Digital Platforms, Blockchains, and Financial Contracts*

Alexander Karaivanov
Department of Economics
Simon Fraser University

March 2024

Abstract

I analyze the mapping between fundamental elements of financial contracts and transactions – property rights, enforcement, commitment, information – and the algorithmic tools and constraints of digital technologies such as blockchain platforms and automated ('smart') contracts. I then describe and formalize the microeconomic foundations, algorithmic building blocks, and possible uses of blockchain digital platforms for implementing constrained-optimal multiperiod contracts in incomplete markets settings with information or enforcement frictions and as collateral mechanism supporting on-platform and off-platform transactions and payments. Specific economic applications are provided and analyzed.

*Correspondence address: akaraiva@sfu.ca. I thank David Andolfatto, Alex Berentsen and participants in the Society for Economic Dynamics conference and the Vienna Macroeconomics workshop for useful discussions and comments. Financial support from the Social Sciences and Humanities Research Council (SSHRC), grant 435-2018-0111 is gratefully acknowledged.

1 Introduction

Digital platforms and markets have grown rapidly with the recent technological advances in collecting, storing, indexing, searching, matching, and quick transmission of large amounts of granular data. Blockchain-based digital platforms are a fast growing segment of digital financial markets. On March 14, 2024 the two leading blockchain platforms, Bitcoin and Ethereum processed over 1.5M daily transactions and had market capitalizations of $1.4T and $465B respectively.

A blockchain is a decentralized and free to join and access networked computer database of transactions and other digital records (e.g., 'smart contracts' or 'tokens'), organized in a sequence of interconnected time-stamped data blocks. Network nodes (user accounts) hold or have access to a complete copy of the database, hence blockchain platforms are often referred to as 'distributed ledgers'. Digital fund ownership and transaction verification (e.g., prevention of double spending of funds or re-writing of the ledger) is performed via consensus algorithms (e.g., 'proof or work') enabled by specialized computer code and cryptography (private-public key pairs, hash functions), without the need for a trusted central party. Economic incentives to verify, execute, and record transactions on the platform are provided by transaction fees paid by the users and by the issuance of digital ('crypto') currency as reward to verifiers.

In this paper I explore the relationship between fundamental concepts and constructs upon which financial contracts and transactions are based (property rights, information, commitment, enforcement) and the algorithmic tools and constraints of digital platforms and blockchain technology, with specific economic examples. Unlike many overview papers, I explicitly describe and model the micro-level details about how on- and off-platform financial transactions and contracts can be implemented via blockchain technology. The ultimate goal is constructing a mapping between theoretical concepts and solutions from mechanism design and the algorithmic building blocks of digital platforms and blockchains (accounts, transactions, single or multi signatures, timelocks). I characterize the strengths and limitations of blockchain technology in implementing and enforcing financial contracts and payments, with applications to settings with financial market frictions and obstacles to credit or risk-sharing (incomplete markets, private information, limited enforcement). I analyze whether and how these financial market frictions can be addressed or mitigated by digital-platform transactions and (smart) contracts via mechanism-design based implementations.

The main takeaways are as follows. Blockchain technology excels at securing, verifying and transferring ownership of digital funds and objects. It also excels at recording, securing and making publicly accessible complete current and historical record of all data written on the blockchain but, naturally, not what happens off it. A key limitation of self-enforcing transactions performed via blockchain-based digital contracts is the need for *collateral* (escrow) for enforcing promised

future payments; that is, the algorithmic tools alone are insufficient. Any promise of payment must be backed up by escrowed/locked funds, otherwise it is unenforceable. However, I show how blockchain technology can be used to enable collateralizing and securing off-platform payments, saving on transaction fees. Leveraging the main advantages of blockchain digital platforms (history and information verifiability and security of proving ownership) with enhanced trust, commitment and enforcement mechanisms (including trusted third parties) can further facilitate the implementation of complex contingent contracts and solutions from mechanism design and contract theory in real-life applications, while saving on collateral costs.

I abstract from issues related to cybersecurity, mining, or achieving consensus on blockchain platforms. These technological elements, while very important, are taken as given in this paper and assumed to operate as intended and as specified by the computer code. All discussion about information, commitment and enforcement in digital contracts and transactions is predicated on this assumption and should be interpreted accordingly, specifically when comparing to other existing technologies or institutions such as courts, formal financial intermediaries, centralized databases, etc. I also do not address monetary or regulation-related issues, e.g., stable coins, monetary policy, the possible impact of blockchain technology on central banks, etc.

Related literature

Much of the early research on blockchains or digital platforms in economics consists of review papers (Koeppl and Kroninck, 2017; Chapman et al., 2017; CPMI, 2017; He et al., 2016; Catalini and Gans, 2016) or focuses on currency and monetary issues (e.g., Yermack, 2014; Raskin and Yermack, 2016; Chiu and Wong, 2014; Andolfatto 2018; Berentsen and Schar, 2018). A separate technical literature emphasizes security and the prevention of double spending (Nakamoto, 2008; Buterin, 2013; Peyrott, 2017), also with game theory applications (Eyal, 2015; Kiayias et al., 2016). Chiu and Koeppl (2019) are among the first to propose an economic model of transaction verification incentives in a blockchain, calibrate it to data, and perform policy analysis. Early conceptualizations of smart contracts and digital ownership are Szabo (1996, 1998). On policy and regulation see Kiviat (2015) and Chapman et al. (2017), among others.

I contribute to this literature by focusing on blockchain platforms as a leading example of digital platforms through which payments, credit and insurance in incomplete market settings can be coded and implemented. In doing so, I draw on a large theoretical literature emphasizing the role of history-dependence and the use of promised utility as state variable in multiperiod settings with private information (Spear and Srivastava, 1987; Abreu et al., 1990; Phelan and Townsend, 1991; Atkeson and Lucas, 1992; Fernandes and Phelan, 2000; Cole and Kocherlakota, 2001; Albanesi and Sleet, 2006; Karaivanov and Townsend, 2014), or with limited commitment (Thomas and Worrall, 1988; Phelan, 1995; Kocherlakota, 1996; Ligon et al. 2000, 2002; Kehoe and Perri,

2002; Broer at al., 2017). I introduce and model blockchain specific and digital-platform based mechanism-design elements, as in Chiu and Koeppl (2019) or Karaivanov et al. (2023).

This paper is most closely related to recent work conceptualizing and analyzing possible current or future applications of distributed ledgers, blockchains, and digital platforms in economic activity and distributed finance (DeFi), drawing on mechanism design and contract theory. Contributions include the comprehensive overview by Townsend (2020), Holden and Malani (2018) on hold-up, Cong and He (2019) on collusion, Gans (2019) on international trade, Routledge and Zetlin-Jones (2018) on currency pegs.[1] I draw on these works but differ by modeling in detail the algorithmic tools and constraints of blockchain technology as related to ownership rights, commitment, enforcement, and information in financial contracts.

2 Blockchain technology and tools

I start by introducing the key elements and tools of blockchain digital platforms in a stylized form, emphasizing their economic significance for enabling financial contracts and transactions. I use terminology that is mostly based on the Bitcoin implementation and borrows from, but is not always equivalent, to the terminology in various blockchain 'white papers' (e.g., Nakamoto, 2008 or Buterin, 2013).

2.1 Transactions

I use the term *blockchain* to mean a database of recorded *transactions* organized in time-stamped data 'blocks', together with associated computer code and algorithmic rules. Cryptographic tools (Merkle-Patricia trees, hash functions, etc.) ensure the integrity of the recorded data and the interconnectedness ('chain') between the data blocks, however, I abstract from these technical issues here.

1. Transactions – inputs and outputs

Transactions are the main element of a blockchain database. Define a transaction τ as the process of unlocking ('spending') pre-existing unspent digital funds x_{t-k} (*inputs*) and locking them into new unspent funds x_t (*outputs*). This can be as simple as sending / transferring funds from one person or account to another, however, more general interpretations, e.g., involving automated ('smart') contracts are possible. New digital funds (cryptocurrency) originate from special 'coinbase' transactions as a reward to 'miners' or validators but I abstract from this here; the focus is on transactions using pre-existing inputs.

[1] See also Chiu et al (2022, 2023), Chiu and Wong (2021), Lee et al (2022), Aronoff and Townsend (2023), Townsend and Zhang (2023).

2. Cryptographic locking scripts (keys)

Funds on the blockchain are secured (locked) by cryptographic locking scripts (e.g., cryptographic private keys). Only the possessor of a matching key/script can unlock (spend) the locked funds. I model locking scripts as the script l_{t-k}, paired with given inputs x_{t-k}.

3. Signing a transaction

The process of supplying an unlocking script, u_t (also known as 'signature') to the inputs of a transaction is called signing the transaction. If the supplied unlocking script matches the locking script for the unspent input funds x_{t-k} (that is, $u_t = l_{t-k}$), then the input funds x_{t-k} can be spent, that is, transferred and re-locked into output x_t by a new specified locking script l_t.[2]

Definition 1: Transaction

A transaction τ is defined as the mapping

$$\tau : (\{x_{t-k,} l_{t-k}\}, u_t) \rightarrow \{x_t, l_t\}$$

where subscripts denote time (block height) and where

- *the transaction inputs, $\{x_{t-k}, l_{t-k}\}$ consist of unspent funds, x_{t-k} with corresponding locking script(s), l_{t-k}. A transaction can have several inputs*
- *the transaction signature, u_t is a supplied unlocking script. A transaction can require several signatures*
- *the transaction outputs, $\{x_t, l_t\}$ are new unspent locked funds, x_t with locking script(s), l_t. A transaction can have several outputs.*

The **blockchain data** $\mathcal{T}$ are the *set of all posted and confirmed transactions* $\{\tau_t^i\}_{i,t}$ on the blockchain, where i indexes a transaction and t indexes its block, up to the current date.[3] These data can be used to trace back all past funds transfers or balances, for example, by sender or recipient account, or using other criteria. There is a natural mapping between the blockchain data $\mathcal{T}$ and the concept of 'history' in contract theory / mechanism design which I will discuss in more detail below.

