Computable Execution Traces & Algorithms

Declan Thompson

21/22 May 2021

Outline

What are algorithms?

What are algorithms?

There is no algorithm to decide whether the domain of Φ_x is empty.
[3, Proposition 2.1.4]

If G is a connected weighted graph then the $Prim(G, v)$-algorithm produces a minimum spanning tree for G.
[6, Theorem 17.3]

What are algorithms?

There is no algorithm to decide whether the domain of Φ_x is empty.
[3, Proposition 2.1.4]

If G is a connected weighted graph then the $Prim(G, v)$-algorithm produces a minimum spanning tree for G.
[6, Theorem 17.3]

- “They’re like a recipe!”

What are algorithms?

There is no algorithm to decide whether the domain of Φ_x is empty.
[3, Proposition 2.1.4]

If G is a connected weighted graph then the $\mathit{Prim}(G, v)$-algorithm produces a minimum spanning tree for G.
[6, Theorem 17.3]

- "They're like a recipe!"
- Turing machines!
  - Or some other classical model of computation.

What are algorithms?

There is no algorithm to decide whether the domain of Φ_x is empty.
[3, Proposition 2.1.4]

If G is a connected weighted graph then the $Prim(G, v)$-algorithm produces a minimum spanning tree for G.
[6, Theorem 17.3]

- “They’re like a recipe!”
- Turing machines!
  - Or some other classical model of computation.
- Programs!

What are algorithms?

There is no algorithm to decide whether the domain of Φ_x is empty.
[3, Proposition 2.1.4]

If G is a connected weighted graph then the $Prim(G, v)$-algorithm produces a minimum spanning tree for G.
[6, Theorem 17.3]

- "They're like a recipe!"
- Turing machines!
  - Or some other classical model of computation.
- Programs!
  - "A concept like 'abstract algorithm' without reference to any algorithmic language does not exist. In order to specify an algorithm one has to give the specifications in some algorithmic language." [5, p. 654]

What are algorithms?

There is no algorithm to decide whether the domain of Φ_x is empty.
[3, Proposition 2.1.4]

If G is a connected weighted graph then the $\mathit{Prim}(G, v)$-algorithm produces a minimum spanning tree for G.
[6, Theorem 17.3]

- "They're like a recipe!"
- Turing machines!
  - Or some other classical model of computation.
- Programs!
  - "A concept like 'abstract algorithm' without reference to any algorithmic language does not exist. In order to specify an algorithm one has to give the specifications in some algorithmic language." [5, p. 654]

What are algorithms?

There is no algorithm to decide whether the domain of Φ_x is empty.
[3, Proposition 2.1.4]

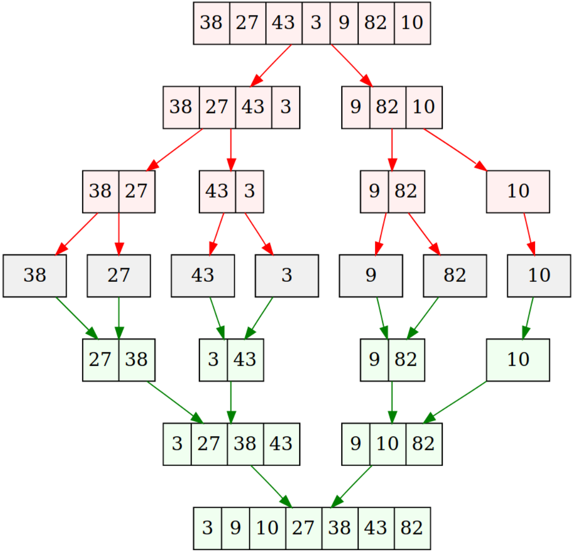
If G is a connected weighted graph then the $\mathit{Prim}(G, v)$-algorithm produces a minimum spanning tree for G.
[6, Theorem 17.3]

- "They're like a recipe!"
- Turing machines!
  - Or some other classical model of computation.
- Programs!
  - "A concept like 'abstract algorithm' without reference to any algorithmic language does not exist. In order to specify an algorithm one has to give the specifications in some algorithmic language." [5, p. 654]

Unambiguous, deterministic, clear sequences of instructions for achieving some task.

What are algorithms?

Mergesort sorts a given list of numbers by first dividing them into two equal halves, sorting each half separately by recursion, and then combining the results of these recursive calls–in the form of the two sorted halves–using the linear time algorithm for merging sorted lists that we saw in Chapter 2. [7, p. 210]



...is that really what algorithms are?

"Unambiguous, deterministic, clear sequences of instructions for achieving some task."

...is that really what algorithms are?

"Unambiguous, deterministic, clear sequences of instructions for achieving some task."

- Which model of computation should be chosen? What are the primitive objects?

...is that really what algorithms are?

"Unambiguous, deterministic, clear sequences of instructions for achieving some task."

- Which model of computation should be chosen? What are the primitive objects?
- Algorithms need not be fully determined.

…is that really what algorithms are?

"Unambiguous, deterministic, clear sequences of instructions for achieving some task."

- Which model of computation should be chosen? What are the primitive objects?
- Algorithms need not be fully determined.
- Programs are language dependent. Algorithms are not.

...is that really what algorithms are?

"Unambiguous, deterministic, clear sequences of instructions for achieving some task."

- Which model of computation should be chosen? What are the primitive objects?
- Algorithms need not be fully determined.
- Programs are language dependent. Algorithms are not.
- Is an algorithm the set of instructions, or the behaviour they describe?

…is that really what algorithms are?

"Unambiguous, deterministic, clear sequences of instructions for achieving some task."

- Which model of computation should be chosen? What are the primitive objects?
- Algorithms need not be fully determined.
- Programs are language dependent. Algorithms are not.
- Is an algorithm the set of instructions, or the behaviour they describe?

...is that really what algorithms are?

"Unambiguous, deterministic, clear sequences of instructions for achieving some task."

- Which model of computation should be chosen? What are the primitive objects?
- Algorithms need not be fully determined.
- Programs are language dependent. Algorithms are not.
- Is an algorithm the set of instructions, or the behaviour they describe?

Equivalence class approach

Algorithms as equivalence classes under some equivalence relation. *Dubious that any adequate equivalence relation exists.*

Generalised programs

Allow arbitrary operations and objects in programs. *Language dependent, and over-determined.*

Computable execution traces

- Classical accounts of computability have a standard form:

Computable execution traces

- Classical accounts of computability have a standard form:
  1. Identify a domain of primitive objects

$$\mathcal{T} = \{[|\mathtt{B}], [\mathtt{11}|\mathtt{B}], [\mathtt{1B1}|\mathtt{B11}], \ldots\}$$

Computable execution traces

- Classical accounts of computability have a standard form:
  1. Identify a domain of primitive objects
  2. Choose primitive actions on that domain

$$\mathcal{T} = \{[|\texttt{B}], [\texttt{11}|\texttt{B}], [\texttt{1B1}|\texttt{B11}], \ldots\}$$

$$\textsc{Read}_1, \textsc{Move}_{\mathsf{B,R}}, \ldots$$

Computable execution traces

- Classical accounts of computability have a standard form:
  1. Identify a domain of primitive objects
  2. Choose primitive actions on that domain
  3. Describe how actions can be combined

$$\mathcal{T} = \{[|\texttt{B}], [\texttt{11}|\texttt{B}], [\texttt{1B1}|\texttt{B11}], \ldots\}$$

$$\textsc{Read}_1, \textsc{Move}_{\mathsf{B},\mathsf{R}}, \ldots$$

δ	1	B
s_0	$\langle 1, \mathsf{R}, s_0 \rangle$	$\langle 1, \mathsf{R}, s_1 \rangle$
s_1	$\langle 1, \mathsf{R}, s_1 \rangle$	$\langle \texttt{B}, \mathsf{L}, s_2 \rangle$
s_2	$\langle \texttt{B}, \mathsf{L}, s_3 \rangle$	—
s_3	—	—

