UCT

Polynomial Hierarchy

Klaus Sutner

Carnegie Mellon University

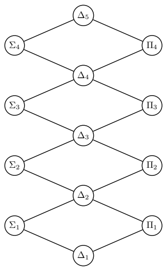
Spring 2022

1 **The Polynomial Hierarchy**

2 **Alternating Turing Machines**

Recall: Arithmetical Hierarchy

Recall that there is a natural hierarchy beyond the fundamental classes of decidable and semidecidable problems.



The hierarchy is proper, and there are fairly straightforward complete problems such as FIN, TOT, REC and so on.

Knowledge Transfer

We can think of $\mathbb{P}$ and $\mathbb{NP}$ as an analogue to decidable versus semidecidable (the classes Δ_1 versus Σ_1 in the AH). This makes it tempting to try and construct a polynomial time hierarchy (PH) analogous to the AH.

We mentioned this PH briefly when we first introduced $\mathbb{NP}$, here is some more information.

> **Dire Warning:** Note that we will not be able to prove an analogue to Post's hierarchy theorem this time around. Still, it may be interesting to study this putative hierarchy: reducibilities, complete problems, relation to other classes . . .

Recall: Definition PH

$$\Sigma_0^p = \Pi_0^p \; = \; \mathbb{P}$$

$$\Sigma_{n+1}^p = \mathsf{proj}_P(\Pi_n^p)$$

$$\Pi_n^p = \mathsf{comp}(\Sigma_n^p)$$

$$\Delta_n^p = \Sigma_n^p \cap \Pi_n^p$$

$$\mathrm{PH} = \bigcup \Sigma_n^p$$

Here proj_P refers to polynomially bounded projections: the witness is required to have length polynomial in the length of the witness. comp is just plain complement (unchanged from AH).

Some Example

Here is a version of the Independent Set problem that makes as much sense, and might be more interesting to a graph theory person:

Problem:	**Maximum Independent Set** (MIS)
Instance:	A ugraph G, a number k.
Question:	Does the largest independent set in G have size k?

We could guess a vertex set and verify that it is independent and has size k, but that does not preclude the existence of an even larger one. The verification would also need to include what looks like an exponential search over all bigger sets.

MIS seems to belong to a class larger than $\mathbb{NP}$.

Boolean Example

It is most natural to try to find the smallest Boolean formula equivalent to a given one.

Problem:	**Minimum Equivalent DNF (**MEQDNF**)**
Instance:	A Boolean DNF formula Φ, a bound k.
Question:	Is Φ equivalent to a DNF formula with at most k literals?

Here we assume that our language has Boolean variables, constants, and the usual connectives. In particular we have constants $\top$ and $\bot$.

We could nondeterministically guess the small formula Φ', but then we need to verify that it is equivalent to Φ, an apparently exponential task.

Where's the Problem?

Access to a MEQDNF oracle would immediately demolish SAT.

Lemma

SAT *is polynomial time Turing reducible to* MEQDNF*.*

Proof.

Let $\Phi = \bigwedge C_i$ be a CNF formula, an instance of SAT.

Define the DNF formula Ψ to be $\neg\Phi = \bigvee \neg C_i$. It is easy to check in polynomial time whether Ψ is a contradiction.

If not, Ψ is a tautology iff it is equivalent to a formula with 0 literals: the constant $\top$, (constant $\bot$ is ruled out since Ψ is not a contradiction).

□

Automata Example

A pattern π is an expression formed from strings in $\mathbf{2}^\star$ and variables, using only concatenation. We can generate a language $\mathcal{L}(\pi)$ by replacing the variables by strings over $\mathbf{2}^\star$. Given (finite) languages $P[N]$ of positive[negative] examples, we would like a pattern that is consistent with P and N.

Problem:	**Pattern Consistency**
Instance:	Two finite languages $P, N \subseteq \mathbf{2}^\star$.
Question:	Is there a pattern π such that $P \subseteq \mathcal{L}(\pi) \subseteq \overline{N}$?

Again, we could perform a simple nondeterministic guess, but it seems that the guess would have to be followed by an exponential brute-force verification.

Graph Example

Problem:	**Node Deletion**
Instance:	Two graphs G and H, a number k.
Question:	Can one delete at most k vertices from G to obtain a graph that does not contain H as a subgraph?

We could nondeterministically guess the vertices that determine the subgraph G', and then verify that G' has no subgraph isomorphic to H. Again, the second part seems exponential.

The Class Σ_2^p

Here is an explicit definition of the class Σ_2^p.

