How to define a set in P?

Mykyta Narusevych

0/1-strings

$\{0,1\}^*$ $\Longleftrightarrow$ $\mathbb{N}$

coding

Natural numbers

Language

$L \subseteq \{0,1\}^*$ $\Longleftrightarrow$ $A \subseteq \mathbb{N}$

Subset of naturals

$L \subseteq \{0,1\}^*$ is p-time

$\Updownarrow$

$\exists$ poly-time TM_L s.t. $s \in L$ iff TM_L accepts s

$\Longleftrightarrow$

$A \subseteq \mathbb{N}$ is p-time

$\Updownarrow$

?

Language of arithmetic

$$\left.\begin{array}{l} 0, 1 \ \ldots \text{ constants} \\ +, \cdot \ \ldots \text{ binary operations} \\ \leq \ \ldots \text{ binary relation} \end{array}\right\} L_{PA}$$

When interpreting numbers as 0/1-strings it is convenient to enrich L_{PA}

$$\left.\begin{array}{l} |\;| \ \ldots \text{ unary function} \\ \qquad \underline{\text{bit-length}} = \lceil \log_2 (x+1) \rceil \\ \\ \lfloor \frac{\cdot}{2} \rfloor \ \ldots \text{ unary function} \\ \qquad \underline{\text{shortening of string}} = \lfloor \frac{x}{2} \rfloor \\ \\ \# \ \ldots \text{ binary function} \\ \qquad \underline{\text{Nelson's smash function}} = 2^{|x| \cdot |y|} \end{array}\right\} + L_{PA} = L_{S_2}$$

Classes of formulas

Let x code some 0/1 string of length n, i.e. $|x| \sim n$.
Let y code some 0/1 string of length n^k.
Then, $y \sim x \underbrace{\# \ldots \#}_{k\text{-times}} x$

Pr: $|y| \sim n^k$, $|x \# \ldots \# x| = |2^{|x|^k}| = |x|^k \sim n^k$ $\blacksquare$

<u>Formula</u> = Formula in L_{S_2}

<u>standard model</u> = $\mathbb{N}$ + standard interpretation of L_{S_2}

<u>Definable set</u> = subset of $\mathbb{N}$ definable by a formula

Given a set $A \subseteq \mathbb{N}$ in P,
can we say something about its definition?

<u>Thm:</u> Every $A \subseteq \mathbb{N}$ which is in P is definable.
Actually, every <u>recursively</u> <u>enumerable</u> set $\subseteq \mathbb{N}$ is definable.

<u>Definability alone is not enough.</u>

<u>Problem:</u>

Not all definable sets are even recursively enumerable

<u>open formula</u> : formula with no quantifiers

A set $A \subseteq \mathbb{N}$ definable by an open formula is in P.

<u>Pr</u>: given numbers $n_1, \dots, n_k$
$m_1, \dots, m_\ell$

$t(n_1, \dots, n_k) = s(m_1, \dots, m_\ell)$

$t(n_1, \dots, n_k) \neq s(m_1, \dots, m_\ell)$ – decidable in

$t(n_1, \dots, n_k) \leq s(m_1, \dots, m_\ell)$ poly-time
in $|n_1| + \dots + |n_k|$
$+$
$|m_1| + \dots + |m_\ell|$

Adding $\wedge$ and $\vee$ does not change complexity ∎

<u>Problem</u>: not all sets defi in P are definable by open formulas.

E.g. PRIMES, ... – may not be true ☹

<u>Bounded quantifier</u> = $\forall x \leq t(x)$
$\exists x \leq t(x)$

<u>sharply bounded quantifier</u> = $\forall x \leq |t(x)|$
$\exists x \leq |t(x)|$

A ~~formula~~ set $A \subseteq \mathbb{N}$ definable by sharply bounded formula is in P

<u>pr</u>: unwind quantifier by exhaustive search

Apply the previous observation. ∎

<u>Problem</u>:

It is not at all clear whether poly-time sets are definable by sharply-bounded f-las.

