You're Too Slow!

The Case for Constant-time Algorithms

Michael Coulombe

Electrical Engineering and Computer Science Department
Massachusetts Institute of Technology
32 Vasser Street, Cambridge, MA
mcoulomb@mit.edu

Abstract

Constant-time algorithms underlie all computation, but are often discarded as mere building blocks for solving inherently-slow problems. In this paper, we demonstrate the astonishing power of various deterministic and nondeterministic models when constrained to $O(1)$ time.

Keywords Theoretical Computer Science, Complexity Theory, Constant Time, Turing Machines, Word RAM, Complexity Classes

1. Introduction

Why study constant-time problems and algorithms?

1. They are the fundamental building blocks of all algorithms.
2. They are practical and scalable beyond compare.
3. If we don't do it now, someone else will scoop us first [RBV17].

1.1 Background

Complexity theory studies how many resources it takes to solve computational problems, most commonly running time [HS65]. The most famous is the P vs NP problem, which even has a million dollar prize for a solution. In this paper, we advance the field by investigating the underappreciated class of problems that run in constant time, in both deterministic and non-deterministic models.

Thanks for reading this far into our paper! As as token of our appreciation, here is a useful fact that we use to simplify our proofs:

Lemma 1.1. *$f(n) \in O(1)$ iff $\exists c. \forall n \in \mathbb{N}. f(n) \le c$.*

Proof. If we only know c, N where $\forall n \ge N. f(n) \le c$, then take

$$c' = \max\left(\{f(n) \mid 0 \le n < N\} \cup \{c\}\right)$$

and we get the upper bound $f(n) \le c'$ for all n. □

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, contact the Owner/Author. Request permissions from permissions@acm.org or Publications Dept., ACM, Inc., fax +1 (212) 869-0481. Copyright 2023 held by Owner/Author. Publication Rights Licensed to ACM.

SIGTBD 2023 April 7st, 2023, Cambridge, Massachusetts, USA.
Copyright © 2023 ACM . . . not really, just the Author (April Fools!). . . $15.00
DOI: http://dx.doi.org/10.1145/(to come)

2. Turing Machine Model

We give a full characterization of constant-time Turing Machines.

Definition 2.1. FIN is the set of finite languages.

Definition 2.2. The concatenation of two sets of languages C_1, C_2 is denoted as $C_1 \circ C_2 = \{L_1 \circ L_2 \mid L_1 \in C_1, L_2 \in C_2\}$.

Definition 2.3. The addition of two sets of languages C_1, C_2 is denoted as $C_1 + C_2 = \{L_1 \cup L_2 \mid L_1 \in C_1, L_2 \in C_2\}$.

Theorem 2.1. $\text{TIME}(1) = \text{FIN} + (\text{FIN} \circ \{\Sigma^*\})$.

Proof. We show each direction separately.

($\subseteq$) Take any $L \in \text{TIME}(1)$ solved by TM M in $f(n) = O(1)$ time, meaning $\exists c. \forall n.\ f(n) \le c$ by Lemma 1.1.
When M runs on any w, it takes at most c steps, thus it can only accept or reject based on the first c characters of w. Thus, for any w_c where $|w_c| = c$ and $w_1, w_2 \in \Sigma^*$, we have $w_c w_1 \in L \iff w_c w_2 \in L$. However, if $|w| < c$, then M has time to detect that w has less than c characters and accept or reject based on the whole of w.
Thus, we have sets $S = \{w \in L \mid c > |w|\}$ and $B = L \setminus S$, so $L = S \cup B \in \text{FIN} + \text{FIN} \circ \{\Sigma^*\}$

($\supseteq$) Take any $L \in \text{FIN} + \text{FIN} \circ \{\Sigma^*\}$, meaning $L = S_1 \cup (S_2 \circ \Sigma^*)$ for $S_1, S_2 \in \text{FIN}$. Clearly, a deterministic TM can decide if $w \in L$ using brute-force, by testing if $w \in S_1$ or if w starts with any $w' \in S_2$. Since they are finite sets of finite strings, it will take finite time, so $L \in \text{TIME}(1)$. □

