

Grassroots Social Contracts: From Legal Text to an App (Full Version)

JAMES GOLIKE, London School of Economics and Political Science, United Kingdom

ANDY LEWIS-PYE, London School of Economics and Political Science, United Kingdom

EHUD SHAPIRO, London School of Economics and Political Science, United Kingdom and Weizmann Institute of Science, Israel

Almost two centuries ago Pierre-Joseph Proudhon envisioned a social contract that is an agreement of man with man; reciprocal; imposing no obligation upon the parties except that which results from their personal promise; subject to no external authority; freely accepted and signed by all the participants; and of the nature of a contract of exchange. He also envisioned that groups formed by a like contract could, by a similar contract, agree with every and all other groups; digitally, that independent platform instances may coalesce by mutual consent. Here we define *grassroots social contracts* as social contracts meeting Proudhon's six conditions and two more: people are free to deal with each other, and there is no external register of people. We show that a grassroots social contract can be transformed into a working smartphone-based grassroots platform through an abstraction cascade: the contract's text is rendered as *act schemas*, a formal language for the acts it describes, verified syntactically to be grassroots by three decidable conditions, *Introduction*, *Provenance* and *Volition*. The schemas are compiled into volition-guarded multiagent atomic transactions, the cascade from which to a working app has already been described. We prove that the protocol realising a syntactically grassroots contract is grassroots, and volitionally so: two instances coalesce only when a member of each wills it. We distinguish three legal modalities of clauses, *enforced*, *attested* and *undertaken*, by whether an agent can breach a clause and what non-repudiable evidence of the breach the contract leaves, and illustrate them with a social network and with grassroots currencies. A checker of the three conditions, written in Grassroots Logic Programs, accompanies the paper.

Additional Key Words and Phrases: grassroots social contract, act schema, compilation, legal modalities, provenance, contract specification language, norm-governed systems, contract formation

1 Introduction

Almost two centuries ago Pierre-Joseph Proudhon envisioned a social contract [46, Fourth Study] that is (1) an agreement of man with man; (2) reciprocal; (3) imposing no obligation upon the parties except that which results from their personal promise; (4) subject to no external authority; (5) freely accepted and signed by all the participants; (6) of the nature of a contract of exchange; and a few other conditions, among them that it include all citizens with their interests and that it increase the well-being and liberty of every citizen. In his words, the social contract is “an agreement of man with man”; “essentially reciprocal: it imposes no obligation upon the parties, except that which results from their personal promise of reciprocal delivery: it is not subject to any external authority: it alone forms the law between the parties: it awaits their initiative for its execution”; “freely discussed, individually accepted, signed with their own hands, by all the participants”; and “of the nature of a contract of exchange: not only does it leave the party free, it adds to his liberty; not only does it leave him all his goods, it adds to his property; it prescribes no labor; it bears only upon exchange”. He also envisioned a property of social contracts [46, Sixth Study], analogous to a property of digital platforms we term *grassroots* [53, 55, 56]: that (i) one could “make a contract with all, as ... with some”; digitally, that a grassroots platform can have multiple instances, and (ii) “each group of citizens ... formed by a like contract ... could thereafter, and always by a similar contract, agree with every and all other groups”; digitally, that independent platform instances may

Authors' Contact Information: James Golike, London School of Economics and Political Science, London, United Kingdom; Andy Lewis-Pye, London School of Economics and Political Science, London, United Kingdom; Ehud Shapiro, London School of Economics and Political Science, London, United Kingdom and Weizmann Institute of Science, Rehovot, Israel.

coalesce by mutual consent, possibly into a single global platform instance. In full: “if I could make a contract with all, as I can with some; if all could renew it among themselves, if each group of citizens, as a town, county, province, corporation, company, &c., formed by a like contract, and considered as a moral person, could thereafter, and always by a similar contract, agree with every and all other groups, it would be the same as if my own will were multiplied to infinity. I should be sure that the law thus made on all questions in the Republic, from millions of different initiatives, would never be anything but my law; and if this new order of things were called government, it would be my government.” We proposed grassroots platforms owned, operated and governed by their participants as a foundation for an equitable and democratic digital realm [55].

A *digital social contract* is one realised by software the agents run [7]. We call the mathematical object a *protocol* and the software implementing it a *platform*. A protocol or a platform *realises* a contract, a contract being an informal text; one abstraction of the cascade *implements* another, and implementations compose, so each implements every abstraction above it [36, 51]. A protocol is *grassroots* when its instances are independent of one another and of every global resource but the net, and can always coalesce, possibly into a single global instance [53, 55, 56], and a *grassroots platform* is the software implementing a grassroots protocol. The grassroots notion is a formal property of a protocol, recalled in Section 4.

Here we define *grassroots social contracts* as social contracts meeting Proudhon’s six conditions above and that (7) people are free to deal with each other; and (8) there is no external register of people, so people can come to know each other either directly or through people they already know. Our main theorem is that the protocol realising such a contract, written as act schemas and meeting three decidable conditions, is grassroots: any two agents may come into relation without anyone’s permission, and two instances formed independently coexist and may later coalesce.