Fix A in poly-time. Let TM_A be the corresponding turing machine.
Let's unwind the definition:

$$a \in A$$

$$\Updownarrow$$

$$TM_A \text{ accept } a$$

$$\Updownarrow$$

$$\exists y \, \underbrace{(y\text{-code of the accepting computation of } TM_A \text{ starting with } a)}_{\text{"O": can be defined by a } \Delta_0^b \text{ f-la (sharply-bounded)}}$$

$$\Updownarrow$$

$$\exists y \, \varphi(a, y) \, , \quad \varphi \in \Delta_0^b$$

$$\Updownarrow \quad |y| \sim |a|^k, \text{ since } TM_A \text{ is poly-time}$$

$$\underline{\exists y \leq t(a) \, \varphi(a, y)} \, , \quad t(a) \sim a \underbrace{\# \ldots \#}_{k\text{-times}} a$$

To sum-up : $a \in A \iff \exists y \leq t(a)\, \varphi(y, a)$,
$\varphi \in \Delta_0^b$

Formulas of type $\exists \bar{x} \leq t(\bar{x})\, \varphi(\bar{x})$ are called Σ_1^b-formulas $\varphi \in \Delta_0^b$.

Formulas of type $\forall \bar{x} \leq t(\bar{x})\, \varphi(\bar{x})$ are called Π_1^b-formulas

O: Any p-time set A is definable by a Σ_1^b f-la.

<u>Problem</u>: NP sets are also Σ_1^b-definable.

Moreover, a set is NP iff it's Σ_1^b-def.

Similarly, coNP coincides with Π_1^b-def.

Is there some difference between f-las defining P and NP sets?

A in P $\Rightarrow$ A^c in P, as well.

<u>Corr:</u> Let A be in P and $\varphi \in \Sigma^b_1$ defining A. Then, there is a Σ^b_1-f-la ψ s.t. $\neg\varphi \Leftrightarrow \psi$

Such formulas are called $\Delta^b_1 = \Sigma^b_1 \cap \Pi^b_1$.

Δ^b_1-definable sets are exactly $NP \cap coNP$

Let's analyze $\neg\varphi \Leftrightarrow \psi$ more.

What do we mean by $\Leftrightarrow$? ... true in $\mathbb{N}$

This lacks syntactical flavour ☹

What if we impose stronger requirements on (=) than just being true?

We want (=) to be provable in a feasible theory.

Feasible reasoning

BASIC = 32 axioms in L_{S_2} describing basic facts and relations between L_{S_2}-symbols.

All axioms of BASIC are some open formulas.

PIND axiom for $\varphi(x)$: $\varphi(0) \wedge \forall x \left[\varphi(\lfloor \frac{x}{2} \rfloor) \to \varphi(x)\right]$

$$\downarrow$$

$$\forall x\, \varphi(x).$$

$$S^1_2 = BASIC + PIND(\varphi)$$

for all Σ^b_1-f.las φ.

S^1_2 formalizes the following:

"given an NP set A (non-empty) there is a member of A with the smallest bit-length"

OR

"to prove that an NP property holds for all strings, it is enough to show that this property holds for ε and it is preserved under prolonging a string by a single 1-bit".

11.

S^1_2 can prove: SAT is NP-complete, PCP-theorem, CSP-dichotomy ...

<u>Thm</u>: given a p-time set A, there are Σ^b_1 f-las φ and ψ s.t. φ-defines A, ψ-defines A^c and $S^1_2 \vdash \neg\varphi \Leftrightarrow \psi$.

<u>Pr</u>: by suitable encoding of sequences + additional bootstrapping of S^1_2 ... $\square$

<u>Thm</u>: assume φ, ψ are Σ^b_1 s.t. $S^1_2 \vdash \neg\varphi \Leftrightarrow \psi$.
Then, the set defined by φ is in P.

So, S^1_2 captures exactly P.
On the other hand, we can view this as a <u>barrier</u>: S^1_2 cannot separate NP$\cap$coNP and P.

It is also possible to view this as an instruction how to show that a set A from NP is in P: prove that it is in NP$\cap$coNP in S^1_2.

We will prove this theorem.

However, we first discuss a similar issue of defining an FP relation.

$R(x,y)$ ... binary relation $\subseteq \mathbb{N}^2$

R in FP ... given x, it is easy to compute y s.t. $R(x,y)$

R in FNP ... given x, it is easy to verify whether some y is in relation to x

For R in FP:

$$(x,y) \in R$$

$$\Updownarrow$$

$$TM_R \text{ on } x \text{ outputs } y$$

$$\Updownarrow$$

$$\exists u \, (u \text{ - code for the computation of } TM_R \text{ starting with } x \text{ and ending with } y)$$

$$\Updownarrow$$

$$\dots$$

$$\exists u \leq t(x) \; \underbrace{\varphi(x,y,u)}_{\Delta_0^b}$$

To sum-up: every FP relation is Σ_1^b-definable.