[2]Technically, signing a transaction asserts cryptographically that a node (e.g., the sender) has the correct private key or script required to unlock the referred input funds but does not reveal that key.

[3]In actual blockchain platforms there could be submitted transactions that are unconfirmed for some time (not recorded on the blockchain), e.g., Bitcoin's *mempool*. I abstract from this issue here and use the term "posted" for transactions that are both submitted and confirmed.

2.1.1 Locking scripts

I model the following types of locking scripts l that exist in major blockchain implementations (e.g., Bitcoin)

(a) **simple key**

$$l_{t-k} = k^a$$

where k^a is a single-user private key required to spend/unlock the inputs x_{t-k}.

(b) **multisig key**

$$l_{t-k} = \text{ any } m \text{ out of } n \text{ keys } \{k^a, k^b, ..., k^n\} \text{ where } m \leq n$$

For example, the '2-of-2 multisig' key,

$$l_{t-k} = k^a \wedge k^b$$

where $\wedge$ is the logical operator "and", means that two private keys, k^a and k^b are required to unlock the inputs x_{t-k}. That is, to be valid the transaction must be signed by two signatures proving possession of both keys.

(c) **composite script**

$$l_{t-k} = s$$

I will use two main types of composite locking scripts:

– *timelock script* – a combination of private keys k^j, timelocks $T^i > 0$, and the logical operators "and" $\wedge$ and "or" $\vee$; for example,

$$s_1 = (k^a, T^1) \text{ or } s_2 = (k^a \wedge k^b, T^1) \text{ or } s_3 = (k^a, T^1) \vee (k^b, T^2)$$

The *timelock* T^i is a period of time (either astronomical time or number of blocks) that must have passed, relative to a fixed starting point (e.g., the block in which the transaction was recorded), for the transaction inputs to be unlockable. Script s_1 indicates that the input funds can be unlocked (used) if the key k^a is supplied and time T^1 has passed. Script s_2 also has timelock T^1 but requires the multisig key $k^a \wedge k^b$. Script s_3 requires key k^a with timelock T^1 or key k^b with timelock T^2.

– *timelocked redeemable secret* (e.g., as implemented in Bitcoin's *Hash Time Locked Contracts, HTLC*):

$$s = H \vee (k, T)$$

where $H = H(\sigma)$ is the hash function value of some secret message σ.[4] Anyone with knowledge

[4]Hash functions $H(w)$ are special mathematical functions that take a message (string) w of any length as input and

of σ can redeem the locked input funds immediately. Alternatively, a user can redeem the inputs by signing with key k after waiting T periods.

2.1.2 Writing vs. posting a transaction

In the following analysis I distinguish between

writing a transaction – the act of defining a transaction's inputs, locking script, signature(s) and outputs, as defined in Definition 1.

posting a transaction – the act of submitting a written transaction for execution through a software node connected to the blockchain platform. Here I abstract from waiting time, effectively assuming for simplicity that all posted transactions are confirmed and recorded on the blockchain without delay.

An important difference between writing and posting a transaction on the blockchain is that writing a transaction does not involve paying transaction fees (reward to miners or validators) while posting the transaction does. Once posted and executed a blockchain transaction is irreversible.[5]

2.1.3 Valid vs. invalid transactions

A transaction τ will be called *valid transaction* if it executes on the blockchain as written and intended, that is, if it results in a successful transformation of past locked funds into new locked funds. Once a posted blockchain transaction is valid it remains valid and cannot expire.

A transaction τ will be called *invalid transaction* if executing it on the blockchain platform/code would result in an error (code exception). Possible reasons can be: the referred input funds have been already spent; the supplied unlocking signatures do not match the locking scripts for all referenced inputs, or coding errors.

It is important to note that a transaction τ can be *invalidated on purpose* by writing and posting another transaction ('double-spend') that spends τ's inputs. A written transaction can only be invalidated in this way *prior to* posting it to the blockchain, not after. Posting a valid transaction therefore can be used not only to transfer funds (by unlocking and re-locking) but also to render invalid other written (but not posted and confirmed) transactions by spending their inputs. I will show that this technological feature of blockchains is very useful for constructing multi-period and multi-party financial transactions.

---

output a value, $H(w)$ of fixed length. Importantly, hash functions are easy to compute but practically impossible to invert.

[5] A new transaction can be constructed to "reverse" the actual ownership of digital funds but such transaction involves a new set of inputs and outputs and will be recorded as a different transaction.

2.1.4 Accounts and balances

For the purposes of this paper, a blockchain account is identified with a simple (private) key, k^a. Account balances can be derived from the record of all transactions on the blockchain (the blockchain data $\mathcal{T}$) as the sum of all funds unlockable by the account's private key (e.g., in Bitcoin); or are retrievable from the blockchain data as part of the virtual machine state (e.g., in Ethereum).

2.2 Algorithmic constraints

Given the definition of transaction (Definition 1), the blockchain platform code imposes and enforces the following constraints on inputs, outputs, locking scripts and signatures for a transaction to be valid and successfully posted on the blockchain:

1. **funds availability** – each input x_t^i is an unspent (available to be unlocked) output x_{t-k}^j from a previous valid transaction

$$x_t^i = x_{t-k}^j \text{ for some } j \text{ and some } t - k < t \tag{C1}$$

2. **no double spending** – no two valid transactions can spend/unlock the same input funds x_{t-k}^i

$$\nexists\ \tau_1, \tau_2 \in \mathcal{T} \text{ and } x_{t-k}^i \text{ such that } x_{t-k}^i \text{ is an input of both } \tau_1 \text{ and } \tau_2 \tag{C2}$$

3. **proof of ownership** – the supplied signature(s) u_t^i for each input x_t^i must match (cryptographically satisfy) the locking script l_{t-k}^j of the previous output, x_{t-k}^j it attempts to unlock/use[6]

$$u_t^i = l_{t-k}^j \text{ for the same } t, i, j, k \text{ in (C1)} \tag{C3}$$

The locking script could be *multisig* in which case two or more signatures may be required to satisfy (C3).

4. **input-output balance** – the total value of a transaction's output funds cannot exceed the total value of its input funds

$$\sum_i x_{t-k}^i \geq \sum_j x_t^j \tag{C4}$$

where the superscripts i and j indicate multiple inputs/outputs, if applicable. Allowing for strict inequality in (C4) can capture *transaction fees* (blockchain processing fees) that normally accrue to the miner of the block in which the transaction is recorded. Below I ignore transaction fees for

[6] In Bitcoin the most common scenario is that a signature applies to all inputs but targeted signatures are technologically feasible too.

simplicity. Note that no new funds are injected via transactions[7] and no funds are "lost" (the funds for miners' fees are locked in their accounts). In Bitcoin one of the outputs could be 'change' – e.g., part of the total input funds going back to the sender (locked by her key).

Constraints C1-C4 prevent 'double' or fraudulent spending which is one of the foundational ideas of blockchain technology (Nakamoto, 2008). The constraints ensure that a transaction cannot spend funds that are not available and that are not unlockable. If an attempt is made to spend/use inputs x^i_{t-k} that have already been spent by a previous posted transaction, the current transaction will be rejected by the blockchain as invalid. In practice, e.g., in the Bitcoin implementation, the blockchain code and node network keeps track of the complete set of unspent funds (the so-called 'UTXO set'), which is updated after each recorded block of transactions. Other blockchain algorithms (e.g., Ethereum) directly keep track of account balances as part of the blockchain virtual machine state.

5. **respecting timelocks** – timelocked outputs x^j_{t-k} cannot be spent before their timelock (if it exists) has expired

$$t \geq t - k + T^j_{t-k} \text{ for the same } t, j, k \text{ in (C1)} \tag{C5}$$

3 Blockchain digital platforms and financial contracts – tools, possibilities and limitations

This section discusses how the algorithmic tools of blockchain digital platforms introduced in Section 2 can assist the design, implementation, and execution of financial contracts and transactions. I discuss each of four fundamental building blocks of contracts – property rights, commitment (trust), enforcement, and information (verifiability). The strengths and limitations of digital blockchain technology with respect to each of these building blocks are summarized at the end of each sub-section.

3.1 Property rights

Contracts require well-defined property rights. In blockchain platforms property rights over digital funds or assets (e.g., unspent Bitcoin or Ethereum balance) are fully algorithmic and are established and verified by cryptographic tools in the form of computer code. Examples of these tools include the *scriptSig, scriptPubKey* functions in Bitcoin or the transaction *nonce* and *ECDSA* signatures in

[7]The focus of this paper is transactions between blockchain nodes and this I also abstract from the special "block reward" (coinbase) transactions that accrue to miners.

Ethereum.

First, using the Section 2 terminology, *locking scripts* secure and establish ownership over digital funds. Matching signatures to locking scripts is used to transfer ownership. A locking script/key cannot be forged – it is either objectively correct or not; this is a technological strength. However, a key or script can be stolen/copied or lost – a limitation. This observation links to a large and important cybersecurity literature which, however, I do not discuss here.

Second, property rights over digital funds or assets in blockchains are normally of the *bearer* type and are not tied to a particular user identity. That is, anyone who produces the required unlocking signature matching the locking script can spend/use the funds (see Section 2). Importantly, in most or all currently existing blockchain platforms there are no external legal property rights; that is, rights or claims existing outside of the digital algorithmic tools just described. The anonymity of accounts and lack of central trusted party or regulation currently rule out other existing methods to record or verify property rights over digital funds and assets. However, this feature of blockchains could be modified in the future.

Third, the *multisig* and *timelock* locking scripts can be used as flexible *covenants* to restrict digital property rights. Multisig locking scripts require the presentation of m out of n signatures, therefore preventing unilateral unauthorized spending. Timelocks can be used to restrict usage property rights, since timelocked funds cannot be spent by anyone before the timelock expiration, for example, supplying the correct signature (private key) k^a alone cannot unlock and use funds that are locked by an unexpired timelock script (k^a, T^a).

