)$ (time)

By recursion on $D(M)$:

(L^s1) $L^s(\mathrm{tt}) = L^s(\mathrm{ff}) = L^s(x) = 0$, $L^s(\phi) = 1$ if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$

(L^s2) $L^s(\phi(M_1, \ldots, M_n)) = L^s(M_1) + L^s(M_2) + \cdots + L^s(M_n) + 1$

(L^s3) If $M \equiv \mathrm{if}\ M_0\ \mathrm{then}\ M_1\ \mathrm{else}\ M_2$, then

$$L^s(M) = \begin{cases} L^s(M_0) + L^s(M_1) + 1 & \text{if } \overline{M}_0 = \mathrm{tt}, \\ L^s(M_0) + L^s(M_2) + 1 & \text{if } \overline{M}_0 = \mathrm{ff} \end{cases}$$

(L^s4) $L^s(\mathrm{p}_i(M_1, \ldots, M_n))$

$$= L^s(M_1) + \cdots + L^s(M_n) + L^s(E_i(\overline{M}_1, \ldots, \overline{M}_n)) + 1$$

$$\boxed{\mathrm{time}_E(\vec{x}) = I^s(\mathbf{A}, E(\vec{x})) =_{\mathrm{df}} L^s(E_0(\vec{x})) \quad (\mathrm{den}^{\mathbf{A}}_E(\vec{x})\downarrow)}$$

- $\mathrm{time}_E(\vec{x})$ counts the number of steps required for the computation of $\mathrm{den}(E)(\vec{x})$ using "the algorithm expressed by" E
- *If E simulates a program E^-, then for some K, L and all $\vec{x}$,*

$$\mathrm{time}_E(\vec{x}) \leq \mathrm{TIME}(E^-, \vec{x}) \leq K\,\mathrm{time}_E(\vec{x}) + L$$

The sequential logical complexity $L^s(M)$ (time)

By recursion on $D(M)$:

(L^s1) $L^s(\mathtt{tt}) = L^s(\mathtt{ff}) = L^s(x) = 0$, $L^s(\phi) = 1$ if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$

(L^s2) $L^s(\phi(M_1, \ldots, M_n)) = L^s(M_1) + L^s(M_2) + \cdots + L^s(M_n) + 1$

(L^s3) If $M \equiv \mathrm{if}\ M_0\ \mathrm{then}\ M_1\ \mathrm{else}\ M_2$, then

$$L^s(M) = \begin{cases} L^s(M_0) + L^s(M_1) + 1 & \text{if } \overline{M}_0 = \mathtt{tt}, \\ L^s(M_0) + L^s(M_2) + 1 & \text{if } \overline{M}_0 = \mathtt{ff} \end{cases}$$

(L^s4) $L^s(\mathrm{p}_i(M_1, \ldots, M_n))$
$= L^s(M_1) + \cdots + L^s(M_n) + L^s(E_i(\overline{M}_1, \ldots, \overline{M}_n)) + 1$

$$\boxed{\mathrm{time}_E(\vec{x}) = I^s(\mathbf{A}, E(\vec{x})) =_{\mathrm{df}} L^s(E_0(\vec{x})) \quad (\mathrm{den}^{\mathbf{A}}_E(\vec{x})\downarrow)}$$

- $\mathrm{time}_E(\vec{x})$ counts the number of steps required for the computation of $\mathrm{den}(E)(\vec{x})$ using "the algorithm expressed by" E
- *If E simulates a program E^-, then for some K, L and all $\vec{x}$,*

$$\mathrm{time}_E(\vec{x}) \leq \mathrm{TIME}(E^-, \vec{x}) \leq K\,\mathrm{time}_E(\vec{x}) + L$$

The sequential logical complexity $L^s(M)$ (time)

By recursion on $D(M)$:

(L^s1) $L^s(\mathtt{tt}) = L^s(\mathtt{ff}) = L^s(x) = 0$, $L^s(\phi) = 1$ if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$

(L^s2) $L^s(\phi(M_1, \ldots, M_n)) = L^s(M_1) + L^s(M_2) + \cdots + L^s(M_n) + 1$

(L^s3) If $M \equiv$ if M_0 then M_1 else M_2, then

$$L^s(M) = \begin{cases} L^s(M_0) + L^s(M_1) + 1 & \text{if } \overline{M}_0 = \mathtt{tt}, \\ L^s(M_0) + L^s(M_2) + 1 & \text{if } \overline{M}_0 = \mathtt{ff} \end{cases}$$

(L^s4) $L^s(\mathrm{p}_i(M_1, \ldots, M_n))$

$$= L^s(M_1) + \cdots + L^s(M_n) + L^s(E_i(\overline{M}_1, \ldots, \overline{M}_n)) + 1$$

$$\boxed{\mathrm{time}_E(\vec{x}) = I^s(\mathbf{A}, E(\vec{x})) =_{\mathrm{df}} L^s(E_0(\vec{x})) \quad (\mathrm{den}^{\mathbf{A}}_E(\vec{x})\downarrow)}$$

- $\mathrm{time}_E(\vec{x})$ counts the number of steps required for the computation of $\mathrm{den}(E)(\vec{x})$ using "the algorithm expressed by" E
- *If E simulates a program E^-, then for some K, L and all $\vec{x}$,*

$$\mathrm{time}_E(\vec{x}) \leq \mathrm{TIME}(E^-, \vec{x}) \leq K\,\mathrm{time}_E(\vec{x}) + L$$

The sequential logical complexity $L^s(M)$ (time)

By recursion on $D(M)$:

(L^s1) $L^s(\mathrm{tt}) = L^s(\mathrm{ff}) = L^s(x) = 0$, $L^s(\phi) = 1$ if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$

(L^s2) $L^s(\phi(M_1, \ldots, M_n)) = L^s(M_1) + L^s(M_2) + \cdots + L^s(M_n) + 1$

(L^s3) If $M \equiv \mathrm{if}\ M_0\ \mathrm{then}\ M_1\ \mathrm{else}\ M_2$, then

$$L^s(M) = \begin{cases} L^s(M_0) + L^s(M_1) + 1 & \text{if } \overline{M}_0 = \mathrm{tt}, \\ L^s(M_0) + L^s(M_2) + 1 & \text{if } \overline{M}_0 = \mathrm{ff} \end{cases}$$

(L^s4) $L^s(\mathrm{p}_i(M_1, \ldots, M_n))$

$$= L^s(M_1) + \cdots + L^s(M_n) + L^s(E_i(\overline{M}_1, \ldots, \overline{M}_n)) + 1$$

$$\boxed{\mathrm{time}_E(\vec{x}) = I^s(\mathbf{A}, E(\vec{x})) =_{\mathrm{df}} L^s(E_0(\vec{x})) \quad (\mathrm{den}^{\mathbf{A}}_E(\vec{x})\downarrow)}$$

- $\mathrm{time}_E(\vec{x})$ counts the number of steps required for the computation of $\mathrm{den}(E)(\vec{x})$ using "the algorithm expressed by" E
- *If E simulates a program E^-, then for some K, L and all $\vec{x}$,*

$$\mathrm{time}_E(\vec{x}) \leq \mathrm{TIME}(E^-, \vec{x}) \leq K\,\mathrm{time}_E(\vec{x}) + L$$

The sequential logical complexity $L^s(M)$ (time)

By recursion on $D(M)$:

(L^s1) $L^s(\mathtt{tt}) = L^s(\mathtt{ff}) = L^s(x) = 0$, $L^s(\phi) = 1$ if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$

(L^s2) $L^s(\phi(M_1, \ldots, M_n)) = L^s(M_1) + L^s(M_2) + \cdots + L^s(M_n) + 1$

