DLCVM

Generalized, Unboundedly Scalable Computation on Bitcoin

Allen Farrington
allen@axiombtc.capital

March 2024

Abstract: DLCVM is a computing paradigm to generalize the domain of computable functions the output of which can determine the transfer of Bitcoin, and to align the computational cost with the deflationary trend of Moore's Law rather than the inflationary trend of competition for block space. This effectively extends the expressivity of Bitcoin Script yet requires no changes to the network's consensus rules.

The proposal works as follows: for any computable function with maximum *n* possible outputs, we construct a Discreet Log Contract ("DLC") to be entered into as a Bitcoin transaction with *n* possible spend paths. Separately, we arrange for an oracle to attest to the result of this computation. By this paradigm, Bitcoin transactions can be constructed to depend on the outcome of any arbitrary computation without requiring that this computation be performed by the network. The only constraint is on a finite and predetermined number of potential outputs, but not on runtime; the computation can run unboundedly in computational resources with no correlated onchain footprint, so long as the offchain physical resources are willingly provided by the parties involved, ideally in a permissionless and decentralized manner.

We posit further that, i) DLCVM generalizes even the concept of a *decentralized VM*, given *any* VM can be used to perform the computations. DLCVM may, therefore, be thought of as not only a virtual machine, but also as a *virtual* virtual machine; and, ii) DLCVM spend paths can be constructed as DLCVMs and "chained" together, enabling a generalization of state channels we call a "virtual network." Given DLCVM enables any computation, by any computer, real or virtual, to determine the outputs to a Bitcoin transaction (the most robust and valuable decentralized asset there is) and to chain such computers together in a virtual network of virtual machines, it follows that DLCVM is, in a relatively strict theoretical sense, the most powerful virtualization scheme that could possibly exist.

1. What Is A Computer?

The Oxford English Dictionary defines a "computer" as,

"A device for performing calculations automatically; originally and chiefly: an electronic device which is capable of receiving, storing, manipulating, and outputting data in accordance with a predetermined program or sequence of instructions."

While helpful for general comprehension, for our purposes this immediately raises questions such as, *what kind of device? what kind of calculations? what kind of data?* and, *what is meant by "manipulating" and "automatically"? Turing Machines*, *Turing Completeness*, and *Computable Functions*, or their equivalent characterizations in terms of λ-calculus or μ-recursion, are typically used to explicate *what kind of computations*. Informally, they mean something like the minimum mathematical tooling needed to compute anything computable, provided with access to an unbounded memory of past states of computation, up to isomorphism. The concept of an *algorithm* both formalizes *manipulation* and captures a specific and purposeful instantiation of such manipulations that may in turn comprise part(s) of a *program*. Building on earlier work of Markov (1954) and Kleene (1967), Knuth (1973) characterizes an *algorithm* by the features of: *finiteness, definiteness, input, output,* and *effectiveness.* Respectively, these informally capture that: an algorithm must terminate after finitely many steps; every step in its execution must be precisely and unambiguously defined; it has zero or more inputs and outputs; and it must be possible in principle to carry out the operations manually.

In answering, *what kind of device,* and *what kind of data* – and mindful of the paradigm shift we will discuss momentarily – we are forced to reckon with the concept of a virtual machine ("VM"). Originally introduced by Popek and Goldberg (1974), the rough idea is to conceptually separate an operating system defined purely in terms of software (the *algorithm*) from the hardware (the *device*) on which it is intended to be executed, so as both to run multiple operating systems on the same hardware and to transplant the same operating system to different hardware.

2. Turing Completeness, Statefulness, and Virtual Machines

The publication of the Ethereum white paper (Buterin, 2014) and subsequent launch of the Ethereum network proved a milestone for the range of answers to the titular question of the preceding section, *what is a computer?* as well as the more specific question, *what is a VM?* Bitcoin had launched several years earlier and is alluded to where relevant by Buterin, but Bitcoin's novel programming language ("Script") used to propose and authenticate valid state transitions was deliberately constructed to be little more than a zero-order predicate verifier, to be minimally stateful, to have minimally expressive smart contracting capability (Szabo, 1994), *not* to be Turing Complete, and to have no VM of which to speak.

The Ethereum Virtual Machine ("EVM") on the other hand has provided rich material for novel interpretations and understandings of the fundamental nature of computation. The EVM exists on distributed physical devices of some subset of network participants. And yet, unlike what we might call "classical" transposable VMs, the computations executed are only capable of being understood as the operations of a single entity forged by synchronized adherence to consensus rules mutually enforced by multilateral incentives.