- Strengths: cryptographically secure proof of ownership; low cost of acquiring and transferring ownership; flexible covenants – multisig, timelocks.
- Limitations: no record/verification outside the blockchain; purely bearer type ownership.

3.2 Commitment and trust

Economists use the term commitment to describe (or assume) a party's ability to make credible/deliverable promises about future actions. For example, a reputable online seller commits to ship the purchased item after receiving payment from a buyer. Related to this, the term *limited commitment* is often used to indicate situations in which only one of the contract parties is bound by their promises or that commitment is possible only in the short-term (e.g., single period), or with probability less than one. A vast economics literature on time inconsistency and lack of commitment exists analyzing these settings.

I abstract from any external/third-party commitment devices or institutions and focus solely on the digital algorithmic tools that can be used to ensure commitment to future payments or

property right transfers. Blockchains are often described as 'trustless' decentralized platforms. However, the term 'trustless' is not entirely precise – while most blockchain platforms (e.g., Bitcoin, Ethereum) indeed do not rely on a trusted authority in the traditional sense (e.g., courts of law, the banking system, etc.), they do strongly rely on the users' trust in the computer code and algorithms on which the platform is based and runs.[8]

Specifically, for a transaction τ, as described in Definition 1, to be valid, i.e., represent a *commitment* to pay/transfer an amount from account A to account B it is essential that:

Condition C1: the referenced *inputs* x_{t-k} *must be available and accessible* at the moment of posting the transaction on the platform. This requires having the requisite signature / unlocking script(s), u_t, including satisfying any timelocks.

The availability of input funds is critical and demonstrates that blockchain transactions and smart contracts are not automatic or set in stone once written or submitted to the platform. A transaction τ can be rendered invalid (effectively canceled) by spending its input funds with another valid transaction, τ' prior to τ being posted and confirmed on the platform. I show an application of this idea in Section 4.

The second implication of Condition C1 is that a necessary prerequisite for any future payment commitment is that the required funds must be **fully collateralized**. That is, the promised funds must be locked and still available to be unlocked at the time the payment or transfer is due. In practice this can be achieved using the algorithmic tools described in Section 2 – *multisig scripts*, *timelocks* and combinations thereof. Multisig locking scripts, e.g., $l_{t-k} = k^A \wedge k^B$ ensure that one of the parties cannot spend or syphon the funds without the agreement of the other. Hence, only when the economic incentives of both parties are aligned can the funds be transferred, ensuring mutual trust. Timelocks directly lock digital funds from use by any party and can algorithmically ensure that the funds are still available at the intended usage or payment date. Note that potential self-control problems are also avoided in this way.

Condition C2: the transaction which establishes the promised funds' availability (known as 'collateral transaction' or 'funding transaction', see Section 4) must be confirmed on the blockchain platform (transaction fees must be paid). This activates the algorithmic tools and secures the digital property rights ensuring the future availability of funds.

- Strengths: commitment and trust can be established and maintained algorithmically.
- Limitations: commitment and trust are limited to the available algorithmic tools (timelocks, multisig) and require locking funds as collateral/escrow (costly).

[8]This includes technical issues such as achieving and maintaining consensus about the blockchain records, susceptibility to malicious attacks, network throughput and others, which I do not discuss here.

3.3 Enforcement

Contract *enforcement* refers to executing the terms of the contract as stated and intended. Here I focus on enforcing digital transactions or contracts that transfer funds between accounts. Digital transactions are enforced algorithmically (automatically) by the platform computer code, as long as the *property rights* and *commitment* conditions ensuring transaction validity are satisfied:

– input funds ownership is proved cryptographically

– all locking script conditions (multisig, timelocks) are satisfied by providing all necessary signatures or posting the transaction at the correct time

– transaction fees to post the transaction on the platform are paid.

Conditional on the above, the receiver of the transaction output(s) does not need to do anything (except possibly wait) after a transaction is posted and validated on the blockchain platform. Contract verifiability and execution is thus algorithmic and automated, via unambiguous computer code. This reduces uncertainly about what is promised (a form of counterparty risk) and also reduces legal interpretation risk, see Holden and Malani (2018) for further discussion.

An enforcement problem could therefore arise only when a transaction (for example, a promise of future payment) is *invalid* and hence rejected by the platform code when submitted. This could happen, for example, if the transaction's input funds have already been spent in another previously validated transaction (in Bitcoin) or, equivalently, if there is insufficient balance in the sender's account at the time of posting (in Ethereum). Note that the recipient has no resort in such situations, since external enforcement is unavailable.

- Strengths: enforcement is automated, without human element; low enforcement costs
- Limitations: only valid transactions can be enforced – the pre-conditions for property rights and commitment must be satisfied (e.g., costly collateral); no third-party / external enforcement exists (courts, arbitration, etc.)

3.4 Information

By design, blockchain digital platforms record complete granular information of all recorded transactions going back to the initial block 0. Also by design, everything that is posted on the platform distributed ledger is public information and free to access. The minimal data recorded on the platform are transactions data (input, output, value) but more complex data can be recorded too (e.g., coded 'smart' contracts or digital tokens and NFTs in Ethereum). The data are organized in an interconnected chain of time-stamped data blocks via hash functions, Merkle trees, etc. Cryptography makes it nearly impossible to modify past written data – recorded data are permanent. The data immutability is a design feature, not a limitation.

The public, complete and permanent nature of blockchain data has a natural parallel with the notion of *history* in mechanism design theory. The transactions data can also be used to determine account balances or other aggregates at any moment of time. Platforms such as Ethereum also allow Turing-complete code execution which can embed complex if/then/else, and/or conditions, loops, and multi-dimensional contingencies based on real-time and historical information recorded on the platform.

While blockchain platforms excel at storing huge amounts of public information via the recorded transaction data, this does not automatically solve or avoid potential contractual problems that may arise from asymmetric information (hidden actions, hidden income, unobserved characteristics). Essentially any information not recorded in the platform database can be private information for other users or counterparties. In addition, most blockchain platforms are permissionless and (pseudo-)anonymous so that users can have multiple accounts which makes difficult matching transactions to specific individuals or firms unless they want to reveal such identifying information. Therefore, further coordination and efficiency gains from required posting or (self-)reporting of information are possible. See Townsend (2020) for further discussion, including on possible gains from obfuscating information, depending on the economic setting.

- Strengths: complete, permanent, and public record of granular current and historical information; capability for automated code execution with complex multi-dimensional contingencies based on the recorded information
- Limitations: off-platform information can remain hidden, including strategically; the immutability of recorded data may cause problems (e.g., permanently locked funds; coding errors)

3.5 Technological strengths and limitations – discussion

Blockchain technology excels at securing and verifying property rights over digital assets via cryptographic locking scripts and signatures. It also excels in recording a complete and immutable history of on-platform transactions (the blockchain data). The leading platform implementations also provide algorithmic tools for making and enforcing future payment commitments (e.g., multisig, timelock, and composite locking scripts).

The main limitation of blockchain platforms in terms of writing and enforcing financial contracts and transactions is the need for full collateral backing of future promised payments.[9] The

[9] In addition, there are costs of posting transactions (transaction fees) and waiting for confirmation, however, these economic costs are likely lower than the corresponding direct and indirect costs of other transaction or payment methods especially among parties who are strangers.

collateral requirement can be costly, as I show in the economic applications in the next section. However, in Section 5 I show that the same collateral (locked funds) can be used together with the algorithmic tools from Section 2, to back *multiple* off-platform payment transactions, hence mitigating this technological limitation.

The essentiality of collateral arises from the anonymity of blockchain accounts and the bearer form of digital property rights. How about reputation as a possible commitment mechanism? It is possible to use a blockchain platform to record reputation-related information, for example, as digital token balance akin to a credit or user rating. However, it is unclear how outsiders (e.g., new contract parties) can verify the authenticity of such reputation ratings, e.g., a trader may self-generate transactions with other accounts that s/he controls and/or rate himself highly, similar to creating or purchasing fraudulent reviews on business rating websites.

Another common commitment mechanism involves using punishment threats not based on collateral, for example, the threat of autarky or no future trade. Contract parties could write such self-enforcing mechanisms using digital platform technology (smart contracts). However, doing so may significantly limit the set of feasible trades and reduce efficiency (see the risk-sharing example in Section 4). It is possible to punish a counterparty with no future trade if a contractual payment is not made, however, without external enforcement or the possibility to restrict access in a permissionless anonymous platform, it is hard to prevent the offender from contracting with another lender, borrower or seller. In addition, such no-trade punishments are often time-inconsistent (both parties would like to re-negotiate ex-post if gains from trade exist) and there is the issue of how to prevent fraudulent punishment.

4 Applications and examples

In this section I present two specific economic applications, multiperiod risk sharing and international trade, to illustrate the main contracting issues and frictions from contract theoretical point of view and discuss how digital platform algorithmic tools can be used to address these issues and frictions. Implementation strategies using the algorithmic tools from Section 2 are then discussed in Sections 5 and 6.

4.1 Risk sharing

Townsend (1982) studies a multiperiod optimal risk sharing problem. A risk-averse agent with preferences over consumption $u(c)$ and discount factor $\beta \in (0,1)$ has an i.i.d. stochastic income process $\{y_t\}_{t=0}^{T}$ which takes values y_j, $j = 1, ..., \#Y$ on the discrete set Y (e.g., low or high income). The agent can enter a contract with a risk-neutral financial intermediary ('the insurer')

who can provide credit or insurance against the income shocks. The agent's income can be public or private information. In the latter case, assuming for simplicity a two-period setting and two possible values for the agent's income, Townsend shows how a multiperiod insurance contract which conditions risk-sharing transfers on the *history* of output realizations dominates in terms of efficiency a simple debt contract while both the insurance contract and the debt contract attain higher utility than autarky.