Theorem 2.2. $\text{TIME}(1) = \text{NTIME}(1)$

Proof. Take any $L \in \text{NTIME}(1)$ solved by NTM N in at most $c \in \mathbb{N}$ steps. We can create a deterministic TM M that decides if $w \in L$ by simulating all $O(|N|)^c = O(1)$ possible paths of the nondeterminism of N on w, thus $L \in \text{TIME}(1)$. The other direction is obvious. □

3. Word RAM Model

Definition 3.1 (The Word RAM Model)**.** Memory is an unbounded array of words, each with $\omega \ge \log n$ bits on inputs of size n. The machine has a constant number of registers and can perform operations on registers and access memory by index. Initially, one register contains n and all memory beyond the input is $\perp$.

Definition 3.2. $L \in \text{RAMTIME}(f(n))$ iff L is decided by a deterministic Word RAM algorithm in time $O(f(n))$.

Definition 3.3. $L \in \text{NRAMTIME}(f(n))$ iff L is accepted by a nondeterministic Word RAM algorithm in time $O(f(n))$, counting the duration of the longest path.

Definition 3.4. $L \in \text{coNRAMTIME}(f(n))$ iff $\overline{L}$ is accepted by a nondeterministic Word RAM algorithm in time $O(f(n))$, counting the duration of the longest path.

Table 1 details the notation for the ***C**onstant Word **R**AM **T**ime* complexity classes we will be focusing on in this paper.

Shorthand	Class
CRT	RAMTIME(1)
NCRT	NRAMTIME(1)
coNCRT	coNRAMTIME(1)

Table 1. Constant-time Word RAM complexity classes.

3.1 The Power of Word RAM

We show that Word RAM allows you to solve more problems in constant-time compared to Turing Machines.

Theorem 3.1. $\text{TIME}(1) \subset \text{CRT}$.

Proof. Just simulate the constant-time Turing Machine. □

Lemma 3.2. $\Sigma^* 1 \in \text{CRT}$

Proof. Just check the last word and see if it is a 1. □

Lemma 3.3. $\Sigma^* 1 \notin \text{TIME}(1)$

Proof. For any Turing Machine M that runs in at most $c \geq 1$ time, M cannot distinguish between $0^{2c}0$ and $0^{2c}1$. □

Theorem 3.4. $\text{TIME}(1) \neq \text{CRT}$.

Proof. By Lemmas 3.2 and 3.3. □

3.2 The Power of Non-determinism

We show that non-determinism greatly improves your power in the constant-time regime.

Lemma 3.5. $\text{CRT} \subset \text{NCRT}$.

Proof. Just don't do non-deterministic things. □

Lemma 3.6. $\overline{0^*} \in \text{NCRT}$.

Proof. Given a w of length n, guess the index $i \in [0, n)$ of the non-zero then do a single read and comparison to verify the answer. □

Lemma 3.7. $\overline{0^*} \notin \text{CRT}$.

Proof. Suppose for the sake of contradiction that an deterministic algorithm M decides $\overline{0^*}$ in at most c steps. On input $w = 0^n$ for $n > c$, M reads at most c words from the input then accepts w, so its answer doesn't depend on the values in $n - c > 0$ positions. If we create w' to be w except all of those positions are 1, M will behave the same and accept w', which is incorrect. Due to this contradiction, M cannot exist. □

Theorem 3.8. $\text{CRT} \neq \text{NCRT}$.

Proof. By Lemma 3.5, non-determinism isn't a weakness, and by Lemmas 3.6, and 3.7, deciding $\overline{0^*}$ in constant time requires non-determinism. □

3.3 Importance of Knowing the Input Length

Lemma 3.9. *If the input size n is not known initially, then non-determinism is required to compute n in constant time.*

Proof. To compute n with non-determinism, we simply guess n then verify that index n contains $\perp$ but the previous index does not.