This is all well and good, but there are some valid critiques we can make of Ethereum. While in theory both the *language* of the EVM is Turing Complete and its statefulness can perhaps be described as "rich" (Buterin, 2017), in practice neither does the EVM operate as a Turing Machine nor can the states to which its programs have access scale unboundedly (Miller, 2016). While clearly no physical computer can *successfully* compute an infinite loop either (in the sense of *definitely* executing an algorithm and *effectively* producing an *output,* per Knuth) or store unbounded data in pursuit of executing even a *finite* algorithm, regular computers can continuously store and compute for as long as resources are provided and until such an algorithm is exogenously terminated. Ethereum, on the other hand, must provide an endogenous termination procedure lest the network immediately succumb to a DOS attack, which it does in the form of "gas" (op. cit. Buterin, 2014).

Given computation is an economic good – and *permissionless, distributed* computation presumably an extremely valuable good – we would ideally like continued capital accumulation, and technological innovation in general, to trigger compounding deflation. However, it is vital for ongoing consensus formation for the gas token to at least sustain speculative value with relative predictability, if not to appreciate. There seems to be a conflation between a fungible-seeming capital good and *money*. Menger (1892) canonically defines money as "the most saleable good," and while relatively scarce digital bearer credits for a consensus-entangled VM are surely *valuable,* it is unclear if or why they would ever be *saleable*.

And so, although practically necessary to ensure *finiteness, definiteness* and *automaticity,* the monetary role played by these credits in creating a marketplace forces modes of socioeconomic and technical consensus which cause the ideal of a VM to run into a number of difficulties. The state transition functions and their fields (Knuth's *inputs, outputs,* and *effectiveness*) have to adhere to this consensus. Updating the consensus mechanism is intrinsically at odds with the ideal of *distribution* of computation and its *definiteness*, although the Ethereum Foundation has historically been effective at managing this process.

Hence, even though we start with permissionless, distributed computation, the need to bend the range of *calculation* and *data* to that which enables the *definiteness* and *automaticity* of consensus formation required for Turing Complete expressivity and rich statefulness arguably pares back our ability to physically manifest a "real" VM within the confines of the distributed ledger to which its output is anchored. As Maxwell (2016) and O'Connor (2017) observe, we may as well have used a total functional programming language (Turner, 2004) given the complexity class of the problem space of verifying execution of total functions is anyways more limited than that required for Turing Complete expressivity. As introduced above, we term this situation "consensus entanglement," and suggest it appears intractable under prior paradigms of cryptoeconomic design.

Returning to Bitcoin may regain a suitably saleable *money*, but as is well understood, Script can only verify zero-order predicates, has minimal state, and has no real VM. That said, recent development efforts in Bitcoin have taken steps to address some of these points. For example, BitVM (Linus, 2023) has opened the door to the verification of decidable programs in Script. An example recently made popular on social media involves Chess being encoded in BitVM, which we know is decidable by Zermelo's Theorem (Zermelo, 1913). However, programs which loop unboundedly seem still to be intractable without contentious network upgrades. Without OP_CAT, for example, it seems unlikely Script could handle a simulation of *The Sims*, given, unlike Chess, *The Sims* is known to allow for infinite loops in some cases of ethically questionable play.

3. Scaling by Oraclizing Offchain Computation

The quantum of capital allocated to the Ethereum ecosystem surely suggests there is a market for, and some merit in, a distributed smart contracting VM. Hence, to avoid the issues identified above, chief amongst them VM-limiting consensus entanglement, we desire a means for onchain verification of functions operating on Bitcoin that are in principle *unknowably decidable* and the execution of which we can shift to a more economically sound and coherent computing domain. In the Discreet Log Contracts ("DLCs") white paper, Dryja (2018) succinctly addresses many of the concerns covered above, writing:

"*Two of the biggest hurdles to their implementation and adoption have been scalability of the smart contracts, and the difficulty in getting data external to the currency system into the smart contract.*"

Dryja goes onto describe a contract and signature scheme whereby an oracle attests to some real-world outcome; a range of partially signed transactions between counterparties Alice and Bob have already been created such that only the transaction corresponding to this outcome is ever "fully" signed, is ever therefore valid, and can be successfully broadcast to close out the "contract." The example in the white paper – and in some sense the canonical use case to date – is that of a floating exchange rate relative to Bitcoin (the monetary asset) such that the counterparties are effectively creating an onchain futures contract that is automatically executed when provided with the oracle's independent attestation.