Using standard arguments, an infinite-horizon contracting problem in this setting can be written recursively in terms of the agent's promised utility (present value of future utility), w as the state variable:[10]

$$\Pi(w) = \max_{\{\rho_j, w'_j\}_{j=1}^{\#Y}} E(-\rho_j + \beta\Pi(w'_j)) \text{ subject to:} \tag{1}$$

$$E(u(y_j + \rho_j) + \beta w'_j) = w \quad \text{[promise keeping]}$$

$$u(y_j + \rho_j) + \beta w'_j \geq u(y_j + \rho_l) + \beta w'_l \text{ for any } j, l = 1, ..., \#Y \quad \text{[truth telling]}$$

where $\Pi(w)$ is the insurer's profit function and the expectation is taken over the income states distribution. This dynamic programming formulation is equivalent to making the optimal transfers ρ_j conditional on the complete history of agent messages about past income realizations $y_1, ..., y_{t-1}$ and $y_t = y_j$ at any period t,

$$\rho_j(w) = \rho(y_1, ..., y_{t-1}, y_j)$$

The first-best outcome is full insurance, that is, the agent receives constant consumption $c_j = \bar{c}$ in all income states, by making or receiving transfer $\rho_j^{fb} = \bar{c} - y_j$. This outcome is, however, not incentive-compatible if the agent's income y_j is private information, since the agent would have incentive to report the income level that results in the largest payout. Therefore 'truth-telling' constraints must be imposed on the transfers to ensure, by the Revelation principle, that the agent would optimally report her actual income, resulting in the on-path contractual transfer (contribution or payout) ρ_j and promised utility w'_j, as opposed to reporting some other y_l (e.g., lower income) and receiving the off-path values ρ_l and w'_l.

This setting is a prototypical application of mechanism-design theory to derive a complex financial contract (a mix of credit and insurance) in the presence of private information. Typically the constrained-optimal contract can be solved only numerically. Therefore, a platform-based digital contract could be coded to compute and implement the optimal allocation solving problem (1), by keeping track of the agent's history of messages about income or, equivalently, by keeping track

[10] Townsend (1982) did not use this recursive formulation but recognized the optimality of conditioning transfers on income history, which is mathematically equivalent to keeping track of promised utility. If the time horizon is finite, all variables are time dependent.

of promised utility w in a token account (see Section 6). Blockchain technology is perfectly suited for storing such granular data and for coding the automatic computation of the insurance transfers (contributions/premia or indemnities/payouts), provided the economic structure (the income process, preferences, etc.) is known or can be estimated or learned. Structural estimation techniques (e.g., Karaivanov and Townsend, 2014) or (machine) learning algorithms may be used to calibrate the code using pre-existing or pilot study data. The optimal contract attains higher welfare than autarky (no insurance) or a simple debt-contract (which also satisfies the truth-telling constraints), see the numerical example below.

Numerical example

Suppose there are two periods and two output levels $y_L = 3$ and $y_H = 5$ each occurring with probability $\pi = 0.5$ and preferences $u(c) = c - .05c^2$ and $\beta = 1$. Then we obtain the following solution for the insurance transfers ρ, for different assumed contractual settings ranging from autarky to full insurance. In Table 1 the superscripts on ρ denote the time period and the subscripts denote the realized income history, e.g. LH means low income in period 1 and high income in period 2.

Table 1. Risk-sharing example

two-period setting	ρ^1_H	ρ^1_L	ρ^2_{HH}	ρ^2_{HL}	ρ^2_{LH}	ρ^2_{LL}	*ex-ante welfare*
1. autarky	0	0	0	0	0	0	6.3
2. hidden income							
a. debt	$-.5$	.5	.5	.5	$-.5$	$-.5$	6.325
b. insurance[11]	$-.56$	.64	.45	.45	$-.53$	$-.53$	6.33
3. full insurance (first best)	-1	1	-1	1	-1	1	6.4

Contracting problem (1) and its solution assume commitment by both parties, that is, the requisite insurance transfers ρ_j can be enforced costlessly. But what if a party does not make the required transfer transaction, e.g., by spending its input funds? As discussed earlier, this can happen if the required funds are not locked or secured in advance. Specifically, suppose that the agent could renege after observing their income, but before having to make the transfer ρ_j and the insurer can commit to punishing such agent with no-trade (autarky) afterwards. Then an additional 'limited enforcement' constraint is introduced in the contracting problem (see Thomas and Worrall, 1988 or Karaivanov and Martin, 2015 for more examples),

$$u(y_j + \rho_j) + \beta w_j \geq u(y_j) + \beta V^a, \forall_j \quad \text{[limited enforcement, agent]}$$

[11]For simplicity Townsend (1982) restricts the per-period insurer profits to zero. Instead I assume zero *ex-ante expected* profits for the insurer, yielding a minor welfare gain.

where $V^a = \sum_{t=0}^{\infty} \beta^t E(u(y_t))$ is the agent's autarky value. Imposing this additional constraint on problem (1) ensures that the agent will not have an incentive to renege on any contributions (insurance premia) but can further the degree of risk-sharing relative to the first-best (see Figure 1 for illustration).

Discussion

Blockchain technology cannot solve the private information (hidden income) problem per se, unless income accrues or is recorded directly on the platform without human intervention. However, a digital contract coded on the platform can implement the private information constrained optimum by keeping track of history or promised utility based on the agents' self-reported messages about income.

Blockchain-based *collateral* can in turn help with the commitment problem. One possible way is to lock sufficient funds ex-ante that allow to support any possible sequence of due transfers. I describe a "payment channel" implementation of this idea in Section 5.2. An alternative is to only lock sufficient funds to ensure that the insurance contract is self-enforcing, that is, a party has no economic incentive to renege at any moment of time and after any history of income realizations. For example, if we focus only on the agent's commitment problem, then locking funds C as collateral, to be repossessed if the agent reneges on a due transfer $\rho_j < 0$, would satisfy the self-enforcement conditions as long as C satisfies, for all j,

$$u(y_j + \rho_j - C) + \beta w_j \geq u(y_j - C) + \beta V^a \tag{2}$$

A similar argument can be made about the insurer, by requiring posting sufficient collateral for the histories with $\rho_j > 0$.

By having the technological ability to lock funds as collateral digital financial platforms (e.g., blockchain-based) can therefore support the optimal history-contingent contract. The algorithmic tools used are a digital token account to record the history of agent messages or promised utility and the cryptographic locking scripts and signatures used to secure the collateral.

4.2 International trade

Consider next an example related to international trade, as in Gans (2019). A buyer B wants to purchase a product from a seller, S. The buyer's value for the product is V. It costs C for the seller to produce and ship the product, where $C < V$. Clearly, a mutually beneficial trade opportunity exists, for some price $P \in (C, V)$. There are two potential enforcement-related issues related to this trade contract:

(i) hold up – the buyer may receive the product but decide not to pay (or offer ex-post to

pay less than agreed upon)

(ii) moral hazard – the seller could ship an inferior product, with lower cost $c \in [0, C)$ and lower value to the buyer $v \in [0, V)$.

Gans (2019) describes a sequential mechanism design solution for this setting based on Moore and Repullo (1988) and argues how digital platform technology (a smart contract) can be used to implement it. The proposed solution is a sequential mechanism consisting of three stages in which the buyer and seller take turns making 'take it or leave it' offers.

Stage 1 (announcement) The buyer B announces the received good's quality: V or v ($v = 0$ could be interpreted as 'not received'). If V is announced then P is paid, end.

Stage 2 (challenge) If v is announced at Stage 1, the seller S can challenge B's claim. If there is a challenge the buyer is fined an amount F which is remitted to the seller. Then, the seller offers the buyer a choice between:

2.1 keep the received product and pay price p where $p < P$, end.

2.2 return the product for refund f.

Stage 3 (settlement) If the buyer B chooses option 2.2, B is issued a refund f by the seller and the seller pays fine $2F$ to a third party, end.

The extensive game form is illustrated on Figure 3. It can be proved that, for appropriate conditions on the parameters, the unique equilibrium is that the seller ships the contracted good (with value V) and the buyer pays the contracted price P (see Gans, 2019 and references therein for details).

Discussion

While blockchain technology and smart contracts can be used to automate the specified payments in the proposed game-theoretic solution, observe that **collateral** (funds locked in escrow) is required to ensure commitment to the contract. First, the buyer B must lock in escrow funds $P + F$ to ensure ability to pay for the good, P and the potential fine F in the off-equilibrium path where B challenges the seller S. The fine is necessary to prevent frivolous challenges in Stage 2. Second, the seller S must post funds $2F$ as collateral to support the settlement Stage 3 (possibly F of these funds could be from the buyer's fine that S received in Stage 2, if appropriately script-locked). The contract also calls for a third party to receive the fine $2F$ in Stage 2 but this could be automated by 'burning' the digital funds (sending to a null address).

The amount of collateral required for the contract implementation can be sizable. Gans (2019) discusses a numerical example with $V = 30$, $v = 0$, $c = 5$, $P = 15$, $C = 10$, $F = f = 6$ and $p = 10$ which satisfies the sufficient parametric conditions for unique equilibrium (V, P). Observe that for these parameters the total surplus from trade is $V - C = 20$ while the total collateral/escrow required from the buyer and the seller is $P + 3F = 33$ (or at least $P + 2F = 27$,

if the buyer's fine can be re-used as part of the seller's fine $2F$ in Stage 3). I discuss a payment channel implementation in Section 5.2.

5 Digital platform payments and collateral

I describe the concept of *payment channel* (Decker and Wattenhofer, 2015) to describe how digital technology algorithmic tools can be used to enable and support incentive-compatible self-enforcing transfers and payments between two agents. An extension to multiple agents, as in Bitcoin's Lightning network, is also briefly discussed. I focus on the economic issues related to incentives, commitment, and enforcement of blockchain-based digital payments and abstract from technical issues, e.g., 50-percent attacks, congestion, or cybersecurity concerns.

In the preceding sections I argued that committing to and enforcing promises of future payments via a blockchain platform requires sufficient collateral. In addition, posting transactions on the blockchain is costly – transaction fees must be paid and there could be throughput constraints and time delays after posting. Therefore, the key challenge is to design how to:

(i) use the blockchain platform as conduit and *collateral mechanism* to support digital payments and

(ii) use the blockchain *minimally,* with the smallest number of posted transactions.