(L^s3) If $M \equiv$ if M_0 then M_1 else M_2, then

$$L^s(M) = \begin{cases} L^s(M_0) + L^s(M_1) + 1 & \text{if } \overline{M}_0 = \mathtt{tt}, \\ L^s(M_0) + L^s(M_2) + 1 & \text{if } \overline{M}_0 = \mathtt{ff} \end{cases}$$

(L^s4) $L^s(\mathrm{p}_i(M_1, \ldots, M_n))$

$$= L^s(M_1) + \cdots + L^s(M_n) + L^s(E_i(\overline{M}_1, \ldots, \overline{M}_n)) + 1$$

$$\boxed{\mathrm{time}_E(\vec{x}) = I^s(\mathbf{A}, E(\vec{x})) =_{\mathrm{df}} L^s(E_0(\vec{x})) \quad (\mathrm{den}^{\mathbf{A}}_E(\vec{x})\downarrow)}$$

- $\mathrm{time}_E(\vec{x})$ counts the number of steps required for the computation of $\mathrm{den}(E)(\vec{x})$ using "the algorithm expressed by" E
- *If E simulates a program E^-, then for some K, L and all $\vec{x}$,*

$$\mathrm{time}_E(\vec{x}) \leq \mathrm{TIME}(E^-, \vec{x}) \leq K\,\mathrm{time}_E(\vec{x}) + L$$

The sequential logical complexity $L^s(M)$ (time)

By recursion on $D(M)$:

(L^s1) $L^s(\mathtt{tt}) = L^s(\mathtt{ff}) = L^s(x) = 0$, $L^s(\phi) = 1$ if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$

(L^s2) $L^s(\phi(M_1, \ldots, M_n)) = L^s(M_1) + L^s(M_2) + \cdots + L^s(M_n) + 1$

(L^s3) If $M \equiv$ if M_0 then M_1 else M_2, then

$$L^s(M) = \begin{cases} L^s(M_0) + L^s(M_1) + 1 & \text{if } \overline{M}_0 = \mathtt{tt}, \\ L^s(M_0) + L^s(M_2) + 1 & \text{if } \overline{M}_0 = \mathtt{ff} \end{cases}$$

(L^s4) $L^s(\mathrm{p}_i(M_1, \ldots, M_n))$
$$= L^s(M_1) + \cdots + L^s(M_n) + L^s(E_i(\overline{M}_1, \ldots, \overline{M}_n)) + 1$$

$$\mathrm{time}_E(\vec{x}) = I^s(\mathbf{A}, E(\vec{x})) =_{\mathrm{df}} L^s(E_0(\vec{x})) \quad (\mathrm{den}^{\mathbf{A}}_E(\vec{x})\downarrow)$$

- $\mathrm{time}_E(\vec{x})$ counts the number of steps required for the computation of $\mathrm{den}(E)(\vec{x})$ using "the algorithm expressed by" E
- *If E simulates a program E^-, then for some K, L and all $\vec{x}$,*

$$\mathrm{time}_E(\vec{x}) \leq \mathrm{TIME}(E^-, \vec{x}) \leq K\,\mathrm{time}_E(\vec{x}) + L$$

The sequential logical complexity $L^s(M)$ (time)

By recursion on $D(M)$:

(L^s1) $L^s(\mathtt{tt}) = L^s(\mathtt{ff}) = L^s(x) = 0$, $L^s(\phi) = 1$ if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$

(L^s2) $L^s(\phi(M_1, \ldots, M_n)) = L^s(M_1) + L^s(M_2) + \cdots + L^s(M_n) + 1$

(L^s3) If $M \equiv$ if M_0 then M_1 else M_2, then

$$L^s(M) = \begin{cases} L^s(M_0) + L^s(M_1) + 1 & \text{if } \overline{M}_0 = \mathtt{tt}, \\ L^s(M_0) + L^s(M_2) + 1 & \text{if } \overline{M}_0 = \mathtt{ff} \end{cases}$$

(L^s4) $L^s(\mathrm{p}_i(M_1, \ldots, M_n))$
$$= L^s(M_1) + \cdots + L^s(M_n) + L^s(E_i(\overline{M}_1, \ldots, \overline{M}_n)) + 1$$

$$\boxed{\mathrm{time}_E(\vec{x}) = I^s(\mathbf{A}, E(\vec{x})) =_{\mathrm{df}} L^s(E_0(\vec{x})) \quad (\mathrm{den}^{\mathbf{A}}_E(\vec{x})\downarrow)}$$

- $\mathrm{time}_E(\vec{x})$ counts the number of steps required for the computation of $\mathrm{den}(E)(\vec{x})$ using "the algorithm expressed by" E
- *If E simulates a program E^-, then for some K, L and all $\vec{x}$,*

$$\mathrm{time}_E(\vec{x}) \leq \mathrm{TIME}(E^-, \vec{x}) \leq K\,\mathrm{time}_E(\vec{x}) + L$$

The sequential logical complexity $L^s(M)$ (time)

By recursion on $D(M)$:

(L^s1) $L^s(\mathtt{tt}) = L^s(\mathtt{ff}) = L^s(x) = 0$, $L^s(\phi) = 1$ if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$

(L^s2) $L^s(\phi(M_1, \ldots, M_n)) = L^s(M_1) + L^s(M_2) + \cdots + L^s(M_n) + 1$

(L^s3) If $M \equiv$ if M_0 then M_1 else M_2, then

$$L^s(M) = \begin{cases} L^s(M_0) + L^s(M_1) + 1 & \text{if } \overline{M}_0 = \mathtt{tt}, \\ L^s(M_0) + L^s(M_2) + 1 & \text{if } \overline{M}_0 = \mathtt{ff} \end{cases}$$

(L^s4) $L^s(\mathrm{p}_i(M_1, \ldots, M_n))$
$$= L^s(M_1) + \cdots + L^s(M_n) + L^s(E_i(\overline{M}_1, \ldots, \overline{M}_n)) + 1$$

$$\boxed{\mathrm{time}_E(\vec{x}) = I^s(\mathbf{A}, E(\vec{x})) =_{\mathrm{df}} L^s(E_0(\vec{x})) \quad (\mathrm{den}^{\mathbf{A}}_E(\vec{x})\downarrow)}$$

- $\mathrm{time}_E(\vec{x})$ counts the number of steps required for the computation of $\mathrm{den}(E)(\vec{x})$ using "the algorithm expressed by" E
- *If E simulates a program E^-, then for some K, L and all $\vec{x}$,*

$$\mathrm{time}_E(\vec{x}) \leq \mathrm{TIME}(E^-, \vec{x}) \leq K\,\mathrm{time}_E(\vec{x}) + L$$

The sequential logical complexity $L^s(M)$ (time)

By recursion on $D(M)$:

(L^s1) $L^s(\mathtt{tt}) = L^s(\mathtt{ff}) = L^s(x) = 0$, $L^s(\phi) = 1$ if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$

(L^s2) $L^s(\phi(M_1, \ldots, M_n)) = L^s(M_1) + L^s(M_2) + \cdots + L^s(M_n) + 1$

(L^s3) If $M \equiv$ if M_0 then M_1 else M_2, then

$$L^s(M) = \begin{cases} L^s(M_0) + L^s(M_1) + 1 & \text{if } \overline{M}_0 = \mathtt{tt}, \\ L^s(M_0) + L^s(M_2) + 1 & \text{if } \overline{M}_0 = \mathtt{ff} \end{cases}$$

(L^s4) $L^s(\mathrm{p}_i(M_1, \ldots, M_n))$
$$= L^s(M_1) + \cdots + L^s(M_n) + L^s(E_i(\overline{M}_1, \ldots, \overline{M}_n)) + 1$$

$$\boxed{\mathrm{time}_E(\vec{x}) = I^s(\mathbf{A}, E(\vec{x})) =_{\mathrm{df}} L^s(E_0(\vec{x})) \quad (\mathrm{den}^{\mathbf{A}}_E(\vec{x})\downarrow)}$$