For the sake of contradiction, suppose a deterministic M can compute n in at most c time. Let $i_1, i_2, \ldots, i_c$ be the indices that M reads when each location returns an initial value of 1, and let $i = \max\{i_j \mid 1 \leq j \leq c\}$. If M on input 1^i correctly determines that it has length i, then it will also incorrectly return i on all longer inputs 1^{i+m} for $m > 0$. By this contradiction, M cannot exist. □

Definition 3.5. $L \in \text{RAMTIME-NO-N}(f(n))$ if it is decided by a deterministic Word RAM algorithm in time $O(f(n))$ without knowing n in advance. Denote RAMTIME-NO-N(1) as CRT-NO-N.

Theorem 3.10. $\text{TIME}(1) = \text{CRT-NO-N}$ *on a finite alphabet* Σ.

Proof. First, you don't need to know the input length to simulate a Turing Machine. Second, for $L \in \text{CRT-NO-N}$, suppose algorithm R decides L in at most c steps without knowing n.

Since in c steps, R can only access c indices of memory, R can only distinguish between at most $|\Sigma \cup \{\perp\}|^c = O(1)$ equivalence classes of strings which share the same values at the accessed indices. Identify these classes as C_s where s is the shortest string in C_s, and let m be the longest length of any s, thus also the largest index that R will ever access.

We can therefore define a Turing Machine M that decides $x \in L$ by simulating R running on the first m characters of the input in at most $cm = O(1)$ steps. □

3.4 Closure

Lemma 3.11. CRT *and* NCRT *are both closed under union and intersection.*

Proof. We are given M_1, M_2 deciding L_1, L_2 in at most c_1, c_2 time. On input w, we run M_1 on w (being sure to reset memory afterwards) then M_2 on w. If we are deciding union, we accept if either accepted, and for intersection we accept if both accepted. Total time is $O(c_1 + c_2) = O(1)$. □

Definition 3.6. $\text{MIDDLE-ELEVEN} = \{x11y \mid x, y \in \Sigma^n\}$.

Lemma 3.12. $\text{MIDDLE-ELEVEN} \in \text{CRT}$.

Proof. Just divide n by 2 (such as by bit-shifting) to get the middle indices then read those words to verify they are 11. □

Lemma 3.13. CRT *is* not *closed under concatenation.*

Proof. We show MIDDLE-ELEVEN ∘ MIDDLE-ELEVEN is not in CRT, which alongside Lemma 3.12 proves the claim. This new language is equivalent to the following:

$$\{x11yu11v \mid x, y \in \Sigma^n \text{ and } u, v \in \Sigma^m\}$$

Like Lemma 3.15, for the sake of contradiction, let M decide it in at most c time, and take any $w_{n,m} = 0^n 110^{n+m} 100^{1+m}$ that M must reject. Since M can only read at most c positions of $w_{n,m}$, it must erroneously accept one of $w_{n+i,m-i}$ for $i \in [0, 2c]$ and $n, m \geq 3c$ to distinguish it from $0^n 110^{n+m} 110^m$. □

Lemma 3.14. NCRT *is closed under concatenation.*

Proof. We are given N_1, N_2 deciding L_1, L_2 in at most c_1, c_2 time. To decide $w \in (L_1 \circ L_2)$ of length n, we let $w = w_1 w_2$, guessing the length $|w_1| \in [0, n)$, and then run N_1 on w_1 (being sure to reset the at-most-c_2 words of memory used afterwards) then N_2 on w_2 (being sure to account for the offset in memory) to verify that $w_1 \in L_1$ and $w_2 \in L_2$. Total time is $O(c_1 + c_2) = O(1)$. □

Lemma 3.15. $0^* \notin$ NCRT

Proof. Similarly to Lemma 3.7, suppose for the sake of contradiction that N decided 0^* in at most c time.