__Problem__: FNP-relations are also Σ_1^b-def. moreover Σ_1^b-def. relations are exactly FNP.

The main difference between FP and FNP is that $\forall x$ there is a guaranteed y s.t. $(x,y) \in R$ .. not true for FNP

$$\forall x \exists y \underbrace{\varphi(x,y)}_{\Sigma_1^b}$$

Such FNP relations are also known as TFNP

__Thm__: $\forall R \in FP \; \exists \varphi(x,y) \in \Sigma_1^b$ s.t. $\varphi(x,y)$ defines FP and $S_2^1 \vdash \forall x \exists y \varphi(x,y)$

__Pr__: by suitable encoding of sequences + additional bootstrapping of S_2^1 ... ∎

We will prove the converse

Thm 1: given $\varphi(x, y) \in \Sigma_1^b$ s.t.

$S_2^1 \vdash \forall x \exists y\, \varphi(x, y)$ it follows that

$\varphi(x, y)$ defines an FP relation

Thm 2: given $\varphi(x), \psi(x) \in \Sigma_1^b$ s.t.

$S_2^1 \vdash \forall x\, \neg\varphi(x) \Leftrightarrow \psi(x)$ it follows that

$\varphi(x)$ defines a P set

S_2^1 syntactically captures exactly P and FP

S_2^1 cannot separate P from $NP \cap coNP \subseteq NP$

and FP from $TFNP \subseteq FNP$

💡 Thm 1 implies Thm 2:

Pr: take $\varphi(x), \psi(x)$ as above. Define

$$R(x, y) \dots \begin{matrix} (x, 0) \in R \Leftrightarrow \varphi(x) \\ (x, 1) \in R \Leftrightarrow \neg\varphi(x) \Leftrightarrow \psi(x). \end{matrix}$$

R is definable by: $\underbrace{[y = 0 \wedge \varphi(x)] \vee [y = 1 \wedge \psi(x)]}_{\Sigma_1^b\text{-fla } \Theta(x,y)}$

$S_2^1 \vdash \forall x \exists y\, \Theta(x, y)$ ... ∎

So, we focus on Thm 1 only.

$S^1_2 \vdash \forall x \exists y \varphi(x,y) \rightsquigarrow$ given x, easy to compute y

Why do we even expect that something like this even holds?

<u>Thm</u>:

$$I\Sigma_1 \vdash \forall x \exists y \varphi(x,y) \quad \text{for } \varphi \in \Sigma_1$$

$$\Downarrow$$

a function computing y for a given x is primitive recursive.

There is a natural correspondence:

Recursion theory	Complexity theory
$\Sigma_1 \rightsquigarrow$ [Buss]	$\rightsquigarrow \Sigma^b_1$
$I\Sigma_1 \rightsquigarrow$ [Buss]	$\rightsquigarrow S^1_2$
primitive recursive $\rightsquigarrow$ [Cobham]	$\rightsquigarrow$ polynomial-time

What is the idea of a proof?

We are given $\varphi(x,y) \ldots \exists u \leq t(x,y)\, \underbrace{\psi(u,x,y)}_{\Delta_0^b}$

$S_2^1 \vdash \forall x\, \exists y\, \exists u \leq t(x,y)\, \psi(u,x,y)$

$S_2^1 \vdash \exists y\, \exists u \leq t(a,y)\, \psi(u,a,y)$ for free variable a.

Consider a proof $P(a)$ of the above. $P(a)$ has fixed length, so one can hope, by plugging some particular n into P to actually be able to compute corresponding y efficiently.

Of course, it cannot be that simple. In general, proofs are impossibly hard to analyze in full generality.

Need a more constructive way to write proofs ...

Sequent Calculus!

