Probabilistic automatic complexity

Kenneth Gill

2024 ASL Annual Meeting
Iowa State University, Ames, IA

5/16/2024

Algorithmic complexity of finite strings

“What’s the smallest possible machine you can build to encode a string x?”

Algorithmic complexity of finite strings

"What's the smallest possible machine you can build to encode a string x?"

- **Kolmogorov complexity**, $K(x)$: smallest size of a Turing machine which outputs x (Kolmogorov, Solomonoff '60s)

Algorithmic complexity of finite strings

"What's the smallest possible machine you can build to encode a string x?"

- **Kolmogorov complexity**, $K(x)$: smallest size of a Turing machine which outputs x (Kolmogorov, Solomonoff '60s)
  Drawbacks: Not computable. Defined up to an additive constant.

Algorithmic complexity of finite strings

"What's the smallest possible machine you can build to encode a string x?"

- **Kolmogorov complexity**, $K(x)$: smallest size of a Turing machine which outputs x (Kolmogorov, Solomonoff '60s)
  Drawbacks: Not computable. Defined up to an additive constant.
- Smallest size of a CFG which generates $\{x\}$ (Diwan '86)
- **Finite state complexity**: analogue of $K(x)$ using finite transducers (Calude-Salomaa-Roblot '11)

Algorithmic complexity of finite strings

"What's the smallest possible machine you can build to encode a string x?"

- **Kolmogorov complexity**, $K(x)$: smallest size of a Turing machine which outputs x (Kolmogorov, Solomonoff '60s)
  Drawbacks: Not computable. Defined up to an additive constant.
- Smallest size of a CFG which generates $\{x\}$ (Diwan '86)
- **Finite state complexity**: analogue of $K(x)$ using finite transducers (Calude-Salomaa-Roblot '11)
- **DFA complexity**, $A_D(x)$: least number of states of a DFA uniquely accepting x among strings of length $|x|$ (Shallit-Wang '01)

Algorithmic complexity of finite strings

"What's the smallest possible machine you can build to encode a string x?"

- **Kolmogorov complexity**, $K(x)$: smallest size of a Turing machine which outputs x (Kolmogorov, Solomonoff '60s) Drawbacks: Not computable. Defined up to an additive constant.
- Smallest size of a CFG which generates $\{x\}$ (Diwan '86)
- **Finite state complexity**: analogue of $K(x)$ using finite transducers (Calude-Salomaa-Roblot '11)
- **DFA complexity**, $A_D(x)$: least number of states of a DFA uniquely accepting x among strings of length $|x|$ (Shallit-Wang '01)
- **NFA complexity**, $A_N(x)$: least number of states of an NFA accepting x and having a unique accepting path of length $|x|$ (Hyde '13)

Probabilistic finite-state automata (PFAs)

The **PFA complexity** of x, $A_P(x)$, is the least number of states of a PFA accepting x with unique highest probability among strings of length $|x|$. (G. '22)

Probabilistic finite-state automata (PFAs)

The **PFA complexity** of x, $A_P(x)$, is the least number of states of a PFA accepting x with unique highest probability among strings of length $|x|$. (G. '22)

A **PFA** (Rabin '63) is a generalization of a DFA which

- has finitely many states, with an initial prob. dist. $\vec{\pi}$ on states and a list $\vec{\eta}$ of accepting states;
- reads an input word sequentially starting in an initial state;

Probabilistic finite-state automata (PFAs)

The **PFA complexity** of x, $A_P(x)$, is the least number of states of a PFA accepting x with unique highest probability among strings of length $|x|$. (G. '22)

A **PFA** (Rabin '63) is a generalization of a DFA which

- has finitely many states, with an initial prob. dist. $\vec{\pi}$ on states and a list $\vec{\eta}$ of accepting states;
- reads an input word sequentially starting in an initial state;
- changes states probabilistically. Formally given by stochastic matrices P_σ with $(P_\sigma)_{ij} =$ prob. of moving from state s_i to state s_j when reading letter σ;

Probabilistic finite-state automata (PFAs)

The **PFA complexity** of x, $A_P(x)$, is the least number of states of a PFA accepting x with unique highest probability among strings of length $|x|$. (G. '22)

A **PFA** (Rabin '63) is a generalization of a DFA which

- has finitely many states, with an initial prob. dist. $\vec{\pi}$ on states and a list $\vec{\eta}$ of accepting states;
- reads an input word sequentially starting in an initial state;
- changes states probabilistically. Formally given by stochastic matrices P_σ with $(P_\sigma)_{ij}$ = prob. of moving from state s_i to state s_j when reading letter σ;
- assigns an **acceptance probability** $\rho(x)$ to each word x, i.e., the prob. of ending in an accepting state after reading x.

PFA complexity

The **gap function** of the PFA M is

$$\mathrm{gap}_M(x) = \min\{\, \rho_M(x) - \rho_M(y) : |y| = |x| \text{ and } y \neq x \,\}.$$

The **PFA complexity** of x is the least number $A_P(x)$ of states of an M with $\mathrm{gap}_M(x) > 0$.