A social contract is a contract among those who adopt it. Participating in a grassroots platform requires adopting the grassroots social contract that defines it, as participating in a global platform requires adopting an End-User Licence Agreement (EULA). A grassroots social contract is mutual and symmetric among all participants; an EULA is between each person and the platform operator. A friendship or a mutual credit line is an act under the contract between particular agents. We show that GSN, a contract for a social network [54], is grassroots, and GC, a contract for a currency of personal coins [52]. Not every social contract is. Examples that are not include a contract without coalescences, not allowing two strangers to come into relation; a contract with named third-party control, e.g. where introduction requires a token only a third party can make; and a contract with an act that changes a stranger’s state without their will and requires no prior relation between the two; each fails one of the conditions.

We show that a grassroots social contract can be transformed into a working smartphone-based grassroots platform through an *abstraction cascade* [36], starting from converting the social contract text to formal act schemas, verified syntactically to be grassroots, and then to volition-guarded multiagent atomic transactions [40], hitherto the most abstract formalism used to specify grassroots platforms. The abstraction cascade from there to a working app has been described elsewhere [36]. A grassroots social contract has two artefacts: a legal text, and a protocol realising it. A clause of the text describes an *act*, in the sense of a juridical act: a declaration by one or more agents intended to create a legal effect. *Act schemas* are a formal language for the acts the contract describes, each naming the agents’ roles, which of them must will the act, and its precondition and effect at each role, with contract agents determined upon signature. Any contract written in this language meets the eight conditions above, provided it is *syntactically grassroots*, satisfying: (i) *Introduction*, that some act can introduce any two willing agents, following which the state of each stores an identifier of the other, as of a person they know, a friend, or the issuer of a coin they hold, and we say that it *records* the other; (ii) *Provenance*, that the state of an agent stores

an identifier of an agent only by an act with that agent, or with an agent whose state stores that identifier; and (iii) *Volition*, that an act requires the consent of an agent in each connected part of the undirected graph formed by identifiers stored by its agents, unless that graph is connected. Conditions (1), (2) and (4) hold of every contract written as act schemas, since every participant in an act takes a role in it and the schemas single out no one, and so does (5), since nobody is bound who has not adopted the contract (Section 3.5); (3) is Volition, (6) and (7) are Introduction, and (8) is Provenance (Section 3.6). The three are decidable, and a checker for them accompanies this paper, at <https://github.com/EShapiro2/GLP/tree/main/programs/gsc>; it certifies both of the contracts we carry through. The compilation into transactions is defined for contracts meeting them. It implements act schemas by volition-guarded transactions.

We prove that the protocol realising a syntactically grassroots contract is grassroots, and volitionally so (Theorem 11). To do so, we first prove that *politeness* – any two agents can always come to interact, and no act joins two instances unless each of them holds a guard of it – is sufficient for the protocol over a contained set of volition-guarded multiagent atomic transactions to be grassroots, and volitionally so: two instances coalesce only when a member of each wills it (Theorem 24). We then prove that a syntactically grassroots social contract compiles into a contained, polite set of volition-guarded transactions (Section 7).

We distinguish three legal modalities of the clauses of a grassroots social contract (Section 8), by whether an agent can breach a clause and what record of it the contract leaves in another agent’s state: a clause is *enforced*, if performing it is an act of the contract and every correct run of it keeps the clause; *attested*, if an agent may breach it, with the contract recording the breach; or *undertaken*, if the contract records who gave the undertaking, but performing the undertaking is carried out outside the contract. A breach of an *enforced* clause is impossible with a correct realisation of the contract; a breach of an *attested* or *undertaken* clause can be taken to court, with non-repudiable evidence attributable to the obligor of the act alleged to constitute the breach, in the case of an attested clause. We illustrate the three modalities with the grassroots social network [54], and with grassroots currencies [52]. The compilation and the conditions are algorithmic, so the only informal step in this paper is the first: rendering a legal text as schemas.

We carry two contracts through, a social network and a currency of personal coins, and certify both from their schemas. A coin can be handed on, so a holding may record an agent its holder has never transacted with; the certification then rests on traceable provenance, and conservation of money is its special case.

Three languages. The paper employs three languages: natural language for the social contract; act schemas; and volition-guarded atomic transactions. Rendering the first as the second is by judgement; compiling the second into the third is algorithmic; and the conditions of the main theorem, that a syntactically grassroots social contract is grassroots, are verified in the second. GSG:1, of the social graph (Section 2.2), reads “Alice and Bob, who know each other personally and are not friends, may jointly decide to become friends by an act of both”. Its schema (Section 3.3) names the act’s two roles, marks each with ? as having to will it, and at each role requires an established identity and knowing the other, forbids the atom that role gains and adds it:

befriend(Alice?, Bob?) :

identity, *knows*(Bob), $\neg$ *friend*(Bob), $+$ *friend*(Bob) *identity*, *knows*(Alice), $\neg$ *friend*(Alice), $+$ *friend*(Ali