- $\mathrm{time}_E(\vec{x})$ counts the number of steps required for the computation of $\mathrm{den}(E)(\vec{x})$ using "the algorithm expressed by" E
- *If E simulates a program E^-, then for some K, L and all $\vec{x}$,*

$$\mathrm{time}_E(\vec{x}) \leq \mathsf{TIME}(E^-, \vec{x}) \leq K\mathrm{time}_E(\vec{x}) + L$$

The number-of-calls complexity $C^s(\Phi_0)(M)$ (calls)

By recursion on $D(M)$, for any $\Phi_0 \subseteq \Phi$:

(C^s1) $C^s(\Phi_0)(\mathrm{tt}) = C^s(\Phi_0)(\mathrm{ff}) = C^s(\Phi_0)(x) = 0 \quad (x \in A)$;
and if $\mathrm{arity}(\phi) = 0$ and $\phi\downarrow$,
$C^s(\Phi_0)(\phi) = 0$ if $\phi \notin \Phi_0$, and $C^s(\Phi_0)(\phi) = 1$ if $\phi \in \Phi_0$.

(C^s2) If $M \equiv \phi(M_1, \ldots, M_n)$, then

$$C^s(\Phi_0)(M) = \begin{cases} C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n) + 1, & \text{if } \phi \in \Phi_0, \\ C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n), & \text{otherwise.} \end{cases}$$

(C^s3) If $M \equiv \text{if } M_0 \text{ then } M_1 \text{ else } M_2$, then

$$C^s(\Phi_0)(M) = \begin{cases} C^s(\Phi_0)(M_0) + C^s(\Phi_0)(M_1), & \text{if } \overline{M}_0 = \mathrm{tt}, \\ C^s(\Phi_0)(M_0) + C^s(\Phi_0)(M_2), & \text{if } \overline{M}_0 = \mathrm{ff}. \end{cases}$$

(C^s4) If $M \equiv \mathrm{p}_i(M_1, \ldots, M_n)$ with p_i a recursive variable of E, then
$C^s(\Phi_0)(M) = C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n) + C^s(\Phi_0)(E_i(\overline{M}_1, \ldots, \overline{M}_n))$

- *If E simulates a program E^-, then $C^s(\Phi_0)(E_0(\vec{x}))$ "agrees" with the number-of-Φ_0-calls complexity of E^-*

The number-of-calls complexity $C^s(\Phi_0)(M)$ (calls)

By recursion on $D(M)$, for any $\Phi_0 \subseteq \Phi$:

(C^s1) $C^s(\Phi_0)(\mathtt{tt}) = C^s(\Phi_0)(\mathtt{ff}) = C^s(\Phi_0)(x) = 0 \quad (x \in A)$;
and if $\mathrm{arity}(\phi) = 0$ and $\phi\downarrow$,
$C^s(\Phi_0)(\phi) = 0$ if $\phi \notin \Phi_0$, and $C^s(\Phi_0)(\phi) = 1$ if $\phi \in \Phi_0$.

(C^s2) If $M \equiv \phi(M_1, \ldots, M_n)$, then

$$C^s(\Phi_0)(M) = \begin{cases} C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n) + 1, & \text{if } \phi \in \Phi_0, \\ C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n), & \text{otherwise.} \end{cases}$$

(C^s3) If $M \equiv \text{if } M_0 \text{ then } M_1 \text{ else } M_2$, then

$$C^s(\Phi_0)(M) = \begin{cases} C^s(\Phi_0)(M_0) + C^s(\Phi_0)(M_1), & \text{if } \overline{M}_0 = \mathtt{tt}, \\ C^s(\Phi_0)(M_0) + C^s(\Phi_0)(M_2), & \text{if } \overline{M}_0 = \mathtt{ff}. \end{cases}$$

(C^s4) If $M \equiv \mathrm{p}_i(M_1, \ldots, M_n)$ with p_i a recursive variable of E, then
$C^s(\Phi_0)(M) = C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n) + C^s(\Phi_0)(E_i(\overline{M}_1, \ldots, \overline{M}_n))$

- *If E simulates a program E^-, then $C^s(\Phi_0)(E_0(\vec{x}))$ "agrees" with the number-of-Φ_0-calls complexity of E^-*

The number-of-calls complexity $C^s(\Phi_0)(M)$ (calls)

By recursion on $D(M)$, for any $\Phi_0 \subseteq \Phi$:

(C^s1) $C^s(\Phi_0)(\mathtt{tt}) = C^s(\Phi_0)(\mathtt{ff}) = C^s(\Phi_0)(x) = 0 \quad (x \in A)$;

and if $\mathrm{arity}(\phi) = 0$ and $\phi\downarrow$,
$C^s(\Phi_0)(\phi) = 0$ if $\phi \notin \Phi_0$, and $C^s(\Phi_0)(\phi) = 1$ if $\phi \in \Phi_0$.

(C^s2) If $M \equiv \phi(M_1, \ldots, M_n)$, then

$$C^s(\Phi_0)(M) = \begin{cases} C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n) + 1, & \text{if } \phi \in \Phi_0, \\ C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n), & \text{otherwise.} \end{cases}$$

(C^s3) If $M \equiv \text{if } M_0 \text{ then } M_1 \text{ else } M_2$, then

$$C^s(\Phi_0)(M) = \begin{cases} C^s(\Phi_0)(M_0) + C^s(\Phi_0)(M_1), & \text{if } \overline{M}_0 = \mathtt{tt}, \\ C^s(\Phi_0)(M_0) + C^s(\Phi_0)(M_2), & \text{if } \overline{M}_0 = \mathtt{ff}. \end{cases}$$

(C^s4) If $M \equiv \mathrm{p}_i(M_1, \ldots, M_n)$ with p_i a recursive variable of E, then
$C^s(\Phi_0)(M) = C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n) + C^s(\Phi_0)(E_i(\overline{M}_1, \ldots, \overline{M}_n))$

- *If E simulates a program E^-, then $C^s(\Phi_0)(E_0(\vec{x}))$ "agrees" with the number-of-Φ_0-calls complexity of E^-*

The number-of-calls complexity $C^s(\Phi_0)(M)$ (calls)

By recursion on $D(M)$, for any $\Phi_0 \subseteq \Phi$:

(C^s1) $C^s(\Phi_0)(\mathtt{tt}) = C^s(\Phi_0)(\mathtt{ff}) = C^s(\Phi_0)(x) = 0 \quad (x \in A)$;
and if $\mathrm{arity}(\phi) = 0$ and $\phi\downarrow$,
$C^s(\Phi_0)(\phi) = 0$ if $\phi \notin \Phi_0$, and $C^s(\Phi_0)(\phi) = 1$ if $\phi \in \Phi_0$.