S^1_2 as LK-style system:

Axioms:

$A \longrightarrow A$ for any f-la A

$\longrightarrow \varphi(\bar{x})$ for open f-las from BASIC

Inference rules:

Weak structural rules

+

Propositional rules

+

The cut rule ... $\dfrac{\Gamma \longrightarrow \Delta, A \quad A, \Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta}$

+

Equality axioms

+

Bounded quantifier rules:

$$\frac{A(t), \Gamma \longrightarrow \Delta}{t \leq s, \forall x \leq s A(x), \Gamma \longrightarrow \Delta} \qquad \frac{b \leq s, \Gamma \longrightarrow \Delta, A(b)}{\Gamma \longrightarrow \Delta, \forall x \leq s A(x)}$$

$$\frac{b \leq s, A(b), \Gamma \longrightarrow \Delta}{\exists x \leq s A(x), \Gamma \longrightarrow \Delta} \qquad \frac{\Gamma \longrightarrow \Delta, A(t)}{t \leq s, \Gamma \longrightarrow \Delta, \exists x \leq s A(x)}$$

Polynomial induction rule :

$$\frac{A(\lfloor \frac{b}{2} \rfloor), \Gamma \longrightarrow \Delta, A(b)}{A(0), \Gamma \longrightarrow \Delta, A(t)}, \quad A \in \Sigma^b_1.$$

Thm : $S^1_2 \vdash \varphi$ in a usual way

$\Updownarrow$

$S^1_2 \vdash \; \longrightarrow \varphi$ in a LKB-style (sequent calculus)

Thm (cut elimination) :

it is enough to have only special type of cuts ... <u>non-free</u>

Cor (subformula property) :

Assume $S^1_2 \vdash \Gamma \longrightarrow \Delta$, s.t. all the formulas among Γ and Δ are Σ^b_1.

Then, we may as well assume that all the formulas in the proof are Σ^b_1, too.

So, we are in a situation:

$$S^1_2 \vdash \forall \exists y \underbrace{\exists u \leq t(a,y)\, \psi(u,a,y)}_{\Sigma^b_1}$$

However, this is not a Σ^b_1-formula!

<u>Thm [Parikh]</u>:

$S^1_2 \vdash \forall x \exists y\, \varphi(x,y) \iff \exists$ term t s.t.

$S^1_2 \vdash \forall x \exists y \leq t(x)\, \varphi(x,y)$, where $\varphi \in \Sigma^b_1$.

Thus, our initial setup is as follows:

$$S^1_2 \vdash \underbrace{\exists y \leq t(a) \exists u \leq s(a,y)\, \psi(u,a,y)}_{\Sigma^b_1}$$

We apply the Corrolary from the previous page to get an LKB-proof of

$\longrightarrow \exists y \leq t(a) \exists u \leq s(a,y)\, \psi(u,a,y)$ s.t.

all the formulas appearing in the proof are Σ^b_1.

The most natural way is to somehow use the induction on the proof length.

$$S^1_2 \vdash \exists y \leq t \, \exists u \leq s \, \psi(a, y, u)$$

compute y by using something computed one step before.

To simplify, we will actually compute both (y, u) ..

$$S^1_2 \vdash \exists \bar{y} \leq t \, \psi(a, \bar{y})$$

given a, need this $\bar{y}$

Such $\bar{y}$ serves as a <u>witness</u> for a formula $\exists \bar{y} \leq t(a) \, \psi(a, \bar{y})$.

We need a more general notion of a witness for sequents $\Gamma \longrightarrow \Delta$ of Σ^b_1 - formulas

Witness:

- $\varphi(\bar{a}) \in \Delta_0^b$ ... witness is just $\varphi(\bar{a})$ itself

- $\exists x \leq t(\bar{a})\, \varphi(x, \bar{a})$ ... witness is a combination of $b \leq t(\bar{a})$ and a witness for $\varphi(b, \bar{a})$

- $\forall x \leq |t(\bar{a})|\, \varphi(x, \bar{a})$ ... witness is a combination of witnesses for each $b \leq |t(\bar{a})|$ of $\varphi(b, \bar{a})$

- for sequent $\Gamma \longrightarrow \Delta$ .. witness to Γ is a witness to $\bigwedge_{\varphi \in \Gamma} \varphi$ and witness to Δ is a witness to $\bigvee_{\varphi \in \Delta} \varphi$

So given $\exists \bar{y} \leq t(a)\, \underbrace{\varphi(\bar{y}, a)}_{\Delta_0^b}$ its witness is precisely what we want to compute.