Compiling it (Section 5.1) yields the volition-guarded transaction $(d \rightarrow d', \{Alice, Bob\})$, for every d with *identity*, *knows*(Bob) $\in d_{Alice}$, *identity*, *knows*(Alice) $\in d_{Bob}$, *friend*(Bob) $\notin d_{Alice}$ and *friend*(Alice) $\notin d_{Bob}$, where $d'_{Alice} = d_{Alice} \uplus \{\text{friend}(\text{Bob})\}$ and $d'_{Bob} = d_{Bob} \uplus \{\text{friend}(\text{Alice})\}$. The rest of the paper concerns the middle line. **The abstraction cascade.** The passage from a legal

text to a running platform goes through the *abstraction cascade* [36] set out in Figure 1; this paper provides (1)→(3). A compiler carries the grassroots social graph and grassroots currencies [52] from their volition-guarded transactions down the cascade to Grassroots Logic Programs that run with scripted persons, and earlier programs of the social graph, the grassroots social network [54] and the currencies run as mini-apps of one app, deployed on a physical smartphone [36]. The screens of Figures 2 to 5 are screenshots of the GKK, GSG, GSN and GC mini-apps, each compiled from its social contract down the abstraction cascade; one instance of the grassroots app on an iPhone simulator runs the mini-app of every agent, the agents communicate over channels within it, and a screenshot shows one person’s screen. All source code and all artefacts, including the social contracts and the artefacts the abstraction cascade produces from them, are in the public repository <https://github.com/EShapiro2/GLP>. A child-safe social network [58] is specified the same way. **The**

Human–Science–AI methodology. The formal objects of the cascade are its abstractions and the implementations among them, each constructed, maintained, evolved and proved correct with AI, in tandem with the paper describing it. The methodology has two kinds of artefact: papers written for competitive peer review, and code. The papers serve as an evolving and cooperatively-constructed interface between us humans and AI, recording our shared understanding of the subject matter, and are the source of authority for coding. Papers are written by people and reviewed, revised and extended with the help of AI; the code, with few exceptions, is written by AI; and both evolve in the process of harmonising them. AI codes down the cascade, from paper to tested code, and every gap coding finds is repaired in the paper before it is repaired in the code. It is at the antipode of “vibe coding”, where a person describes the intended behaviour informally and AI guesses the rest [48]. Spec-driven development makes the specification the primary artefact, maintained by people while the code is regenerated from it [19, 23]; here that artefact is a scientific paper, and the feedback from coding revises it. The AI is Claude, Anthropic’s models, among them Claude Opus 5.5 and Claude Fable 5.1. Through Claude Code it wrote, each from the paper that specifies it, the checker and the compiler of act schemas, the compilers of the cascade below them, the apps of Figures 2 to 5, and the tests that run the apps and draw their screens; and it revised and extended the text of this paper, its definitions and proofs included, at the authors’ direction. **Relation to contract**

specification languages. Formalising legal text has a large literature, on contracts as automata, as defeasible deontic rules and as domain-specific languages, and on statute as code, and Section 10 reviews it. Two languages formalise legal contracts in a way close enough to ours to say plainly how they differ. *Symboleo* [45, 64] specifies a contract as obligations and powers with statechart lifetimes, for verification and for monitoring; *Stipula* [10] is a domain-specific language whose contracts are state machines over parties, fields and linear assets, with an agreement operator and a formal semantics. Both bind their parties at contract formation and both keep one global contract state. Our question is about the family of systems a contract induces over every finite set of people, and about what happens when two instances that formed independently meet; it cannot be posed in either language, and each requires a global resource that closure forbids. Section 10 compares the two languages, together with the literature on norms and on contract languages more broadly.

Structure. Section 2 presents the contracts and Section 3 the syntax of act schemas, with listed roles, by which a leader closes a group with all its members, and the contracts’ schemas. Section 4 recalls what is needed of [40] and Section 5 compiles the schemas into its transactions. Section 6 defines politeness and proves it sufficient for a protocol to be grassroots and volitionally grassroots, and Section 7 states the conditions on the schemas and certifies the contract from them. Section 8 explains the legal modalities of contract clauses, Section 9 how an agent enters the contract and what the code they run warrants, and Section 10 reviews related work. Appendix A states the