Computable execution traces

- Classical accounts of computability have a standard form:
  1. Identify a domain of primitive objects
  2. Choose primitive actions on that domain
  3. Describe how actions can be combined

$$\mathcal{T} = \{[|\texttt{B}], [\texttt{11}|\texttt{B}], [\texttt{1B1}|\texttt{B11}], \dots\}$$

$$\textsc{Read}_1, \textsc{Move}_{\mathsf{B},\mathsf{R}}, \dots$$

δ	1	B
s_0	$\langle 1, \mathsf{R}, s_0\rangle$	$\langle 1, \mathsf{R}, s_1\rangle$
s_1	$\langle 1, \mathsf{R}, s_1\rangle$	$\langle \texttt{B}, \mathsf{L}, s_2\rangle$
s_2	$\langle \texttt{B}, \mathsf{L}, s_3\rangle$	—
s_3	—	—

$$(\!|[|\texttt{1B11}],$$

Computable execution traces

- Classical accounts of computability have a standard form:
  1. Identify a domain of primitive objects
  2. Choose primitive actions on that domain
  3. Describe how actions can be combined

$$\mathcal{T} = \{[|\texttt{B}], [\texttt{11}|\texttt{B}], [\texttt{1B1}|\texttt{B11}], \ldots\}$$

$$\textsc{Read}_1, \textsc{Move}_{\texttt{B},\mathsf{R}}, \ldots$$

δ	1	B
s_0	$\langle 1, \mathsf{R}, s_0 \rangle$	$\langle 1, \mathsf{R}, s_1 \rangle$
s_1	$\langle 1, \mathsf{R}, s_1 \rangle$	$\langle \texttt{B}, \mathsf{L}, s_2 \rangle$
s_2	$\langle \texttt{B}, \mathsf{L}, s_3 \rangle$	—
s_3	—	—

$$(\![|\texttt{1B11}],[\texttt{1}|\texttt{B11}],$$

Computable execution traces

- Classical accounts of computability have a standard form:
  1. Identify a domain of primitive objects
  2. Choose primitive actions on that domain
  3. Describe how actions can be combined

$$\mathcal{T} = \{[|\mathtt{B}], [\mathtt{11}|\mathtt{B}], [\mathtt{1B1}|\mathtt{B11}], \ldots\}$$

$$\textsc{Read}_1, \textsc{Move}_{\mathtt{B},\mathsf{R}}, \ldots$$

δ	1	B
s_0	$\langle 1, \mathsf{R}, s_0\rangle$	$\langle 1, \mathsf{R}, s_1\rangle$
s_1	$\langle 1, \mathsf{R}, s_1\rangle$	$\langle \mathtt{B}, \mathsf{L}, s_2\rangle$
s_2	$\langle \mathtt{B}, \mathsf{L}, s_3\rangle$	—
s_3	—	—

$$(\!([|\mathtt{1B11}],[\mathtt{1}|\mathtt{B11}],[\mathtt{11}|\mathtt{11}],$$

Computable execution traces

- Classical accounts of computability have a standard form:
  1. Identify a domain of primitive objects
  2. Choose primitive actions on that domain
  3. Describe how actions can be combined

$$\mathcal{T} = \{[|\mathtt{B}], [\mathtt{11}|\mathtt{B}], [\mathtt{1B1}|\mathtt{B11}], \ldots\}$$

$$\textsc{Read}_1, \textsc{Move}_{\mathtt{B},\mathsf{R}}, \ldots$$

δ	1	B
s_0	$\langle 1, \mathsf{R}, s_0 \rangle$	$\langle 1, \mathsf{R}, s_1 \rangle$
s_1	$\langle 1, \mathsf{R}, s_1 \rangle$	$\langle \mathtt{B}, \mathsf{L}, s_2 \rangle$
s_2	$\langle \mathtt{B}, \mathsf{L}, s_3 \rangle$	—
s_3	—	—

$$([|\mathtt{1B11}], [\mathtt{1}|\mathtt{B11}], [\mathtt{11}|\mathtt{11}], [\mathtt{111}|\mathtt{1}],$$

Computable execution traces

- Classical accounts of computability have a standard form:
  1. Identify a domain of primitive objects
  2. Choose primitive actions on that domain
  3. Describe how actions can be combined

$$\mathcal{T} = \{[|\mathtt{B}], [\mathtt{11}|\mathtt{B}], [\mathtt{1B1}|\mathtt{B11}], \dots\}$$

$$\textsc{Read}_1, \textsc{Move}_{\mathtt{B},\mathsf{R}}, \dots$$

δ	1	B
s_0	$\langle 1, \mathsf{R}, s_0 \rangle$	$\langle 1, \mathsf{R}, s_1 \rangle$
s_1	$\langle 1, \mathsf{R}, s_1 \rangle$	$\langle \mathtt{B}, \mathsf{L}, s_2 \rangle$
s_2	$\langle \mathtt{B}, \mathsf{L}, s_3 \rangle$	—
s_3	—	—

$$(\!([|\mathtt{1B11}], [\mathtt{1}|\mathtt{B11}], [\mathtt{11}|\mathtt{11}], [\mathtt{111}|\mathtt{1}],$$
$$[\mathtt{1111}|\mathtt{B}],$$

Computable execution traces

- Classical accounts of computability have a standard form:
  1. Identify a domain of primitive objects
  2. Choose primitive actions on that domain
  3. Describe how actions can be combined

$$\mathcal{T} = \{[|\mathtt{B}], [\mathtt{11}|\mathtt{B}], [\mathtt{1B1}|\mathtt{B11}], \ldots\}$$

$$\textsc{Read}_1, \textsc{Move}_{\mathtt{B},\mathsf{R}}, \ldots$$

δ	$\mathtt{1}$	$\mathtt{B}$
s_0	$\langle \mathtt{1}, \mathsf{R}, s_0 \rangle$	$\langle \mathtt{1}, \mathsf{R}, s_1 \rangle$
s_1	$\langle \mathtt{1}, \mathsf{R}, s_1 \rangle$	$\langle \mathtt{B}, \mathsf{L}, s_2 \rangle$
s_2	$\langle \mathtt{B}, \mathsf{L}, s_3 \rangle$	—
s_3	—	—

$$(\!([|\mathtt{1B11}], [\mathtt{1}|\mathtt{B11}], [\mathtt{11}|\mathtt{11}], [\mathtt{111}|\mathtt{1}],$$
$$[\mathtt{1111}|\mathtt{B}], [\mathtt{111}|\mathtt{1B}],$$

Computable execution traces

- Classical accounts of computability have a standard form:
  1. Identify a domain of primitive objects
  2. Choose primitive actions on that domain
  3. Describe how actions can be combined

$$\mathcal{T} = \{[|\mathtt{B}], [\mathtt{11}|\mathtt{B}], [\mathtt{1B1}|\mathtt{B11}], \ldots\}$$

$$\textsc{Read}_1, \textsc{Move}_{\mathtt{B},\mathsf{R}}, \ldots$$

δ	1	B
s_0	$\langle 1, \mathsf{R}, s_0 \rangle$	$\langle 1, \mathsf{R}, s_1 \rangle$
s_1	$\langle 1, \mathsf{R}, s_1 \rangle$	$\langle \mathtt{B}, \mathsf{L}, s_2 \rangle$
s_2	$\langle \mathtt{B}, \mathsf{L}, s_3 \rangle$	—
s_3	—	—

$$(\!([|\mathtt{1B11}],[\mathtt{1}|\mathtt{B11}],[\mathtt{11}|\mathtt{11}],[\mathtt{111}|\mathtt{1}],[\mathtt{1111}|\mathtt{B}],[\mathtt{111}|\mathtt{1B}],[\mathtt{11}|\mathtt{1BB}])\!)$$