(C^s2) If $M \equiv \phi(M_1, \ldots, M_n)$, then

$$C^s(\Phi_0)(M) = \begin{cases} C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n) + 1, & \text{if } \phi \in \Phi_0, \\ C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n), & \text{otherwise.} \end{cases}$$

(C^s3) If $M \equiv \text{if } M_0 \text{ then } M_1 \text{ else } M_2$, then

$$C^s(\Phi_0)(M) = \begin{cases} C^s(\Phi_0)(M_0) + C^s(\Phi_0)(M_1), & \text{if } \overline{M}_0 = \mathtt{tt}, \\ C^s(\Phi_0)(M_0) + C^s(\Phi_0)(M_2), & \text{if } \overline{M}_0 = \mathtt{ff}. \end{cases}$$

(C^s4) If $M \equiv \mathrm{p}_i(M_1, \ldots, M_n)$ with p_i a recursive variable of E, then

$$C^s(\Phi_0)(M) = C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n) + C^s(\Phi_0)(E_i(\overline{M}_1, \ldots, \overline{M}_n))$$

- *If E simulates a program E^-, then $C^s(\Phi_0)(E_0(\vec{x}))$ "agrees" with the number-of-Φ_0-calls complexity of E^-*

The number-of-calls complexity $C^s(\Phi_0)(M)$ (calls)

By recursion on $D(M)$, for any $\Phi_0 \subseteq \Phi$:

(C^s1) $C^s(\Phi_0)(\mathtt{tt}) = C^s(\Phi_0)(\mathtt{ff}) = C^s(\Phi_0)(x) = 0 \quad (x \in A)$;
and if $\mathrm{arity}(\phi) = 0$ and $\phi\downarrow$,
$C^s(\Phi_0)(\phi) = 0$ if $\phi \notin \Phi_0$, and $C^s(\Phi_0)(\phi) = 1$ if $\phi \in \Phi_0$.

(C^s2) If $M \equiv \phi(M_1, \ldots, M_n)$, then

$$C^s(\Phi_0)(M) = \begin{cases} C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n) + 1, & \text{if } \phi \in \Phi_0, \\ C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n), & \text{otherwise.} \end{cases}$$

(C^s3) If $M \equiv \text{if } M_0 \text{ then } M_1 \text{ else } M_2$, then

$$C^s(\Phi_0)(M) = \begin{cases} C^s(\Phi_0)(M_0) + C^s(\Phi_0)(M_1), & \text{if } \overline{M}_0 = \mathtt{tt}, \\ C^s(\Phi_0)(M_0) + C^s(\Phi_0)(M_2), & \text{if } \overline{M}_0 = \mathtt{ff}. \end{cases}$$

(C^s4) If $M \equiv \mathrm{p}_i(M_1, \ldots, M_n)$ with p_i a recursive variable of E, then
$C^s(\Phi_0)(M) = C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n) + C^s(\Phi_0)(E_i(\overline{M}_1, \ldots, \overline{M}_n))$

- *If E simulates a program E^-, then $C^s(\Phi_0)(E_0(\vec{x}))$ "agrees" with the number-of-Φ_0-calls complexity of E^-*

The number-of-calls complexity $C^s(\Phi_0)(M)$ (calls)

By recursion on $D(M)$, for any $\Phi_0 \subseteq \Phi$:

(C^s1) $C^s(\Phi_0)(\mathrm{tt}) = C^s(\Phi_0)(\mathrm{ff}) = C^s(\Phi_0)(x) = 0 \quad (x \in A)$;
and if $\mathrm{arity}(\phi) = 0$ and $\phi\downarrow$,
$C^s(\Phi_0)(\phi) = 0$ if $\phi \notin \Phi_0$, and $C^s(\Phi_0)(\phi) = 1$ if $\phi \in \Phi_0$.

(C^s2) If $M \equiv \phi(M_1, \ldots, M_n)$, then

$$C^s(\Phi_0)(M) = \begin{cases} C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n) + 1, & \text{if } \phi \in \Phi_0, \\ C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n), & \text{otherwise.} \end{cases}$$

(C^s3) If $M \equiv \text{if } M_0 \text{ then } M_1 \text{ else } M_2$, then

$$C^s(\Phi_0)(M) = \begin{cases} C^s(\Phi_0)(M_0) + C^s(\Phi_0)(M_1), & \text{if } \overline{M}_0 = \mathrm{tt}, \\ C^s(\Phi_0)(M_0) + C^s(\Phi_0)(M_2), & \text{if } \overline{M}_0 = \mathrm{ff}. \end{cases}$$

(C^s4) If $M \equiv \mathrm{p}_i(M_1, \ldots, M_n)$ with p_i a recursive variable of E, then
$C^s(\Phi_0)(M) = C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n) + C^s(\Phi_0)(E_i(\overline{M}_1, \ldots, \overline{M}_n))$

- *If E simulates a program E^-, then $C^s(\Phi_0)(E_0(\vec{x}))$ "agrees" with the number-of-Φ_0-calls complexity of E^-*

The number-of-calls complexity $C^s(\Phi_0)(M)$ (calls)

By recursion on $D(M)$, for any $\Phi_0 \subseteq \Phi$:

(C^s1) $C^s(\Phi_0)(\mathtt{tt}) = C^s(\Phi_0)(\mathtt{ff}) = C^s(\Phi_0)(x) = 0 \quad (x \in A)$;
and if $\mathrm{arity}(\phi) = 0$ and $\phi\downarrow$,
$C^s(\Phi_0)(\phi) = 0$ if $\phi \notin \Phi_0$, and $C^s(\Phi_0)(\phi) = 1$ if $\phi \in \Phi_0$.

(C^s2) If $M \equiv \phi(M_1, \ldots, M_n)$, then

$$C^s(\Phi_0)(M) = \begin{cases} C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n) + 1, & \text{if } \phi \in \Phi_0, \\ C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n), & \text{otherwise.} \end{cases}$$

(C^s3) If $M \equiv \text{if } M_0 \text{ then } M_1 \text{ else } M_2$, then

$$C^s(\Phi_0)(M) = \begin{cases} C^s(\Phi_0)(M_0) + C^s(\Phi_0)(M_1), & \text{if } \overline{M}_0 = \mathtt{tt}, \\ C^s(\Phi_0)(M_0) + C^s(\Phi_0)(M_2), & \text{if } \overline{M}_0 = \mathtt{ff}. \end{cases}$$

(C^s4) If $M \equiv \mathrm{p}_i(M_1, \ldots, M_n)$ with p_i a recursive variable of E, then
$C^s(\Phi_0)(M) = C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n) + C^s(\Phi_0)(E_i(\overline{M}_1, \ldots, \overline{M}_n))$

- *If E simulates a program E^-, then $C^s(\Phi_0)(E_0(\vec{x}))$ "agrees" with the number-of-Φ_0-calls complexity of E^-*

The number-of-calls complexity $C^s(\Phi_0)(M)$ (calls)

By recursion on $D(M)$, for any $\Phi_0 \subseteq \Phi$:

(C^s1) $C^s(\Phi_0)(\mathtt{tt}) = C^s(\Phi_0)(\mathtt{ff}) = C^s(\Phi_0)(x) = 0 \quad (x \in A)$;
and if $\mathrm{arity}(\phi) = 0$ and $\phi\downarrow$,
$C^s(\Phi_0)(\phi) = 0$ if $\phi \notin \Phi_0$, and $C^s(\Phi_0)(\phi) = 1$ if $\phi \in \Phi_0$.