On any input $w_{n,m} = 0^n 1 0^m$, on every non-deterministic path N must reject. However, in order to distinguish $w_{n,m}$ from 0^{n+1+m}, N must read the 1 in the middle on every path. Since N can only read from c indices at once on a single path, N must erroneously accept on one of $w_{n+i,m-i}$ for $i \in [0, 2c]$ and $n, m \geq 3c$. By this contradiction, N cannot exist. □

Since clearly the single-string language $0 \in$ NCRT, we get the following corollary.

Corollary 3.16. NCRT *is* not *closed under star.*

Theorem 3.17. NCRT *is* not *closed under complement.*

Proof. By Lemmas 3.6 and 3.15. □

3.5 Regularity

We show that the regular languages aren't a good model for constant-time algorithms.

Theorem 3.18. CRT *and* NCRT *on finite* Σ *are incomparable with the regular languages.*

Proof. By Lemma 3.15, the regular $0^* \notin$ NCRT, and by Lemma 3.12, MIDDLE-ELEVEN $\in$ CRT even though it is not regular. □

3.6 Context-freedom

We show that some non-context-free languages can be decided in constant-time.

Definition 3.7. POW-TWO $= \{\Sigma^n \mid n = 2^m, m \in \mathbb{N}\}$, which is clearly not context-free.

Lemma 3.19. POW-TWO $\in$ CRT.

Proof. Just test if n has only one bit set. □

3.7 Hardness

Definition 3.8. B is CRT-hard if any language $A \in$ CRT can be reduced to B by a constant-time transducer, denoted $A \leq_1 B$. We define NCRT-hard similarly. B is complete if it is also a member of the class.

Definition 3.9. $\text{ACCEPT}_{\text{RAM}}$ is the following language:

$$\{(x, M, c) \mid \text{RAM algorithm } M \text{ accepts } x \text{ in at most } c \text{ time}\}$$

Theorem 3.20. $\text{ACCEPT}_{\text{RAM}}$ *is* CRT*-hard.*

Proof. Take any $A \in$ CRT, which is decided by a deterministic algorithm M that runs in at most c time. We can decide if M accepts x in at most c time by testing if $(x, M, c) \in \text{ACCEPT}_{\text{RAM}}$, using the transducer which simulates a longer input where M and c are written after x in memory. □

Similarly, we get the following analogous results.

Corollary 3.21. $\text{ACCEPT}_{\text{NRAM}}$ *is* NCRT*-hard.*

Corollary 3.22. $\text{ACCEPT}_{\text{coNRAM}}$ *is* coNCRT*-hard.*

Conjecture 3.1. $\text{ACCEPT}_{\text{RAM}}$ *is not in* CRT.

Conjecture 3.2. *Nothing is* CRT*-complete.*

3.8 Characterization

Theorem 3.23. NCRT *and* coNCRT *are incomparable language classes.*

Proof. By Lemmas 3.6 and 3.15 and definition of "co-", we have that $0^* \notin$ NCRT so $\overline{0^*} \notin$ coNCRT, and that $\overline{0^*} \in$ NCRT so $0^* \in$ coNCRT. □

4. Constant Time with Oracles

4.1 Secondary Tape Model

Classic objects of study for computational complexity are so-called *oracle* machines, where a machine is augmented with an additional tape to write queries to some other language, which are answered magically in one step. However, these are entirely insufficient for studying constant-time problems, as seen by the following results.

Lemma 4.1. *A Turing machine augmented with a secondary tape of constant size can be simulated by a single-tape Turing machine with no time overhead.*

Proof. Store the secondary tape in the state. □

Theorem 4.2. $\text{TIME}(1)^B = \text{TIME}(1)$ *for all languages* B.

Proof. Let M^B be an oracle Turing machine that decides A in at most c steps, including oracle steps.

If M^B makes any queries to $x \in B$, then it must be that $|x| \leq c$ since there isn't enough running time to write down more symbols on the oracle tape. Thus, M^B is equivalent to $M^{B'}$ where the finite language $B' = \{x \mid x \in L, |x| \leq c\}$.