The following exposition mainly follows the Bitcoin implementation of payment channels, however, this is not essential for the main results and conclusions.

5.1 State transitions and payment channels

Consider a payment contract between two agents, A and B. We can think of the contract as the agents starting at an initial state (a_t, b_t) where a_t is A's balance and b_t is B's balance at time t, and having to transition to a new balance state (a_{t+1}, b_{t+1}) in an incentive-compatible and self-enforcing way. A simple example of such *state transition* problem is a one-time payment p from A to B, in which case

$$a_{t+1} = a_t - p \text{ and } b_{t+1} = b_t + p.$$

Below I outline an incentive-compatible blockchain-based implementation to the state transition problem known as a *payment channel,* based on Antonopoulos (2017) in the context of Bitcoin. The key idea is to use a transaction posted on the blockchain to collateralize the set of all possible balances between the two parties, that is, the set of all values $(a, b) = (A - z, z)$ for a fixed total sum A, called the channel capacity, and any $z \in [0, A]$.

5.1.1 One-way payment channel

To illustrate the idea of using blockchain-based collateral to enforce payments, consider the simplest case of a one-way payment channel – that is, transfers go only one way (e.g., A's balance a_t always decreases while B's balance b_t always increases over time). Such one-way payment channel can be implemented as the following sequence of steps using the the algorithmic tools from Section 2.

1. collateral set-up. Write a collateral (funding) transaction which establishes the maximum payment that can be implemented:

$$\tau_c : (\{x_0, l_0\}, u_0) \rightarrow \{c, l_c\}$$

For simplicity, assume that the input funds $\{x_0\}$ of τ_c are provided by A, locked by his private key $l_0 = k^a$, and are unlockable by signing with $u_0 = k^a$. The value $c = \sum_j x_0^j$ is the collateral amount which is locked by the 2-of-2 *multisig* locking script

$$l_c = k^a \wedge k^b$$

The use of a multisig locking script is important, as it ensures that no single party can unlock and use the funds without the other party's signature/key.

2. refund set-up. Before posting the collateral transaction τ_c on the blockchain platform the contract parties also write a second 'refund' transaction:

$$\tau_r : (\{c, l_c\}, u_c) \rightarrow \{c, l_r\}$$

in which the collateral funds c are secured by a *timelock* $T > 0$ (e.g., 20 days in the future) which establishes the maximum duration of the payment channel, that is,

$$l_r = (k^a, T)$$

Person A passes the transaction τ_r to B to sign (that is, to supply the k^b part of the required unlock signature $u_c = l_c$). B agrees to sign since presumably she has something to gain from the contract. The refund transaction τ_r, once signed by B, protects A in case B disappears. It is is kept by A as a guarantee and not posted on the blockchain.

3. posting collateral. The collateral transaction τ_c is posted on the blockchain by A signing it by $l_0 = k^a$. This locks the funds c (requiring the multisig script l_c to unlock) and establishes the payment channel. Note that both parties' private keys are needed to unlock/use the

funds.

4. state transitions. After steps 1 through 3 the payment channel is established and ready to support state transitions of the form

$$(a, c-a) \rightarrow (a', c-a')$$

where the first argument in the brackets is A's balance and the second argument is B's balance and where $a' < a$, that is, A is sending a payment of size $a - a'$ to B. These state transitions correspond to unilateral transfers of digital funds (payments) from A to B.

To perform a state transition the parties write the following blockchain transaction with two outputs corresponding respectively to the balances due to A and B.

$$\tau_s : (\{c, k^a \wedge k^b\}, k^a) \rightarrow \left\{ \begin{array}{l} a', k^a \\ c-a', k^b \end{array} \right\}$$

A signs transaction τ_s by his key k^a and passes it to B. B can post τ_s on the blockchain by signing it by k^b (which would satisfy the required locking script $l_c = k^a \wedge k^b$) or she could wait, e.g., if she is due additional payments from A. Further state transitions can be made by writing new transactions (separate communication between the parties may be required, or the process may be automated) which spend the same input funds $\{c, l_c\}$ but have different outputs, for example, a'' and $c - a''$ where $a'' < a'$ and so on. Note that, along this transaction chain, whenever B holds a transaction entitling him/her to a larger balance (e.g., $c - a''$ vs. $c - a'$), then B no longer has incentive to post a previous/older transaction. In addition, A is prevented from posting any old transactions by not having B's signature/key.

5. settlement. The final state transition ('settlement') transaction, $\bar{\tau}_s$ must be posted on the blockchain before expiry of the timelock T set in step 2.

Observe that in the described mechanism the blockchain platform is used to post only two transactions – the collateral transaction τ_c and the final settlement transaction $\bar{\tau}_s$, while multiple payments from A to B can be completed in the meantime. All these payments are fully secured and incentive-compatible by using the algorithmic tools from Section 2, specifically:

– the *multisig* locking script $l_c = k^a \wedge k^b$ in the collateral transaction τ_c ensures that either party cannot expropriate the collateral funds c; in particular, it prevents A from posting an old state (an old transaction τ_s) which gives A larger balance than the current state (e.g., prevents A from skipping or not making a payment for services provided by B). This protects B.

– the *timelock* T in the refund transaction's locking script $l_r = (k^a, T)$ protects A by ensuring that A can get their funds back if B unilaterally quits (does not sign and post any settlement

transaction).

5.1.2 Bilateral payment channel

The unilateral payment mechanism described in Section 5.2.1 is only self-enforcing when A is paying to B (that is, more and more of c becomes due to B over time). To see that, suppose the intended state transition was $(a, c-a) \rightarrow (a', c-a')$ with $a' > a$ (that is, B needs to make a payment to A). But then nothing would prevent B to post the old transaction already signed by A (that is, the transaction giving B the larger previous balance, $c-a$) and benefit. To prevent such deviations and allow state transitions in both directions, the two parties must hold (asymmetric) refund transactions, as described below.

A bilateral payment channel supporting any state transition $(a, c-a) \rightarrow (a', c-a')$ with $a > a'$ or $a < a'$ can be implemented using the algorithmic tools of Section 2 as follows:

b1. collateral. The first step is to write and post a collateral (funding) transaction of the form

$$\tau_c : (\{a_0, k^a; b_0, k^b\}, u_c) \rightarrow \{c, l_c\}$$

Note that now the collateral transaction has two inputs, (a_0, k^a) and (b_0, k^b), originating respectively from A and B and unlockable by their corresponding signatures k^a and k^b. It is possible that one of a_0 or b_0 equal zero – that is, all collateral may be provided by a single party. As in Section 5.2.1, the collateral transaction has a single output with value c and is locked by a multisig key l_c, requiring both parties to sign. Here, using (C3), we have

$$c = a_0 + b_0 \text{ and } l_c = k^a \wedge k^b.$$

b2. state transitions. The next step is to perform the transition from balance state $(a, c-a)$ to state $(a', c-a')$, where initially $a = a_0$. Here and later on in the state transitions chain, both states with $a' > a$ or $a' < a$ are possible, e.g., A pays B or B pays A. The state transitions must be enforceable by the algorithmic tools. That is, each party must be able to obtain their contractual due amount without relying on trust or cooperation from the other party or third-party enforcement. This is achieved by writing and exchanging two transactions τ_{sa} and τ_{sb}, held respectively by A and B and counter-signed by the other party's key (τ_{sa} is signed by k^b and τ_{sb} is signed by k^a):

$$\tau_{sa} : (\{c, k^a \wedge k^b\}, k^b) \rightarrow \left\{ \begin{array}{c} a', (k^a, T) \vee r^b \\ c-a', k^a \end{array} \right\} \quad \text{(A)}$$

and

$$\tau_{sb} : (\{c, k^a \wedge k^b\}, k^a) \rightarrow \left\{ \begin{array}{c} a', k^b \\ c - a', (k^b, T) \vee r^a \end{array} \right\} \quad \text{(B)}$$

These transactions do not need to be posted on the platform, but would be valid if they are signed and posted. In (A) and (B) above, r^a and r^b are special 'revocation' scripts (to be explained below), that are exchanged in every state transition and held by A and B respectively. Note that the transactions τ_{sa} and τ_{sb} serve simultaneously two roles: (i) a potential refund/guarantee (via the timelock T and revocation script r^j) and (ii) implementing the intended state transition (payment).

How do (A) and (B) enable the state transition? Transaction τ_{sa} held by A has input funds c and has been pre-signed with B's key, k^b as part of its required unlock signature u_c. Hence, if A also signs τ_{sa} using her key k^a, its input funds c would be unlocked, since they are locked by the script $l_c = k_b \wedge k_a$. Transaction τ_{sa} has two outputs. The second output, $(c - a', k^a)$ would release B's contractual balance $c - a'$ immediately if A signs, since its locking script is $l^1_{sa} = k^a$ without any timelock. However, A's due funds a' (the first output of τ_{sa}) remain locked by the composite locking script

$$s_a \equiv (k^a, T) \vee r^b.$$

This output can be unlocked by A using her private key k^a but only after the timelock T expires, or alternatively, it can be unlocked immediately by using the revocation script r^b (held by B). Importantly, the script r^b must remain in possession of B until the parties agree to transition to another state. An analogous argument applies for transaction τ_{sb} by exchanging the a and b superscripts accordingly.

To transition to a new balance state, e.g., to $(a'', c-a'')$ the parties first create and give each other revocation keys r^a and r^b (so that the previous state can be revoked if posted) and then counter-sign new transactions of the form (A) and (B) with the same inputs but new output balances and new revocation keys. The revocation key exchange and counter-signing must be done algorithmically and simultaneously to avoid any hold up.