We prove the following Lemma:

Lemma: given sequents Γ, Δ of Σ_1^b formulas s.t. $S_2^1 \vdash \Gamma \longrightarrow \Delta$, there is a poly-time machine h which, given parameters $\overline{a}$ (assignment to free variables to formulas in Γ, Δ) and a witness to $\Gamma(\bar{a})$, outputs a witness to $\Delta(\bar{a})$.

This lemma immediately implies the Thm 1:

$$S_2^1 \vdash \longrightarrow \exists \bar{y} \leq t(a) \varphi(a, \bar{y})$$

we have a p-time machine h, which on a outputs $\bar{b}$ s.t. $\bar{b} \leq t(a)$ and $\varphi(a, \bar{b})$.

Proof of the lemma (sketch):

By induction on proof length

Case 1: $A \to A$ and $\to \varphi$, $\varphi \in BASIC$

is straightforward $\blacksquare$

Case n+1: weak structural rules, propositional rules, equality axioms .. easy. $\blacksquare$

Non-free cut:
$$\frac{\Gamma \to \Delta, A \qquad A, \Gamma \to \Delta}{\Gamma \to \Delta}.$$

$A \in \Sigma^b_1$, since it is anchored

given witness to Γ we compute

a witness to $\Delta, A = \bigvee_{\varphi \in \Delta} \varphi \vee A$.

If it witnesses Δ (easily checkable), then we are done.

If it witnesses A, then we actually have a witness to A, Γ and of which we can efficiently compute a witness to Δ $\blacksquare$

Bounded quantifier:

We show only the hardest case:

$$\frac{b \leq s, \Gamma \longrightarrow \Delta, A(b)}{\Gamma \longrightarrow \Delta, \forall x \leq s\, A(x)} \,, \quad b \text{ appears only in } A.$$

Given witness to Γ, we automatically have a witness to $b \leq s, \Gamma$.

If the output witnesses Δ, we are done.

So assume we witness $A(b)$.

The only interesting case is when $A \in \Sigma^b_1$ and not in Δ^b_0.

In that case, however, s is necessary $|t|$, otherwise $\forall x \leq s\, A(x)$ is not Σ^b_1 anymore.

So:

$$\frac{b \leq |t|, \Gamma \longrightarrow \Delta, A(b)}{\Gamma \longrightarrow \Delta, \forall x \leq |t|\, A(x)} .$$

Induction hypothesis tells us, that, given a witness to

$$n \le |t| \wedge \bigwedge_{\varphi \in \Gamma} \varphi \quad \text{for any choice of } n$$

you can feasibly produce a witness to

$$\bigvee_{\varphi \in \Delta} \varphi \vee A(n).$$

Now, we are given a witness to Γ and need to output a witness to

$$\bigvee_{\varphi \in \Delta} \varphi \vee \forall x \le |t| A(x).$$

Loop through all choices of $n \le |t|$ and see whether at some point you output a witness for $\bigvee_{\varphi \in \Delta} \varphi$ by using the machine from the previous step. If you do, then this serves as a witness to $\Delta, \forall x \le |t| A(x)$, if not, then you have a witness to $\forall x \le |t| A(x)$ $\blacksquare$

Finally, let's analyze induction inference:

$$\frac{A(\lfloor \frac{b}{2} \rfloor), \Gamma \longrightarrow \Delta, A(b)}{A(0), \Gamma \longrightarrow \Delta, A(t)} \, , \quad b \text{ appears in } A \text{ only.}$$

Again, the only interesting case is when $A \in \Sigma^b_1$ and not in Δ^b_0.

Take a witness to $A(0), \Gamma$.

Output a witness to $\Delta, A(1)$.

If you have witnessed Δ .. done.

Otherwise, you have a witness to $A(1), \Gamma$.

Proceed by always multiplying b by 2 until you either hit a witness to $A(t)$ or a witness to Δ ... $|t|$ many steps)

$\blacksquare$

Final remarks:

The mentioned Thm 1 and Thm 2 hold more generally:

Thm 1*: Every Δ^b_i $(= \Sigma^b_i \cap \Pi^b_i)$-definable set (predicate) of S^i_2 is in the i-th level Δ^p_i of polynomial hierarchy

Thm 2*: Every Σ^b_i-definable function of S^i_2 is in the $\square^p_i$ level of functional polynomial hierarchy.

Proof are exactly the same as for i=1 case. One needs to modify the witness definition