Definition

L is in Σ_2^p if there is a polynomial time decidable relation V and a polynomial p such that

$$x \in L \iff \exists\, u \in \mathbf{2}^{p(|x|)}\, \forall\, v \in \mathbf{2}^{p(|x|)}\; V(u, v, x).$$

So from the definition $\mathbb{NP}, \text{co-}\mathbb{NP} \subseteq \Sigma_2^p$, requiring only one block of quantifiers.

All the examples from above are in Σ_2^p: use the existential quantifiers for guessing and the universal ones for verfication. For fixed u and v, the actual verification can be handled by a deterministic polynomial time TM.

Completeness

All of our examples of problems in Σ_2^p are complete for this class:

- Maximum Independent Set
- Minimum Equivalent DNF
- Pattern Consistency
- Node Deletion

So Σ_2^p seems to be a reasonable class that contains at least some RealWorld$^{\text{TM}}$ problems.

More Classes

For Σ_2^p we have one block of existential quantifiers followed by one block of universal quantifiers. Of course, this generalizes:

- Σ_k^p: k alternating blocks of quantifiers, starting with existential
- Π_k^p: k alternating blocks of quantifiers, starting with universal

So e.g. L is in Π_3^p iff

$$x \in L \iff \forall\, u \in \mathbf{2}^{p(|x|)}\, \exists\, v \in \mathbf{2}^{p(|x|)}\, \forall\, w \in \mathbf{2}^{p(|x|)}\, R(u, v, w, x).$$

Note that this description also lends itself nicely to finding a class in the PH that contains a particular given problem: write things down concisely, then count quantifiers.

Validity Testing

It is clear from the definitions that Σ_k^p is closely connected to the problem of testing the validity of a Σ_k Boolean formula (and similarly Π_k for Π_k^p): we can express the workings of the Turing machine acceptor as a Boolean formula as in the Cook-Levin theorem.

Theorem

Validity of Σ_k Boolean formulae is Σ_k^p-complete wrto polynomial time reductions.

Validity of Π_k Boolean formulae is Π_k^p-complete wrto polynomial time reductions.

So just like for the AH, we can find reasonable complete problems for the PH. But we don't know that they get more and more complicated.

Example: Integer Expressions

Define an integer expression to be composed of numbers (written in binary), and binary operations $\cup$ and $\oplus$ where

$$A \oplus B = \{\, a + b \mid a \in A, b \in B \,\}$$

Write $\mathcal{L}(E) \subseteq \mathbb{N}$ for the finite set associated with the expression E. An interval in $\mathcal{L}(E)$ is a subset $[a, b] \subseteq \mathcal{L}(E)$.

Problem:	**Integer Expression Intervals (IEI)**
Instance:	An integer expression E, a number k.
Question:	Does $\mathcal{L}(E)$ have an interval of length k?

Note that the cardinality of $\mathcal{L}(E)$ is not polynomially bounded by $|E|$, so we cannot simply evaluate the expression in polynomial time.

Proof Membership

Lemma

Integer Expression Intervals is Σ_3^p-complete.

Assume that all inputs are at most ℓ-bit and the expression has depth d. Then the largest number m in $\mathcal{L}(E)$ is at most $\ell + d$ bits and thus polynomial in $|E|$.

Think of E as a parse tree. To check $a \in \mathcal{L}(E)$ for $a \leq m$ we can guess a subtree T of E. T is node-labeled by $\mathbb{N}$, contains the root with label a, and has the following properties. Let ν be a node in T, then:

- If ν is a union node labeled b, then T contains exactly one child node of ν, also labeled b.
- If ν is a sum node labeled b, then T contains both child nodes of ν, labeled b_1 and b_2, where $b = b_1 + b_2$.
- If ν is a leaf, it's label is the same as the number stored there.

Proof, Contd

Then there is an interval of length k iff

$$\exists\, a, b \leq m\, \forall\, x \leq m\, \big(b = a + k - 1 \wedge a \leq x \leq b \Rightarrow x \in \mathcal{L}(E)\big).$$

We have just seen $x \in \mathcal{L}(E)$ is in $\mathbb{NP}$, so IEI is in Σ_3^p. □

Alas, completeness is much harder.

A similar argument shows that inequality of integer expressions is Σ_2^p.

More Oracles

We know how to attach an oracle $A \subseteq \Sigma^\star$ to a Turing machine $\mathcal{M}$. If we do this systematically for all machines in a certain class $\mathcal{C}$, we obtained a relativized class $\mathcal{C}^A$.