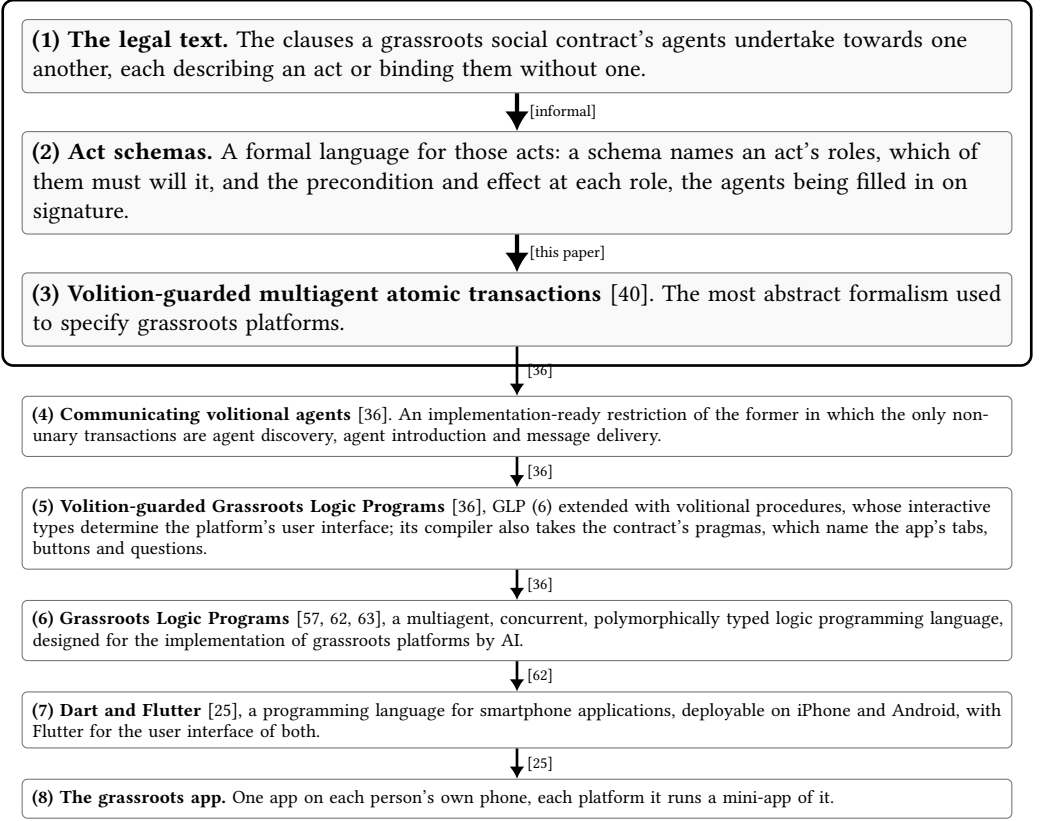


Fig. 1. The abstraction cascade, from a legal text to a running platform. The framed (1)–(3) are this paper. The first arrow, the realisation of the legal text, is left to judgement, and the second is the compilation of this paper. Each arrow from (3) to (6) is an implementation proved correct in the reference it carries, and they compose [36]; the GLP engine in Dart and the app are built and tested, and not proved. A condition on the transactions of (3), politeness, is sufficient for the platform they specify to be a grassroots platform, and conditions on the schemas of (2) are sufficient in turn for politeness.

framework of [40] formally, citing its proofs; Appendix B holds the proofs; Appendix C defines the values act schemas allow; and Appendix D expands the related work.

2 Grassroots Social Contracts

A grassroots social contract is a finite list of clauses its agents undertake towards one another. A clause either describes an *act*, which the protocol realising the contract carries out, or binds the agents without describing one. Each contract is in a box of its own, and a clause is cited by the contract's name and its number, as GSG:1; a contract that extends another cites its clauses and does not restate them. Each clause opens with its legal modality, as the box below sets out (Section 8). A box ends with the contract's pragmas, which the app built from it follows. They open with a line for each tab of the app, naming the tab and what it lists. Each clause then has a line naming it, then the person who proposes an act of it, a button for each act they propose, in brackets, bearing its act's name, and any field they fill, in angle brackets; a line for each other person who proposes an act of the clause, written the same way; and a line for each other person whose will the act needs,

their name, a colon, and the question they are asked. A button is pressed in the row of an entry of a tab, and that entry’s person takes a role of the act; the fields are what the proposer fills besides, so GKK’s Introduction has one field, Carol, Bob being the contact in whose row the button is pressed. A question line belongs to the nearest button line above it, so where a proposer’s acts ask questions each act takes a line, and buttons share a line only when no question follows. The contracts of this section are examples: the syntax of Section 3, the compilation of Section 5 and the conditions of Section 7 apply to any contract written the same way.

The clauses are stated here in prose, as a contract states them. The sections that follow give the acts a formal syntax, compile the schemas into transactions, and certify the contracts from the schemas.

Legal Modalities of Contract Clauses

- **Enforced:** no act of the contract can breach it.
- ◆ **Attested:** an agent may breach it, and the contract records the breach.
- **Undertaken:** its performance lies outside the contract, which records only the undertaking.

An agent running correct code, code that correctly implements the contract, cannot breach an enforced clause.

2.1 The Keys

A *person* is a legally culpable entity: a natural person of age, or a legal person of good standing, such as a business. A person takes the pronoun ‘they’; Alice, Bob and Carol are variables naming distinct persons. A person *knows* another when they hold the other’s public key.

GKK: Grassroots Keys and Knowledge

- *Identity.* Alice may act once to establish their identity by choosing their unique key pair. They make themselves known by the public key, sign their acts with the private key, and are accountable for every act signed with it.
- *Meeting.* Alice and Bob may come to know each other by an act of both.
- *Introduction.* If Alice and Bob know each other and Alice knows Carol, then an act of Alice and Bob can introduce Carol to Bob, following which Bob knows Carol.

Pragmas

Contacts: the persons Alice knows.
Identity: Alice [Establish]
Meeting: Alice [Meet] <person>
Bob: Alice wishes that you know each other.
Introduction: Alice [Introduce] <person>
Bob: Alice asks to introduce Carol to you.