Since $B' \in \text{FIN} \subseteq \text{TIME}(1)$, there is some Turing Machine which decides B' in at most d steps, thus we can replace the oracle with an concrete algorithm that runs on a c-sized secondary tape in $O(dc)$ steps. By Lemma 4.1, we can simulate this with a single-tape Turing machine with no overhead, therefore $A \in \text{TIME}(1)$. □

The same technique holds for all other models: in constant time, only constant-size queries can be written on a secondary tape/memory, so all oracles are equivalent to oracles for finite languages and thus can be replaced by constant-time algorithms.

Corollary 4.3. $\text{NTIME}(1)^B = \text{NTIME}(1)$ *for all languages* B.

Corollary 4.4. $\text{CRT}^B = \text{CRT}$ *for all languages* B.

Corollary 4.5. $\text{NCRT}^B = \text{NCRT}$ *for all languages* B.

Corollary 4.6. $\text{coNCRT}^B = \text{coNCRT}$ *for all languages* B.

4.2 Transducer Model

To study oracles in a non-trivial way, we must abandon the oracle tape model and find a way to give the oracle super-constant-sized inputs without having to write them down explicitly. Our solution is an oracle-oracle, an oracle that we give to the oracle that answers queries about which symbols are in the query we want to ask.

Definition 4.1. $\text{TRAMTIME}(g(n))$ is the class of functions $f(x) = y$ for which there is a Word RAM transducer t such that computing $t(x, i) = y_i$ takes $O(g(n))$ time.

Definition 4.2. An *oracle-oracle machine* M^B is a Word RAM machine with an oracle O_B for language B, where queries are made by providing O_B with an oracle f_x for the symbols of the query x, implemented as a transducer with read-only access to memory.

Definition 4.3. $\text{RAMTIME}(f(n))\triangle^{C}_{\text{TRAMTIME}(g(n))}$ is the class of languages decided by oracle-oracle machines that run in $O(f(n))$ time, are augmented with oracle-oracles for language or complexity class C, and use transducers that run in $O(g(n))$ time. Similar oracle-oracle classes are defined and notated similarly.

As a sanity check, we see the following result:

Lemma 4.7. *$A \in \text{CRT}\triangle^{A}_{\text{TRAMTIME}(1)}$ for any language A.*

Proof. We create an oracle-oracle machine M^A which on input x queries $x \in A$ with the identity transducer. M^A just makes one query and mirrors the result, and the transducer takes $O(1)$ steps to read and return the value at a given index, so M^A decides A and we get that $A \in \text{CRT}\triangle^{A}_{\text{TRAMTIME}(1)}$. □

5. Finest-Grained Complexity

The often-exciting field of "Fine-grained" complexity claims to study the exact amount of time required to solve problems, though often results are still limited to distinguishing between asymptotic bounds like $O(n^2)$ vs $\Omega(n^3)$.

In this section, we consider truly-fine-grained complexity and arrive at stunning and unconditional hierarchy results.

Theorem 5.1. *Define $L_t = \{w \mid w \text{ starts with } t \text{ ones}\}$. For all $t \in \mathbb{N}$, L_t can be decided in at most t steps, and requires at least t steps in the worst case, on a Turing machine.*

Proof. Let algorithm A_t implement the regular expression $1^t\Sigma^*$:

$$Q = \{q_0, q_1, \cdots, q_{t-1}, q_{\text{ACCEPT}}, q_{\text{REJECT}}\}$$
$$\delta(q_i, 1) = (q_{i+1}, 1, \text{R}) \text{ if } i < t-1$$
$$\delta(q_{t-1}, 1) = (q_{\text{ACCEPT}}, 1, \text{R})$$
$$\delta(q_i, \sigma) = (q_{\text{REJECT}}, \sigma, \text{R}) \text{ for all } \sigma \neq 1$$