Computable execution traces

- Classical accounts of computability have a standard form:
  1. Identify a domain of primitive objects
  2. Choose primitive actions on that domain
  3. Describe how actions can be combined
  4. Ask *what*, not *how*

$$\mathcal{T} = \{[|\mathtt{B}], [\mathtt{11}|\mathtt{B}], [\mathtt{1B1}|\mathtt{B11}], \dots\}$$

$$\textsc{Read}_1, \textsc{Move}_{\mathsf{B},\mathsf{R}}, \dots$$

δ	$\mathtt{1}$	$\mathtt{B}$
s_0	$\langle \mathtt{1}, \mathsf{R}, s_0 \rangle$	$\langle \mathtt{1}, \mathsf{R}, s_1 \rangle$
s_1	$\langle \mathtt{1}, \mathsf{R}, s_1 \rangle$	$\langle \mathtt{B}, \mathsf{L}, s_2 \rangle$
s_2	$\langle \mathtt{B}, \mathsf{L}, s_3 \rangle$	—
s_3	—	—

$$(\!|[|\mathtt{1B11}], [\mathtt{1}|\mathtt{B11}], [\mathtt{11}|\mathtt{11}], [\mathtt{111}|\mathtt{1}], [\mathtt{1111}|\mathtt{B}], [\mathtt{111}|\mathtt{1B}], [\mathtt{11}|\mathtt{1BB}]|\!)$$

Computable execution traces

- Classical accounts of computability have a standard form:
  1. Identify a domain of primitive objects
  2. Choose primitive actions on that domain
  3. Describe how actions can be combined
  4. Ask *what*, not *how*
  5. Show extensional equivalence with other accounts $\rightsquigarrow$ Church-Turing Thesis

$$\mathcal{T} = \{[|\mathtt{B}], [\mathtt{11}|\mathtt{B}], [\mathtt{1B1}|\mathtt{B11}], \ldots\}$$

$$\textsc{Read}_1, \textsc{Move}_{\mathtt{B},\mathsf{R}}, \ldots$$

δ	1	B
s_0	$\langle 1, \mathsf{R}, s_0\rangle$	$\langle 1, \mathsf{R}, s_1\rangle$
s_1	$\langle 1, \mathsf{R}, s_1\rangle$	$\langle \mathtt{B}, \mathsf{L}, s_2\rangle$
s_2	$\langle \mathtt{B}, \mathsf{L}, s_3\rangle$	—
s_3	—	—

$$(\!|[|\mathtt{1B11}],[\mathtt{1}|\mathtt{B11}],[\mathtt{11}|\mathtt{11}],[\mathtt{111}|\mathtt{1}], [\mathtt{1111}|\mathtt{B}],[\mathtt{111}|\mathtt{1B}],[\mathtt{11}|\mathtt{1BB}]|\!)$$

Computable execution traces (cont.)

In all cases, we get a model which takes input and moves through a (possibly infinite) sequences of stages. This gives a set of *execution traces*, built from the bottom up.

$$\textsc{Run}_{\mathfrak{M}} = \left\{ \begin{array}{c} ([|\mathrm{B}], [1|\mathrm{B}], [|1\mathrm{B}], [|\mathrm{BB}]) \\ ([|1], [1|\mathrm{B}], [11|\mathrm{B}], [1|1\mathrm{B}], [|1\mathrm{BB}]) \\ ([|1\mathrm{B}1], [1|\mathrm{B}1], [11|1], [111|\mathrm{B}], [11|1\mathrm{B}], [1|1\mathrm{BB}]) \\ \vdots \end{array} \right\}$$

The focus is on what can be computed, not the way the computation proceeds.

Motivating question

What is required for a given set of sequences to be the execution trace set of some computable process?

Notation

- $\sigma, \tau, \upsilon, \ldots$ are sequences.

Notation

- $\sigma, \tau, \upsilon, \ldots$ are sequences.
- $\mathrm{F}, \mathrm{G}, \mathrm{H}$ are *tasks*: sets of sequences.

Notation

- $\sigma, \tau, \upsilon, \ldots$ are sequences.
- $\mathrm{F}, \mathrm{G}, \mathrm{H}$ are *tasks*: sets of sequences.
- $\sigma[0, \alpha + 1) = \llparenthesis \sigma[0], \sigma[1], \ldots \sigma[\alpha] \rrparenthesis$ if $\alpha < |\sigma|$, otherwise σ.

$$\llparenthesis [|\mathtt{B}], [\mathtt{1}|\mathtt{B}], [|\mathtt{1B}], [|\mathtt{BB}] \rrparenthesis[0, 3) = \llparenthesis [|\mathtt{B}], [\mathtt{1}|\mathtt{B}] \rrparenthesis$$

Notation

- $\sigma, \tau, \upsilon, \ldots$ are sequences.
- $\mathrm{F}, \mathrm{G}, \mathrm{H}$ are *tasks*: sets of sequences.
- $\sigma[0, \alpha+1) = (\!|\sigma[0], \sigma[1], \ldots \sigma[\alpha]|\!)$ if $\alpha < |\sigma|$, otherwise σ.

$$(\!|[|\mathtt{B}], [\mathtt{1}|\mathtt{B}], [|\mathtt{1B}], [|\mathtt{BB}]|\!)[0,3) = (\!|[|\mathtt{B}], [\mathtt{1}|\mathtt{B}]|\!)$$

- $\boxed{\mathrm{F}} = \{\sigma[0, \alpha+1) \mid \sigma \in \mathrm{F}, \alpha < \omega\}$.

Notation

- $\sigma, \tau, \upsilon, \ldots$ are sequences.
- $\mathrm{F}, \mathrm{G}, \mathrm{H}$ are *tasks*: sets of sequences.

- $\sigma[0, \alpha+1) = (\!|\sigma[0], \sigma[1], \ldots \sigma[\alpha]|\!)$ if $\alpha < |\sigma|$, otherwise σ.

$$(\!|[|\mathtt{B}], [\mathtt{1}|\mathtt{B}], [|\mathtt{1B}], [|\mathtt{BB}]|\!)[0,3) = (\!|[|\mathtt{B}], [\mathtt{1}|\mathtt{B}]|\!)$$

- $\boxed{\mathrm{F}} = \{\sigma[0, \alpha+1) \mid \sigma \in \mathrm{F}, \alpha < \omega\}$.

- If $\sigma[-1] = \tau[0]$ then $\sigma \circ \tau = (\!|\sigma[0], \sigma[1], \ldots \sigma[-1], \tau[1], \tau[2], \ldots|\!)$.

$$(\!|[|\mathtt{B}], [\mathtt{1}|\mathtt{B}], [|\mathtt{1B}]|\!) \circ (\!|[\mathtt{1}|\mathtt{B}], [|\mathtt{BB}]|\!) = (\!|[|\mathtt{B}], [\mathtt{1}|\mathtt{B}], [|\mathtt{1B}], [|\mathtt{BB}]|\!)$$

Notation

- $\sigma, \tau, \upsilon, \ldots$ are sequences.
- $\mathrm{F}, \mathrm{G}, \mathrm{H}$ are *tasks*: sets of sequences.