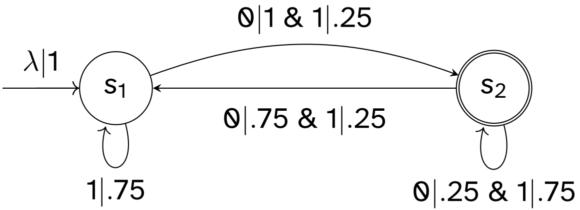
PFA complexity

The **gap function** of the PFA M is

$$\mathrm{gap}_M(x) = \min\{\, \rho_M(x) - \rho_M(y) : |y| = |x| \text{ and } y \neq x \,\}.$$

The **PFA complexity** of x is the least number $A_P(x)$ of states of an M with $\mathrm{gap}_M(x) > 0$.

Example: the NFA complexity of $x = 0^6 1^8 0$ is 8, but $A_P(x) = 2$ via



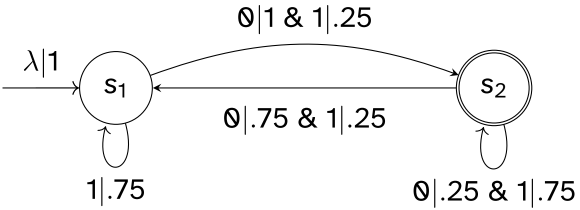
PFA complexity

The **gap function** of the PFA M is

$$\mathrm{gap}_M(x) = \min\{\, \rho_M(x) - \rho_M(y) : |y| = |x| \text{ and } y \neq x \,\}.$$

The **PFA complexity** of x is the least number $A_P(x)$ of states of an M with $\mathrm{gap}_M(x) > 0$.

Example: the NFA complexity of $x = 0^6 1^8 0$ is 8, but $A_P(x) = 2$ via



For this PFA, $\rho(x) \approx 0.625$ with $\mathrm{gap}(x) \approx 4 \times 10^{-6}$ (!).

PFA complexity with required gap

The **PFA complexity with gap** δ is the least number $A_{P,\delta}(x)$ of states of a PFA having gap$(x) > \delta$, where $\delta \in [0, 1)$ is a real-valued parameter. (G. '22)

PFA complexity with required gap

The **PFA complexity with gap δ** is the least number $A_{P,\delta}(x)$ of states of a PFA having $\text{gap}(x) > \delta$, where $\delta \in [0, 1)$ is a real-valued parameter. (G. '22)

$$A_P(x) = A_{P,0}(x) \leq A_{P,\delta}(x) \leq A_D(x) \qquad \text{for all } x, \delta,$$

where A_D = DFA complexity. Note $A_{P,\delta}(x)$ is increasing in δ.

PFA complexity with required gap

The **PFA complexity with gap** δ is the least number $A_{P,\delta}(x)$ of states of a PFA having gap$(x) > \delta$, where $\delta \in [0, 1)$ is a real-valued parameter. (G. '22)

$$A_P(x) = A_{P,0}(x) \leq A_{P,\delta}(x) \leq A_D(x) \qquad \text{for all } x, \delta,$$

where $A_D =$ DFA complexity. Note $A_{P,\delta}(x)$ is increasing in δ.

$$A_P(x) \leq A_N(x) + 1 \qquad \text{for all } x,$$

where $A_N =$ NFA complexity.

PFA complexity with required gap

The **PFA complexity with gap** δ is the least number $A_{P,\delta}(x)$ of states of a PFA having $\text{gap}(x) > \delta$, where $\delta \in [0, 1)$ is a real-valued parameter. (G. '22)

$$A_P(x) = A_{P,0}(x) \le A_{P,\delta}(x) \le A_D(x) \qquad \text{for all } x, \delta,$$

where A_D = DFA complexity. Note $A_{P,\delta}(x)$ is increasing in δ.

$$A_P(x) \le A_N(x) + 1 \qquad \text{for all } x,$$

where A_N = NFA complexity.
Not a tight bound! Only met by constant strings so far.
In fact $A_P(x) \le 3 \ \forall\, |x| \le 9$.

$A_{P,\delta}$ is computable

A_P is not known to be computable—yet—but $A_{P,\delta}$ is (“almost everywhere”):

Theorem (G. ’23)

For any finite alphabet Σ, the function $(\delta, x) \mapsto A_{P,\delta}(x)$ is computable on $[0, 1) \times \Sigma^*$ except at:

$A_{P,\delta}$ is computable

A_P is not known to be computable—yet—but $A_{P,\delta}$ is ("almost everywhere"):

Theorem (G. '23)

For any finite alphabet Σ, the function $(\delta, x) \mapsto A_{P,\delta}(x)$ is computable on $[0, 1) \times \Sigma^*$ except at:

- *A countable c.e. set of discontinuities;*
- *Possibly $\delta = 0$ (it's at least continuous there).*

$A_{P,\delta}$ is computable

A_P is not known to be computable—yet—but $A_{P,\delta}$ is ("almost everywhere"):

Theorem (G. '23)

For any finite alphabet Σ, the function $(\delta, x) \mapsto A_{P,\delta}(x)$ is computable on $[0, 1) \times \Sigma^*$ except at:

- *A countable c.e. set of discontinuities;*
- *Possibly $\delta = 0$ (it's at least continuous there).*

In particular, for every x, $A_{P,\delta}(x)$ is computable for all but at most $A_D(x) - 2$ many values of δ.

$A_{P,\delta}$ is computable

A_P is not known to be computable—yet—but $A_{P,\delta}$ is ("almost everywhere"):

Theorem (G. '23)

For any finite alphabet Σ, the function $(\delta, x) \mapsto A_{P,\delta}(x)$ is computable on $[0,1) \times \Sigma^*$ except at:

- *A countable c.e. set of discontinuities;*
- *Possibly $\delta = 0$ (it's at least continuous there).*

In particular, for every x, $A_{P,\delta}(x)$ is computable for all but at most $A_D(x) - 2$ many values of δ.