Any Turing machine algorithm that takes fewer than t steps cannot distinguish between 1^t and $1^{t-1}0$ since there is not enough time to reach the end of the input. □

Theorem 5.2. *Define $L_t = \{w \mid w \text{ starts with } t \text{ ones}\}$. For all $t \in \mathbb{N}$, L_t can be decided in at most $c_1 t + c_2$ steps, and requires at least t steps in the worst case, in (non)deterministic Word RAM for some constants $c_1, c_2 \in \mathbb{N}$.*

Proof. Let algorithm A_t simulate the Turing machine algorithm from Theorem 5.1. No matter how you count steps, it is linear in the constant t. If some Word RAM algorithm could always take fewer than t steps, then it cannot distinguish between 1^t and 1^i01^{t-1-i} for some $i \in [0, t]$ since it doesn't have time to read all of the first t symbols. □

6. Pushing the Boundaries of Constant

We have so far been using the following definition of constant time:

Definition 6.1. An algorithm runs in *strongly constant time* if its runtime is $T(n) = O(1)$ unconditionally.

In this section, we extend traditional Word RAM variations to the constant time regime.

6.1 Pseudo-Constant Time

Here, we allow the runtime to be super-constant with respect to the numeric values in the input.

Definition 6.2. An algorithm runs in *pseudo-constant time* if its parameterized runtime is $T_V(n) = f(V)$ where V is the maximum value in the input and f is constant with respect to n.

Lemma 6.1. *Factoring one word takes pseudo-constant time.*

Proof. The trial division algorithm on an integer x that fits into one word runs in $O(x)$ time, which does not vary with respect to the input size $n = 1$ (notably, 1 cannot vary). □

Theorem 6.2. *For any decidable language L, the following language*

$$L' = \{(n, x) \mid x \in L, n = |x|\}$$

is pseudo-constant time.

Proof. Let M decide L in at most $T(n)$ time (monotonic). The maximum word in the input V is at least n, so there is an M' which ignores n and runs M on x will run in pseudo-constant $O(T(n)) = O(T(V)) = T_V(n)$ time. □

6.2 Weakly Constant Time

Now, we will consider the case where the runtime can be super-constant with respect to the word size ω.

Definition 6.3. An algorithm runs in *weakly constant time* if its parameterized runtime is $T_\omega(n) = f(\omega)$ where ω is the number of bits per word and f is constant with respect to n.

Theorem 6.3. *The languages of deterministic Linear Bounded Automata are decidable in weakly constant time.*

Proof. Given a dLBA M for a language L in n space and $T(n)$ time, we can simulate it in Word RAM on input x with length n in its size $2^\omega \geq n$ memory and in time $s(T(n)) = O(s(T(2^\omega))) = T_\omega(n)$ time for some simulation overhead function s. □

Corollary 6.4. *The languages of non-deterministic Linear Bounded Automata, the Context-Sensitive Languages, are recognizable in weakly constant non-deterministic time.*

7. Conclusion

We have introduced a variety of constant-time models of computation, and solved many of the most critical questions in complexity theory with regards to them.

However, there is still much room for future work. Here are some promising avenues of inquiry:

- Randomized algorithms, such as expected constant time and always correct, or worst-case constant time and high probability
- Approximation algorithms: what are optimal approximation factors achievable in constant time for various problems?
- Parallel algorithms: what can multiple processors or distributed agents compute in constant time?
- Cryptography: what encryption is optimal against constant-time adversaries? What problems have zero-knowledge proofs for constant-time verifiers? Best protocols for verifiable delegation of constant-time computation?
- Quantum algorithms, such as constant-depth quantum circuits or a classical machine with a constant number of qubits
- Generative AI: forget Large Language Models, what about Constant-size Language Models?

References

[HS65] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. volume 117, pages 285–305, 1965.

[RBV17] Oscar Reparaz, Josep Balasch, and Ingrid Verbauwhede. Dude, is my code constant time? In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1697–1702, 2017.