- $\sigma[0, \alpha+1) = (\!|\sigma[0], \sigma[1], \ldots \sigma[\alpha]|\!)$ if $\alpha < |\sigma|$, otherwise σ.

$$(\!|[|\mathtt{B}], [\mathtt{1}|\mathtt{B}], [|\mathtt{1B}], [|\mathtt{BB}]|\!)[0,3) = (\!|[|\mathtt{B}], [\mathtt{1}|\mathtt{B}]|\!)$$

- $\boxed{\mathrm{F}} = \{\sigma[0, \alpha+1) \mid \sigma \in \mathrm{F}, \alpha < \omega\}$.

- If $\sigma[-1] = \tau[0]$ then $\sigma \circ \tau = (\!|\sigma[0], \sigma[1], \ldots \sigma[-1], \tau[1], \tau[2], \ldots|\!)$.

$$(\!|[|\mathtt{B}], [\mathtt{1}|\mathtt{B}], [|\mathtt{1B}]|\!) \circ (\!|[\mathtt{1}|\mathtt{B}], [|\mathtt{BB}]|\!) = (\!|[|\mathtt{B}], [\mathtt{1}|\mathtt{B}], [|\mathtt{1B}], [|\mathtt{BB}]|\!)$$

- $\mathrm{F} \circ \mathrm{G} = \{\sigma \circ \tau \mid \sigma \in \mathrm{F},\ \tau \in \mathrm{G}\}$.
  - Q is a *test* if $|\sigma| = 1$ for all $\sigma \in \mathrm{Q}$. $\mathrm{F} \circ \mathrm{Q} = \{\sigma \in \mathrm{F} \mid \sigma[-1] \in \mathrm{Q}\}$
  - P is an *operation* if $|\sigma| = 2$ for all $\sigma \in \mathrm{P}$. $\mathrm{F} \circ \mathrm{P} = \{\sigma \mid \sigma[0, -1) \in \mathrm{F},\ (\!|\sigma[-2], \sigma[-1]|\!) \in \mathrm{P}\}$

Which tasks could be execution trace sets?

Idea: Assemble tasks from primitive operations

- Q is a *test* if $|\sigma| = 1$ for all $\sigma \in \mathrm{Q}$
- P is an *operation* if $|\sigma| = 2$ for all $\sigma \in \mathrm{P}$
- $\mathrm{Q} \circ \mathrm{P}$ is a *guarded operation*
- $\mathrm{F} \circ \mathrm{Q} \circ \mathrm{P} = \{\sigma \mid \sigma[0, -1) \in \mathrm{F},\ \sigma[-2] \in \mathrm{Q},\ (\!|\sigma[-2], \sigma[-1]|\!) \in \mathrm{P}\}$

- Classical accounts utilise a finitary control mechanism.
- This gives an equivalence relation on stages of a computation, similar to a bisimulation:
  1. $\sigma[0, 1) \leftrightarrows \tau[0, 1)$ for all $\sigma, \tau \in \mathrm{F}$
  2. if $\sigma[0, \alpha + 1) \leftrightarrows \tau[0, \alpha + 1)$ then "the same thing" happens in each

Control Equivalence

Control Equivalence

- Assume we're given a set of tests $\mathfrak{q}$ and a set of operations $\mathfrak{p}$: the *actions* we can use.

Control Equivalence

- Assume we're given a set of tests $\mathfrak{q}$ and a set of operations $\mathfrak{p}$: the *actions* we can use.

$$
\begin{aligned}
\mathrm{TT}_{\mathsf{a}} &:= \{ (\![\mathtt{u} | \mathtt{av}]\!) \mid \mathtt{u}, \mathtt{v} \in \Gamma^* \} \\
\mathrm{TM}_{\mathsf{a},\mathsf{R}} &:= \{ (\![\mathtt{ub} | \mathtt{cv}], [\mathtt{uba} | \mathtt{v}]\!) \mid \mathtt{u}, \mathtt{v} \in \Gamma^* \ \mathsf{b}, \mathsf{c} \in \Gamma \} \\
\mathrm{TM}_{\mathsf{a},\mathsf{L}} &:= \{ (\![\mathtt{ub} | \mathtt{cv}], [\mathtt{u} | \mathtt{bav}]\!) \mid \mathtt{u}, \mathtt{v} \in \Gamma^* \ \mathsf{b}, \mathsf{c} \in \Gamma \} \\
\mathfrak{q} &= \{ \mathrm{TT}_{\mathsf{a}} \mid \mathsf{a} \in \Gamma \} \\
\mathfrak{p} &= \{ \mathrm{TM}_{\mathsf{a},\mathsf{R}}, \mathrm{TM}_{\mathsf{a},\mathsf{L}} \mid \mathsf{a} \in \Gamma \}
\end{aligned}
$$

Control Equivalence

- Assume we're given a set of tests $\mathfrak{q}$ and a set of operations $\mathfrak{p}$: the *actions* we can use.
- Define an equivalence relation $\leftrightarrows$ on $\boxed{\mathrm{F}}$, the stages of computation of an algorithm.

Control Equivalence

- Assume we're given a set of tests $\mathfrak{q}$ and a set of operations $\mathfrak{p}$: the *actions* we can use.
- Define an equivalence relation $\leftrightarrows$ on $\boxed{\mathrm{F}}$, the stages of computation of an algorithm.

Set $\sigma \leftrightarrows \tau$ iff they correspond to the same internal state of $\mathfrak{M}$.

Control Equivalence

- Assume we're given a set of tests $\mathfrak{q}$ and a set of operations $\mathfrak{p}$: the *actions* we can use.
- Define an equivalence relation $\leftrightarrows$ on $\boxed{\mathrm{F}}$, the stages of computation of an algorithm.
- $\sigma[0,1) \leftrightarrows \tau[0,1)$ for all $\sigma, \tau \in \mathrm{F}$.

Control Equivalence

- Assume we're given a set of tests $\mathfrak{q}$ and a set of operations $\mathfrak{p}$: the *actions* we can use.
- Define an equivalence relation $\leftrightarrows$ on $\boxed{\mathrm{F}}$, the stages of computation of an algorithm.
- $\sigma[0,1) \leftrightarrows \tau[0,1)$ for all $\sigma, \tau \in \mathrm{F}$.

$$(\![[|\mathtt{B}]]\!) \leftrightarrows (\![[|\mathtt{1}]]\!) \leftrightarrows (\![[|\mathtt{1B1}]]\!) \leftrightarrows \ldots$$

Control Equivalence

- Assume we're given a set of tests $\mathfrak{q}$ and a set of operations $\mathfrak{p}$: the *actions* we can use.
- Define an equivalence relation $\leftrightarrows$ on $\boxed{\mathrm{F}}$, the stages of computation of an algorithm.
- $\sigma[0,1) \leftrightarrows \tau[0,1)$ for all $\sigma, \tau \in \mathrm{F}$.
- If $\sigma \leftrightarrows \tau$ then "the same thing" should happen at each.
  - How to capture this?

Control Equivalence (cont.)

Take an equivalence class C under $\leftrightarrows$.

Control Equivalence (cont.)

Take an equivalence class C under $\leftrightarrows$.

- There is a *construction set* of guarded operations $\mathfrak{F}_C \subseteq \mathfrak{q} \times \mathfrak{p}$ such that
  1. Every guarded operation in $\mathfrak{F}_C$ gets applied to every $\sigma \in C$; *and*
  2. If a guarded operation is successfully applied to σ and τ, the results are equivalent under $\leftrightarrows$.

Control Equivalence (cont.)

Take an equivalence class C under $\leftrightarrows$.

- There is a *construction set* of guarded operations $\mathfrak{F}_C \subseteq \mathfrak{q} \times \mathfrak{p}$ such that
  1. Every guarded operation in $\mathfrak{F}_C$ gets applied to every $\sigma \in C$; *and*
  2. If a guarded operation is successfully applied to σ and τ, the results are equivalent under $\leftrightarrows$.