Proof via computable analysis. (Thanks to Jake Canel for suggesting the approach.)

Classification of binary strings with $A_P = 2$

Theorem (G. '23)

For a binary string w, $A_P(w) = 2 \iff w$ is of the form

$$0^n 1^m, \qquad 0^n 1^m 0, \qquad 0^n (10)^m, \qquad \text{or} \quad 0^n 1 (01)^m$$

for some $n, m \geq 0$, or is the bit-flip of one of the above.

Classification of binary strings with $A_P = 2$

Theorem (G. '23)

For a binary string w, $A_P(w) = 2 \iff w$ is of the form

$$0^n1^m, \qquad 0^n1^m0, \qquad 0^n(10)^m, \qquad \text{or} \quad 0^n1(01)^m$$

for some $n, m \geq 0$, or is the bit-flip of one of the above.

To compare, $A_N(w) = 2$ iff $w = (01)^m$, 0^m1, or 01^m (Hyde).

Classification of binary strings with $A_P = 2$

Theorem (G. '23)

For a binary string w, $A_P(w) = 2 \iff w$ is of the form

$$0^n 1^m, \qquad 0^n 1^m 0, \qquad 0^n (10)^m, \qquad \textit{or} \quad 0^n 1 (01)^m$$

for some $n, m \geq 0$, or is the bit-flip of one of the above.

To compare, $A_N(w) = 2$ iff $w = (01)^m$, $0^m 1$, or 01^m (Hyde).

Since $A_N(0^n 1^n)$ is unbounded, we can save arbitrarily many states by switching to a PFA from an NFA. (At the cost of a more complex automaton...)

Classification of binary strings with $A_P = 2$

Theorem (G. '23)

For a binary string w, $A_P(w) = 2 \iff w$ is of the form

$$0^n 1^m, \qquad 0^n 1^m 0, \qquad 0^n (10)^m, \qquad \text{or} \quad 0^n 1 (01)^m$$

for some $n, m \geq 0$, or is the bit-flip of one of the above.

To compare, $A_N(w) = 2$ iff $w = (01)^m$, $0^m 1$, or 01^m (Hyde).

Since $A_N(0^n 1^n)$ is unbounded, we can save arbitrarily many states by switching to a PFA from an NFA. (At the cost of a more complex automaton...)

The proof of this theorem shows that a generic 2-state PFA describes an infinite family of strings of similar structure.

The IFS correspondence

Both directions of the proof rely heavily on a correspondence between PFAs and iterated function systems (IFSs).

The IFS correspondence

Both directions of the proof rely heavily on a correspondence between PFAs and iterated function systems (IFSs).

Let any 2-state binary PFA be given. Then there are maps

$$f_0(x) = a + bx \quad \text{and} \quad f_1(x) = c + dx$$

on $[0, 1]$ and a number x_0 such that for any w,

$$\rho(w) = f_{w(n)} \circ f_{w(n-1)} \circ \cdots \circ f_{w(0)}(x_0).$$

The reverse correspondence is also true. (This generalizes!)

The IFS correspondence

Both directions of the proof rely heavily on a correspondence between PFAs and iterated function systems (IFSs).

Let any 2-state binary PFA be given. Then there are maps

$$f_0(x) = a + bx \quad \text{and} \quad f_1(x) = c + dx$$

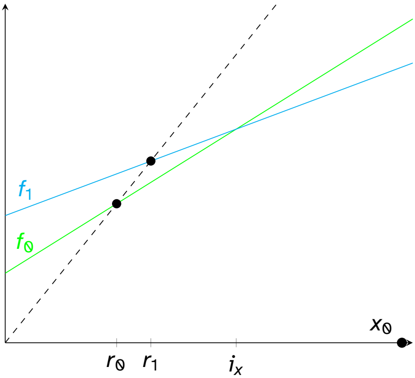
on $[0, 1]$ and a number x_0 such that for any w,

$$\rho(w) = f_{w(n)} \circ f_{w(n-1)} \circ \cdots \circ f_{w(0)}(x_0).$$

The reverse correspondence is also true. (This generalizes!)

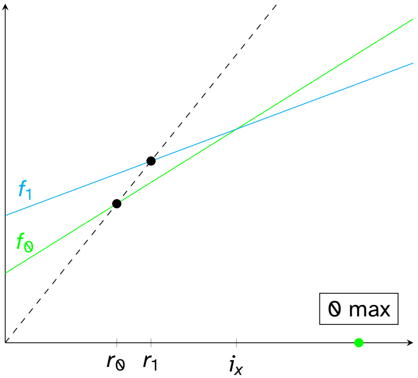
Idea of forward direction: For each n, find the sequence of n compositions of f_0 and f_1 attaining the highest possible value.

Illustration of forward direction: Positive slopes



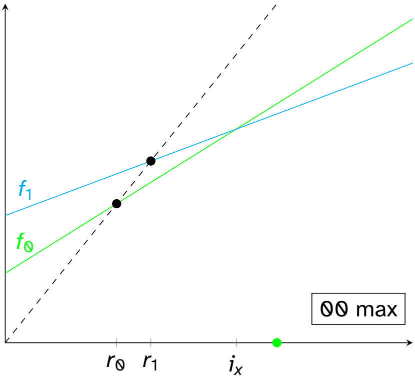
- A max prob is always the image of either a max prob (under a map of pos. slope) or a min prob (neg. slope).