(C^s2) If $M \equiv \phi(M_1, \ldots, M_n)$, then

$$C^s(\Phi_0)(M) = \begin{cases} C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n) + 1, & \text{if } \phi \in \Phi_0, \\ C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n), & \text{otherwise.} \end{cases}$$

(C^s3) If $M \equiv \text{if } M_0 \text{ then } M_1 \text{ else } M_2$, then

$$C^s(\Phi_0)(M) = \begin{cases} C^s(\Phi_0)(M_0) + C^s(\Phi_0)(M_1), & \text{if } \overline{M}_0 = \mathtt{tt}, \\ C^s(\Phi_0)(M_0) + C^s(\Phi_0)(M_2), & \text{if } \overline{M}_0 = \mathtt{ff}. \end{cases}$$

(C^s4) If $M \equiv \mathtt{p}_i(M_1, \ldots, M_n)$ with $\mathtt{p}_i$ a recursive variable of E, then
$C^s(\Phi_0)(M) = C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n) + C^s(\Phi_0)(E_i(\overline{M}_1, \ldots, \overline{M}_n))$

- *If E simulates a program E^-, then $C^s(\Phi_0)(E_0(\vec{x}))$ "agrees" with the number-of-Φ_0-calls complexity of E^-*

The mergesort algorithm

- Suppose L is ordered by $\leq$, L^* is the set of all sequences from L, $\mathbf{L^*}_{\mathrm{ms}} = (\mathbf{L}^*, \mathrm{half}_1, \mathrm{half}_2, \leq)$ is the indicated expansion of the standard Lisp structure $\mathbf{L}^*$ and consider the recursive program

$$p(u) \text{ where } \left\{ \begin{array}{l} p(u) = \text{ if } (\mathrm{tail}(u) = \mathrm{nil}) \text{ then } u \text{ else } q(p(\mathrm{half}_1(u)), p(\mathrm{half}_2(u))), \\ q(w, v) = \text{ if } (w = \mathrm{nil}) \text{ then } v \text{ else } \text{ if } (v = \mathrm{nil}) \text{ then } w \\ \qquad \text{else } \text{ if } (\mathrm{head}(w) \leq \mathrm{head}(v)) \text{ then } \mathrm{cons}(\mathrm{head}(w), q(\mathrm{tail}(w), v)) \\ \qquad \text{else } \mathrm{cons}(\mathrm{head}(v), q(w, \mathrm{tail}(v))) \end{array} \right\}$$

which expresses the mergesort algorithm on $\mathbf{L^*}_{\mathrm{ms}}$

- *The mergesort computes the sorted version* $\mathrm{sort}(u)$ *of each sequence* $u \in L^*$ *so that*

$$\mathrm{calls}(\leq)(u) \leq |u| \log |u| \quad (|u| \geq 2)$$

- It is well-known that this is asymptotically the best upper bound for sorting by "deterministic comparison algorithms"

The mergesort algorithm

- Suppose L is ordered by $\leq$, L^* is the set of all sequences from L, $\mathbf{L^*}_{\mathrm{ms}} = (\mathbf{L^*}, \mathrm{half}_1, \mathrm{half}_2, \leq)$ is the indicated expansion of the standard Lisp structure $\mathbf{L^*}$ and consider the recursive program

$$p(u) \text{ where } \left\{ \begin{array}{l} p(u) = \text{ if } (\mathrm{tail}(u) = \mathrm{nil}) \text{ then } u \text{ else } q(p(\mathrm{half}_1(u)), p(\mathrm{half}_2(u))), \\ q(w,v) = \text{ if } (w = \mathrm{nil}) \text{ then } v \text{ else } \text{ if } (v = \mathrm{nil}) \text{ then } w \\ \qquad \text{else } \text{ if } (\mathrm{head}(w) \leq \mathrm{head}(v)) \text{ then } \mathrm{cons}(\mathrm{head}(w), q(\mathrm{tail}(w), v)) \\ \qquad \text{else } \mathrm{cons}(\mathrm{head}(v), q(w, \mathrm{tail}(v))) \end{array} \right\}$$

which expresses the mergesort algorithm on $\mathbf{L^*}_{\mathrm{ms}}$

- *The mergesort computes the sorted version* $\mathrm{sort}(u)$ *of each sequence* $u \in L^*$ *so that*

$$\mathrm{calls}(\leq)(u) \leq |u| \log |u| \quad (|u| \geq 2)$$

- It is well-known that this is asymptotically the best upper bound for sorting by "deterministic comparison algorithms"

The mergesort algorithm

- Suppose L is ordered by $\leq$, L^* is the set of all sequences from L, $\mathbf{L}^*_{\mathrm{ms}} = (\mathbf{L}^*, \mathrm{half}_1, \mathrm{half}_2, \leq)$ is the indicated expansion of the standard Lisp structure $\mathbf{L}^*$ and consider the recursive program

$$p(u) \text{ where } \Big\{ p(u) = \text{ if } (\mathrm{tail}(u) = \mathrm{nil}) \text{ then } u \text{ else } q(p(\mathrm{half}_1(u)), p(\mathrm{half}_2(u))),$$

$$q(w, v) = \text{ if } (w = \mathrm{nil}) \text{ then } v \text{ else if } (v = \mathrm{nil}) \text{ then } w$$

$$\text{else if } (\mathrm{head}(w) \leq \mathrm{head}(v)) \text{ then } \mathrm{cons}(\mathrm{head}(w), q(\mathrm{tail}(w), v))$$

$$\text{else } \mathrm{cons}(\mathrm{head}(v), q(w, \mathrm{tail}(v))) \Big\}$$

which expresses the mergesort algorithm on $\mathbf{L}^*_{\mathrm{ms}}$

- *The mergesort computes the sorted version* $\mathrm{sort}(u)$ *of each sequence* $u \in L^*$ *so that*

$$\mathrm{calls}(\leq)(u) \leq |u| \log |u| \quad (|u| \geq 2)$$

- It is well-known that this is asymptotically the best upper bound for sorting by "deterministic comparison algorithms"

The mergesort algorithm

- Suppose L is ordered by $\leq$, L^* is the set of all sequences from L, $\mathbf{L}^*_{\mathrm{ms}} = (\mathbf{L}^*, \mathrm{half}_1, \mathrm{half}_2, \leq)$ is the indicated expansion of the standard Lisp structure $\mathbf{L}^*$ and consider the recursive program

$$p(u) \text{ where } \Big\{ p(u) = \text{ if } (\mathrm{tail}(u) = \mathrm{nil}) \text{ then } u \text{ else } q(p(\mathrm{half}_1(u)), p(\mathrm{half}_2(u))),$$

$$q(w, v) = \text{ if } (w = \mathrm{nil}) \text{ then } v \text{ else if } (v = \mathrm{nil}) \text{ then } w$$

$$\text{else if } (\mathrm{head}(w) \leq \mathrm{head}(v)) \text{ then } \mathrm{cons}(\mathrm{head}(w), q(\mathrm{tail}(w), v))$$

$$\text{else } \mathrm{cons}(\mathrm{head}(v), q(w, \mathrm{tail}(v))) \Big\}$$

which expresses the mergesort algorithm on $\mathbf{L}^*_{\mathrm{ms}}$

- *The mergesort computes the sorted version* $\mathrm{sort}(u)$ *of each sequence* $u \in L^*$ *so that*

$$\mathrm{calls}(\leq)(u) \leq |u| \log |u| \quad (|u| \geq 2)$$

- It is well-known that this is asymptotically the best upper bound for sorting by "deterministic comparison algorithms"

The mergesort algorithm

- Suppose L is ordered by $\leq$, L^* is the set of all sequences from L, $\mathbf{L}^*_{\mathrm{ms}} = (\mathbf{L}^*, \mathrm{half}_1, \mathrm{half}_2, \leq)$ is the indicated expansion of the standard Lisp structure $\mathbf{L}^*$ and consider the recursive program

$$p(u) \text{ where } \Big\{ p(u) = \text{ if } (\mathrm{tail}(u) = \mathrm{nil}) \text{ then } u \text{ else } q(p(\mathrm{half}_1(u)), p(\mathrm{half}_2(u))),$$

$$q(w, v) = \text{ if } (w = \mathrm{nil}) \text{ then } v \text{ else if } (v = \mathrm{nil}) \text{ then } w$$

$$\text{else if } (\mathrm{head}(w) \leq \mathrm{head}(v)) \text{ then } \mathrm{cons}(\mathrm{head}(w), q(\mathrm{tail}(w), v))$$

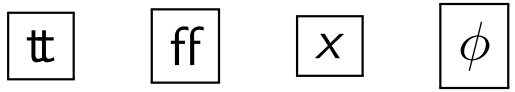
$$\text{else } \mathrm{cons}(\mathrm{head}(v), q(w, \mathrm{tail}(v))) \Big\}$$

