Blockchains, Collateral and Financial Contracts

Alexander Karaivanov
Simon Fraser University

January 2021

Abstract

I map the link between financial contracts and the algorithmic tools and constraints of blockchain technology related to property rights, information, commitment, and enforcement. I describe and formalize the microfoundations and possible use of blockchains as direct conduit for implementing financial contracts in incomplete markets settings and as collateral mechanism for on- and off-chain transactions and contracts.

1 Introduction

Internet-based digital markets and platforms have grown rapidly with recent technological advances making possible previously prohibitively costly search, indexing, matching, storage and quick transmission of large amounts of data. Blockchain internet platforms are a fast growing segment of digital markets. On January 28, 2021 the two largest blockchain platforms, Bitcoin and Ethereum, had market capitalizations of $632bn and $152bn respectively and 100,000s of daily transactions.

A *blockchain* is a decentralized (most often free to join and access) networked computer database of transactions and other data (e.g., 'smart contracts' or 'tokens') among pseudo-anonymous accounts, organized in a sequence of connected time-stamped blocks. Network nodes or users each hold or have free access to a full copy of the database, hence blockchains are often called 'distributed ledgers'. A user may have many blockchain accounts. Ownership of funds and transaction verification (e.g., prevention of double spending or re-writing the ledger) is done via consensus algorithms (computer code) and cryptographic technology (private-public key pairs, hash functions[1]), without the need for a trusted central party. Economic incentives to verify, perform and record transactions on the blockchain are provided by transaction fees and the issuance of digital tokens (cryptocurrency).

In this paper I explore and map the relationship between key economic concepts and constructs upon which financial contracts and transactions are based (property rights, information, commitment, enforcement) and the algorithmic tools and constraints of blockchain technology, with specific economic examples. Unlike other papers in this literature, I explicitly model actual micro-level details regarding how different types of on- and off-chain transactions can be enabled by blockchain technology, with the ultimate goal of constructing a correspondence between mechanism design concepts from the theory literature and the actual algorithmic building blocks of blockchains (accounts, transactions, single or multi signatures and timelocks).

I characterize the main advantages and limitations of blockchain technology in implementing and enforcing financial contracts and payments, with applications to settings with financial market frictions (exogenously incomplete markets, private information, limited enforcement). I analyze how these financial market frictions could be addressed by transactions and (smart) contracts defined on a blockchain network.

The main takeaways are as follows. Blockchain technology excels at locking, unlocking and tracking funds, that is, verifying and transferring ownership. Blockchains are also excellent in

[1]Hash functions are mathematical functions which are easy to evaluate but nearly impossible to invert without many brute force tries (known as 'proof of work'). For example, solving a 10,000-piece jigsaw puzzle is easy to verify but very time-consuming to do.

recording and making publicly accessible complete current and historical information of what happened on the blockchain but, naturally, not what happens off it. An important limitation of self-enforcing transactions performed via blockchain technology is the need for **on-blockchain collateral** (escrow) for enforcing promised payments; that is, the algorithmic tools alone are insufficient. Any payment promise must be backed up by locked funds (penalty, guarantee), otherwise it is unenforceable. I show, however, how blockchain technology can be used to enable collateralizing and securing off-chain payments, saving on transaction fees. Leveraging the main advantages of blockchains (history and information verifiability and security of proving ownership) with trust/commitment/enforcement mechanisms (including trusted third parties) can facilitate implementing complex abstract notions and solutions from mechanism design and general equilibrium theory into reality while saving on collateral costs.

In this paper I abstract from issues related to cybersecurity, mining, or achieving consensus on the blockchain. These blockchain technology ingredients are treated as exogenous and assumed to function as intended and specified by the computer code. All following discussion about information, commitment and enforcement in blockchains is predicated on this assumption and should be interpreted accordingly, especially when comparing to other existing technologies or institutions such as courts, formal financial intermediaries, centralized databases, etc. I also do not analyze monetary issues, e.g., stable coins, monetary policy questions, the possible impact on blockchain technology on central banks, or regulation issues.

Related literature

Much of the early economic research on blockchains consists of review papers (Koeppl and Kroninck, 2017; Chapman et al., 2017; CPMI, 2017; He et al., 2016; Catalini and Gans, 2016; Tasca et al., 2016) focuses on currency and/or monetary issues (e.g., Yermack, 2014 on Bitcoin; Raskin and Yermack, 2016 on digital currency central banking; Chiu and Wong, 2014 on e-money and system stability; Andolfatto 2018; Berentsen and Schar, 2018). There is also a technical literature emphasizes security and prevention of double spending (Nakamoto, 2008; Buterin, 2013; Antonopoulos, 2017; Peyrott, 2017), also with game theory applications (Eyal, 2015; Kiayias et al., 2016). Chiu and Koeppl (2017) are among the first authors to propose an economic model of blockchains and transaction verification incentives, calibrate it to data and perform policy analysis. Early conceptualizations of smart contracts and digital ownership can be found in Szabo (1996, 1997, 1998). On the blockchain as 'public memory', see Andolfatto (2016a,b). On policy and regulation see Kiviat (2015), Chapman et al., 2017 or Szabo (2017) among others.

I relate to the above literature but abstract from monetary aspects and instead focus on blockchains as digital platforms onto which payments, credit and insurance in incomplete market settings can be mapped and implemented. Doing so, I draw on the theory 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; Green, 1987; Abreu et al., 1990; Phelan and Townsend, 1991; Atkeson and Lucas, 1992; Phelan, 1998; Fernandes and Phelan, 2000; Cole and Kocherlakota, 2001; Albanesi and Sleet, 2006), or with limited commitment (Thomas and Worrall, 1988, 1994; Phelan, 1995; Kocherlakota, 1996, 1998; Krueger, 1999; Alvarez and Jermann, 2001; Ligon et al. 2000, 2002; Kehoe and Perri, 2002; Broer at al., 2017). I also use blockchain-specific mechanism design elements, as in Chiu and Wong (2015) or Chiu and Koeppl (2017) and linear programming representations and solution methods (Prescott and Townsend, 1984; Doepke and Townsend, 2006; Ligon et al.,2000; Karaivanov and Townsend, 2014).

This paper is most closely related to recent work trying to conceptualize the possible (future) applications of distributed ledgers and blockchains in economic activity, drawing on results from mechanism design and contract theory. Contributions include the agenda-setting review by Townsend (2019) and the applications in Holden and Malani (2019) on the hold-up problem, Cong and He (2019) on collusion, or Gans (2019) on international trade.[2] I draw on this literature but differ by exploring in detail the algorithmic tools and constraints of blockchain technology as related to property rights, commitment, enforcement and information.