Illustration of forward direction: Positive slopes



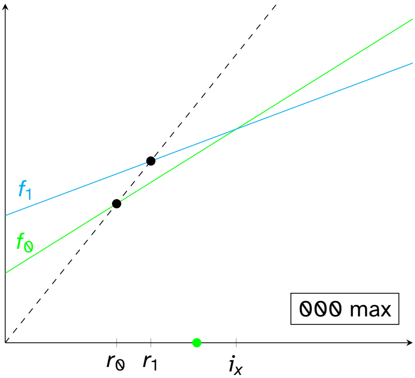
- A max prob is always the image of either a max prob (under a map of pos. slope) or a min prob (neg. slope).

Illustration of forward direction: Positive slopes



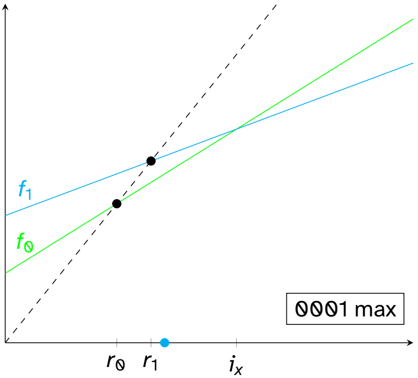
- A max prob is always the image of either a max prob (under a map of pos. slope) or a min prob (neg. slope).

Illustration of forward direction: Positive slopes



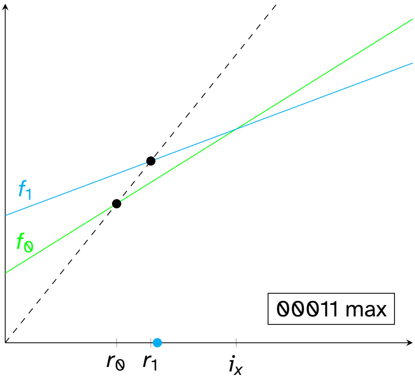
- A max prob is always the image of either a max prob (under a map of pos. slope) or a min prob (neg. slope).

Illustration of forward direction: Positive slopes



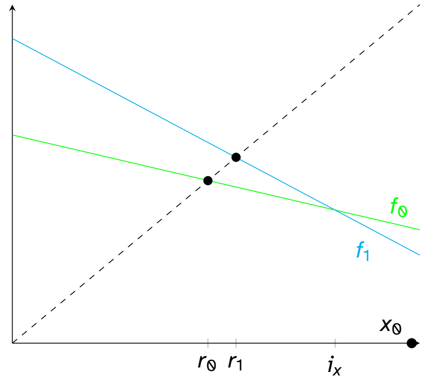
- A max prob is always the image of either a max prob (under a map of pos. slope) or a min prob (neg. slope).
- Here, iterating f_0 gives maxes until $x < i_x$, then f_1 is max. Witness $0^n 1^m$ $\forall m$.

Illustration of forward direction: Positive slopes



- A max prob is always the image of either a max prob (under a map of pos. slope) or a min prob (neg. slope).
- Here, iterating f_0 gives maxes until $x < i_x$, then f_1 is max. Witness 0^n1^m $\forall m$.

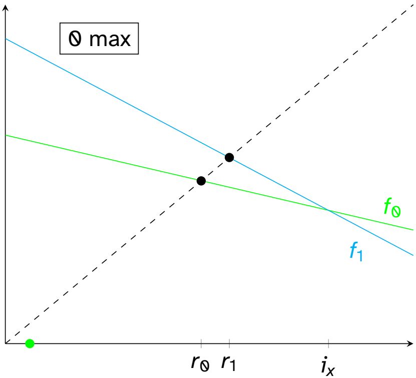
Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

Max-min-max-min pattern

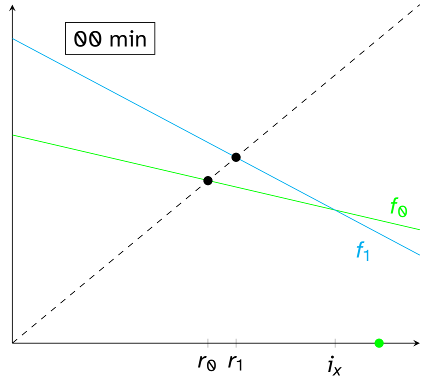
Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

Max-min-max-min pattern

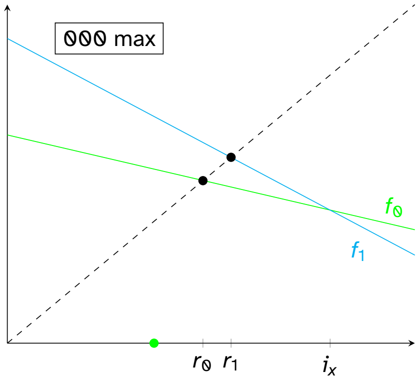
Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