δ	1	B
s_0	$\langle 1, \mathsf{R}, s_0 \rangle$	$\langle 1, \mathsf{R}, s_1 \rangle$
s_1	$\langle 1, \mathsf{R}, s_1 \rangle$	$\langle \mathtt{B}, \mathsf{L}, s_2 \rangle$
s_2	$\langle \mathtt{B}, \mathsf{L}, s_3 \rangle$	—
s_3	—	—

$$\mathfrak{F}_C = \begin{cases} \{\langle \mathrm{TT}_1, \mathrm{TM}_{1,\mathsf{R}} \rangle, \langle \mathrm{TT}_\mathtt{B}, \mathrm{TM}_{1,\mathsf{R}} \rangle\} & s = s_0 \\ \{\langle \mathrm{TT}_1, \mathrm{TM}_{1,\mathsf{R}} \rangle, \langle \mathrm{TT}_\mathtt{B}, \mathrm{TM}_{\mathtt{B},\mathsf{L}} \rangle\} & s = s_1 \\ \{\langle \mathrm{TT}_1, \mathrm{TM}_{\mathtt{B},\mathsf{L}} \rangle\} & s = s_2 \\ \emptyset & s = s_3. \end{cases}$$

$$(\![\,[|1\mathtt{B}11], [1|\mathtt{B}11], [11|11]\,]\!) \leftrightarrows (\![\,[|1\mathtt{B}11], [1|\mathtt{B}11], [11|11], [111|1], [1111|\mathtt{B}]\,]\!) \qquad (s_1)$$

Control Equivalence (cont.)

Take an equivalence class C under $\leftrightarrows$.

- There is a *construction set* of guarded operations $\mathfrak{F}_C \subseteq \mathfrak{q} \times \mathfrak{p}$ such that
  1. Every guarded operation in $\mathfrak{F}_C$ gets applied to every $\sigma \in C$; *and*
  2. If a guarded operation is successfully applied to σ and τ, the results are equivalent under $\leftrightarrows$.

δ	1	B
s_0	$\langle 1, \mathsf{R}, s_0 \rangle$	$\langle 1, \mathsf{R}, s_1 \rangle$
s_1	$\langle 1, \mathsf{R}, s_1 \rangle$	$\langle \mathsf{B}, \mathsf{L}, s_2 \rangle$
s_2	$\langle \mathsf{B}, \mathsf{L}, s_3 \rangle$	—
s_3	—	—

$$\mathfrak{F}_C = \begin{cases} \{\langle \mathrm{TT}_1, \mathrm{TM}_{1,\mathsf{R}} \rangle, \langle \mathrm{TT}_\mathrm{B}, \mathrm{TM}_{1,\mathsf{R}} \rangle\} & s = s_0 \\ \{\langle \mathrm{TT}_1, \mathrm{TM}_{1,\mathsf{R}} \rangle, \langle \mathrm{TT}_\mathrm{B}, \mathrm{TM}_{\mathsf{B},\mathsf{L}} \rangle\} & s = s_1 \\ \{\langle \mathrm{TT}_1, \mathrm{TM}_{\mathsf{B},\mathsf{L}} \rangle\} & s = s_2 \\ \emptyset & s = s_3. \end{cases}$$

$$(\!|[|1B11], [1|B11], [11|11]|\!) \leftrightarrows (\!|[|1B11], [1|B11], [11|11], [111|1], [1111|B]|\!) \qquad (s_1)$$

$$(\!|[|1B11], [1|B11], [11|11], [111|1]|\!)$$

Control Equivalence (cont.)

δ	1	B
s_0	$\langle 1, \mathsf{R}, s_0\rangle$	$\langle 1, \mathsf{R}, s_1\rangle$
s_1	$\langle 1, \mathsf{R}, s_1\rangle$	$\langle \mathtt{B}, \mathsf{L}, s_2\rangle$
s_2	$\langle \mathtt{B}, \mathsf{L}, s_3\rangle$	—
s_3	—	—

Take an equivalence class C under $\leftrightarrows$.

- There is a *construction set* of guarded operations $\mathfrak{F}_C \subseteq \mathfrak{q} \times \mathfrak{p}$ such that
  1. Every guarded operation in $\mathfrak{F}_C$ gets applied to every $\sigma \in C$; *and*
  2. If a guarded operation is successfully applied to σ and τ, the results are equivalent under $\leftrightarrows$.

$$\mathfrak{F}_C = \begin{cases} \{\langle \mathrm{TT}_1, \mathrm{TM}_{1,\mathsf{R}}\rangle, \langle \mathrm{TT}_{\mathrm{B}}, \mathrm{TM}_{1,\mathsf{R}}\rangle\} & s = s_0 \\ \{\langle \mathrm{TT}_1, \mathrm{TM}_{1,\mathsf{R}}\rangle, \langle \mathrm{TT}_{\mathrm{B}}, \mathrm{TM}_{\mathsf{B},\mathsf{L}}\rangle\} & s = s_1 \\ \{\langle \mathrm{TT}_1, \mathrm{TM}_{\mathsf{B},\mathsf{L}}\rangle\} & s = s_2 \\ \emptyset & s = s_3. \end{cases}$$

$$(\!|[|\mathtt{1B11}], [\mathtt{1}|\mathtt{B11}],[\mathtt{11}|\mathtt{11}]|\!) \leftrightarrows (\!|[|\mathtt{1B11}], [\mathtt{1}|\mathtt{B11}],[\mathtt{11}|\mathtt{11}],[\mathtt{111}|\mathtt{1}],[\mathtt{1111}|\mathtt{B}]|\!) \qquad (s_1)$$

$$(\!|[|\mathtt{1B11}], [\mathtt{1}|\mathtt{B11}],[\mathtt{11}|\mathtt{11}],[\mathtt{111}|\mathtt{1}]|\!) \qquad (\!|[|\mathtt{1B11}], [\mathtt{1}|\mathtt{B11}],[\mathtt{11}|\mathtt{11}],[\mathtt{111}|\mathtt{1}],[\mathtt{1111}|\mathtt{B}],[\mathtt{111}|\mathtt{1B}]|\!)$$

Control Equivalence (cont.)

Take an equivalence class C under $\leftrightarrows$.

- There is a *construction set* of guarded operations $\mathfrak{F}_C \subseteq \mathfrak{q} \times \mathfrak{p}$ such that
  1. Every guarded operation in $\mathfrak{F}_C$ gets applied to every $\sigma \in C$; *and*
  2. If a guarded operation is successfully applied to σ and τ, the results are equivalent under $\leftrightarrows$.
- If $\sigma \in C \cap \mathrm{F}$, there must be a test (to show it halts).

Control Equivalence (cont.)

Take an equivalence class C under $\leftrightarrows$.

- There is a *construction set* of guarded operations $\mathfrak{F}_C \subseteq \mathfrak{q} \times \mathfrak{p}$ such that
  1. Every guarded operation in $\mathfrak{F}_C$ gets applied to every $\sigma \in C$; *and*
  2. If a guarded operation is successfully applied to σ and τ, the results are equivalent under $\leftrightarrows$.
- If $\sigma \in C \cap \mathrm{F}$, there must be a test (to show it halts).