GKK is the base contract, which every contract below extends: a person comes to know another by meeting them or through someone they know, which is condition (8) of Section 1. A key pair chosen at random is unique with high probability, and [12] replaces one that is lost or stolen. The GKK app is a mini-app of the grassroots app, compiled down the cascade from GKK; Figure 2 shows it realising GKK:1 and GKK:3.



Fig. 2. The GKK app: Alice, before establishing their identity, with the button Join (GKK:1); Bob’s contacts after Alice introduces Carol to Bob, with the sentence for an introduction of Bob’s own (GKK:3).



Fig. 3. The GSG app: Alice asks to be Bob’s friend, and Bob answers Accept or Decline (GSG:1); Alice’s friends after Bob accepts.

2.2 The Social Graph

GSG: Grassroots Social Graph
1. ♦ <i>Friendship</i>. Alice and Bob, who know each other personally and are not friends, may jointly decide to become friends by an act of both. 2. ■ <i>Ending a friendship</i>. Alice may end their friendship with Bob by their own act.
Pragmas
<i>Friends</i>: the friends of Alice. <i>Friendship</i>: Alice [Befriend] Bob: Alice asks to be your friend. <i>Ending a friendship</i>: Alice [Unfriend]

GSG extends GKK and is the grassroots social graph [54]. The GSG app is a mini-app of the grassroots app, compiled down the cascade from GSG; Figure 3 shows it realising GSG:1.



Fig. 4. The GSN app, on Bob’s phone: Alice asks Bob to join her group *friends*, and Bob answers Join or Decline (GSN:3); the group *friends*, led by Alice (GSN:2), with Bob’s message to the group as Alice delivered it to him (GSN:5).

2.3 The Social Network

GSN: Grassroots Social Network
1. ■ <i>Message</i>. Alice may send a message to their friend Bob. 2. ■ <i>Group</i>. Alice may create a group, and is then its leader and first member. 3. ■ <i>Joining</i>. Alice, the leader of a group, may add their friend Bob to it, by an act of both. 4. ■ <i>Leaving</i>. Bob, a member of a group, may leave it by their own act; when the leader leaves, the group ends. A member of a group who is not a friend of its leader is removed from it. 5. ■ <i>Group message</i>. Bob, a member of a group, may send a message to the group, and the leader delivers it to every current member.
Pragmas
Groups: the groups Alice is a member of. Message: Alice [Message] <text> Group: Alice [Create] <name> Joining: Alice [Join] Bob: Alice asks you to join their group. Leaving: Bob [Leave] Alice [Close] Group message: Bob [Post] <text> Alice [Post own] <text>

GSN extends GSG with messages and groups [54], and the paper carries it through. The GSN app is a mini-app of the grassroots app, compiled down the cascade from GSN; Figure 4 shows it realising GSN:2, GSN:3 and GSN:5. The accountability in GKK:1 and the personal acquaintance GSG:1 requires describe no act and bind the persons; the other clauses describe acts and determine the contract’s transactions. Section 8 says what a realisation does with a clause that describes no act. In the repository GSN says ‘agent’ for ‘person’, so that children [58] and AI agents [59] can be added to it.

2.4 The Currency

A *coin* is a unit of debt naming its issuer, and an *Alice-coin* is a coin issued by Alice.



Fig. 5. The GC app: Alice’s holdings after minting an Alice-coin (GC:1); Bob’s after swapping a Bob-coin for Alice’s Alice-coin (GC:2), the coin with the button Pay (GC:3).

GC: Grassroots Currencies [52]
1. ■ <i>Minting</i>. Alice may decide to issue Alice-coins. 2. ■ <i>Swap</i>. Alice and Bob, who know each other, may jointly decide to swap coins they hold. 3. ■ <i>Payment</i>. Alice may decide to pay Bob-coins they hold to Bob, whom they know. 4. ■ <i>Redemption</i>. Alice may decide to redeem a Bob-coin they hold against any coin held by Bob, whom they know. 5. ○ <i>Pricing</i>. Alice prices their offerings in Alice-coins and accepts them in payment at those prices.
Pragmas
<i>Coins</i>: the coins Alice holds. <i>Minting</i>: Alice [Mint] <amount> <i>Swap</i>: Alice [Swap] Bob: Alice asks to swap coins with you. <i>Payment</i>: Alice [Pay] <amount> <i>Redemption</i>: Alice [Redeem]

GC is the grassroots currencies contract [52], quoted from its owner’s copy; it extends GKK and not GSG, no act of it requiring a friend, and the paper carries it through beside GSN. The first four clauses describe acts and determine the contract’s transactions; a swap in which Alice gives an Alice-coin and Bob gives a Bob-coin opens a *mutual credit line* between them, each thereafter holding a coin of the other. The fifth describes no act: it gives a coin its worth. The GC app is a mini-app of the grassroots app, compiled down the cascade from GC; Figure 5 shows it realising GC:1 and GC:2.