The oracle may vary over a class $\mathcal{D}$:

$$\mathcal{C}^\mathcal{D} = \bigcup_{A \in \mathcal{D}} \mathcal{C}^A$$

For example, $\mathbb{P}^\mathbb{P} = \mathbb{P}$ and $\mathbb{P}^{\mathbb{NP}} = \mathbb{P}^{\mathsf{SAT}}$.

Similar definitions can be obtained for function classes, and function oracles.

Alternative Polynomial Hierarchy

We can now check that our original definition of PH in terms of projections and complements can also be expressed in terms of oracles like so:

$$\begin{aligned}
\Sigma_0^p &= \Pi_0^p \; = \; \mathbb{P} \\
\Delta_{n+1}^p &= \mathbb{P}^{\Sigma_n^p} \\
\Sigma_{n+1}^p &= \mathbb{NP}^{\Sigma_n^p} \\
\Pi_n^p &= \text{comp}(\Sigma_n^p) \\
\mathrm{PH} &= \bigcup \Sigma_n^p
\end{aligned}$$

Exercise

Verify that our two definitions are really equivalent.

There is no analogue to the hierarchy theorem for the arithmetical hierarchy, but it is still true that

$$\Sigma_k^p \cup \Pi_k^p \;\subseteq\; \Delta_{k+1}^p \;\subseteq\; \Sigma_{k+1}^p \cap \Pi_{k+1}^p$$

Also, if there is a glitch at some level $k \geq 1$, then the rest of the hierarchy collapses:

$$\begin{aligned}
\Sigma_k^p = \Pi_k^p &\quad \text{implies} \quad \Sigma_k^p = \mathrm{PH} \\
\Sigma_k^p = \Sigma_{k+1}^p &\quad \text{implies} \quad \Sigma_k^p = \mathrm{PH}
\end{aligned}$$

This leaves open the possibility that, say, the first 42 levels are proper, and the rest collapses. Perish the thought.

Versus PSPACE

Lemma

$\mathrm{PH} \subseteq \mathrm{PSPACE}$.

$\mathrm{PH} = \mathrm{PSPACE}$ *implies the polynomial hierarchy collapses.*

Proof.

To see why, recall that QBF Validity is in PSPACE, and subsumes all the Σ_k Validity problems.

If we had equality, then QBF Validity would already be equivalent to Σ_k Validity for some k.

□

Completeness?

We have Σ_k^p-complete problems for all levels $k \geq 1$, generic as well as concrete.

But note that the last argument seems to rule out the existence of PH-complete problems: if L were PH-complete, then $L \in \Sigma_k^p$ for some k simply by the definition of the polynomial hierarchy.

But then the hierarchy collapses at level k—which sounds less than plausible. Famous last words.

1 The Polynomial Hierarchy

2 **Alternating Turing Machines**

Machine Models

We defined the polynomial hierarchy in analogy to the arithmetical hierarchy: by applying suitable projections and complements to polynomial time decidable sets. An alternative definition can be given in terms of oracles.

It is natural to ask whether there is yet another definition that focuses on a machine model instead: more precisely, we would like some class of Turing machines that defines precisely the languages in PH.

We will have to push the envelope a bit to make this happen, the resulting machines are a bit more complicated that ordinary Turing machines, or even plain nondeterministic ones.

Turing Machines

- Plain, deterministic Turing machines are slightly irritating to use, but still a fairly direct formalization of computation. They certainly are physically realizable (maybe if and only if, at least if we ignore wild quantum physics, black holes and the like).
- Nondeterministic Turing machines add a significant level of abstraction: we are now dealing with a tree of computations; realizability fades away.
- Probabilistic Turing machines return to the realm of realizable computation; we could use physical random sources for a precise implementation.
- Alternating Turing machines push the level of abstraction much higher.

Warm-Up: Finite State Machines

In a DFA there is exactly one trace for each input.

In an NFA there may be exponentially many, and acceptance is determined by an existential quantification: is there a run .... So here is a tempting question:

Question: Is there a useful notion of acceptance based on "for all runs such that such and such"?

One problem is whether these "universal" automata are more powerful than ordinary FSMs. As we will see, we still only get regular languages.

Of course, this raises the question of how the state complexities compare.

Forall Automata

How would one formally define a type of FSM $\mathcal{A} = \langle Q, \Sigma, \delta; I, F \rangle$ where acceptance means all runs have a certain property?

The underlying transition system $\langle Q, \Sigma, \delta \rangle$ will be unaffected, it is still a labeled digraph.