Max-min-max-min pattern

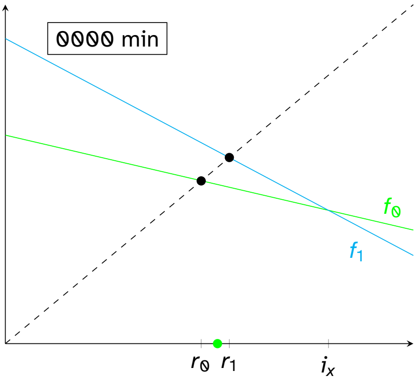
Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

Max-min-max-min pattern

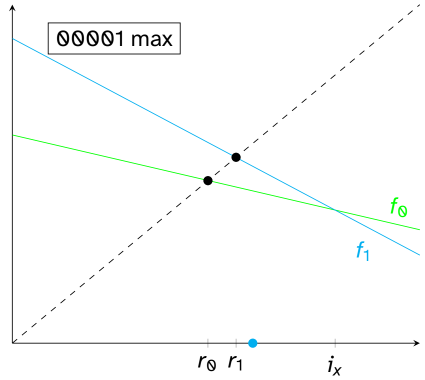
Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

Max-min-max-min pattern

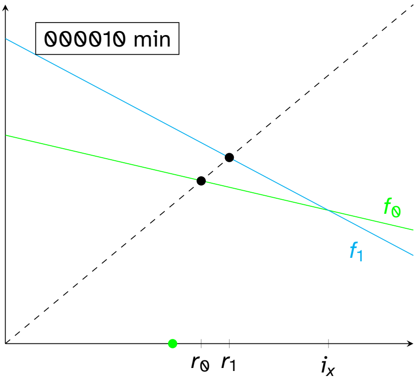
Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

Max-min-max-min pattern

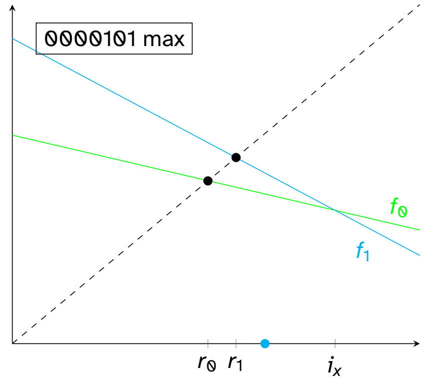
Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

Max-min-max-min pattern

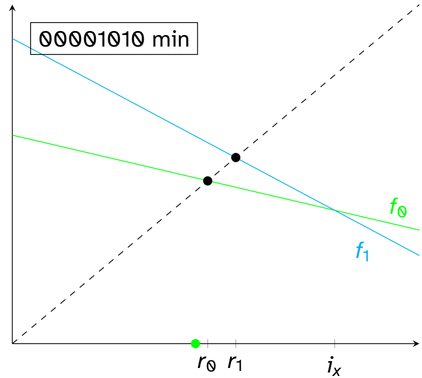
Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

Max-min-max-min pattern

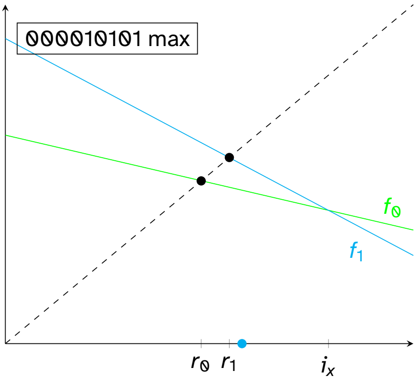
Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

Max-min-max-min pattern

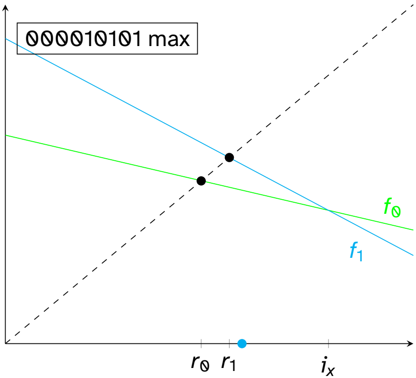
Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