Why are the state transitions (payments) implemented by τ_{sa} and τ_{sb} self-enforcing? Suppose A attempted to renege on the contract and posted (by signing with k^a) the transaction τ^{-1}_{sa} corresponding to a previous/old balance state (e.g., because such action would give A more funds, a^{-1} than currently due). Then B would receive $c - a^{-1}$ immediately and has T periods (the timelock on A's output in τ^{-1}_{sa}, see (A)) to detect A's deviation and use transaction τ^{-1}_{sa}'s revocation key r^b to claim the output a^{-1} as well, essentially seizing A's share of the collateral and punishing the deviation. The same argument applies for B in a symmetric situation. Because of these collateral-enabled penalties neither party would have an incentive to post an old transaction. However, what if a party disappears, that is, does not exchange a revocation key or does not counter-sign the next

state transition transaction? Then the other party can still post the last signed valid transaction after waiting for its timelock to expire and so is protected against such deviations as well.

5.2 Payment channels – applications

I return to the risk sharing and trade applications introduced in Section 4 and show how collateral-backed payment channels can be used to implement the constrained-efficient outcome in each setting. A debt contract setting is also analyzed.

Multiperiod insurance

In the multiperiod risk sharing setting of Townsend (1982), think of the agent and insurer starting at state $(0,0)$ (each has zero balance due) and some initial promised utility w_0 (stored in a digital token account). Then, depending on the realized history of agent's income $(y_{j_1}, y_{j_2}, ...)$, the following chain of state transitions is implemented:

$$(0,0) \rightarrow (\rho^1_{j_1}, -\rho^1_{j_1}) \rightarrow (\rho^1_{j_1} + \rho^2_{j_2}, -\rho^1_{j_1} - \rho^2_{j_2}) ... \rightarrow (\sum_{t=1}^{T} \rho^t_{j_t}, -\sum_{t=1}^{T} \rho^t_{j_t})$$

where $\rho^t_{j_t}$ denotes the optimal insurance transfer to the agent (payout if positive and contribution/premium if negative) at time t if the reported income state is j_t. For this series of state transitions to be implementable via a blockchain-based payment channel, the parties need posted collateral which spans the maximum possible balance due to either side. It is sufficient to set

$$c = \max\{|\frac{\rho_{\min}}{1-\beta}|, |\frac{\rho_{\max}}{1-\beta}|\}$$

where $\rho_{\min}$ is the largest possible transfer from the agent in the mechanism-design solution and $\rho_{\min}$ is the largest possible transfer from the insurer, see Section 4.1.

Trade

Consider a repeated version of the Gans (2019) setting in which the buyer and seller engage in a maximum of T trades. Assume that if a party reneges on the contracted outcome (the seller S sends good with value V and the buyer B pays price P), then the relationship is terminated. In such case, as discussed in Section 4.2, a payment channel with collateral

$$c = TP + 2F$$

can support the required state transitions:

$$(0,0) \rightarrow (-P, P) \rightarrow ... \rightarrow (-TP, TP)$$

where the first state is the buyer's balance. The additional collateral amount $2F$ is needed in case of a deviation, see Section 4.2 for details.

Debt contract

A payment channel can be also used to set up a simple debt contract (one-period loan agreement). Suppose a lender A and a borrower B wish to enter a blockchain-based loan contract to provide B with funds l for a fixed term (e.g., one year) at interest rate r. As assumed throughout this paper, suppose that no external enforcement is possible. That is, the contract must be implementable solely via the algorithmic tools and constraints from Section 2. This implies that in order for A to be certain that she will be repaid $(1+r)l$, the minimum amount b_0 that B must post in the payment channel collateral transaction τ_c must satisfy:

$$b_0 \geq (1+r)l.$$

Suppose also that the lender A provides a_0 in τ_c where $a_0 > l$, i.e., A has enough funds to enable the loan. Disbursing the loan can be written as the following state transition, where the output funds l are immediately redeemable by B (using the private key k^b)

$$(a_0, b_0) \rightarrow (a_0 - l, b_0 + l)$$

If the loan is repaid as intended (i.e., the borrower B repays the contracted amount $(1+r)l$ to the lender A), the following balance state is reached at the end of the loan term:

$$(a_0 + rl, b_0).$$

Essentially, A gains the interest on the loan, rl while B keeps his collateral funds. This is achieved by a settlement transaction in which A is paid $(1+r)l$ and B's collateral value $(1+r)l$ is released/unlocked back to B.

If, instead, B fails to repay within the specified time in the contract (a timelock is used), the following alternative state is reached in which the lender A seizes (part of) B's collateral:

$$(a_0 + rl, b_0 - (1+r)l)$$

A real-world example for digital-platform loans is MakerDAO's borrowing facility implemented on the Ethereum blockchain. A user can lock Ethereum cryptocurrency (ETH) as collateral and receive a DAI (token) denominated loan at a minimum 1.5-to-1 collateral-loan ratio ($150 worth of ETH gives $100 worth of DAI). If the loan is repaid the ETH collateral is released back to the borrower; if not, the collateral is liquidated in favor of the platform (the lender).

5.3 Multi-party transactions

The basic idea behind bilateral state transition payments described above can be extended to transactions involving more than two parties (an example is the Lightning Network in Bitcoin). Essentially, if a path of bilateral collateralized payment channels can be constructed between any two platform clients (nodes) C and D, then these two nodes can make payments to each other even if they do not have a direct payment channel between them. The maximum possible transfer amount is determined by the 'weakest link', that is, the bilateral channel with the lowest collateral (capacity) on the path.

Relying only on internal enforcement, via algorithmic and cryptographic tools, the main challenge of such transfers hopping over anonymous nodes is how to guarantee that each node will pass the funds or digital assets ownership forward and *prove* that it has done so. Notably, each intermediate node between the end-nodes C and D should not be able to unlock and use the funds because in such case the sender C would have no recourse (there is no external enforcement).

A way to solve this enforcement problem is an algorithmic tool called *timelocked redeemable secret (TRS)* script, $H(\sigma)$ (called HTLC in Bitcoin) where $H(\sigma)$ denotes the hash function of some secret statement σ. A hash function is a mathematical function that is easy to compute/verify (it is easy to compute $H(\sigma)$ when one knows σ) but nearly impossible to invert, that is, one cannot find σ from just knowing $H(\sigma)$. When payment is agreed upon, the intended final recipient/payee D creates the secret σ and sends its hash $H(\sigma)$ (but not σ itself!) to the sender/payer C. The sender C then creates a transaction with output funds F locked by the TRS script $l_1 = H(\sigma) \vee (k^C, T^1)$ and passes it to the first intermediate node, I_1 on the chain between C and D. Here C is assumed to have an established a collateralized payment channel with I_1. Note that node I_1 cannot redeem the funds F (since I_1 does not know the secret σ needed to obtain $H(\sigma)$) but she can be incentivized, by means of a small fee added to F, to pass along the TRS-locked funds F to another node, I_2 with whom I_1 has an established collateralized channel and locked by the script $l_2 = H(\sigma) \vee (k^{I_1}, T^2)$ with $T^2 < T^1$, and so on. The path $I_1, I_2, ...$ is constructed algorithmically. The role of the timelocks T^i is to ensure that each node along the chain would be able to get their funds/fee back in case the secret is never provided by D.

When D is finally reached, e.g., from some node I_n then, since D knows the secret statement σ, she claims the output F by debiting it from its payment channel state balance with I_n. Claiming F algorithmically triggers sending the secret σ to I_n who in turn claims F (plus a small fee) from her payment channel state balance with I_{n-1} and so on, until we reach C. The end result of the chain of bilateral state transitions is a payment of F from C to D, exactly as contractually intended. Note that no additional use of the blockchain is required in the process, except for posting collateral for establishing the pre-existing bilateral channels on the payment chain path.

6 Digital financial contracts – a roadmap

In this section I outline how a wide range of mechanism-design solutions can be implemented via a digital platform (e.g., using blockchain technology) and digital ('smart') contracts, i.e., computer code executable on the digital platform. Unlike in the previous sections, I do not describe in detail the algorithmic and cryptographic tools used but assume that they are available in the background, for example, via the implementation methods in Section 5, including the ability to post and use platform-based collateral. Instead, in this section I analyze more generally how digital platform accounts and information can be used to implement (constrained-) optimal contract allocations in several settings with exogenously or endogenously incomplete financial markets.

Consider agents $i = 1, ...N$ who transact via a digital platform (e.g., blockchain-based). Each agent i has two accounts, an expenditure account with balance a_t^i and a digital *token* account with balance w_t^i. The token account will be used to record history-dependent state variables, including non-monetary, (e.g., 'promised utility', credit rating, or debt level). Define transaction τ_t^{ij} to mean a transfer of funds from account i to account j at time t.

The recorded history (platform data) $\mathcal{T}$ up to time T consists of all transactions $\Theta^T \equiv \{\tau_t^{ij}\}_{ij,t}$ and commonly agreed upon/observed external states, e_t (e.g., date, GDP, weather, aggregate shock) for $t = 1, .., T-1$. Account balances a_t^i and w_t^i, summarized in the vectors A_t and W_t, can be constructed from the transactions data going back to $t = 0$ (as in Bitcoin) or queried from the platform (as in Ethereum). Define the platform state at time t as $\Sigma_t \equiv \{A_t, W_t, e_t\}$. Using superscripts to denote history up to T, all public digital data are $h^T = [\Sigma^T, \Theta^T]$.

A financial digital contract with initial date t_1 and end date t_2 among agents $\mathcal{J} \subset \{1, ..N\}$ is defined as the mapping

$$\phi(\mathcal{J}, t_1, t_2) : \Sigma \to \mathcal{T}$$

from (a subset of) the time-t state Σ_t to a matrix of contractual transactions (transfers) $\mathcal{T}_t$, $\forall t \in [t_1, t_2]$. The contract terms can be contingent on both past and future events (e.g., $a_{t_1}^i > 5$, $e_{t_2} = 3$). Once initiated, the contract is automatically executed and cannot be modified or terminated unless a pre-specified stopping condition is reached. Transactions are valid if the payment/transfer amount is available and its ownership is successfully cryptographically verified or invalid otherwise (see Section 2).