δ	1	B
s_0	$\langle 1, \mathsf{R}, s_0 \rangle$	$\langle 1, \mathsf{R}, s_1 \rangle$
s_1	$\langle 1, \mathsf{R}, s_1 \rangle$	$\langle \mathrm{B}, \mathsf{L}, s_2 \rangle$
s_2	$\langle \mathrm{B}, \mathsf{L}, s_3 \rangle$	—
s_3	—	—

$$\mathfrak{G}_C = \begin{cases} \emptyset & s = s_0 \\ \emptyset & s = s_1 \\ \{\mathrm{TT_B}\} & s = s_2 \\ \{\mathrm{TT_1}, \mathrm{TT_B}\} & s = s_3. \end{cases}$$

$$(\!|[|1\mathrm{B}11], [1|\mathrm{B}11], [11|11], [111|1], [1111|\mathrm{B}], [111|1\mathrm{B}], [11|1\mathrm{BB}]|\!) \qquad (s_3)$$

Control Equivalence

Definition (Control Equivalence)

Let $\mathfrak{q}$ be a set of tests and $\mathfrak{p}$ of operations. Let F be a task. A *control equivalence* $\leftrightarrows$ on F under $\mathfrak{q}$ and $\mathfrak{p}$ is an equivalence relation on $\boxed{\mathrm{F}}$ satisfying:

Starting State $\sigma[0,1) \leftrightarrows \tau[0,1)$ for all $\sigma, \tau \in \mathrm{F}$.

Construction For each $C \in \boxed{\mathrm{F}}/\leftrightarrows$ there is a *construction set* $\mathfrak{F}_C \subseteq \mathfrak{q} \times \mathfrak{p}$ such that both:

- **Composition** $\{\sigma \in \boxed{\mathrm{F}} \mid \sigma[0,-1) \in C\} = \bigcup_{\langle \mathrm{Q},\mathrm{P}\rangle \in \mathfrak{F}_C} C \circ \mathrm{Q} \circ \mathrm{P}$; *and*
- **Consistency** if $\langle \mathrm{Q}, \mathrm{P}\rangle \in \mathfrak{F}_C$ then $C \circ \mathrm{Q} \circ \mathrm{P} \subseteq D$ for some $D \in \boxed{\mathrm{F}}/\leftrightarrows$.

Halting For each $C \in \boxed{\mathrm{F}}/\leftrightarrows$ there is a *halting set* $\mathfrak{G}_C \subseteq \mathfrak{q}$ such that $C \cap \mathrm{F} = \bigcup_{\mathrm{Q} \in \mathfrak{G}_C} C \circ \mathrm{Q}$.

Definition (Trace set)

A *trace set* for $\mathfrak{q}$ and $\mathfrak{p}$ is a pair $\mathrm{A} = (\mathrm{F}, \leftrightarrows)$, where F is a task and $\leftrightarrows$ is a control equivalence on F under $\mathfrak{q}$ and $\mathfrak{p}$.

Finite Control Computability

Proposition

For every task F *over* $\mathfrak{D}$ *there is a test set* $\mathfrak{e}$*, an operation set* $\mathfrak{w}$ *and a control equivalence* $\leftrightarrows$ *such that* $\mathrm{A} = (\mathrm{F}, \leftrightarrows)$ *is a trace set for* $\mathfrak{e}$ *and* $\mathfrak{w}$*.*

Finite Control Computability

Proposition

For every task F *over* $\mathfrak{D}$ *there is a test set* $\mathfrak{e}$*, an operation set* $\mathfrak{w}$ *and a control equivalence* $\leftrightarrows$ *such that* $\mathrm{A} = (\mathrm{F}, \leftrightarrows)$ *is a trace set for* $\mathfrak{e}$ *and* $\mathfrak{w}$*.*

Definition (Finite control computable)

A trace set A for $\mathfrak{q}$ and $\mathfrak{p}$ is *finite control computable* if $\boxed{\mathrm{A}}/{\leftrightarrows_{\mathrm{A}}}$ is finite, $\mathfrak{q}$ and $\mathfrak{p}$ are finite, and every $\mathrm{Q} \in \mathfrak{q}$ and $\mathrm{P} \in \mathfrak{p}$ is computable.

Finite Control Computability

Proposition

For every task $\textsc{f}$ *over* $\mathfrak{D}$ *there is a test set* $\mathfrak{e}$*, an operation set* $\mathfrak{w}$ *and a control equivalence* $\leftrightarrows$ *such that* $\textsc{a} = (\textsc{f}, \leftrightarrows)$ *is a trace set for* $\mathfrak{e}$ *and* $\mathfrak{w}$*.*

Definition (Finite control computable)

A trace set $\textsc{a}$ for $\mathfrak{q}$ and $\mathfrak{p}$ is *finite control computable* if $\boxed{\textsc{a}}/\leftrightarrows_{\textsc{a}}$ is finite, $\mathfrak{q}$ and $\mathfrak{p}$ are finite, and every $\textsc{q} \in \mathfrak{q}$ and $\textsc{p} \in \mathfrak{p}$ is computable.

Theorem

If $\mathfrak{M} = \langle S, \Gamma, \delta, s_0 \rangle$ *is a Turing machine then* $\textsc{Run}_{\mathfrak{M}}$ *is finite control computable.*

Does finite control computable imply Turing computable?

Does finite control computable imply Turing computable?

Is every FCC trace set the trace set of some Turing machine?

No!

1. FCC allows arbitrary computable tests and operations
2. FCC allows more domains than machine tapes

Does finite control computable imply Turing computable?

Is every FCC trace set the trace set of some Turing machine?

No!

1. FCC allows arbitrary computable tests and operations
2. FCC allows more domains than machine tapes

Theorem

Let $\mathrm{A} = (\mathrm{F}, \leftrightarrows)$ *be a trace set for* $\mathfrak{t}$ *and* $\mathfrak{m}$ *and* $\Gamma \subseteq \mathbb{G}$ *a finite alphabet such that*

1. *For every* $\sigma \in \mathrm{A}$ *and* $\alpha < |\sigma|$, $\sigma[\alpha] = [\mathrm{u}|\mathrm{v}]$ *for some* $\mathrm{u}, \mathrm{v} \in \Gamma^*$*; and*
2. *For every* $\mathrm{u}, \mathrm{v} \in \Gamma^*$ *there is a unique* $\sigma \in \mathrm{A}$ *with* $\sigma[0] = [\mathrm{u}|\mathrm{v}]$*; and*
3. A *is fully deterministic (*$\sigma[0] = \tau[0]$ *implies* $\sigma = \tau$*).*

Then there is a Turing machine $\mathfrak{M} = \langle S, \Gamma, \delta, s_0 \rangle$ *such that* $\mathrm{RUN}_{\mathfrak{M}} = \mathrm{A}$ *iff* A *is finite control computable.*

Does finite control computable imply Turing computable?

Is every FCC trace set the trace set of some Turing machine?

No!

1. FCC allows arbitrary computable tests and operations
2. FCC allows more domains than machine tapes

Theorem

Let $\mathrm{A} = (\mathrm{F}, \leftrightarrows)$ *be a trace set for* $\mathfrak{t}$ *and* $\mathfrak{m}$ *and* $\Gamma \subseteq \mathbb{G}$ *a finite alphabet such that*

1. *For every* $\sigma \in \mathrm{A}$ *and* $\alpha < |\sigma|$, $\sigma[\alpha] = [\mathrm{u}|\mathrm{v}]$ *for some* $\mathrm{u}, \mathrm{v} \in \Gamma^*$*; and*
2. *For every* $\mathrm{u}, \mathrm{v} \in \Gamma^*$ *there is a unique* $\sigma \in \mathrm{A}$ *with* $\sigma[0] = [\mathrm{u}|\mathrm{v}]$*; and*
3. A *is fully deterministic (*$\sigma[0] = \tau[0]$ *implies* $\sigma = \tau$*).*

Then there is a Turing machine $\mathfrak{M} = \langle S, \Gamma, \delta, s_0 \rangle$ *such that* $\mathrm{RUN}_{\mathfrak{M}} = \mathrm{A}$ *iff* A *is finite control computable.*

Can we do better? Can we allow arbitrary computable tests and operations?