3 Act Schemas

A clause of a contract describes an *act*, in the sense of a juridical act: a declaration by one or more agents intended to create a legal effect. This section defines their formal syntax. A *contract* C is a finite set of act schemas over a set Σ of predicates, and S_C is its set of states. An agent’s state is a multiset of atoms; a schema names an act’s roles, which of them must will it, and the atoms it requires, forbids, adds and deletes at each role.

3.1 Atoms and States

Write Π for a countable, potentially infinite set of at least two *agents*, of which every contract and every run concerns a finite subset, and Δ for a countable set of *utterances* disjoint from it: a message, a post or an item, taken as an identity and carrying no signature. A *speech act* is an utterance or a speech act, signed by an agent: D is the least set with $\Delta \times \Pi \subseteq D$ and $D \times \Pi \subseteq D$, and the *signer* of $(x, p) \in D$ is $\text{signer}(x, p) := p$ [7, 24]; a speech act signed again carries its earlier signers inside it, and $D_P := \{d \in D : \text{signer}(d) \in P\}$ for $P \subseteq \Pi$. A *community name* is formed from an agent and speech acts: the set $\mathcal{N}$ of community names is disjoint from $\Pi \cup D$ and generated freely by an injection $p \mapsto \langle p \rangle$ of Π into it and an injection $(v, y) \mapsto v \cdot y$ of $\mathcal{N} \times D$ into it with disjoint images; the *agents* of a name are $\text{ppl}(\langle p \rangle) := \{p\}$ and $\text{ppl}(v \cdot y) := \text{ppl}(v) \cup \{\text{signer}(y)\}$, and $\mathcal{N}_P := \{v \in \mathcal{N} : \text{ppl}(v) \subseteq P\}$. Write Σ for a set of *predicate symbols*, *predicates* for short, and $\alpha : \Sigma \rightarrow \mathbb{N}$ for their arities; a predicate of arity n applied to n arguments is an atom, a syntactic object with no truth value. The arguments of an atom of a state are agents, speech acts and community names; those of an atom of a schema are agent roles, variables and name terms. An atom *names* what is among its arguments, the signer of a speech act among them and the agents of a community name among them: in a state an agent, and in a schema an agent role or a variable, a name term naming its base and the speech-act variables in it.

Definition 1 (Atoms and states). For any set W of arguments put

$$\Sigma[W] = \{e(u_1, \dots, u_{\alpha(e)}) : e \in \Sigma, u_i \in W\},$$

call its members the *atoms over W* , and call $u_1, \dots, u_{\alpha(e)}$ the *arguments* and e the *predicate* of $e(u_1, \dots, u_{\alpha(e)})$. An atom over $\Pi \cup D \cup \mathcal{N}$ is *ground*, a *local state* is a finite multiset of ground atoms, the initial state is $s_0 = \emptyset$, and

$$S_C(P) = \{s : s \text{ a finite multiset over } \Sigma[P \cup D_P \cup \mathcal{N}_P]\} \quad (P \subset \Pi).$$

Three predicates carry the example through this section: *identity*, of arity zero, and *knows* and *friend*, of arity one. After Alice establishes an identity, comes to know Bob and befriends him, the state of Alice is the multiset $\{\text{identity}, \text{knows}(\text{Bob}), \text{friend}(\text{Bob})\}$, which lies in $S_C(P)$ for every P holding Bob and in none omitting him.

Multiset operations are written $\uplus$, $\setminus$, $\cap$ and $\subseteq$: multiplicities are added, subtracted, minimised and compared. A set is a multiset in which every multiplicity is one.

A state is local, so its holder is no argument of its atoms: *friend(Bob)* in the state of Alice says that Bob is a friend of Alice, and $\text{c}(u)$ in the state of Alice is a coin issued by u and held by Alice; a friend set is a set of the former and a holding a multiset of the latter.

3.2 The Syntax of a Schema

Fix disjoint sets R of *agent roles* and X of *agent variables*, both disjoint from Π , and a set Y of *speech-act variables* disjoint from D . A schema is written over atoms whose arguments are agent roles, variables and name terms.

A schema is a contract template: the role names are its parameters, and the identities of the agents assuming them are filled in on signature. The mark $?$ is a question the machine puts to the agent in that role [7].

For example, take the predicates *identity*, of arity zero, and *knows* and *friend*, of arity one. GSG:1, that Alice and Bob, who know each other personally and are not friends, may jointly decide to

become friends, is the schema

befriend(*Alice?*, *Bob?*) :

identity, *knows*(*Bob*), $\neg$ *friend*(*Bob*), *+friend*(*Bob*) *identity*, *knows*(*Alice*), $\neg$ *friend*(*Alice*), *+friend*(*Ali*

Read it role by role. Each role requires *identity*, an identity established under GKK:1, and *knows* of the other, held after meeting under GKK:2 or an introduction under GKK:3. At Alice the atom *friend*(*Bob*) is forbidden, so Alice must not already have Bob as a friend, and the same atom is added, which is the effect of befriending; at Bob the same holds with Alice and Bob exchanged. Both roles carry ?, so the act takes the will of both. Nothing is deleted, and both agents change state.

Closing a group is one act of its leader and of every member, and a schema whose participants are only the agents its roles are bound to cannot state it. A *listed role* takes as participants every agent another role's state lists, and a *name term* names a community, such as a group.