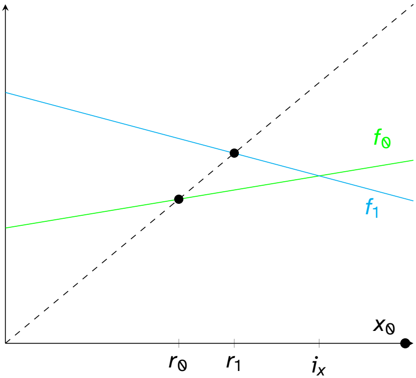
Max-min-max-min pattern

Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

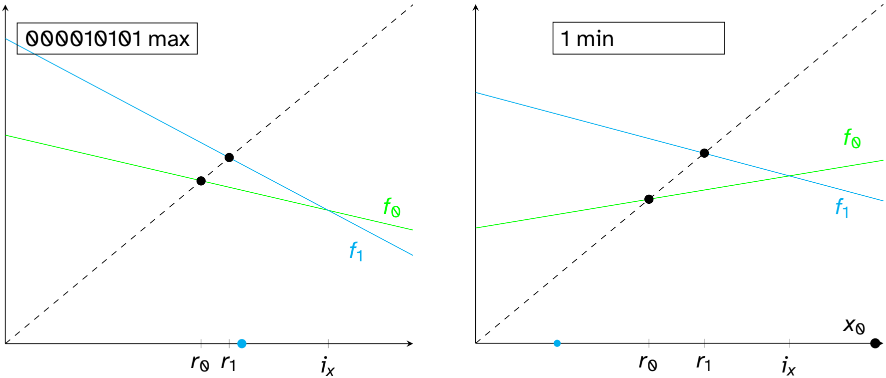
Max-min-max-min pattern



Mixed slopes: 1^n0^m1 $\forall m$

Stay min as long as possible, then apply f_1

Other subcases of the forward direction



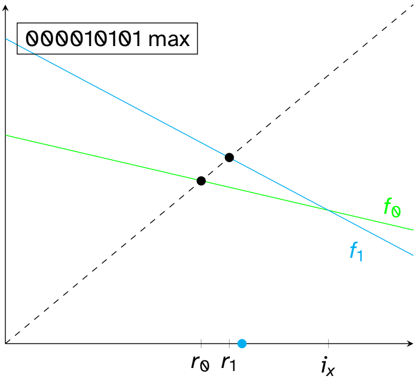
Negative slopes: $1^n(01)^m$, $0^n1(01)^m\ \forall m$

Max-min-max-min pattern

Mixed slopes: $1^n0^m1\ \forall m$

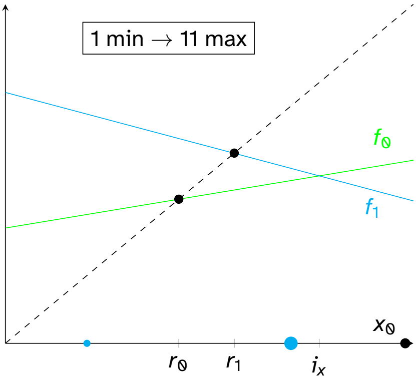
Stay min as long as possible, then apply f_1

Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

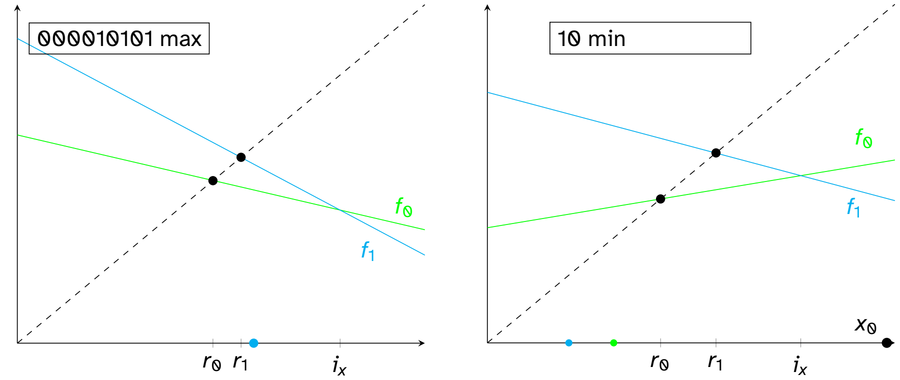
Max-min-max-min pattern



Mixed slopes: 1^n0^m1 $\forall m$

Stay min as long as possible, then apply f_1

Other subcases of the forward direction



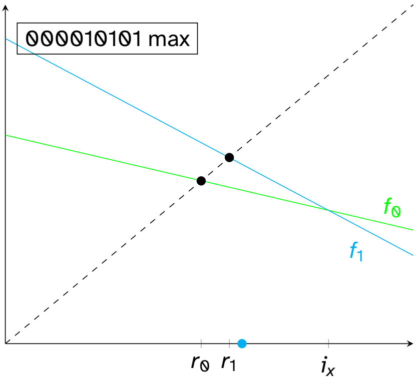
Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

Max-min-max-min pattern

Mixed slopes: 1^n0^m1 $\forall m$

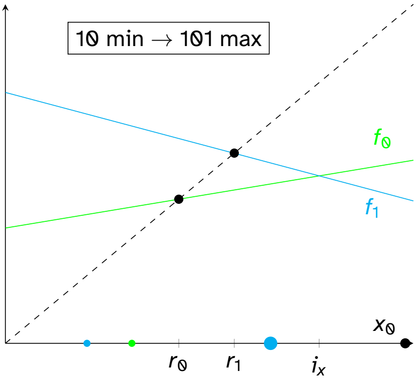
Stay min as long as possible, then apply f_1

Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

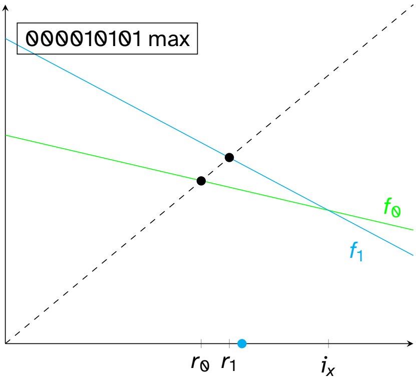
Max-min-max-min pattern



Mixed slopes: 1^n0^m1 $\forall m$

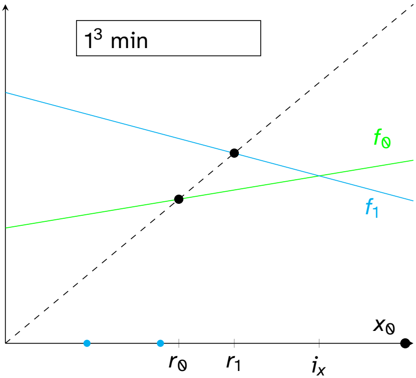
Stay min as long as possible, then apply f_1

Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

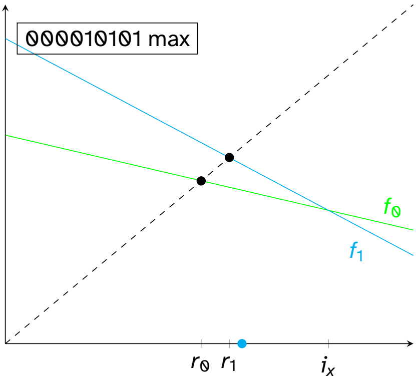
Max-min-max-min pattern



Mixed slopes: 1^n0^m1 $\forall m$

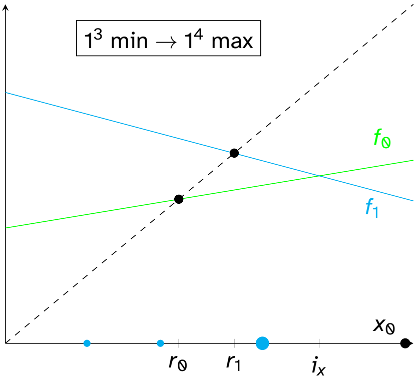
Stay min as long as possible, then apply f_1

Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n 1(01)^m$ $\forall m$

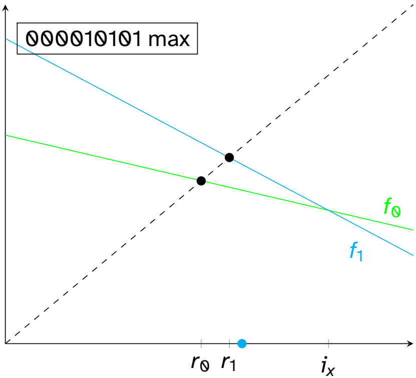
Max-min-max-min pattern



Mixed slopes: $1^n 0^m 1$ $\forall m$

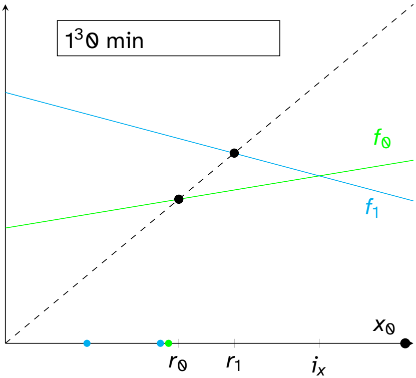
Stay min as long as possible, then apply f_1

Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

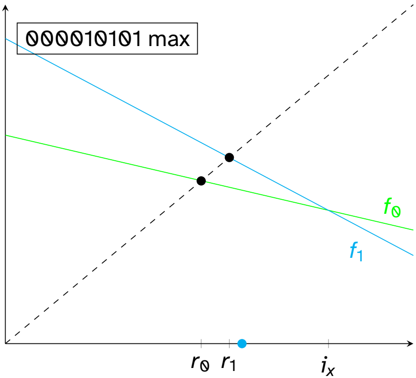
Max-min-max-min pattern



Mixed slopes: $1^n 0^m 1$ $\forall m$

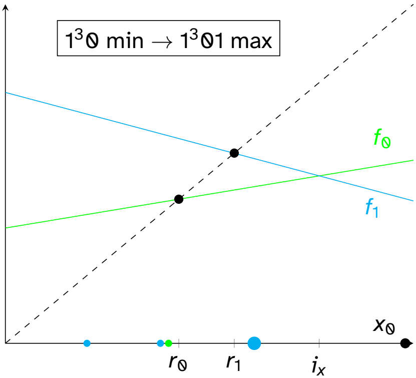
Stay min as long as possible, then apply f_1

Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

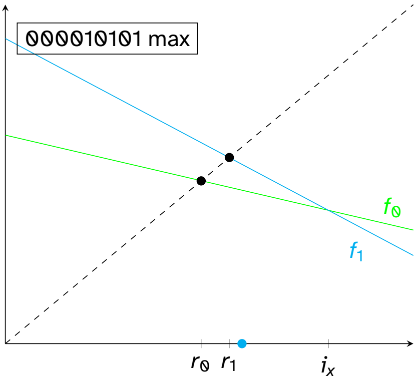
Max-min-max-min pattern



Mixed slopes: $1^n 0^m 1$ $\forall m$

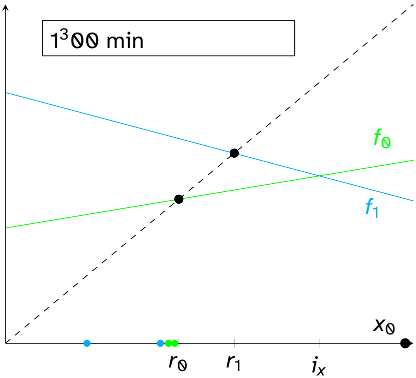
Stay min as long as possible, then apply f_1

Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

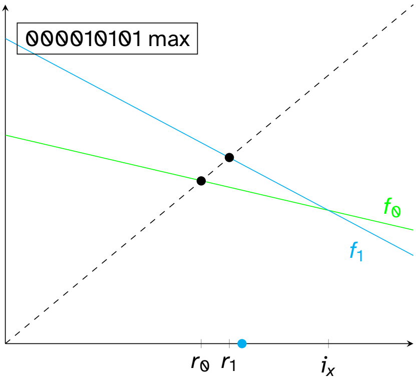
Max-min-max-min pattern



Mixed slopes: $1^n 0^m 1$ $\forall m$

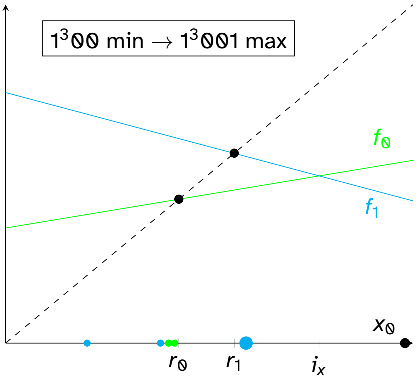
Stay min as long as possible, then apply f_1

Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

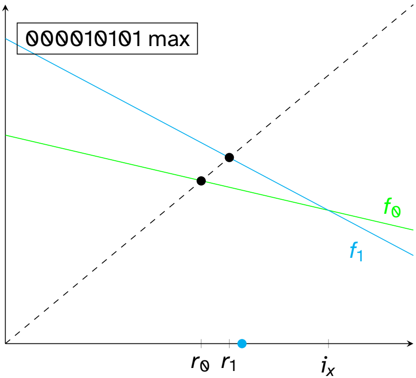
Max-min-max-min pattern



Mixed slopes: 1^n0^m1 $\forall m$

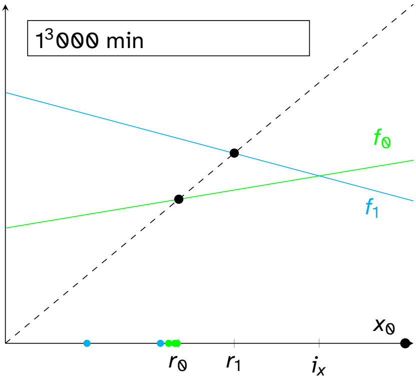
Stay min as long as possible, then apply f_1

Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

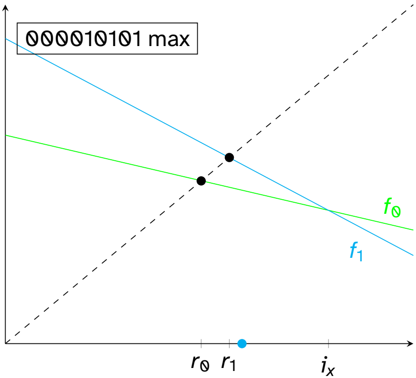
Max-min-max-min pattern



Mixed slopes: 1^n0^m1 $\forall m$

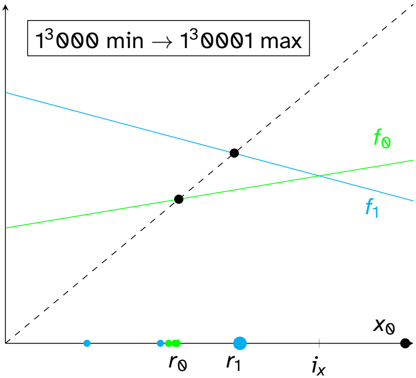
Stay min as long as possible, then apply f_1

Other subcases of the forward direction



Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

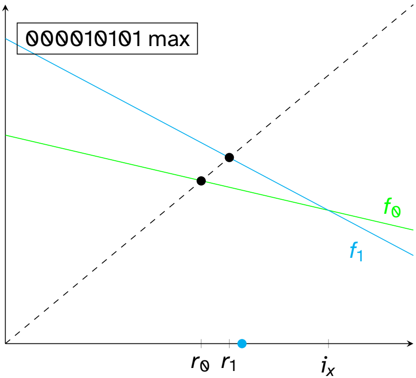
Max-min-max-min pattern

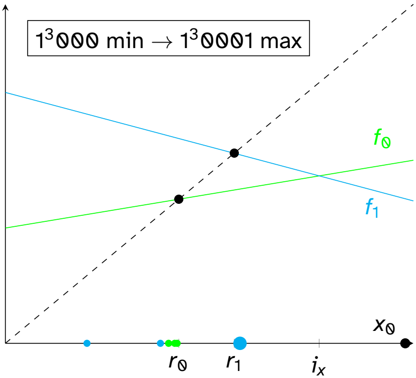


Mixed slopes: 1^n0^m1 $\forall m$

Stay min as long as possible, then apply f_1

Other subcases of the forward direction





Negative slopes: $1^n(01)^m$, $0^n1(01)^m$ $\forall m$

Max-min-max-min pattern

Mixed slopes: $1^n 0^m 1$ $\forall m$

Stay min as long as possible, then apply f_1

(Proof: *really* long!)

Further directions

- A_P computable? (likely yes: details pending)
- A_P unbounded? If so, tight asymptotic bound? Same questions for $A_{P,\delta}$.
- Use the max-gap function $\gamma^k(w)$ as a computable parametrized complexity measure instead?

References

- K. Gill, Probabilistic automatic complexity of finite strings, submitted (2024).
- K. Gill, *Two studies in complexity*, Ph.D. dissertation, Penn State University, 2023.
- K. Hyde, *Nondeterministic finite state complexity*, MA thesis, University of Hawai'i, Manoa, 2013.
- J. Shallit and M.-w. Wang, Automatic complexity of strings, *J. Autom. Lang. Comb.* **6**(4) (2001), 537–554.

This research was supported in part by NSF grant DMS-1854107.