We propose utilizing oracle attestations in DLCs with a different goal in mind: as the basis for a VM in Bitcoin. Consider that any computable function has a theoretical maximum of m possible effective outputs (Radó, 1962) and therefore a real maximum n, such that $n \leq m$. Hence, we first construct a DLC with n possible spend paths and arrange for an oracle to attest to the result of this computation. The model of blockchain-anchored, generalized computation that emerges we term the "DLCVM," and the individual contracts from which DLCVMs are built, "computationally unbounded contracts" or CUCs. How the oracle for a given CUC is to be implemented is left to the reader as it is outside the scope of this paper.

DLCVM provides several benefits over the prevailing paradigm of onchain computation. Smart contracts on Bitcoin can be constructed in this manner to depend on the outcome of any arbitrary computation that is entirely outside the constraints of network consensus. All we require is that the offchain resources are provided willingly by the parties involved. The *devices* and the *data* can be anything on which the counterparties agree. An unbounded range of independently parameterized Knuthian algorithmic design can be brought to bear, covering any degree (or not) of Turing Complete expressivity, computational complexity, decidability, statefulness, and the like, given the only bottleneck rests with verification that can be achieved in Bitcoin today, without a consensus upgrade. Freed from this constraint, computation becomes truly decentralized and permissionless. In contrast to more restrained "layer separation" (Buterin, 2019) we propose to have bypassed consensus entanglement with a novel cryptoeconomic design heuristic we instead term "radical separation." This paradigm also aligns the cost of computation as an economic good with deflationary trends in capital formation: most notably, Moore's Law, rather than the steady appreciation to be expected of well-functioning *money*.

Properly understood, we posit further that DLCVM not only generalizes the space of *algorithms* that can determine Bitcoin transactions, but that, in fact, DLCVM generalizes the concept of a distributed VM itself. The fact any *device* and *data* can be used to execute the algorithms means, by extension, that any VM can as well. We posit DLCVM can be understood not only as a virtual machine but also as a *virtual* virtual machine, which we term "metavirtualization."

4. Scaling by Recursive Virtualization

Supposing we construct every CUC spend path so as to up its own CUC (of which the same is true, and so on, *i*-many times), *including itself*, we may get several benefits: i) this implies offchain state transitions such that the onchain settlement can be pushed out to at least *i*-many state transitions later, hence *i* can be chosen to be arbitrarily high so as to effectively indefinitely defer settlement, both as a technical nuisance and an economic cost, ii) a given state transition to a new child CUC could be contingent on verifiable proof of such a transition in a separate DLCVM (and vice versa) such that atomic state swaps can be enforced, creating an analogue of a state channel network we term a "virtual network," and; iii) the maximum realistically desirable states of either one DLCVM or a network of such DLCVMs can be radically reduced by instead considering every plausible route through states, and building the spend paths to enable cycling around the network state space without ever closing onchain.

We believe this appropriately generalizes the concept of a "state channel" to a "virtual channel" given not only the state but all elements of computation – the state transition function itself, including its own data, its own future computational range, and, crucially, its own VM – are faithfully transferred across time and across execution counterparties without hitting the chain. This naturally gives rise to constructs we term "virtual state flow" and "metavirtual liquidity" and is arguably the truest manifestation of a virtual virtual machine: virtualized not only across computational paradigms but also across itself.

5. Applications and Further Research

We suggest, as a modest starting point, that any smart contract scheme in a cryptoeconomic system other than Bitcoin can now be replicated at a fraction of the cost and can operate on a more robust digital asset. Beyond this, there may be even more exciting applications in store. It would appear that a range of highly technical and controversial Bitcoin scaling proposals can be implemented today, from zk-STARKs to Simplicity to Two-Way Pegged Sidechains (or even three- or four-way, for the sake of argument). The risk and complexity of the network upgrades thought to be necessary to enable these, and others, can be *radically separated* from Bitcoin's consensus layer under the DLCVM paradigm. We can imagine further economic and technical benefits to be realized by embedding the CUCs constituting a DLCVM in a Lightning Channel (Le Guilly, 2022) rather than onchain. Equally, the DLC adaptor signature authorizing the correct execution could be reimagined as a FROST signature (Komlo & Goldberg, 2020), or even as the output of, or cryptographically contingent on, some other multi-party computational process. This process could potentially store and/or execute on a separate VM entirely (i.e. EVM, Cairo, Clarity, JVM, etc.). Finally, we anticipate OP_CTV would improve the efficiency of this proposal, but we haven't worked out the details.