2 Blockchain technology - key elements

I start by introducing the key elements (tools) of blockchains in a stylized form, emphasizing their economic significance for enabling contracts and transactions. The terminology used borrows from, but is not always equivalent to the terminology in the various blockchain 'white papers'.

2.1 Transactions

In this paper I use the term *blockchain* to mean a dataset of recorded *transactions* organized in time-stamped data 'blocks', together with the associated computer code/algorithms. Cryptographic tools (Merkle-Patricia trees, hash functions, etc.) ensure the integrity of the data and the interconnectedness of the blocks (hence the "chain" in blockchain), however, I abstract from these technical issues here. Transactions are the main building block of blockchains.

1. Transactions – input and output funds

Define a transaction τ as the process of unlocking ('spending') pre-existing unspent funds x_{t-k} (*input funds*) and locking them into new unspent funds x_t (*output funds*). This can be thought

[2]The work by Routledge and Zetlin-Jones (2018) on self-fulfilling currency attacks is also broadly related, to the extent that it demonstrates how smart contracts on the Ethereum blockchain can be used to implement a currency peg.

of sending or transferring funds from one person or account to another but more general interpretations are possible. New blockchain funds (cryptocurrency) originate from special ‘coinbase’ transactions as a reward for miners but I abstract from this here; the focus is on transacting pre-existing funds.

2. Cryptographic locking scripts (keys)

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

3. Signing a transaction

The process of supplying an unlocking script (‘signature’), u_t to the input funds of a transaction is called signing the transaction. If the supplied unlocking script matches the locking script for some 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 x_t by a new specified locking script l_t.[3]

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* $\{\tau_t^i\}_{i,t}$ where i indexes a transaction and t indexes its block, up to the current date.[4] These data can be used to trace back all past funds transfers or balances, e.g., by account or other criteria. There is a natural mapping between $\mathcal{T}$ and the concept of ‘history’ in contract theory / mechanism design which I will use below.

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

[4]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 forms of locking scripts l that are present in major blockchain implementations (e.g., Bitcoin)

(a) **simple key**

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

where k^a is a single private key required to spend/unlock the input funds 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, a 2-of-2 multisig key,

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

means that two private keys, k^a and k^b are required to spend the input funds. That is, to be valid the transaction must be signed by two signatures proving possession of both keys.

(c) **composite script**

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

In this paper I will use two main types of composite locking scripts:

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

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

– *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.[5] Anyone with knowledge of σ can redeem the locked funds immediately. Alternatively, one can redeem the funds by signing with key k after T time periods.

2.1.2 Writing vs. posting a transaction

In the following analysis I distinguish between

[5]Hash functions $H(w)$ are special mathematical functions that take a message (string) w of any length as input and output a value, $H(w)$ of fixed length. Importantly, hash functions are easy to compute but practically impossible to invert.

writing a transaction – the act of defining a transaction's input funds, locking scripts, signature(s) and output funds, as defined in Definition 1.

posting a transaction on the blockchain – the act of submitting a written transaction for execution through a software node connected to the blockchain network. Here, I abstract from waiting time, effectively assuming that all posted transactions are confirmed (recorded on the blockchain).

An important difference between writing and posting a transaction is that writing a transaction does not involve paying blockchain transaction fees (reward to miners) while posting it does. Once posted and executed a transaction is irreversible.[6]

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, 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 network/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 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 input funds. I will show that this technological feature of blockchains is very useful in constructing multiperiod or multi-party financial transactions.

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 fund balances can be derived from the records of all transactions on the blockchain (henceforth, the blockchain data $\mathcal{T}$) as the sum of all funds locked by given key (e.g., in Bitcoin); or are retrievable from the blockchain as part of the virtual machine state (e.g., in Ethereum).

[6]A new transaction could be constructed to "reverse" the actual ownership of funds but this involves a new set of inputs and outputs and will be recorded as a different transaction.

2.2 Algorithmic constraints

Given the definition of a transaction (Definition 1), the blockchain algorithm imposes and enforces the following constraints on input funds, output funds, 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 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) for each input x_t^i must match (cryptographically satisfy) the locking script l_{t-k}^j of the previous output it attempts to unlock/spend[7]

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

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 inequality in (C4) captures *transaction fees* (blockchain network processing fees) that normally accrue to the miner of the block in which a transaction is included. Below I ignore those fees for simplicity. Note that no new funds are injected via transactions[8] 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 (e.g., Nakamoto, 2008). These constraints essentially ensure that a transaction cannot spend funds that are not available and that are not unlockable. If an attempt is

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

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

made to spend funds 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 algorithm and network keep track of the complete set of unspent funds (so-called UTXO set), which is updated after every recorded block of transactions. Other blockchain algorithms (e.g. Ethereum) directly keep track of actual account balances as the blockchain 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 Blockchains and financial contracts – possibilities and limitations

This section discusses how the key ingredients and algorithmic tools of blockchains help or constrain writing and enforcing financial contracts and transactions. The focus is on the key building blocks of contracts – property rights, information (verifiability), commitment (trust) and enforcement. The strengths and limitations of 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 funds / assets (e.g., amount of unspent Bitcoin or Ethereum balance) are fully digital (algorithmic) and are established and verified by cryptographic tools. Examples of the latter tools include the *scriptSig*, *scriptPubKey* functions in Bitcoin or the transaction nonce and ECDSA signatures in Ethereum.

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

The property rights over funds in blockchains are of 'bearer' type and not tied to a particular user identity. That is, anyone who produces the required signature matching the locking script can spend the funds. Importantly, in most (all?) currently existing blockchain platforms there are no external legal property rights, that is, rights or claims existing outside of the digital algorithmic rights just described. The anonymity of accounts and lack of central parties and regulation currently rule out other existing methods to record property rights but this is a feature of blockchains that could change in the future.

Multisig and *timelock* locking scripts can be used as flexible covenants to constrain property rights over funds. Multisig locking scripts require the presentation of m out of n signatures (unlocking scripts), thus constraining unilateral use. Timelocks can be used to constrain *usage* property rights, since timelocked funds cannot be spent by anyone before expiration of the timelock, for example, a ‘correct’ private key k^a cannot unlock/spend funds locked by the unexpired timelock script (k^a, T^a) with $T^a > 0$.

- Strengths: cryptographically secure proof of ownership; low costs of acquiring or 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 one’s ability to make credible/deliverable promises about future actions. For example, an online seller can commit to ship the purchased item after receiving payment. Related to this, the term *limited commitment* is used to assume that only one of two contract parties is bound by their promises or that commitment is possible only in the short-term (e.g., single period) or with some probability less than one. A huge literature on time inconsistency and lack of commitment exists analyzing those issues.