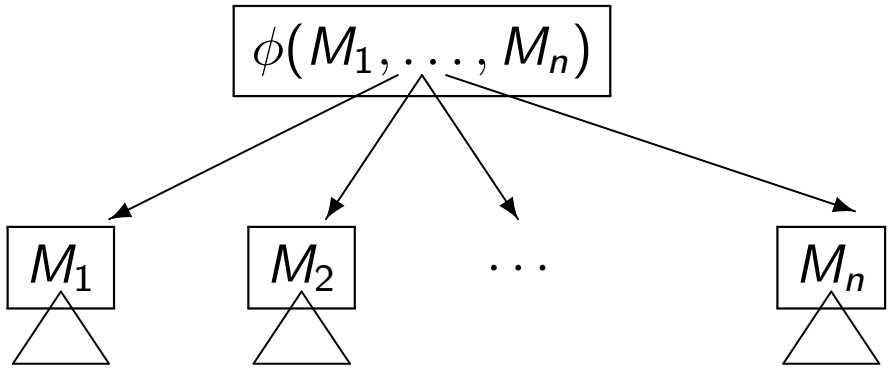
which expresses the mergesort algorithm on $\mathbf{L}^*_{\mathrm{ms}}$

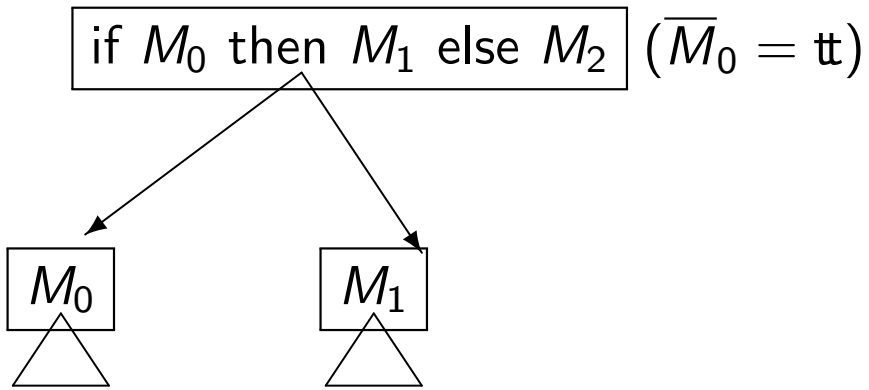
- *The mergesort computes the sorted version* $\mathrm{sort}(u)$ *of each sequence* $u \in L^*$ *so that*

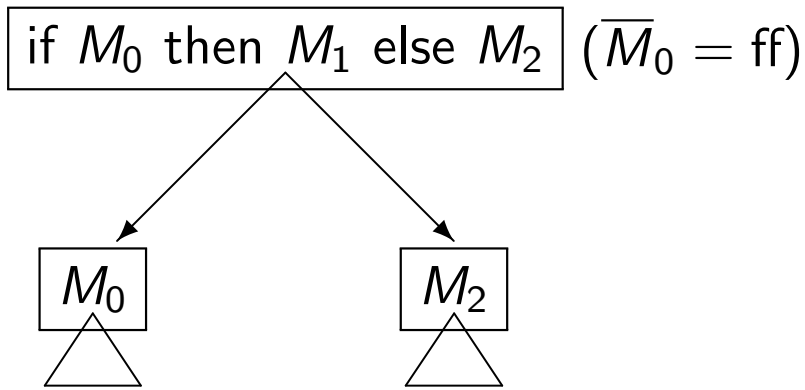
$$\mathrm{calls}(\leq)(u) \leq |u| \log |u| \quad (|u| \geq 2)$$

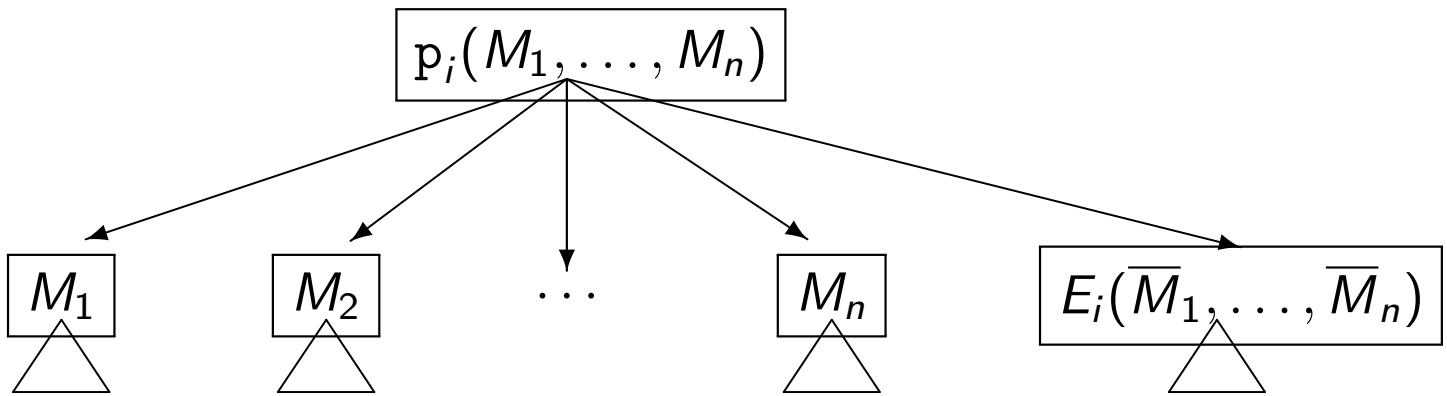
- It is well-known that this is asymptotically the best upper bound for sorting by "deterministic comparison algorithms"











- $D(M)$ is the depth of the computation tree for M

Complexity inequalities and Tserunyan's Theorem

- *For a fixed Φ-structure $\mathbf{A}$ and a recursive Φ-program E*

where calls $=$ calls(Φ), $d(\vec{x}) = D(E_0(\vec{x})$ and $\ell =$ largest arity in E

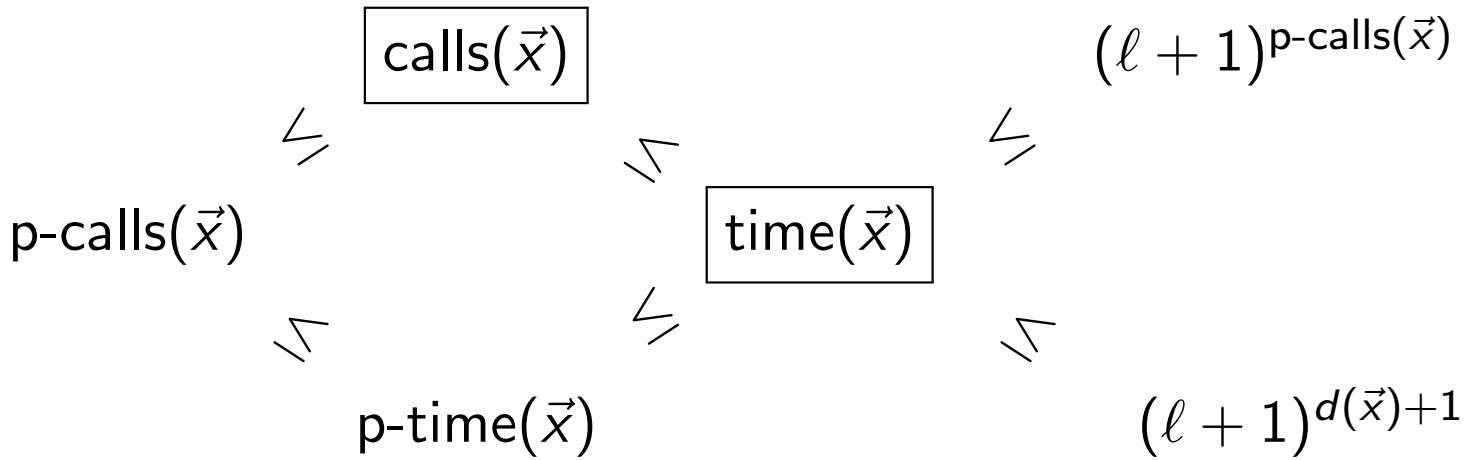
★ (Tserunyan 2013) *For every recursive Φ-program E, there is a constant K_s such that for every Φ-structure $\mathbf{A}$ and every $\vec{x} \in A^n$, if* $\text{den}(\mathbf{A}, E(\vec{x}))\downarrow$, *then*

$$\text{calls}_E(\vec{x}) \leq \text{time}_E(\vec{x}) \leq K_s + K_s \text{calls}_E(\vec{x})$$