The acceptance condition now reads:

$\mathcal{A}$ accepts x if all runs of $\mathcal{A}$ on x starting at I end in F.

Let's call these machines ∀FA.

Read: universal FA. Actually, don't: this collides with the standard use where universal means "accepting all inputs." Just look at the beautiful symbol and don't say anything. Wittgenstein would approve.

By the same token, a NFA would be a ∃FA.

Logical And

As an example consider the mod counter languages

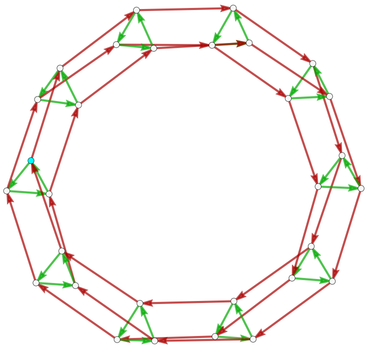
$$K_{a,m} = \{ x \in \mathbf{2}^\star \mid \#_a x = 0 \pmod{m} \}$$

with state complexity m. For the union $K_{0,m} \cup K_{1,n}$ we have a natural NFA of size $m + n$. However, for the intersection $K_{0,m} \cap K_{1,n}$ we only have a product machine that has size mn.

More importantly, note that nondeterminism does not seem to help with intersection: there is no obvious way to construct a smaller NFA for $K_{0,m} \cap K_{1,n}$.

This happens on occasion: there are regular languages where nondeterminism seems utterly useless.

Example Modcounter Intersection



But we can build a ∀FA of size just $m + n$: take the disjoint union and declare the acceptance condition to be universal.

A Hidden Product

What is really going on here?

Let's assume that Q_1 and Q_2 are disjoint. Starting at $\{q_{01}, q_{02}\}$ we update both components. So after a while we are in state

$$\{p, q\}$$

where $p \in Q_1$ and $q \in Q_2$. In the end we accept iff $p \in F_1$ and $q \in F_2$.

This is really no different from a product construction, we just don't spell out all the product states explicitly: instead, we use a succinct representation.

Choosing clever representations is sometimes critically important. Trust me.

That's Not Insane

Note that acceptance testing for a ∀FA is no harder than for an NFA: we just have to keep track of the set of states $\delta(I, x) \subseteq Q$ reachable under some input and change the notion of acceptance: this time we want $\delta(I, x) \subseteq F$.

For example, if some word x crashes all possible computations so that $\delta(I, x) = \emptyset$, then x is accepted.

Likewise we can modify the Rabin-Scott construction that builds an equivalent DFA: as before calculate the (reachable part of the full) powerset and adjust the notion of final state:

$$F' = \{ P \subseteq Q \mid P \subseteq F \}$$

Wisdom

> A mathematician is a person who can find analogies between theorems; a better mathematician is one who can see analogies between proofs and the best mathematician can notice analogies between theories. One can imagine that the ultimate mathematician is one who can see analogies between analogies.
>
> S. Banach

An Analogy (pace Banach)

We can think of the transitions in a NFA as being disjunctions:

$$\delta(p, a) = q_1 \lor q_2$$

We can pick q_1 or q_2 to continue. Similarly, in a ∀FA, we are dealing with conjunctions:

$$\delta(p, a) = q_1 \land q_2$$

meaning: We must continue at q_1 and at q_2. So how about

$$\delta(p, a) = (q_1 \lor q_2) \land (q_3 \lor q_4)$$

Or perhaps

$$\delta(p, a) = (q_1 \lor \neg q_2) \land q_3$$

Does this make any sense?

Threads

Think of threads: both ∧ and ∨ correspond to launching multiple threads. The difference is only in how we interpret the results returned from each of the threads.

For ¬ there is only one thread, and we flip the answer bit.

In other words, a "Boolean" automaton produces a computation tree rather than just a single branch. This is actually not much more complicated than an NFA.

For historical reasons, these devices are called alternating automata.

Alternating Automata

In an alternating finite automaton (AFA) we admit transitions of the form

$$\delta(p, a) = \varphi(q_1, q_2, \ldots, q_n)$$

where φ is an arbitrary Boolean formula over Q, even one containing negations.

How would such a machine compute? Initially we are in "state"

$$q_{01} \lor q_{02} \lor \ldots \lor q_{0k}$$

the disjunction of all the initial states.