Here I abstract from any external commitment devices or institutions and focus solely on the blockchain algorithmic tools that can be used to ensure commitment to future payments. Blockchains are often described as ‘trustless’ decentralized platforms on which economic transactions can be made. The term ‘trustless’ is not fully correct. While most blockchain applications (e.g., Bitcoin, Ethereum) indeed do not rely on a trusted authority in the traditional sense (court system, banking system, etc.), they rely strongly on trust in the computer code and algorithm on which the platform is based and run.[9]

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

Condition (i): the referenced *input funds* x_{t-k} *must be available* and accessible at the moment of posting the transaction on the blockchain. Obviously, this requires having the requisite *signature* / unlocking script(s), u_t.

[9]This includes technical issues such as reaching and maintaining consensus (e.g., about the longest transaction blockchain), susceptibility to malicious attacks, network throughput, etc. which I do not discuss here.

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

The second implication of (i) is that a necessary condition for any *future* transfer commitments is that the required funds must be **fully collateralized**. That is, the promised funds must be locked until needed and still available to be unlocked at the time the transfer is due. In practice this can be achieved using the algorithmic tools reviewed 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. Timelocks directly lock the funds from use by *any party* and can algorithmically ensure that the funds are still available at the intended redemption date (note that potential self-control problems are avoided too).

Condition (ii): the transaction which established the promised funds availability (known as collateral or funding transaction, see Section 4) must be *posted and confirmed on the blockchain* (transaction fees must be paid) – this activates the algorithmic tools and ensures the future availability of funds.

- Strengths: commitment and trust can be generated algorithmically
- Limitations: commitment and trust is limited to the available algorithmic tools (timelocks, multisig) and require locking funds as collateral (costly)

3.3 Enforcement

Contract *enforcement* refers to executing the terms of the contract as stated and/or intended. Specifically, my focus in this paper is on enforcing transactions transferring funds between blockchain accounts (unlocking and re-locking funds).

Blockchain transactions are enforced algorithmically (automatically) by software, as long as the *property rights* and *commitment* conditions ensuring transaction validity are satisfied:

– input funds ownership is proved cryptographically

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

– transaction fees are paid.

Conditional on the above, the receiver of the transaction output does not need to do anything (except possibly wait) after a transaction is posted and successfully validated on the blockchain. Contract verifiability and execution is algorithmic, via unambiguous computer code. The latter reduces uncertainly about what is promised (a form of counterparty risk) and legal (interpretation) risk; see Holden and Malani (2019) for further discussion.

An enforcement problem could therefore arise only when a transaction is *invalid* and hence rejected by the computer code when submitted to the blockchain. This could happen, for example, when the transaction's input funds has already been spent in another previously validated transaction (e.g. in Bitcoin) or, equivalently, if there is insufficient balance in the sending account at the time of posting (in Ethereum). Note that the recipient has no resort in such case since external enforcement is ruled out.

- Strengths: enforcement is automatic, no 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 (courts, arbitration, etc.)

3.4 Information

By design a blockchain records *complete information of all posted and confirmed transactions* going back to the initial block 0. Also by design, everything that is posted on the blockchain is *public information* that is free to access. Typically, the minimum amount of data written on the blockchain are transactions (input, output, value), organized in time-stamped data blocks that are organized in an inter-connected chain (via hash functions, Merkle trees, etc.). Cryptography makes it nearly impossibly hard to modify past written data – the recorded data are permanent – this data immutability is a design feature, not a limitation. The public, complete and permanent nature of the blockchain data has a natural parallel with the notion of *history* in mechanism design.

The transactions information on the blockchain could also be used to construct account balances at any moment of time (block height). Depending on the platform, other data may be written too – e.g., smart contracts, gas used, gas limit and gas price in Ethereum.

While blockchains provide a vast amount of public information in the form of the recorded transaction data, obviously they do not necessarily solve or avoid important economic problems arising from asymmetric information such as hidden actions, hidden income, unobserved type. Essentially, anything that is not recorded on the blockchain can be private information for other users or counterparties. In addition, most blockchain platforms are permissionless and anonymous and users can have multiple addresses/accounts which renders difficult relating transactions to actual individuals / firms unless they want to reveal such information (e.g., in Bitcoin by default most wallet software would create a new account for each transaction). Coordination and other efficiency gains from required posting of information are possible.[10]

- Strengths: complete, permanent and public record of historical information, similar to *history* in mechanism design (see the next section for application)

- Limitations: any off-blockchain information can remain hidden, including strategically; the immutability of recorded data might cause problems in practice (e.g., permanently locked funds; coding errors)

3.5 Blockchains strengths and limitations – discussion

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

The main limitation of blockchains in writing and enforcing economic contracts and transactions appears to be the need for *full collateral* regarding future promised payments.[11] This collateral requirement can be costly, as shown in the economic applications in the next section. However, in Section 5 I show that collateral (locked funds) posted on the blockchain can be used, together with the algorithmic tools in Section 2, to back multiple off-chain 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 commitment mechanism? It is certainly possible to use a blockchain to *record* reputation-related information, for example, as token balance. However, it is unclear how outsiders (e.g., new contract parties) could verify the authenticity of such reputation ratings (e.g., a trader may self-generate transactions with other accounts s/he controls and/or rate himself highly, similar to creating or purchasing fake reviews on business rating websites).

Another common commitment mechanism from the economics literature involves using punishment threats not based on collateral, for example, the threat of autarky or no trade. Contract parties could indeed write such self-enforcing mechanisms using blockchain technology (smart contracts could be very helpful in this). However, doing so imposes additional constraints and

[10]See Townsend (2019) for more discussion on this issue and also on the possible gains from obfuscating certain information, depending on the economic setting.

[11]In addition, there are costs of posting blockchain transactions (transaction fees) and waiting for confirmation but these economic costs are often lower than the direct or implied costs in other transaction methods.

may severely limit the set of feasible trades and efficiency (see the next section for a risk-sharing example). It is indeed possible to punish a counterparty with no future trade if a contractual payment is not made but, without external enforcement or the possibility to restrict access, how can one prevent the offender from contracting with another lender, borrower or seller? In addition, such punishments are often time-inconsistent and there is the issue of how to prevent fraudulent punishment – for example, banning user accounts is ineffective in a permissionless anonymous blockchain.

4 Applications

I next present two specific examples of economic settings (multiperiod risk sharing and international trade) to illustrate the main contracting issues and frictions from the theoretical point of view and show how the tools of blockchain technology can be used to address them.

4.1 Risk sharing

Townsend (1982) studies a multi-period 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 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 and insurance against the income shocks. The agent's income can be public or private information. In the latter case, assuming a two-period model and two possible values of 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. Also, both the insurance and debt contracts attain higher utility than autarky.