- $\text{time}_E(\vec{x}) = \#(\text{logical calls})_E(\vec{x}) + \text{calls}(\Phi)_E(\vec{x}) \approx K_s \text{calls}(\Phi)_E(\vec{x})$
- Perhaps this is why lower bound results are most often proved by counting calls to the primitives

Complexity inequalities and Tserunyan's Theorem

- *For a fixed* Φ*-structure* $\mathbf{A}$ *and a recursive* Φ*-program* E



where calls $=$ calls(Φ), $d(\vec{x}) = D(E_0(\vec{x})$ and $\ell =$ largest arity in E

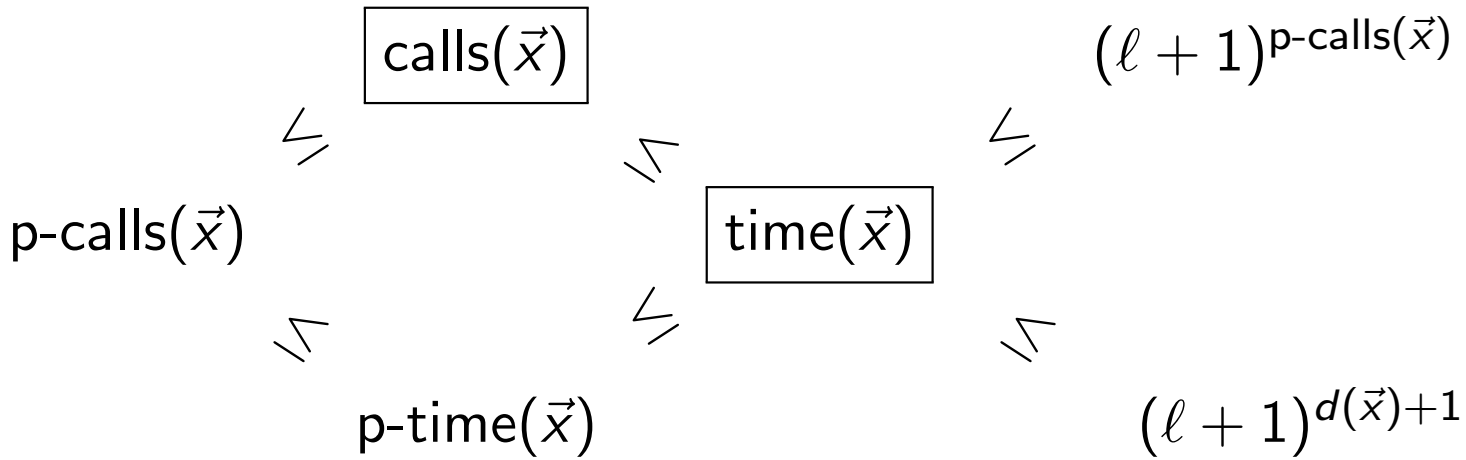
- (Tserunyan 2013) *For every recursive* Φ*-program* E*, there is a constant* K_s *such that for every* Φ*-structure* $\mathbf{A}$ *and every* $\vec{x} \in A^n$*, if* $\text{den}(\mathbf{A}, E(\vec{x}))\downarrow$*, then*

$$\text{calls}_E(\vec{x}) \leq \text{time}_E(\vec{x}) \leq K_s + K_s \text{calls}_E(\vec{x})$$

- $\text{time}_E(\vec{x}) = \#(\text{logical calls})_E(\vec{x}) + \text{calls}(\Phi)_E(\vec{x}) \approx K_s \text{calls}(\Phi)_E(\vec{x})$
- Perhaps this is why lower bound results are most often proved by counting calls to the primitives

Complexity inequalities and Tserunyan's Theorem

- *For a fixed* Φ*-structure* $\mathbf{A}$ *and a recursive* Φ*-program* E



where calls $=$ calls(Φ), $d(\vec{x}) = D(E_0(\vec{x})$ and $\ell =$ largest arity in E

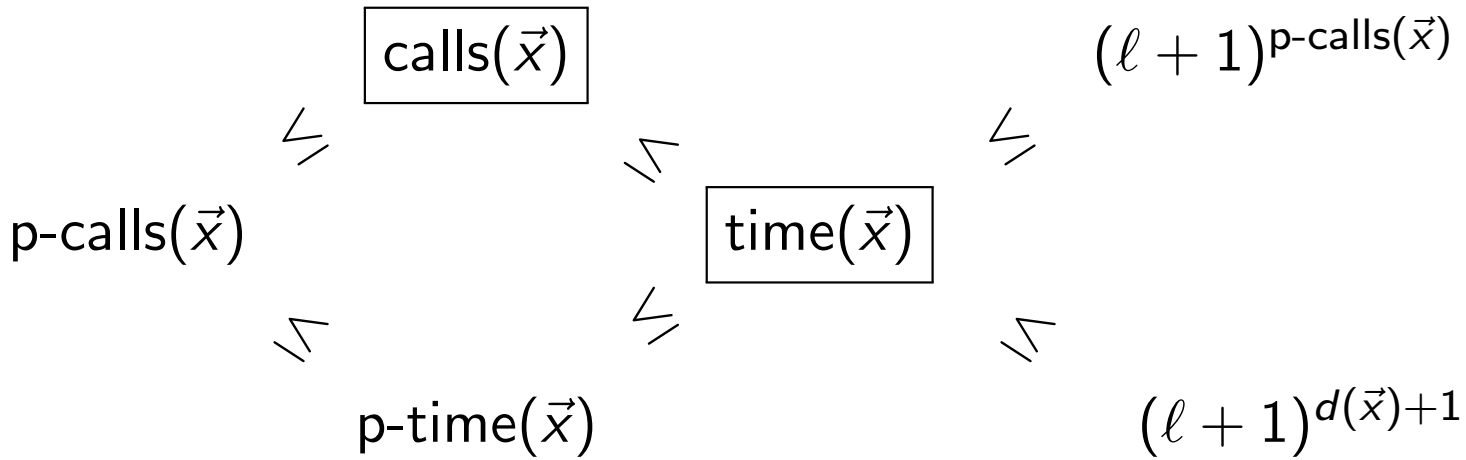
★ (Tserunyan 2013) *For every recursive* Φ*-program* E*, there is a constant* K_s *such that for every* Φ*-structure* $\mathbf{A}$ *and every* $\vec{x} \in A^n$*, if* $\text{den}(\mathbf{A}, E(\vec{x}))\downarrow$*, then*

$$\text{calls}_E(\vec{x}) \leq \text{time}_E(\vec{x}) \leq K_s + K_s\text{calls}_E(\vec{x})$$

- $\text{time}_E(\vec{x}) = \#(\text{logical calls})_E(\vec{x}) + \text{calls}(\Phi)_E(\vec{x}) \approx K_s\text{calls}(\Phi)_E(\vec{x})$
- Perhaps this is why lower bound results are most often proved by counting calls to the primitives