Focus on bilateral financial digital contracts, defined as state-contingent transactions $\mathcal{T}$ using the following four digital platform accounts:

(1) node 1's expenditure account (e.g., insured or borrower)

(2) node 2's expenditure account (e.g., an insurer or lender)

(3) an escrow account (node 3)

(4) a token account for node 1 (transactions using it are denoted τ^{11})

The optimal digital contract specifies contingent transfers solving the expected-payoff maximization problem,

$$\max_{\mathcal{T}(s)} \; EU(\mathcal{T}(s))$$
$$\text{s.t.} \quad \mathcal{T}(s) \in \Gamma(s)$$

where the state variable s is (a subset of) the current platform state Σ and U is a payoff function for node 1 or node 2, depending on the application). The feasible set $\Gamma(s)$ imposes restrictions on the transactions $\mathcal{T}(s)$ (see examples below) such as: payment channel state transition, financing constraints (e.g., borrowing limit), promise keeping constraint, incentive-compatibility constraints, or truth-telling constraints. Both one-period and multiperiod smart contracts (using dynamic programming, by defining $U(\mathcal{T}(s)) = u(\tau(s)) + \beta V(s')$ where u is the current payoff, s' is the next-period state, and $V(s')$ is the next-period value) can be incorporated.

Example A (working capital loan)

An entrepreneur with preferences $u(c)$ defined over consumption c uses technology $f(k)$ mapping working capital k into stochastic output $y \in \{y_1, .., y_{\#Y}\}$. In this example the state variable is the (commonly observed) output realization, $s = e = y$. The entrepreneur has no assets and needs to borrow the working capital k. Call the borrower's account node 1 and let node 2 be the lender's account. A one-period debt contract with interest $r > 0$ solves the following problem:

$$\max_{\tau^{21}} \; Eu(f(\tau^{21}) - \tau^{12})$$
$$\text{s.t. } \tau^{12} = (1+r)\tau^{21}$$

where τ^{21} is the loan size (a transaction from the lender 2 to the borrower 1) and $\tau^{12} = (1+r)\tau^{21}$ is the repayment amount (a transaction from node 1 to node 2). Involuntary default, when the loaned amount exceeds the available output, $\tau^{12} > y$, can be incorporated by adjusting the interest rate τ^{12}/τ^{21} accordingly. Strategic default can be addressed by using the escrow account assuming the borrower can post collateral.

Example B (defaultable debt)

Consider a *defaultable debt* setting between a borrower, node 1 and a lender, node 2. Let 1 have initial assets $b \geq 0$ and balance w in his digital token account (this can be interpreted as 'credit rating'). The borrower puts $\tau^{13} \in [0, b]$ as collateral into the escrow account (node 3), secured by the algorithmic tools discussed earlier. The contract then releases a loan of size $\tau^{21}(\tau^{13}, w, e)$

(where $\tau^{21} < \tau^{13}$) and stipulates a repayment (principal plus interest) $\hat{\tau}^{12}(\tau^{21}, w, e)$ where e is a verifiable external state (e.g., the central bank reference interest rate).

At the end of the period the agent decides to repay τ^{12}. If $\tau^{12} = \hat{\tau}^{12}$ (the agent repays the contracted amount) then the loan is deemed repaid in full, the agent's token account (credit rating) is updated to $w+\tau^{11}$, and the collateral in the escrow account is returned to the borrower, $\tau^{31} = \tau^{13}$. If instead the agent defaults, that is, $\tau^{12} = 0$ (partial default can be incorporated too), the lender receives the contracted payment from the escrow account via transaction $\tau^{32} = \hat{\tau}^{12}$ and the agent's token balance is updated, i.e., his credit rating is 'downgraded', to $w - \tilde{\tau}^{11}$. It is possible to distinguish between strategic and involuntary default by recording the appropriate information (e.g., agent's income or business profits) on the platform.

Example C (moral hazard)

Consider a risk-neutral insurer and a risk-averse agent with preferences $u(c, z)$ where c is consumption and z is a costly action (effort, diligence) unobserved by the insurer. The agent receives stochastic income $y \in \{y_1, .., y_{\#Y}\}$ which is i.i.d. over time. Let $P(y_j|z)$ denote the probability of realizing income level y_j given action level z. Income is observable and recorded in the public state e, so the digital state is $s \equiv (w, y)$ where y is current income and w is promised utility. The optimal risk-sharing contract $\mathcal{T}(s)$ can be implemented by using the token account (4) to store promised utility w as a digital token balance (similar to 'experience rating') encoding income history. In the beginning of each period, given the contract $\mathcal{T}(s)$, the agent chooses the action level z. Next, income y is realized and remitted to the insurer as $\tau^{12}(s)$. Then the digital contract transfers $\tau^{21}(s)$ to the agent's expenditure account (consumption) and $\tau^{11}(s)$ to the agent's token account. The constraints $\Gamma(s)$ include

$$\tau^{12}(w, y_l) = y_l \text{ for any } l \in 1, ..., \#Y$$

the incentive-compatibility constraint ensuring that the agent will supply the contracted action level z as opposed to some other level $\hat{z}$,

$$E_z[u(\tau^{21}(s), z) + \beta(w + \tau^{11}(s))] \geq E_{\hat{z}}(u(\tau^{21}(s), \hat{z}) + \beta(w + \tau^{11}(s))) \text{ for all } \hat{z} \neq z$$

and the promise keeping constraint ensuring that the agent is appropriately rewarded or sanctioned depending on the realized income history

$$E(u(\tau^{21}(s), z) + \beta(w + \tau^{11}(s))) = w.$$

Example D (*hidden income)*

An infinite-horizon version of the hidden income problem can be also implemented using a token account for promised utility. Consider a risk-averse agent and preferences $u(c_t)$ where c_t is consumption and a risk-neutral insurer. The agent's income stream $\{y_t\}$ is i.i.d. over time and y can take values $y_1, ..y_{\#Y}$. Unlike in the previous example, there is no costly action but the agent's income is unobserved by the insurer. The risk-sharing problem can be written as a dynamic program using the agent's promised utility w as state variable that encodes the history of output realizations.

$$\Pi(w) = \max_{\{\mathcal{T}\}} E(-\tau_j^{21} + \beta\Pi(w + \tau_j^{11}))$$
$$\text{s.t. } u(y_j + \tau_j^{21}) + \beta(w + \tau_j^{11}) \geq u(y_j + \tau_l^{21}) + \beta(w + \tau_l^{11}) \text{ for any } j, l = 1, ..\#Y$$
$$E(u(y_j + \tau_j^{21}) + \beta(w + \tau_j^{11})) = w$$

where the first constraint is the truth-telling constraint (income y_j is realized but the agent considers reporting y_l) and the second constraint is the promise-keeping constraint. The initial promised utility token balance w_0 can be chosen to give zero ex-ante profits to the insurer or satisfy an ex-ante participation constraint for the agent.

7 Conclusions

I describe how key economic concepts underlying (financial) contracts – property rights, information, commitment, and enforcement – map to and can be implemented though digital algorithmic tools and code – transactions and full history thereof, simple and multisig locking scripts, signatures and timelocks. Blockchain technology excels at recording, securing (locking) and digital verification (unlocking) of funds ownership and other information. The main limitation is the need for collateral – any promised future payments must be fully backed and secured. Complex locking scripts including multiple signatures and timelocks can be used to enable payments, based on posted collateral funds. While the need for digital collateral appears a costly limitation, I show how that collateral can be leveraged to back multiple off-platform transactions via pre-established payment channels which saves on transaction fees and waiting time costs.

The main takeaways are as follows. Digital platforms such as blockchains and digital smart contracts have the potential to be a very powerful tool for implementing complex mechanism-design solutions with multiple contingencies and conditionalities and based on current real-time or recorded information. Their usefulness is predicated but also limited by the algorithmic tools and constraints on which the platform computer code is based. Specifically, enforcement of payments has to be secured by collateral. Conditional on that, digital platforms can be used to implement

constrained optima and automate complex mechanism design solutions in real-world settings with private information (e.g., unobserved characteristics or actions) and limited enforcement.

My main goal in this paper was to distill and break down blockchain technology and digital contracts to their fundamental building blocks and provide detailed analysis of their strengths and limitations, as opposed to using the terms 'blockchain' or 'smart contract' as catchall phrases for implementing digital transactions. I show how economic theory, specifically mechanism design, can be used to assist digital platform developers and programmers regarding what types of algorithmic tools to include in future implementations or upgrades, for instance, basic ready-made financial contracts such as loans or contingent insurance products. Combining the technological strengths of digital (blockchain) platforms in granular information recording and verifiability, proof of ownership, security and privacy with enhanced enforcement and trust mechanisms (possibly including trusted third parties) can yield further benefits while saving on collateral requirements and transaction costs.

References

[1] Abreu, D., D. Pearce and E. Stacchetti (1990), "Toward a Theory of Discounted Repeated Games with Imperfect Monitoring", Econometrica 58: 1041-63

[2] Albanesi, S. and Sleet, C. (2006), "Dynamic Optimal Taxation with Private Information", Review of Economic Studies, 73: 1-30

[3] Andolfatto, D. (2108), "Blockchain: What It Is, What It Does, and Why You Probably Don't Need One". Federal Reserve Bank of St. Louis Review.

[4] Aronoff, D. and R. Townsend (2023), "A Smart Contract to Increase Intermediation Capacity in the Repo Market", working paper, MIT

[5] Atkeson, A. and R. Lucas, (1992), "On Efficient Distribution with Private Information", Review of Economic Studies, 59: 427-53

[6] Antonopoulos, A. (2017), *Mastering Bitcoin: Programming the Open Blockchain*, O'Reilly

[7] Berentsen, A. and F. Schar (2018), "A Short Introduction to the World of Cryptocurrencies", Federal Reserve Bank of St. Louis Review.