Using standard arguments, an infinite-horizon contracting problem in Townsend (1982)'s setting can be written recursively[12] in terms of the agent's promised utility (present value of future utility), w as the state variable:[13]

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

$$\text{s.t. } E(u(y_j + \rho_j) + \beta w'_j) = w \quad \text{[promise keeping]}$$

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

[12]All variables are time-dependent if T is finite.

[13]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.

where $\Pi(w)$ is the insurer's profit function and the expectation is taken over the income states. This formulation is equivalent to making the consumption transfers ρ_j conditional on the complete history of agent messages about her income realizations, i.e.,

$$\rho_{jt} = \rho(y_1, y_2, ..., y_t)$$

The first-best outcome is full-insurance – the agent receives constant consumption $c_j = \bar{c}$ across all income states. In contrast, when the agent's income y_j is private information, the truth-telling constraint must be imposed to ensure, by the Revelation principle, that the agent would optimally report her income truthfully and earn the on-path current transfer ρ_j and promised utility w'_j as opposed to reporting y_l (e.g., lower income) and receiving the off-path values ρ_l and w'_l.

Townsend's setting is a prototypical application of mechanism design theory to derive a complex financial contract (credit mixed with insurance) in the presence of private information. Typically the constrained-optimal contract in such settings can be solved only numerically. Therefore, a blockchain-based smart contract could be coded to compute and implement the optimal allocation solving problem (1), either by keeping track of the agent's history of messages about income y_j or, equivalently, by keeping track of promised utility w in a blockchain token account (see Section 6). Blockchain technology is perfectly suited for storing such data and also for coding the automatic computation of the needed transfers (e.g., via smart contract), provided the required economic structure is known as assumed in the theory literature (that is, the distribution of income, preferences, etc.).

Note that contracting problem (1) and its solution (like many other similar examples from the literature) *assume commitment* by both parties. That is, it is assumed that the requisite insurance transfers ρ_j can be enforced costlessly. This is an issue that blockchain technology cannot address automatically. What if a party cheats and does not make the required transfer, e.g., by spending its input funds? As discussed earlier, this can happen if the required funds are not locked or secured in advance.

More specifically, suppose the agent could renege after observing her income, but before having to make the transfer ρ_j and the insurer can commit to punishing with no-trade 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).

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

where $V^a = \sum \beta^t E(u(y_t))$ is the agent's autarky value. Imposing this additional constraint on problem (1) limits the set of feasible trades and reduces efficiency.

Discussion

The private information and commitment problems can lead to a significant reduction in surplus (see Figure 1 for illustration). The blockchain cannot solve the private information (hidden income) problem unless income accrues / is recorded directly on the blockchain, without human intervention. The blockchain can help implement the private information constrained optimum by keeping track of history or promised utility.

Blockchain-based **collateral** can help with the commitment problem. One (possibly costly) way would be to lock sufficient funds ex-ante that allow to pay any possible sequence of due transfers. I describe a "payment channel" implementation using this idea in Section 5.2.

Another way would be to only lock sufficient funds to ensure that the insurance contract is self-enforcing, that is both parties have economic incentive to not renege on the contract, at any moment of time after any history. For example, if we focus only on the agent's commitment problem, then locking funds F_j as collateral, to be taken away if the agent reneges on a due transfer $\rho_j < 0$, would satisfy the self-enforcement condition in state j as long as F_j satisfies

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

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

By providing the technological possibility to lock funds as collateral blockchains can support the optimal history-contingent contract. The tools used are the blockchain data, a blockchain account to record the history of agent messages $\tilde{y}^t$ or promised utility $w(\tilde{y}^t)$ and the cryptographic locking scripts and signatures, to secure the collateral.

Numerical example

Assume $T = 2$, two output levels $y_L = 3$ and $y_H = 5$ each with probability $\pi = 1/2$ and preferences $u(c) = c - .05c^2$ and $\beta = 1$. Then we have the following mechanism design solutions, depending on the assumed contracting environment / economic frictions (superscripts denote time and subscripts denote income history)

Risk-sharing example, Townsend (1982)

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. first best	-1	1	-1	1	-1	1	6.4
3. hidden income							
a. debt	$-.5$	$.5$	$.5$	$.5$	$-.5$	$-.5$	6.325
b. insurance[14]	$-.56$	$.64$	$.45$	$.45$	$-.54$	$-.54$	6.33

In this example the constrained-optimal insurance contract 3b. in the hidden income setting does not satisfy the self-enforcement constraint (2). Hence it cannot be implemented (without collateral) if there is no commitment. See Figures 1 and 2 for further illustration of the impact of the information and commitment frictions.

4.2 International trade

Gans (2019) considers an example of an international trade setting. 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, mutually beneficial trade opportunity exists for some price $P \in (C, V)$. Gans introduces two potential issues related to a trade contract enforcement:

(i) 'hold up' – the buyer may receive the product but not pay for it (or try to pay less ex-post)

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

The author then discusses a mechanism-design solution to the trade problem (based on work by Moore and Repullo, 1988 and Moore, 1992 on sequential mechanisms) and argues how blockchain technology (smart contract) can be used to implement this solution. 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.

[14]For simplicity Townsend (1982) restricts per-period insurer's profits to zero. Here, zero ex-ante expected profits for the insurer are assumed instead, yielding a minor welfare gain.

2.2 return the product for refund f.

Stage 3 (settlement) If the buyer B chose 2.2 then B is issued 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 contract 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 mechanism-design solution, observe that **collateral** (funds locked in escrow) is required to ensure commitment to the contract. First, the buyer must post/lock funds $P + F$ to ensure payment for the good, P and the potential payment of the fine F in the off-equilibrium path where B challenges S. The latter is needed to prevent frivolous challenges. Second, the seller must post funds $2F$ as collateral in the off-equilibrium case of refund in Stage 3 (possibly F of this could be the buyer's fine that S received in Stage 2, if properly script-locked).

The amount of collateral required for the implementation can be significant. Gans (2019) shows a numerical example with $V = 30$, $v = 0$, $c = 5$, $P = 15$, $C = 10$, $F = f = 6$ and $p = 10$ which satisfies the parametric sufficient 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 case 2.2).

5 Blockchain collateral and off-chain payments

I explore the idea of *payment channels* (e.g., Antonopoulos, 2017) to describe how the blockchain tools and algorithmic constraints can be used to make incentive-compatible and self-enforcing transfers between two nodes. An extension to multiple nodes, as in Bitcoin's Lightning network, is also briefly discussed. The focus here is on the economic issues related to incentives, commitment, and enforcement of blockchain payments. I therefore abstract from any technical issues, e.g., 50-percent attacks, network congestion, processing delays or cybersecurity concerns.