Expansion mapping

$$\llparenthesis [|\mathtt{1B11}], [\mathtt{11}|\mathtt{11}], [\mathtt{111}|\mathtt{1B}], [\mathtt{11}|\mathtt{1BB}] \rrparenthesis$$

Expansion mapping

$$(\!([\,|\texttt{1B11}],\,[\texttt{11}|\texttt{11}],[\texttt{111}|\texttt{1B}],[\texttt{11}|\texttt{1BB}])\!)$$

No Turing Machine can do this.

Expansion mapping

$$\llparenthesis[|1B11], [11|11], [111|1B], [11|1BB]\rrparenthesis$$

No Turing Machine can do this. But $\mathfrak{M}$ can fill in the gaps:

$$\llparenthesis{\color{red}[|1B11]}, [1|B11], {\color{red}[11|11],} [111|1], [1111|B], {\color{red}[111|1B], [11|1BB]}\rrparenthesis$$

Expansion mapping

$$(\![\,|1B11], [11|11], [111|1B], [11|1BB]\,]\!)$$

No Turing Machine can do this. But $\mathfrak{M}$ can fill in the gaps:

$$(\![\,|1B11], [1|B11], [11|11], [111|1], [1111|B], [111|1B], [11|1BB]\,]\!)$$

Define a function h mapping prefixes of the original sequence to prefixes of the expanded sequence.

$$h((\![\,|1B11]\!)) = (\![\,|1B11]\!)$$
$$h((\![\,|1B11], [11|11]\!)) = (\![\,|1B11], [1|B11], [11|11]\!)$$
$$h((\![\,|1B11], [11|11], [111|1B]\!)) = (\![\,|1B11], \ldots [111|1B]\!)$$
$$\vdots$$

Expansion mapping

$$(\!|[|1B11], [11|11],[111|1B],[11|1BB]|\!)$$

No Turing Machine can do this. But $\mathfrak{M}$ can fill in the gaps:

$$(\!|[|1B11], [1|B11],[11|11],[111|1],[1111|B],[111|1B],[11|1BB]|\!)$$

Define a function h mapping prefixes of the original sequence to prefixes of the expanded sequence.

$$h((\!|[|1B11]|\!)) = (\!|[|1B11]|\!)$$
$$h((\!|[|1B11], [11|11]|\!)) = (\!|[|1B11], [1|B11],[11|11]|\!)$$
$$h((\!|[|1B11], [11|11],[111|1B]|\!)) = (\!|[|1B11], \ldots [111|1B]|\!)$$
$$\vdots$$

Expansion Mapping

$\mathrm{F} \ll \mathrm{G}$ if there is an injective $h : \boxed{\mathrm{F}} \to \boxed{\mathrm{G}}$ such that σ is a subsequence of $h(\sigma)$ and $h(\mathrm{F}) = \mathrm{G}$.

Expansion mapping is not enough

Example

Take $\textsc{Primes}$, enumerating all prime in increasing order:

$$\{ (\!| [\mathtt{11}|\mathtt{B}], [\mathtt{111}|\mathtt{B}], [\mathtt{11111}|\mathtt{B}], [\mathtt{1111111}|\mathtt{B}], \ldots |\!) \}$$

Take $\mathfrak{M} = \langle \{s\}, \{1, B\}, \delta, s \rangle$ with $\delta(s, 1) = \delta(s, \mathtt{B}) = \langle 1, \mathsf{R}, s \rangle$.

$$\{ (\!| [|\mathtt{B}], [\mathtt{1}|\mathtt{B}], [\mathtt{11}|\mathtt{B}], [\mathtt{111}|\mathtt{B}], [\mathtt{1111}|\mathtt{B}], \ldots |\!) \}$$

Carrying out

Two requirements:

1. We need to be able to recover the original task from the expanded task.
2. If $\mathfrak{M}$ carries out a task, that task needs to be computable!

Carrying out

Two requirements:

1. We need to be able to recover the original task from the expanded task.
2. If $\mathfrak{M}$ carries out a task, that task needs to be computable!

Definition (Carrying out)

We say A is *carried out* by B if there is an expansion mapping h from A to B such that $C \in \boxed{\mathrm{A}}/\leftrightarrows_{\mathrm{A}}$ iff $h(C) \in \boxed{\mathrm{B}}/\leftrightarrows_{\mathrm{B}}$.

Theorem

Let $\mathrm{A} = (\mathrm{F}, \leftrightarrows)$ *be a trace set for* $\mathfrak{q}$ *and* $\mathfrak{p}$ *such that* A *is finite control and fully deterministic;* $\mathfrak{q}$ *and* $\mathfrak{p}$ *are both finite and Turing computable; there is a finite alphabet* Γ *such that*

1. *for every* $\sigma \in \mathrm{A}$, $\sigma[0] = [\mathrm{u}|\mathrm{v}]$ *for some* $\mathrm{u} \in \Gamma^*, \mathrm{v} \in \Gamma^+$*;*
2. *for every* $\mathrm{u} \in \Gamma^*, \mathrm{v} \in \Gamma^+$ *there exists* $\sigma \in \mathrm{A}$ *with* $\sigma[0] = [\mathrm{u}|\mathrm{v}]$*.*

Then there is a Turing machine $\mathfrak{M}$ *such that* $\mathrm{Run}_{\mathfrak{M}}$ *carries out* A*.*

FCC and Turing Computability

Theorem

Let $\mathrm{A} = (\mathrm{F}, \leftrightarrows)$ *be a trace set for* $\mathfrak{q}$ *and* $\mathfrak{p}$ *such that* A *is finite control and fully deterministic;* $\mathfrak{q}$ *and* $\mathfrak{p}$ *are both finite and Turing computable; there is a finite alphabet* Γ *such that*

1. *for every* $\sigma \in \mathrm{A}$, $\sigma[0] = [\mathrm{u}|\mathrm{v}]$ *for some* $\mathrm{u} \in \Gamma^*, \mathrm{v} \in \Gamma^+$;
2. *for every* $\mathrm{u} \in \Gamma^*, \mathrm{v} \in \Gamma^+$ *there exists* $\sigma \in \mathrm{A}$ *with* $\sigma[0] = [\mathrm{u}|\mathrm{v}]$.

Then there is a Turing machine $\mathfrak{M}$ *such that* $\mathrm{RUN}_{\mathfrak{M}}$ *carries out* A.

Theorem

Let A *be a trace set and* $\mathfrak{M} = \langle S, \Gamma, \delta, s_0 \rangle$ *be a Turing machine. If* $\mathrm{RUN}_{\mathfrak{M}}$ *carries out* A *then* A *is finite control computable.*

Outline

1. What are algorithms?
2. Computable Execution Traces
   - Finite control computability
   - Carrying out trace sets
3. Algorithms as trace sets

Algorithms as trace sets

Idea: Model algorithms as trace sets.

- "Functionalism about functions"

Algorithms as trace sets

Idea: Model algorithms as trace sets.

- "Functionalism about functions"
- An extensional account of algorithms that doesn't reduce to functional equivalence.

Algorithms as trace sets

Idea: Model algorithms as trace sets.

- "Functionalism about functions"
- An extensional account of algorithms that doesn't reduce to functional equivalence.
- Avoid commitments to computability, allowing for arbitrary basic operations and non-terminating algorithms.

Algorithms as trace sets

Idea: Model algorithms as trace sets.