Definition 2 (Name term, role). A *name term* of a schema is $\langle \pi \rangle$ for an agent role π of it, or $\sigma \cdot y$ for a name term σ of it and a speech-act variable y of it, and its *base* is π and the base of σ respectively. A *role* of a schema is an agent role $\pi \in R$, or a *listed role* $\pi[\rho(*, u_2, \dots, u_r)]$ of an agent role π of it, a predicate ρ of arity $r \geq 1$ and arguments $u_2, \dots, u_r$ of it. The *base* of an agent role is itself, and of a listed role $\pi[\rho(*, u_2, \dots, u_r)]$ the agent role π .

A group of GSN (Section 2.3) is named $\langle \textit{Alice} \rangle \cdot n$, by its leader Alice and a speech act n Alice signs, and Alice is its one agent. Closing it has two roles: the agent role *Alice*, its leader, and the listed role *Alice*[*listed*($*$, n)], every agent Alice lists as a member of the group n .

Definition 3 (Act schema). An *act schema* is written

$$\tau(\pi_1, \dots, \pi_k; x_1, \dots, x_m) : E_1 \quad \dots \quad E_k$$

with τ its name, $\pi_1, \dots, \pi_k$ distinct roles of it, an agent role possibly carrying the mark ? and those that do its *guarding roles*, $k \geq 1$ its *arity*, $x_1, \dots, x_m \in X \cup Y$ distinct, its *variables*, and each E_i a finite multiset of *marked atoms*, an atom whose arguments are its agent roles, its variables and its name terms, carrying at most one of the marks $-$, $+$, $\neg$. For each role π_i the marks divide E_i into four finite multisets of atoms, $+(i)$, $-(i)$, $\neg(i)$, $=(i)$, its members marked $+$, marked $-$, marked $\neg$, and unmarked respectively; the atoms π_i *requires* are $=(i) \uplus -(i)$. It is required that $+(i) \cap -(i) = \emptyset$, that $+(i) \uplus -(i) \neq \emptyset$ at every role that is not a guarding role and at some role.

At the role *Alice* of *befriend* the marks give $+(1) = \{\textit{friend}(\textit{Bob})\}$ and $\neg(1) = \{\textit{friend}(\textit{Bob})\}$, $=(1) = \{\textit{identity}, \textit{knows}(\textit{Bob})\}$ and $-(1)$ empty, and at *Bob* the same with the two roles exchanged. Both roles carry ?, so both guard, and $+(i) \uplus -(i) \neq \emptyset$ at each: befriending changes the state of both agents.

A guarding role may add and delete nothing; the agent in that role then wills the act, and their state is a precondition of it and does not change.

Act schemas allow values of any type: an argument of an atom may be a value or a value variable, a schema may carry a constraint on its value variables, which every binding satisfies, and an atom added or deleted may carry a multiplicity, a count of copies and so a positive whole number, as the bonds of [60] need for maturity dates and amounts (Appendix C). A value names no agent, so the conditions below do not concern it.

A contract also carries pragmas, which the app built from it follows: its tabs, and for each act who proposes it, the button proposing it and the question the mark ? puts to each other agent in a guarding role.

Definition 4 (Binding). A binding β for a schema τ maps its agent roles and agent variables to Π and Y to D , sending two agent roles to one agent only if neither of them adds or deletes an atom naming the other, and is extended to name terms by $\beta(\langle\pi\rangle) := \langle\beta(\pi)\rangle$ and $\beta(\sigma \cdot y) := \beta(\sigma) \cdot \beta(y)$, and to atoms and to multisets of them componentwise. It is *admissible* if every speech-act variable of τ that occurs in an atom added at a role, as an argument or in a name term, and in no atom required at a role is mapped to a speech act signed by the agent in a guarding role of τ .

A binding of *befriend* sends *Alice* to *p* and *Bob* to *q*, distinct because each role adds an atom naming the other, and carries $+(1)$ to $\{friend(q)\}$ and $+(2)$ to $\{friend(p)\}$. Befriending has no speech-act variable, so every binding of it is admissible.

Two roles sent to one agent are carried out by that agent together, as are a listed role and another role whose agent it lists, as Definition 13 states. An agent role that adds or deletes an atom naming another cannot share its agent, so no act relates an agent to themselves through two of its agent roles.

An act schema is a STRIPS operator [16] in multiagent form: what a role requires and forbids is a precondition, what it adds and deletes are an add list and a delete list, and a schema carries one of each per role, where STRIPS has one precondition and one pair of lists over a single global state. The marks are also the four kinds of arc of a contextual net transition [43]: an atom required and deleted is consumed, one required and kept is a positive context condition, one forbidden is a negative context condition, and one added is produced. We take no result from either theory.