In the preceding sections I argued that committing to future payments and enforcing promises of payments via the blockchain requires posting 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. The challenge is therefore to design how to

(i) use the blockchain as collateral mechanism to support payments and

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

The exposition below mainly follows the Bitcoin blockchain implementation but this is not essential for the main results and conclusions.

5.1 State transitions and payment channels

Consider the basic economic problem of a bilateral contract between agents A and B. The agents start at some initial state (a_t, b_t) where a_t is A's balance and b_t is B's balance at time t, and must transition to a new state (a_{t+1}, b_{t+1}) in an incentive-compatible and self-enforcing way using the blockchain.

The simplest possible example of such *state transition problem* is a transfer (payment) p from A to B, in which case

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

I outline an incentive-compatible blockchain solution to the state transition problem as described by Antonopoulos (2017) among others, in the context of Bitcoin. This implementation is known as a *payment channel*. The key idea is to use a transaction posted on the blockchain to *collateralize* the set of all possible balances between two parties, that is, the set of values $(a, b) = (A - z, z)$ for some 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 collateral to enforce payments, consider the simplest case of a one-way (unilateral) 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). A one-way payment channel can be implemented as follows:

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

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

The multisig locking script is important as it ensures that no single party can unlock the funds without the other party's signature.

2. [refund set-up] Before posting the collateral transaction τ_c on the blockchain the parties write a second, refund transaction:

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

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

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

Person A gives 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 (and she is not due any payment yet). The refund transaction τ_r, once signed by B, protects A in case B disappears. It is is kept by A as 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 k^a. This locks the funds c and establishes the payment channel.

4. [state transitions] After steps 1 through 3 the payment channel is set up and ready to support state transitions of the type

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

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

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

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 $l_c = k^a \wedge k^b$) but she could also wait, if she is due additional payments from A (as assumed). Further state transitions can be then made by writing new transactions (communication between the parties may be required for this, or the process may be automated) that 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 process whenever B holds a transaction entitling him to (sufficiently) larger balance (e.g., $c - a''$ vs. $c - a'$) she no longer has incentive to post a previous transaction. On the other hand, A is prevented from posting any old transactions by not having B's signature.

5. [settlement] The final state transition (settlement) transaction τ_s must be posted on the blockchain before expiry of the timelock T set in step 2.

Observe that in the proposed mechanism the blockchain is used to post two transactions only – the collateral transaction τ_c and the final settlement transaction, while multiple payments from A to B can be completed in the meantime. All payments are secured by the blockchain 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.

– the *timelock* T in the refund transaction's locking script $l_r = (k^a, T)$ ensures that A can get his funds back if B unilaterally quits.

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 makes a payment to A). Then nothing would prevent B to post the old transaction already signed by A (that is, the transaction giving B balance $c - a$) and benefit. To prevent such deviation and allow any direction of state transitions both 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 $a < a'$ is implemented as follows:

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

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

This transaction has two inputs, (a_0, k^a) and (b_0, k^b) originating respectively from A and B and unlockable by the corresponding signatures k^a and k^b. It is possible that a_0 or b_0 equals zero – that is, all collateral could be provided by a single party. As in Section 5.2.1, the collateral transaction has a single output with value c, a multisig key l_c, requiring both parties to sign and no timelock. Here, using (C3), we have (ignoring transaction fees)

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

2. [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 $a' > a$ or $a' < a$ are possible, e.g. A pays B or B pays A. The state transitions must be enforceable by the blockchain algorithmic tools. That is, each party is able to obtain their contractual due amount without relying on trust or cooperation from the other party. 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):

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

and

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

Crucially, these transactions do not need to be posted on the blockchain, 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 transactions τ_{sa} and τ_{sb} serve simultaneously as potential refund/guarantee (via the timelock T and revocation keys r^j) and also implement the state transition.

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 the 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 (bottom) output $(c - a', k^a)$ would release B's contractual balance $c - a'$ immediately since its locking script is $l^1_{sa} = k^a$ without any timelock. However, A's due funds a' (the first/top output of τ_{sa}) are 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 *unlocked immediately by whoever has the key* r^b. Importantly, the key r^b must remain in possession of B until the parties agree to transition to another state. An analogous argument holds for transaction τ_{sb} by changing the a and b superscripts accordingly.

To transition to a new balance state, e.g., to $(a'', c - a'')$ the parties first exchange the 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 should be done algorithmically and simultaneously to avoid any hold up.

Why are state transitions achieved by the transactions τ_{sa} and τ_{sb} self-enforcing? Suppose A attempted to renege and posted on the blockchain (by signing with k^a) the transaction τ_{sa}^{-1} corresponding to a previous non-current 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_{sa}^{-1}, see (A)) to detect A's deviation and use transaction τ_{sa}^{-1}'s revocation key r^b to claim the output a^{-1} as well, essentially seizing A's share of the collateral. The same argument applies for B in a symmetric situation. Because of these collateral-enabled penalties (funds seizure) neither party has an incentive to post an old transaction. 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.

Future work. The described mechanism implements the state transition but are there other self-enforcing mechanisms, e.g., involving less harsh penalties or delays? To do – formalize the problem using the elements and constraints in Section 3 and characterize the set of solutions to the mechanism design problem.

5.2 Applications

Multiperiod insurance

Going back to the economic examples from Townsend (1982) and Gans (2019) I show how payment channels and the collateral backing them can be used to implement the constrained-efficient outcome.

In the multiperiod risk sharing setting of Townsend one can think of the agent and insurer starting at balance state $(0, 0)$ and some initial promised utility w_0. Then, depending on the history of agent's income realizations $(y_{j_1}, y_{j_2}, ...)$ a chain of state transitions is implemented:

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

where $\rho^t_{j_t}$ is the contractual insurance transfer to the agent (positive or negative) at time t if the realized income state is j_t. For this state transition sequence to be implementable via a blockchain payment channel the parties would need collateral that spans the maximum possible balance due to either party. Setting

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

where $\rho_{\min}$ is the largest possible transfer from the agent and $\rho_{\min}$ is the largest possible transfer from the insurer would be sufficient.

Trade

Consider next a repeated version of the Gans (2019) setting, e.g., the parties do a maximum of T trades. Assume that if a party reneges on the contracted outcome (S sends the good with value V and B pays price P) the relationship is terminated. In that case, as argued in Section 4.2, a payment channel with collateral

$$c = TP + 2F$$

can support the required state transitions:

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

The additional collateral $2F$ 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). 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 blockchain algorithmic tools and constraints. This implies that in order for A to be certain that she will be repaid $(1 + r)l$, the minimum funds c that B must post in the collateral transaction τ_c must satisfy:

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

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

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

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

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

Essentially, A gains the interest rl while B recoups his collateral funds.