Beyond these and other practical applications, we imagine avenues for further theoretical research into computability theory. We believe most of the results and argumentation herein can be explicated in terms of Post's Theorem, in particular due to our demonstrated links between recursive enumerability, oracles, and Turing Completeness. Consider also that Gödel's Second Incompleteness Theorem fundamentally limits the space of the provable decidability of the first-order predicate schemes within which we may want to compute. We proposed a means to engineer these offchain in order to move our execution domain away from simpler zero-order predicates, even though the constraints of this execution paradigm nonetheless impose computational cost. Hence, we leave open the prospect that robust second-order languages (and higher) may extend the computational power of DLCVM. Utilizing radical separation to break free of consensus entanglement may lay the groundwork for even more powerful programming paradigms to one day be brought to Bitcoin.

References

Buterin, Vitalik, 2014, *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*, available: https://ethereum.org/content/whitepaper/whitepaper-pdf/Ethereum_Whitepaper_-_Buterin_2014.pdf

Buterin, Vitalik, 2017, Reddit comment, available: https://www.reddit.com/r/btc/comments/6ldssd/comment/djt6opz/

Buterin, Vitalik, 2019, *Base Layers and Functional Escape Velocity*, available: https://vitalik.eth.limo/general/2019/12/26/mvb.html

Dryja, Thaddeus, 2018, *Discreet Log Contracts*, available: https://static1.squarespace.com/static/59aae5e9a803bb10bedeb03e/t/5a85cf21e4966bb735a9f757/1518718768309/discrete+log+contracts.pdf

Knuth, Donald Ervin, 1973, *The Art of Computer Programming*, Second Edition, Reading, MA: Addison-Wesley

Komlo, Chelsea & Goldberg, Ian, 2020, *FROST: Flexible Round-Optimized Schnorr Threshold Signatures*, available: https://eprint.iacr.org/2020/852.pdf

Le Guilly, Thibaud, 2022, *DLCs on Lightning*, available: https://medium.com/crypto-garage/dlc-on-lightning-cb5d191f6e64

Linus, Robin, 2023, *BitVM: Compute Anything on Bitcoin*, available: https://bitvm.org/bitvm.pdf

Markov, Andrey, 1954, *Theory of Algorithms*, Tr. Mat. Inst. Steklov 42, pp. 1–14. trans. by Edwin Hewitt in *American Mathematical Society Translations*, Series 2, Vol. 15 (1960)

Maxwell, Gregory, 2016, *Turing Completeness and State for Smart Contract*, BitcoinTalk post, available: https://bitcointalk.org/index.php?topic=1427885.msg14601127#msg14601127

Menger, Carl, 1892, *On the Origins of Money*, *Economic Journal* 2 (1892): 239-55; translation by C.A. Foley, available: https://cdn.mises.org/On%20the%20Origins%20of%20Money_5.pdf

Miller, Andrew, 2016, *Ethereum Isn't Turing Complete, and it Doesn't Matter Anyway*, available: https://www.youtube.com/watch?v=cGFOKTm_8zk

O'Connor, Russell, 2017, *Post's Theorem and Blockchain Languages*, available: https://www.youtube.com/watch?v=TGE6jrVmt_I

Popek, Gerald J. & Goldberg, Robert P., 1974, *Formal Requirement for Virtualizable Third Generations Architectures*, available: https://www.cs.cornell.edu/courses/cs6411/2018sp/papers/popek-goldberg.pdf

Radó, Tibor, 1962, *On Non-Computable Functions, Bell System Technical Journal*, 41 (3): 877–884, doi:10.1002/j.1538-7305.1962.tb00480.x

Szabo, Nick, 1994, *Smart Contracts*, available: https://nakamotoinstitute.org/smart-contracts/

Turner, D.A., 2004, *Total Functional Programming*, *Journal of Universal Computer Science*, 10(7): 751-768, doi:10.3217/jucs-010-07-0751

Zermelo, Ernst, 1913, *On an Application of Set Theory to the Theory of the Game of Chess*