Complexity inequalities and Tserunyan's Theorem

- *For a fixed* Φ*-structure* $\mathbf{A}$ *and a recursive* Φ*-program* E



where calls $=$ calls(Φ), $d(\vec{x}) = D(E_0(\vec{x})$ and $\ell =$ largest arity in E

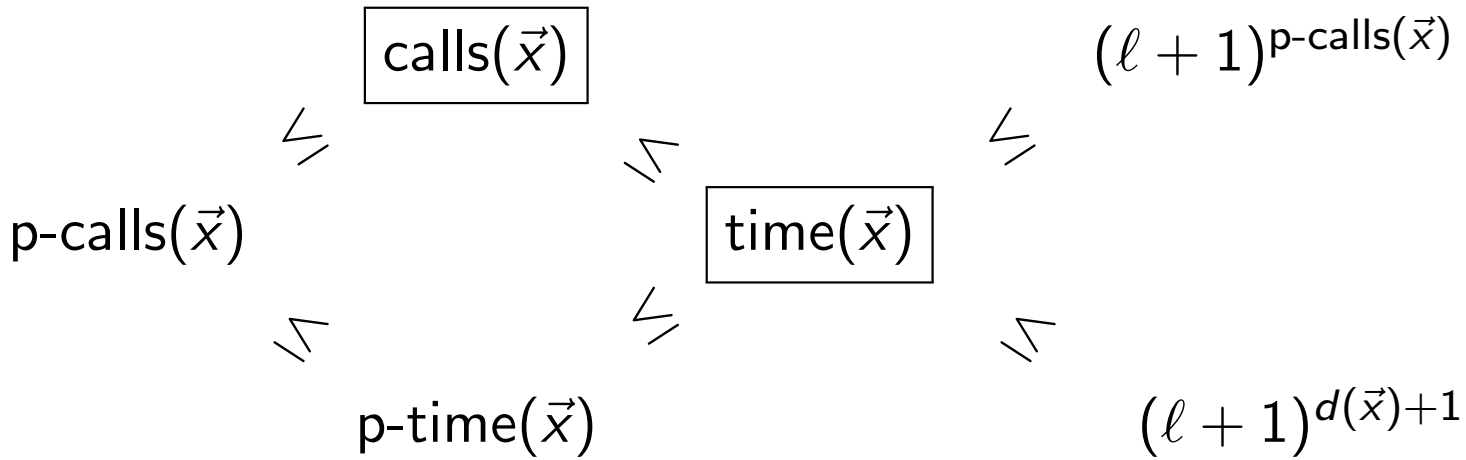
★ (Tserunyan 2013) *For every recursive* Φ*-program* E*, there is a constant* K_s *such that for every* Φ*-structure* $\mathbf{A}$ *and every* $\vec{x} \in A^n$*, if* $\text{den}(\mathbf{A}, E(\vec{x}))\downarrow$*, then*

$$\text{calls}_E(\vec{x}) \leq \text{time}_E(\vec{x}) \leq K_s + K_s\text{calls}_E(\vec{x})$$

- $\text{time}_E(\vec{x}) = \#(\text{logical calls})_E(\vec{x}) + \text{calls}(\Phi)_E(\vec{x}) \approx K_s\text{calls}(\Phi)_E(\vec{x})$
- Perhaps this is why lower bound results are most often proved by counting calls to the primitives

Complexity inequalities and Tserunyan's Theorem

- *For a fixed* Φ*-structure* $\mathbf{A}$ *and a recursive* Φ*-program* E



where calls $=$ calls(Φ), $d(\vec{x}) = D(E_0(\vec{x})$ and $\ell =$ largest arity in E

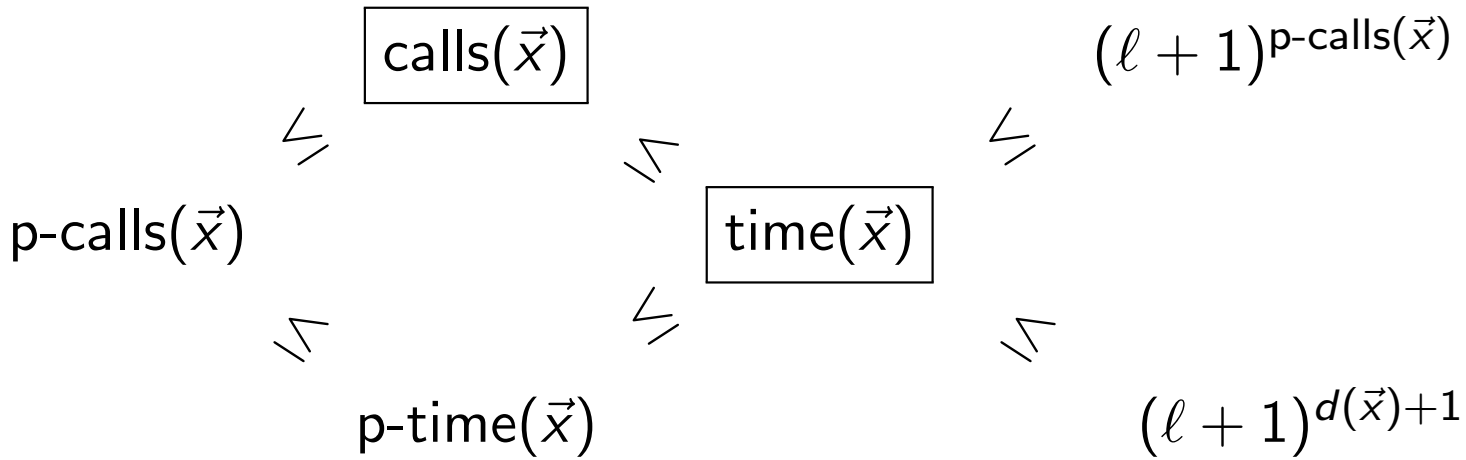
★ (Tserunyan 2013) *For every recursive* Φ*-program* E*, there is a constant* K_s *such that for every* Φ*-structure* $\mathbf{A}$ *and every* $\vec{x} \in A^n$*, if* $\text{den}(\mathbf{A}, E(\vec{x}))\downarrow$*, then*

$$\text{calls}_E(\vec{x}) \leq \text{time}_E(\vec{x}) \leq K_s + K_s\text{calls}_E(\vec{x})$$

- $\text{time}_E(\vec{x}) = \#(\text{logical calls})_E(\vec{x}) + \text{calls}(\Phi)_E(\vec{x}) \approx K_s\text{calls}(\Phi)_E(\vec{x})$
- Perhaps this is why lower bound results are most often proved by counting calls to the primitives

Complexity inequalities and Tserunyan's Theorem

- *For a fixed* Φ*-structure* $\mathbf{A}$ *and a recursive* Φ*-program* E



where calls $=$ calls(Φ), $d(\vec{x}) = D(E_0(\vec{x})$ and $\ell =$ largest arity in E

★ (Tserunyan 2013) *For every recursive* Φ*-program* E*, there is a constant* K_s *such that for every* Φ*-structure* $\mathbf{A}$ *and every* $\vec{x} \in A^n$*, if* $\text{den}(\mathbf{A}, E(\vec{x}))\downarrow$*, then*

$$\text{calls}_E(\vec{x}) \leq \text{time}_E(\vec{x}) \leq K_s + K_s\text{calls}_E(\vec{x})$$

- $\text{time}_E(\vec{x}) = \#(\text{logical calls})_E(\vec{x}) + \text{calls}(\Phi)_E(\vec{x}) \approx K_s\text{calls}(\Phi)_E(\vec{x})$
- Perhaps this is why lower bound results are most often proved by counting calls to the primitives