If instead B fails to repay within the specified time, the following alternative state is reached in which A effectively seizes part of B's collateral:

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

An actual real-world example 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 returned the collateral is released; if not, the collateral is liquidated.

To do: suppose the amount of collateral is constrained in the above settings, what is the best outcome that can be supported?

5.3 Multi-party transactions

The basic idea of bilateral state transitions 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 nodes C and D, these two nodes can make transfers to each other, even if they do not have a direct channel between them. The maximum possible transfer is determined by the 'weakest link', that is, the bilateral channel with the lowest collateral on the path.

In the blockchain setting, relying only on internal enforcement via algorithmic constrains and cryptographic tools, the main challenge of such transfers hopping over anonymous nodes is how to guarantee that each node will pass the funds forward and *prove* that it has done so. Notably, each intermediate node between C and D should not be able to unlock the funds (because it can expropriate them and the sender C has no recourse).

A way to solve this limited enforcement problem is to use a *timelocked redeemable secret (TRS)* locking script, $H(\sigma)$ (called HTLC in Bitcoin). Here $H(\sigma)$ is the hash function of a secret statement σ. A hash functions 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 knowing $H(\sigma)$. When the payment is agreed upon, the intended recipient D creates the secret σ and sends its hash $H(\sigma)$ (but not σ itself!) to the sender 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 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 supply $H(\sigma)$) but she can be incentivized (by means of a small fee added to F) to pass the TRS-locked funds F along 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 sender along the chain would be able to get their funds back in case the secret is never provided by D.

What happens when D is finally reached, e.g., from some node I_n? D knows the secret statement σ and so she claims the output F by debiting it from its state balance with I_n. Claiming F algorithmically sends the secret σ to I_n who claims F from her state balance with I_{n-1} and so on, until we reach C. The end result of this chain of bilateral state transitions is a transfer of F from C to D as intended. Note that no additional use of the blockchain is required in the process, except as collateral for the pre-existing bilateral channels on the path.

6 Blockchain-enabled financial contracts

In this section I explore more generally how a wide range of mechanism designs problems can be implemented via blockchain technology and smart contracts. Unlike in the previous sections, I do not describe in micro-level detail the blockchain algorithmic tools and constraints, but treat them as available in the background, using the implementations in Section 5. Instead, here I focus on the role that blockchain accounts and information can play in implementing (constrained-) optimal allocations of contracts in settings with exogenously and endogenously incomplete financial markets. The role of blockchain-based collateral is also taken as given. Namely, it is assumed that sufficient collateral can be posted to implement the required transfers (for example, via a payment channel implementation).

Consider agents $i = 1, ...N$ who transact via a blockchain network. Each agent i has two accounts, an expenditure account with balance a_t^i and a 'token' account enabled by a smart contract with balance w_t^i. The token account will be used to record various history-dependent state variables, possibly non-monetary, (e.g., 'promised utility', credit rating or debt level). Focus on transactions τ_t^{ij} that transfer funds from account i to account j.

The recorded history (blockchain data) $\mathcal{T}$ up to time T consists of all transactions $\Theta^T \equiv \{\tau_t^{ij}\}_{ij,t}$ and (possibly) commonly agreed 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 derived from the blockchain (as in Ethereum). Define the blockchain state at time t as $\Sigma_t \equiv \{A_t, W_t, e_t\}$. Using superscripts to denote history up to T, all public blockchain data are $h^T = [\Sigma^T, \Theta^T]$.

A financial *smart contract (SC)* 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 blockchain state Σ_t to a matrix of contractual transactions (transfers) $\mathcal{T}_t$, $\forall t \in [t_1, t_2]$. The smart contract terms can be contingent on both past and future events (e.g., $a^i_{t_1} > 5$, $e_{t_2} = 3$). Once initiated, the contract is automatically executed and cannot be modified or terminated unless a stopping condition is pre-specified. Transactions are valid if the transacted amount is available and its ownership is successfully verified or invalid otherwise.

Below I focus on *bilateral financial smart contracts,* defined as state-contingent transactions $\mathcal{T}$ between four blockchain accounts:

(1) node 1's expenditure account

(2) node 2's expenditure account (this could be an insurer or lender)

(3) an escrow account (node 3)

(4) a 'token' account (transactions will be denoted τ^{11})

The smart contract solves 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 blockchain state Σ and U is a payoff function. The feasible set $\Gamma(s)$ imposes restrictions on the transactions $\mathcal{T}(s)$ (see examples below) such as: state transition, exogenous financial constraints (e.g., borrowing limit), promise keeping, incentive-compatibility constraints, or truth-telling constraints. Both one- and multi-period 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 modeled.

Example A (one-period debt)

An agent with preferences over consumption $u(c)$ uses technology $f(k)$ that maps working capital k into stochastic output $y \in \{y_1, .., y_{\#Y}\}$. Here the only state variable is the output realization, $s = e = y$. The agent has no assets and must borrow the working capital k. Node 1 is the borrower's account and node 2 is 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 and $\tau^{12} = (1 + r)\tau^{21}$ is the repayment. Involuntary default, when $\tau^{12} > y$, can be incorporated by adjusting the interest rate τ^{12}/τ^{21} accordingly. Strategic default can be addressed by using the escrow account.

Example B (moral hazard)

Consider a risk-neutral insurer and a risk-averse agent with preferences $u(c,z)$ where c is consumption and z is costly effort unobserved by the insurer. The agent has positive stochastic income $y \in \{y_1, .., y_{\#Y}\}$ that is i.i.d. over time and $P(y_j|z)$ denotes the probability of realizing income y_j given effort z. Income is assumed observable and recorded in the public state e, so here $s \equiv (w, y)$. The constrained-optimal allocation can be implemented as smart contract by using the token account to store promised utility w which encodes income history. In the beginning of each period, given the contract $\mathcal{T}(s)$, the agent chooses effort z. Next, income y is realized and remitted to the insurer as $\tau^{12}(s)$. Then the smart 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 $\tau^{12}(w, y_l) = y_l$ for all w, the incentive-compatibility constraint:

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

and a promise keeping constraint,

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

Example C (hidden income)

The infinite-horizon version of Townsend (1982)'s hidden income problem can be implemented via 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 $y_1, ..y_{\#Y}$. The agent's income is unobserved by the insurer. The risk-sharing problem can be written as the following dynamic program using the agent's promised utility w as state variable that encodes the history of output realizations.

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

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.

Collateral and/or credit rating can be incorporated too (e.g., if short-term debt contracts are used), by using an 'escrow' blockchain account. For example, 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 'token account' (e.g., credit rating). The borrower puts $\tau^{13} \in [0, b]$ as collateral into the escrow account (node 3). The contract then releases loan size $\tau^{21}(\tau^{13}, w, e)$ and requests repayment (principal plus interest) $\hat{\tau}^{12}(\tau^{21}, w, e)$ where e is a verifiable external state (e.g., the central bank reference 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 agent, $\tau^{31} = \tau^{13}$.