One Step and Acceptance

Suppose we are in state Φ, some Boolean formula over Q. Then under input a the next state is defined by substituting formulae for the variables:

$$\Phi[\, p_1 \mapsto \delta(p_1, a), \ldots, p_n \mapsto \delta(p_n, a) \,]$$

Thus, all variables $q \in Q$ are replaced by $\delta(q, a)$, yielding a new Boolean formula. In the end we accept if

$$\Phi[F \mapsto 1, \overline{F} \mapsto 0] = 1$$

Exercise

Verify that for NFA and ∀FA this definition behaves as expected.

Why Alternating?

The name “alternating automaton” may sound a bit strange.

The original paper by Chandra, Kozen and Stockmeyer that introduced these machines in 1981 showed that one can eliminate negation without reducing the class of languages.

One can then think of alternating between existential states (at least one spawned process must succeed) and universal states (all spawned processes must succeed).

In a moment, we will apply this idea of alternation to Turing machines.

nil novis sub solem

Theorem

Alternating automata accept only regular languages.

Proof.

Let Bool(Q) be the collection of all Boolean formulae with variables in Q and Bool$_0(Q)$ a subset where one representative is chosen in each class of equivalent formulae.

Choose a map $\mathrm{norm} :$ Bool$(Q) \to$ Bool$_0(Q)$; say, $\mathrm{norm}(\varphi)$ is the length-lex minimal formula in DNF equivalent to φ.

We can build a DFA over the state set Bool$_0(Q)$, hence the DFA has state complexity at most 2^{2^n} (the accessible part may be smaller, of course).

The DFA

- The initial state is $\mathrm{norm}(\bigvee_{q \in I} q)$.
- Transitions are $\Delta(\boldsymbol{p}, a) = \mathrm{norm}(\boldsymbol{p} \mapsto \delta(\boldsymbol{p}, a))$.
- The final states are $\{\, \varphi \in \text{Bool}_0(Q) \mid \varphi[F \mapsto \text{tt}, \overline{F} \mapsto \text{ff}] = \text{tt} \,\}$.

It is easy to see that the new, ordinary DFA is equivalent to the given, alternating one.

□

But note that the cost of eliminating alternation is potentially doubly exponential, significantly worse that for determinization (corresponding to logical-or only).

Why Would Anyone Care?

Because an AFA can be much, much smaller that the minimal DFA. In fact, the 2^{2^n} bound is tight: there are AFAs on n states where the minimal equivalent DFA is doubly exponential in n.

So we have a succinct representation for a regular language, but one that still behaves reasonably well under the usual algorithms. Avoiding the standard DFA representation is often critical for feasibility: in reality we cannot actually construct the full DFA in many cases. Laziness is a good idea in this context.

BTW, this is even true in pattern matching: DFAs should be avoided unless they are absolutely necessary (because the pattern contains a negation).

Alternating Turing Machines (ATM)

So we understand alternating finite state machines, but we really need to talk about alternating Turing machines.

As before, we assume there are two transition functions

$$\delta_i : Q \times \varSigma \to Q \times \varSigma \times \{-1, 0, 1\}$$

$i = 0, 1$ that we can choose either one at each step.

Here is the critical idea: every state is labeled by $\exists$ or $\forall$.

These labels are used in the definition of acceptance.

The notion of alternation comes from the fact that one could alternate between $\exists$ and $\forall$ states in a computation.

The Computation Graph

Suppose $\mathcal{M}$ is an ATM and let $\mathfrak{C}(\mathcal{M}, x)$ be the computation graph defined as usual.

Now define the following class of “accepting nodes” in $\mathfrak{C}(\mathcal{M}, x)$:

- Accepting configurations are accepting nodes.
- If state p in configuration C is labeled $\exists$, and there is a successor of C that is accepting, then C is also accepting.
- If state p in configuration C is labeled $\forall$, and all the successors of C are accepting, then C is also accepting.

We say that $\mathcal{M}$ accepts x if the initial configuration q_0x is accepting.

We can think of each path in the computation graph as a thread, operating under the usual rules.

ATIME

Given an ATM $\mathcal{M}$, we say that $\mathcal{M}$ runs in alternating time $t(n)$ if every path in the computation graph $\mathfrak{C}(\mathcal{M}, x)$, $|x| = n$, has length at most $t(n)$.

Similarly we define alternating space.

In symbols: ATIME(t) and ASPACE(s)

Also set AP = ATIME(poly) and ALOG = ASPACE($\log$).

The big question is how these alternating polynomial time/space classes relate to ordinary complexity classes. We will show in particular

Lemma

ALOG $= \mathbb{P}$ *and* AP $= \mathrm{PSPACE}$

This will follow easily from the results below.