[8] Broer, T., M. Kapicka and P. Klein (2017), "Consumption risk sharing with private information and limited enforcement", Review of Economic Dynamics 23: 170-90

[9] Buterin, V. (2013), “Ethereum White Paper”, manuscript

[10] Catalini, C. and J. Gans (2016), “Some Simple Economics of the Blockchain”, NBER Working Paper 22952

[11] Chapman, J., R. Garratt, S. Hendry, A. McCormack and W. McMahon (2017), “Project Jasper: Are Distributed Wholesale Payment Systems Feasible Yet?”, Bank of Canada Financial System Review

[12] Chiu, J. and T. Koeppl (2019), “The Economics of Cryptocurrencies: Bitcoin and Beyond”, Staff Working Paper 2019-40, Bank of Canada

[13] Chiu, J. and T-N. Wong (2014), “E-Money: Efficiency, Stability and Optimal Policy”, Working Paper 2014-16, Bank of Canada

[14] Chiu, J. and T-N. Wong (2021), “Payments on Digital Platforms: Resiliency, Interoperability and Welfare”, Staff Working Paper 2021-19, Bank of Canada

[15] Chiu, J., C. Kahn and T. Koeppl, (2022), “Grasping De(centralized) Fi(nance) Through the Lens of Economic Theory”, Staff Working Paper 2022-43, Bank of Canada

[16] Chiu, J., E. Ozdenoren, K. Yuan and S. Zhang (2023), “On the Fragility of DeFi Lending”, Staff Working Paper 2023-14, Bank of Canada

[17] Cole, H. and N. Kocherlakota (2001), “Efficient Allocations with Hidden Income and Hidden Storage”, Review of Economic Studies, 68: 523-42

[18] Committee on Payments and Market Infrastructures, CPMI (2017), “Distributed Ledger Technology in Payment, Clearing and Settlement: An Analytical Framework”, BIS

[19] Cong, L. and Z. He (2019), “Blockchain Disruption and Smart Contracts”, Review of Financial Studies

[20] Decker, C. and R. Wattenhofer (2015), “A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels”, in Pelc, A., Schwarzmann, A. (eds) *Stabilization, Safety, and Security of Distributed Systems*.

[21] Eyal, I. (2015), “The Miner’s Dilemma”, 2015 IEEE Symposium on Security and Privacy

[22] Fernandes, A. and C. Phelan (2000), “A Recursive Formulation for Repeated Agency with History Dependence”, Journal of Economic Theory, 91: 223-247

[23] Gans, J. (2019), “The Fine Print in Smart Contracts”, NBER Working Paper 25443

[24] He, D., K. Habermeier, R. Leckow, V. Haksar, Y. Almeida, M. Kashima, N. Kyriakos-Saad, H. Oura, T. Sedik, N. Stetsenko, and C. Verdugo-Yepes (2016), “Virtual Currencies and Beyond: Initial Considerations”, IMF Staff Discussion Note 16-03.

[25] Holden, R. and A. Malani (2018), “Can Blockchains Solve the Holdup Problem in Contracts”, working Paper, No.2018-12, Becker-Friedman Institute, Chicago

[26] Karaivanov, A. and F. Martin (2015), “Dynamic Optimal Insurance and Lack of Commitment”, Review of Economic Dynamics 18(2), p.287-305

[27] Karaivanov, A. and R. Townsend (2014), “Dynamic Financial Constraints: Distinguishing Mechanism Design from Exogenously Incomplete Regimes”, Econometrica 82:887-959

[28] Karaivanov, A., B. Mojon, L. Pereira da Silva and R. Townsend (2023), “Digital Safety Nets – A Roadmap”, BIS Papers 139

[29] Kehoe, P. and F. Perri (2002), “International Business Cycles with Endogenous Incomplete Markets”, Econometrica, 70: 907-28

[30] Kiayias A., E. Koutsoupias, M. Kyropoulou, Y. Tselekounis (2016), “Blockchain Mining Games”, 2016 ACM Conference on Economics and Computation, 365-382

[31] Kiviat, T, (2015), “Beyond Bitcoin: Issues in Regulating Blockchain Transactions.” Duke Law Journal. Vol. 65: 569

[32] Kocherlakota, N. (1996), “Implications of Efficient Risk Sharing Without Commitment”, Review of Economic Studies, 63: 595-609

[33] Koeppl, T. and J. Kronick (2017), “Blockchain Technology – What’s in Store for Canada’s Economy and Financial Markets?”, Commentary No.468, C.D. Howe Institute

[34] Lee, M., A. Martin and R. Townsend (2022), “Optimal Design of Tokenized Markets”, MIT

[35] Ligon, E., J. Thomas and T. Worrall, (2000), “Mutual Insurance, Individual Savings and Limited Commitment”, Review of Economic Dynamics 3: 216-246.

[36] Ligon, E., J. Thomas and T. Worrall, (2002), “Mutual Insurance and Limited Commitment: Theory and Evidence in Village Economies”, Review of Economic Studies, 69: 209-244.

[37] Nakamoto, S. (2008), “Bitcoin: A Peer-to-Peer Electronic Cash System”, white paper

[38] Peyrott, S. (2017), “An Introduction to Ethereum and Smart Contracts”, manuscript

[39] Phelan, C. (1995), “Repeated Moral Hazard and One-Sided Commitment”, Journal of Economic Theory 66: 488-506.

[40] Phelan, C. and R. Townsend (1991), “Computing Multi-Period, Information-Constrained Equilibria”, Review of Economic Studies, 58: 853-81.

[41] Raskin, M. and D. Yermack (2016), “Digital Currencies, Decentralized Ledgers and the Future of Central Banking”, NBER Working Paper 22238

[42] Routledge, B. and A. Zetlin-Jones (2018), “Currency Stability Using Blockchain Technology”, working paper

[43] Spear, S. and S. Srivastava (1987), “On Repeated Moral Hazard with Discounting”, Review of Economic Studies 53: 599-617

[44] Szabo, N. (1996), “Smart Contracts: Building Blocks for Digital Markets”, Extropy 16

[45] Szabo, N. (1998), “Secure Property Titles with Owner Authority”, manuscript

[46] Thomas, J. and T. Worrall (1988), “Self-Enforcing Wage Contracts”, Review of Economic Studies, 55: 541-55

[47] Townsend, R. (1982), “Optimal Multiperiod Contracts and the Gain from Enduring Relationships under Private Information”, Journal of Political Economy 82:1166-96

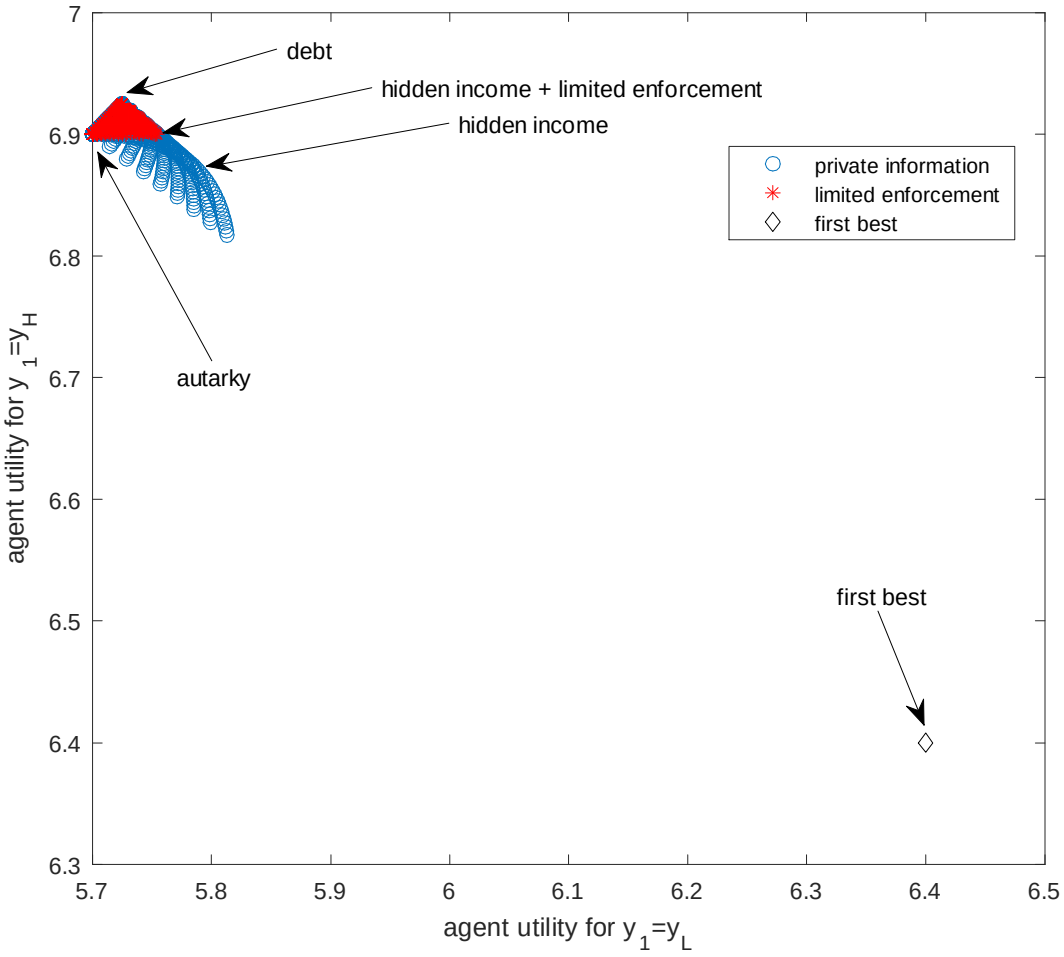
[48] Townsend, R. (2020), *Distributed Ledgers: Design and Regulation of Financial Infrastructure and Payment Systems*, MIT Press

[49] Townsend, R. and N. Zhang (2023), “Technologies that replace a central planner”, AEA Papers and Proceedings 113:257-262.

[50] Yermack, D. (2014), “Is Bitcoin a Real Currency? An Economic Appraisal”, NBER Working Paper 19747

[51] Yermack, D. (2016), “Corporate Governance and Blockchains”, NBER Working Paper 21802

Figure 1: Multi-period Risk Sharing



Note: The Figure illustrates the two-period numerical risk-sharing example from Section 4.1 for different financial settings: autarky, debt, private information (hidden income), limited enforcement, and full insurance (first best). The arrows point to the optimal allocation (in utility terms) in the different settings conditional on the agent's first-period income y_1 (high or low).