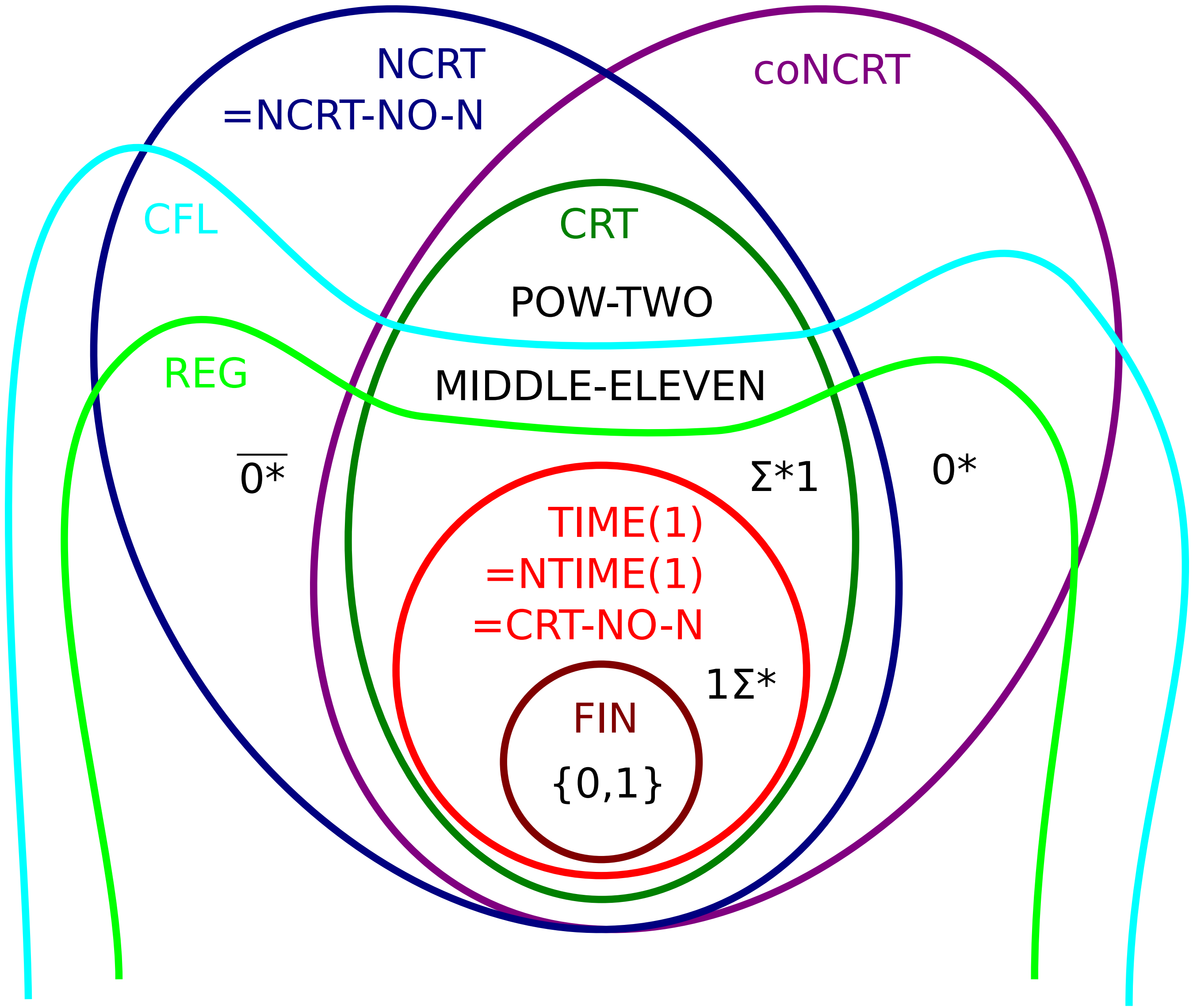

Figure 1. Venn Diagram of our results.