- "Functionalism about functions"
- An extensional account of algorithms that doesn't reduce to functional equivalence.
- Avoid commitments to computability, allowing for arbitrary basic operations and non-terminating algorithms.

Extant accounts of algorithms

- What model/what objects?

Trace set account

- Allow arbitrary domains and operations

Algorithms as trace sets

Idea: Model algorithms as trace sets.

- "Functionalism about functions"
- An extensional account of algorithms that doesn't reduce to functional equivalence.
- Avoid commitments to computability, allowing for arbitrary basic operations and non-terminating algorithms.

Extant accounts of algorithms

- What model/what objects?
- Hard to get ambiguity, under-determinism

Trace set account

- Allow arbitrary domains and operations
- No requirement for full determinism

Algorithms as trace sets

Idea: Model algorithms as trace sets.

- "Functionalism about functions"
- An extensional account of algorithms that doesn't reduce to functional equivalence.
- Avoid commitments to computability, allowing for arbitrary basic operations and non-terminating algorithms.

Extant accounts of algorithms

- What model/what objects?
- Hard to get ambiguity, under-determinism
- Language dependence

Trace set account

- Allow arbitrary domains and operations
- No requirement for full determinism
- Purely semantic

Algorithms as trace sets

Idea: Model algorithms as trace sets.

- "Functionalism about functions"
- An extensional account of algorithms that doesn't reduce to functional equivalence.
- Avoid commitments to computability, allowing for arbitrary basic operations and non-terminating algorithms.

Extant accounts of algorithms

- What model/what objects?
- Hard to get ambiguity, under-determinism
- Language dependence
- A set of instructions

Trace set account

- Allow arbitrary domains and operations
- No requirement for full determinism
- Purely semantic
- A set of behaviours

Algorithms as trace sets: further details

- Algorithms are always presented as a (potential) solution to some problem, in a particular context.
  - A problem in context provides a test set $\mathfrak{q}$ and operation set $\mathfrak{p}$.
  - This provides the level of abstraction for the algorithm.

Algorithms as trace sets: further details

- Algorithms are always presented as a (potential) solution to some problem, in a particular context.
  - A problem in context provides a test set $\mathfrak{q}$ and operation set $\mathfrak{p}$.
  - This provides the level of abstraction for the algorithm.
- (Sequential) algorithms specify a sequence of steps from inputs, possibly to outputs.
  - This is precisely what a task does.
  - Model an algorithm not as instructions, but as what it *does*. cf. functions.

Algorithms as trace sets: further details

- Algorithms are always presented as a (potential) solution to some problem, in a particular context.
    - A problem in context provides a test set $\mathfrak{q}$ and operation set $\mathfrak{p}$.
    - This provides the level of abstraction for the algorithm.
- (Sequential) algorithms specify a sequence of steps from inputs, possibly to outputs.
    - This is precisely what a task does.
    - Model an algorithm not as instructions, but as what it *does*. cf. functions.
- Algorithms are generally taken to be semantic, not syntactic objects.
    - What this means depends on your philosophical persuasions.
    - Tasks are sets of sequences of mathematical objects - whatever they are, and whatever that means.
    - Tasks avoid issues around equivalence of control flow structures.

Algorithms as trace sets: further details

- Algorithms are always presented as a (potential) solution to some problem, in a particular context.
  - A problem in context provides a test set $\mathfrak{q}$ and operation set $\mathfrak{p}$.
  - This provides the level of abstraction for the algorithm.
- (Sequential) algorithms specify a sequence of steps from inputs, possibly to outputs.
  - This is precisely what a task does.
  - Model an algorithm not as instructions, but as what it *does*. cf. functions.
- Algorithms are generally taken to be semantic, not syntactic objects.
  - What this means depends on your philosophical persuasions.
  - Tasks are sets of sequences of mathematical objects - whatever they are, and whatever that means.
  - Tasks avoid issues around equivalence of control flow structures.
- Algorithms should be specifiable via finitary means.
  - Finite control ensures this for trace sets.

Algorithms as trace sets: further details

- Algorithms are always presented as a (potential) solution to some problem, in a particular context.
  - A problem in context provides a test set $\mathfrak{q}$ and operation set $\mathfrak{p}$.
  - This provides the level of abstraction for the algorithm.
- (Sequential) algorithms specify a sequence of steps from inputs, possibly to outputs.
  - This is precisely what a task does.
  - Model an algorithm not as instructions, but as what it *does*. cf. functions.
- Algorithms are generally taken to be semantic, not syntactic objects.
  - What this means depends on your philosophical persuasions.
  - Tasks are sets of sequences of mathematical objects - whatever they are, and whatever that means.
  - Tasks avoid issues around equivalence of control flow structures.
- Algorithms should be specifiable via finitary means.
  - Finite control ensures this for trace sets.
- Algorithms are ambiguous!

What else?

What else?

1. Ask me for details about:
   - Encoding: changing the domain over which a trace set is defined.
   - Resolution: removing ambiguity from a trace set; introducing tie-breakers.
   - Implementation: increasing the level of specification; the relationship between programs and algorithms.
   - Computable functions vs. algorithms.

Thank you!

What else?

1. Ask me for details about:
   - Encoding: changing the domain over which a trace set is defined.
   - Resolution: removing ambiguity from a trace set; introducing tie-breakers.
   - Implementation: increasing the level of specification; the relationship between programs and algorithms.
   - Computable functions vs. algorithms.
2. Ask me to ramble about:
   - Trace sets with arbitrary tasks (not just tests and operations)
   - Recursive algorithms
   - Concurrent/parallel computation
   - Interactive algorithms
   - Transfinite sequences
   - Probabilistic algorithms
   - Complexity theory

Thank you!

Thank you!

[1] Markus Bläser. “Metric TSP”. In: *Encyclopedia of Algorithms*. Ed. by Ming-Yang Kao. Boston, MA: Springer US, 2008, pp. 517–519. ISBN: 978-0-387-30162-4.

[2] Thomas H. Cormen et al., eds. *Introduction to Algorithms*. 3rd ed. Cambridge, Mass: MIT Press, 2009. 1292 pp. ISBN: 978-0-262-03384-8 978-0-262-53305-8.

[3] Rodney G. Downey and Denis R. Hirschfeldt. *Algorithmic Randomness and Complexity*. Theory and Applications of Computability. New York, NY: Springer New York, 2010. ISBN: 978-0-387-95567-4 978-0-387-68441-3.

[4] Yuri Gurevich. “Sequential Abstract-State Machines Capture Sequential Algorithms”. In: *ACM Transactions on Computational Logic* 1.1 (1 July 2000), pp. 77–111.

[5] Hartmut G. M. Huber. “Algorithm and Formula”. In: *Communications of the ACM* 9.9 (1966), pp. 653–654.

[6] Bakhadyr Khoussainov and Nodira Khoussainova. *Lectures on Discrete Mathematics for Computer Science*. Algebra and Discrete Mathematics v. 3. New Jersey: World Scientific, 2012. 346 pp. ISBN: 978-981-4340-50-2.

[7] Jon. Kleinberg and Éva. Tardos. *Algorithm Design*. Boston: Pearson/Addison-Wesley, 2006. 838 pp. ISBN: 978-0-321-29535-4.

[8] Jr Lomonaco. *Shor's Quantum Factoring Algorithm*. 8 Oct. 2000. arXiv: quant-ph/0010034. URL: http://arxiv.org/abs/quant-ph/0010034 (visited on 17/02/2020).

[9] Robert Sedgewick and Kevin Daniel Wayne. *Algorithms*. 4th ed. Upper Saddle River, NJ: Addison-Wesley, 2011. 955 pp. ISBN: 978-0-321-57351-3.