If instead the agent defaults, that is, $\tau^{12} = 0$ (partial default can be incorporated too), the lender receives the collateral, from the escrow account via transaction $\tau^{32} = \tau^{13}$ and the agent's token account is updated (his credit rating is 'downgraded') to $w + \tilde{\tau}^{11}$. It may also be possible to distinguish between strategic and involuntary default by recording the appropriate information (e.g., on agent's income or business profits) to the blockchain. Aggregate shocks can be incorporated in the external state e.

Possible extensions: endogenous choice of what information to record on the blockchain; incentives for blockchain vs. off-blockchain transactions

7 Conclusions

I describe how key economic concepts underlying (financial) contracts – property rights, information, commitment, enforcement relate to and can be implemented with the building blocks of blockchain technology – 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. Its main limitation is the need for collateral – any promised future payments must be fully backed and secured. Multisig or more complex locking scripts and timelocks can be used to enable promised payments, based on posted collateral funds. While the need for blockchain collateral appears a costly limitation, I discuss how the collateral can be leveraged to back multiple off-blockchain transactions (e.g., via payment channels). This saves on blockchain transaction fees and waiting time costs.

The main takeaways are as follows. Blockchains and smart contracts are not a 'silver bullet' that can automatically implement any complicated mechanism design solution. Their usefulness is limited by the algorithmic tools and constraints on which the computer code is based. Enforcement of blockchain-based (smart) contracts has to be secured by posting collateral. Private information (adverse selection or moral hazard related to off-chain unobserved characteristics or actions) and limited enforcement may be still present and cause inefficiency. Subject to these caveats blockchain-enabled contracts can indeed be used to implement constrained optima and automate very complex mechanism design solutions.

My main goal in this paper was to distill or break down some of the main blockchain technology and smart contract ingredients to their most essential building blocks (not simply use ‘blockchain’ or ‘smart contract’ as catchall phrases). Economic theory could inform computer programmers what new algorithmic tools to include in future blockchain implementations or improvements (for instance, ready-made basic financial contracts). Combining the history and information verifiability, proof of ownership and security functions of blockchains with trust, commitment and enforcement mechanisms (possibly including trusted third parties) can be beneficial for bringing abstract mechanism design theory and constructs into reality while saving on collateral 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] Alvarez, F. and U. Jermann (2001), “Quantitative Asset Pricing Implications of Endogenous Solvency Constraints,” Review of Financial Studies, 14: 1117-52

[4] Andolfatto, D. (2016a), “Why the Blockchain Should Be Familiar to You”, MacroMania blog, http://andolfatto.blogspot.com/

[5] Andolfatto, D. (2016b), “Can the Blockchain Kill Fake News?”, MacroMania blog, http://andolfatto.blogspot.com/

[6] 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.

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

[8] Antonopoulos, A. (2017), Mastering Bitcoin: Programming the Open Blockchain, O’Reilly Media

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

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

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

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

[13] 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, June 2017

[14] Chiu, J. and T. Koeppl (2017), “The Economics of Cryptocurrencies: Bitcoin and Beyond”, working paper

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

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

[17] Committee on Payments and Market Infrastructures, CPMI (2017), “Distributed Ledger Technology in Payment, Clearing and Settlement: An Analytical Framework” Bank for International Settlements

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

[19] Doepke, M. and R. Townsend (2006), “Dynamic Mechanism Design with Hidden Income and Hidden Actions”, Journal of Economic Theory, 126: 235-85

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

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

[22] Gans, J. (2018), “The Fine Print in Smart Contracts”, working paper, University of Toronto

[23] Green, E. (1987), “Lending and the smoothing of uninsurable income”, in E. Prescott and N. Wallace, eds. Contractual Arrangements for International Trade, Minnesota Press

[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 SDN 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(3), p.887-959

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

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

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

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

[32] Kocherlakota, N. (1998), “Money is Memory,” Journal of Economic Theory 81, 232-51

[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] Krueger, D. (1999), “Risk sharing in economies with incomplete markets”, manuscript, Stanford.

[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”, manuscript

[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. (1998), “On the long run implications of repeated moral hazard”, Journal of Economic Theory, 79: 174-191.

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

[42] Prescott, E.C. and R. Townsend (1984), “General Competitive Analysis in an Economy with Private Information”, International Economic Review, 25: 1-20.

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

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

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

[46] Szabo, N. (1997), “Formalizing and Securing Relationships on Public Networks”, First Monday 2(9)

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

[48] Szabo, N. (2017), “Money, Blockchains and Social Scalability”, Unenumerated blog

[49] Tasca, P., T. Thanabalasingham and C. Tessone (2016), “Ontology of Blockchain Technologies: Principles of Identification and Classification”, working paper, UCL

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

[51] Thomas, J. and T. Worrall (1994), “Foreign Direct Investment and the Risk of Expropriation”, Review of Economic Studies 61: 81-108

[52] Townsend, R. (2019), "Distributed Ledgers: Innovation and Regulation in Financial Infrastructure and Payment Systems", working paper, MIT

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

[54] Yermack, D. (2016), "Corporate Governance and Blockchains", NBER Working Paper 21802

Figure 1: Multi-period Risk Sharing – Townsend (1982)