Alternation versus Plain

Theorem

Suppose $f(n) \geq n$ is reasonable. Then

$$\text{ATIME}(f) \subseteq \mathrm{SPACE}(f) \subseteq \text{ATIME}(f^2)$$

Theorem

Suppose $f(n) \geq \log n$ is reasonable. Then

$$\text{ASPACE}(f) = \mathrm{TIME}(2^{O(f)}).$$

Proof 1

ATIME$(f) \subseteq \mathrm{SPACE}(f)$

Suppose $\mathcal{M}$ is alternating, $O(f)$ time. Construct a simulator $\mathcal{M}'$ that performs a DFS traversal of the computation tree $\mathfrak{T}_{\mathcal{M}}(x)$ of $\mathcal{M}$ on input x to determine acceptance.

We have to make sure that $\mathcal{M}'$ needs only $O(f)$ deterministic space. The naive approach would produce a recursion stack of depth $O(f)$, with stack frames of size $O(f)$, yielding space complexity $O(f^2)$.

To avoid this, simply record the nondeterministic choice at each step (a single bit), and recompute the actual configuration when needed (we are focused on space, not time).

Proof 2

$\mathrm{SPACE}(f) \subseteq$ ATIME(f^2)

This is similar to the approach in Savitch's theorem. Instead of the plain recursion used there, our alternating machine will branch existentially to find the "middle point", and then universally to verify both pieces.

We check for paths of length $2^{cf(n)}$ where c is large enough so this number bounds the total number of configurations. Hence the total damage is $O(f^2)$ alternating time.

Proof 3

ASPACE$(f) \subseteq \mathrm{TIME}(2^{O(f)})$

Suppose $\mathcal{M}$ is alternating, $O(f)$ space. Construct a simulator $\mathcal{M}'$ that builds the computation graph $\mathfrak{C}_{\mathcal{M}}(x)$ of $\mathcal{M}$ on input x, first ignoring alternation. Nodes have size at most $cf(n)$ for some constant c.

Then run a marking algorithm that labels nodes as accepting when appropriate, working backwards from the leaves. Accept x if the initial configuration is ultimately marked.

The size of the graph is $2^{O(f)}$, and one round of marking is linear in the size. There are at most $2^{O(f)}$ rounds, yielding our deterministic time bound.

Proof 4

$\mathrm{TIME}(2^{O(f)}) \subseteq$ ASPACE(f)

This time $\mathcal{M}$ is an ordinary deterministic machine, time $2^{O(f)}$. The alternating machine $\mathcal{M}'$ must obey a $O(f)$ space bound, so we cannot simply construct a computation graph. This is rather tricky, here is a sketch of the proof.

Think of the computation of $\mathcal{M}$ on input x as being laid out in a $N \times N$ square, $N = 2^{O(f)}$, much as in the homework problem on square tilings (except here we don't have to bother with tilings, each row will just be a configuration represented as a word in $\Sigma^\star Q \Sigma^\star$). The square itself is too large, but we can keep pointers to individual cells.

We guess the position of the accepting state in (i, t). Then, for all $s = t, t-1, \ldots, 1$, we guess existentially the three cells in the row below, check that they are good, and universally verify that the parents are also good. In the bottom row we verify against the initial configuration of $\mathcal{M}$.

Close, but no Cigar

It seems plausible that $\mathrm{PH} \subsetneq \mathrm{PSPACE}$, so our ATMs are a bit too powerful to characterize PH.

To fix this, we can impose more constraints: call an ATM Σ_k if the initial state is labeled $\exists$, and every path in $\mathfrak{C}(\mathcal{M}, x)$ alternates at most $k-1$ times between different labels. This produces a notion of $\Sigma_k\mathrm{TIME}$.

$$\Sigma_k^p = \bigcup \Sigma_k\mathrm{TIME}(n^c).$$

It is unbounded alternation that seems to push us a bit too far.

Oracles, Again

Kozen has shown that, for every k, one can construct an oracle A such that

$$(\Sigma_k^p)^A \neq (\Sigma_{k+1}^p)^A = \mathrm{PSPACE}^A$$

Very little is known about collapsing oracles à la Baker-Gill-Solovay. For example, it is an open problem whether

$$\mathrm{Pr}_A[\mathbb{P}^A = \mathbb{NP}^A] = 1.$$

The BGS theorem just shows that there is some A for which equality holds, and another for which it fails.