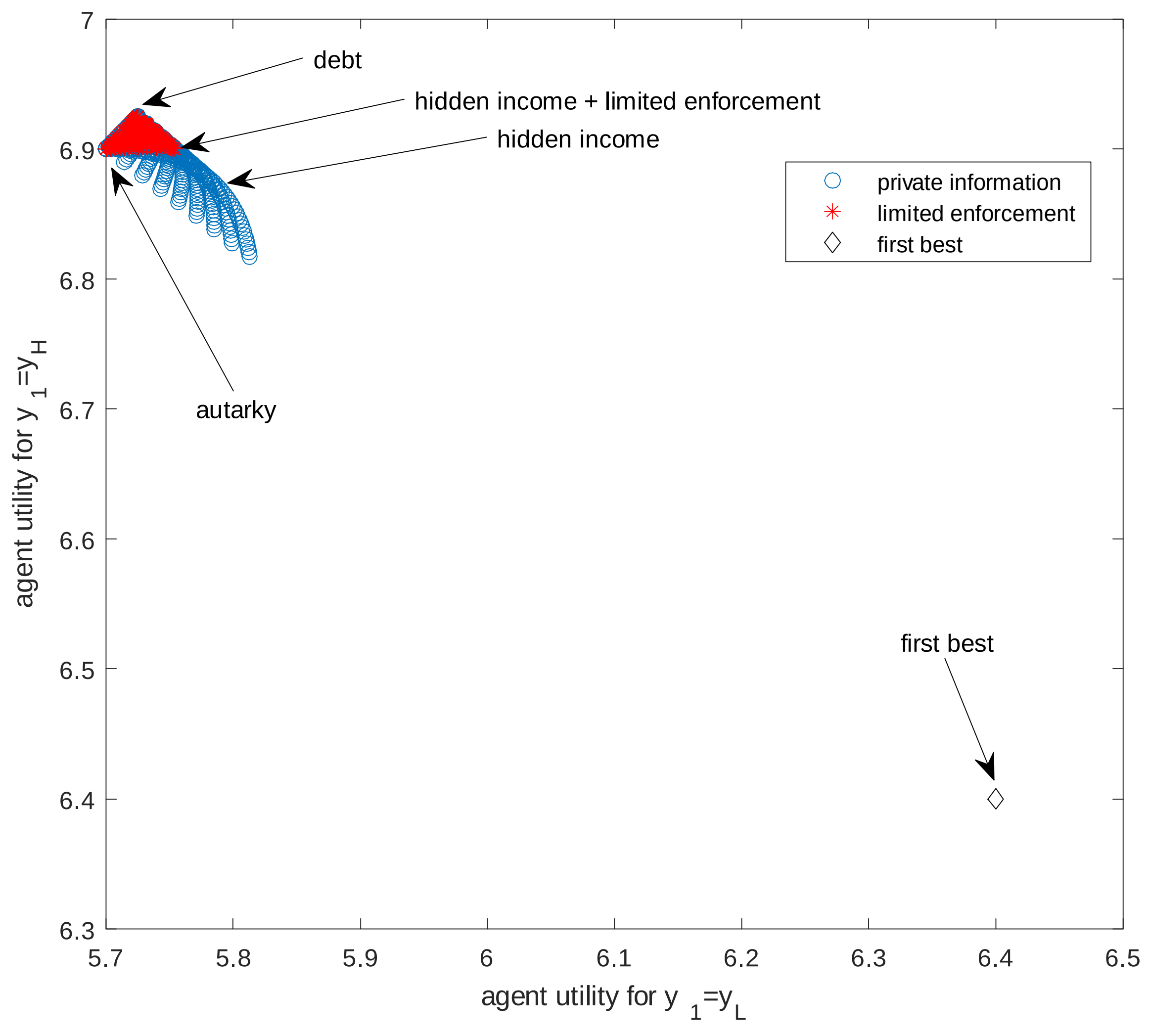
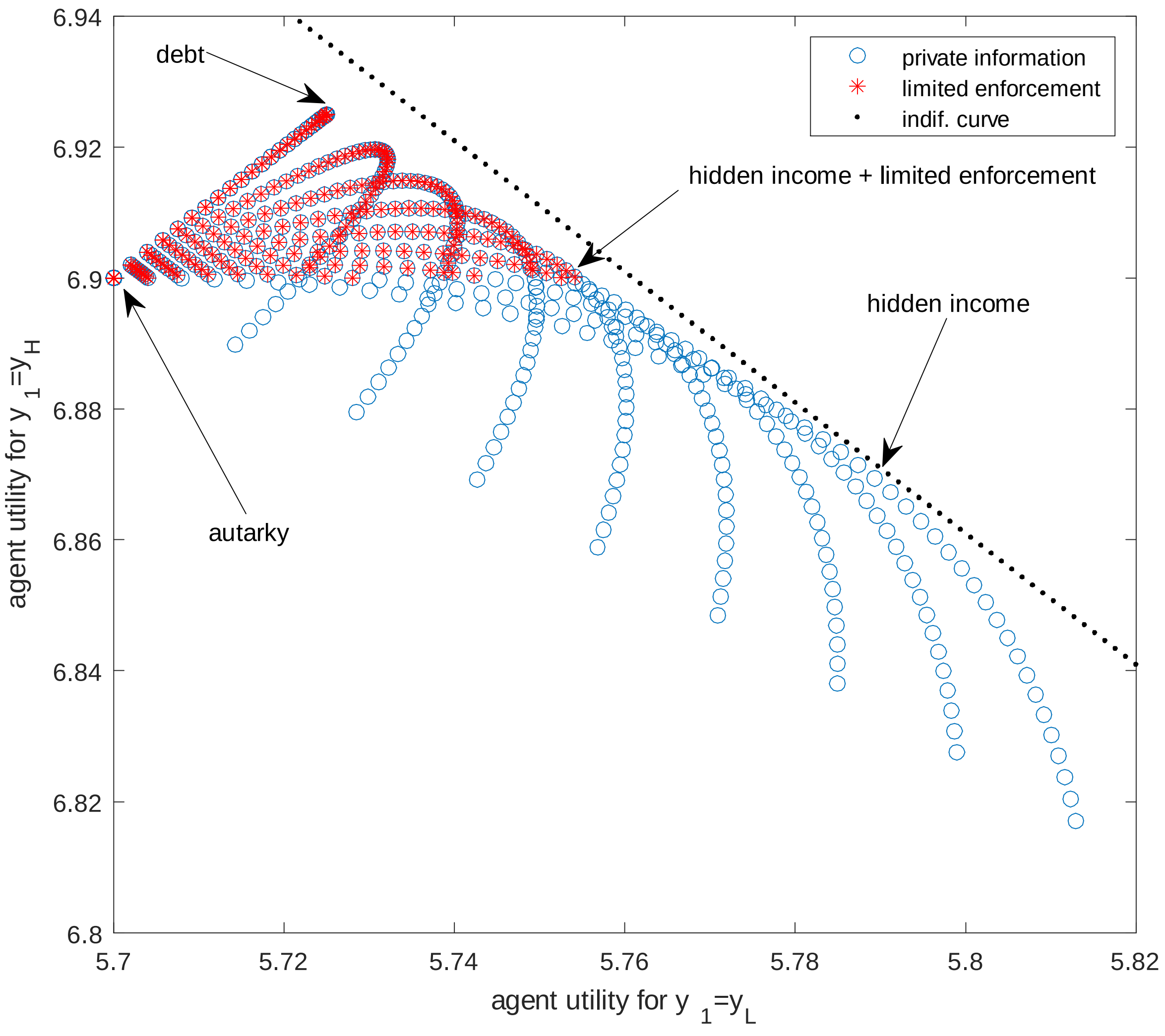


Figure 2: Multi-period risk sharing – constrained allocations


agent utility for y 1=yH
agent utility for y 1=yL
private information
limited enforcement
indif. curve
debt
autarky
hidden income + limited enforcement
hidden income