3.3 The Schemas of the Social Network

The schemas of GSN are those of GKK and GSG, a message and the schemas of a group. Ten predicates. *identity*, of arity zero: in the state of Alice it says that Alice has established an identity; every role of every schema but *establish*'s requires it, and the tables show it only in *establish*. *knows*, of arity one: *knows(Bob)* in the state of Alice says that Alice holds Bob's public key; nothing deletes or forbids it, a second copy meaning the same. *friend*, of arity one: *friend(Bob)* in the state of Alice says that Bob is a friend of Alice. *received*, of arity two: *received(x, Alice)* in the state of Bob is a speech act *x* Bob received from Alice, signed by Alice, what Alice received from others signed inside it. The roles are Alice and Bob, and *c* is an agent variable.

schema	at Alice	at Bob
<i>establish</i> (Alice?) :	$\neg identity, +identity$	
<i>meet</i> (Alice?, Bob?) :	$+knows(Bob)$	$+knows(Alice)$
<i>introduce</i> (Alice?, Bob?, c) :	$knows(Bob), knows(c)$	$knows(Alice), +knows(c)$
<i>befriend</i> (Alice?, Bob?) :	$knows(Bob), \neg friend(Bob), +friend(Bob)$	$knows(Alice), \neg friend(Alice), +friend(Alice)$
<i>unfriend</i> (Alice?, Bob) :	$\neg friend(Bob)$	$\neg friend(Alice)$
<i>message</i> (Alice?, Bob; x) :	$friend(Bob)$	$+received(x, Alice)$

A group is named by the community name $\langle Alice \rangle \cdot n$ (Section 3.1), Alice its leader and *n* a speech act Alice signs: *member*($\langle Alice \rangle \cdot n$) in the state of Bob says that Bob is a member of the group *n* of Alice, and in the state of Alice *listed*(Bob, *n*) says the same, *closed*(*n*) that the group has ended, *posted*(*x*, *n*) that *x* was posted to it, and *delivered*(*x*, Bob, *n*) that Alice delivered *x* to Bob in it; *received_in*(*x*, Alice, *n*) in the state of Bob is a speech act *x* Alice delivered to Bob in the group *n*, signed by the member who posted it. In the schemas of a group Alice is the leader. A member's post adds the posted item at the leader, as the member signed it, and delivery adds it at each member. Drop, which no role guards, ends the membership of a member who is not the leader's friend. Close takes the listed role *Alice*[*listed*(*, *n*)] (Definition 2), every member Alice lists, so closing a group ends every membership in one act.

<i>schema</i>	<i>at Alice, the leader</i>	<i>at Bob</i>
<i>create</i> (<i>Alice?</i> ; <i>n</i>) :	$\neg \text{member}(\langle \text{Alice} \rangle \cdot n), \neg \text{closed}(n),$ $+\text{member}(\langle \text{Alice} \rangle \cdot n), +\text{listed}(\text{Alice}, n)$	
<i>join</i> (<i>Alice?</i> , <i>Bob?</i> ; <i>n</i>) :	$\text{member}(\langle \text{Alice} \rangle \cdot n), \text{friend}(\text{Bob}),$ $+\text{listed}(\text{Bob}, n)$	$\neg \text{member}(\langle \text{Alice} \rangle \cdot n),$ $+\text{member}(\langle \text{Alice} \rangle \cdot n)$
<i>leave</i> (<i>Alice</i> , <i>Bob?</i> ; <i>n</i>) :	$\neg \text{listed}(\text{Bob}, n)$	$\neg \text{member}(\langle \text{Alice} \rangle \cdot n)$
<i>drop</i> (<i>Alice</i> , <i>Bob</i> ; <i>n</i>) :	$\neg \text{friend}(\text{Bob}), \neg \text{listed}(\text{Bob}, n)$	$\neg \text{member}(\langle \text{Alice} \rangle \cdot n)$
<i>close</i> (<i>Alice?</i> , <i>Alice</i> [<i>listed</i> (<i>*</i> , <i>n</i>)] ; <i>n</i>) :	$+\text{closed}(n), \neg \text{listed}(\text{Alice}, n)$	$\text{at Alice}[\text{listed}(*, n)]:$ $\neg \text{member}(\langle \text{Alice} \rangle \cdot n)$
<i>post_own</i> (<i>Alice?</i> ; <i>x</i> , <i>n</i>) :	$\text{member}(\langle \text{Alice} \rangle \cdot n), \neg \text{posted}(x, n),$ $+\text{posted}(x, n)$	
<i>post</i> (<i>Bob?</i> , <i>Alice</i> ; <i>x</i> , <i>n</i>) :	$\text{member}(\langle \text{Alice} \rangle \cdot n), \text{listed}(\text{Bob}, n),$ $\text{friend}(\text{Bob}), \neg \text{posted}(x, n),$ $+\text{posted}(x, n)$	$\text{member}(\langle \text{Alice} \rangle \cdot n)$
<i>deliver</i> (<i>Alice</i> , <i>Bob</i> ; <i>x</i> , <i>n</i>) :	$\text{member}(\langle \text{Alice} \rangle \cdot n), \text{posted}(x, n),$ $\text{listed}(\text{Bob}, n), \text{friend}(\text{Bob}),$ $\neg \text{delivered}(x, \text{Bob}, n),$ $+\text{delivered}(x, \text{Bob}, n)$	$\text{member}(\langle \text{Alice} \rangle \cdot n),$ $+\text{received_in}(x, \text{Alice}, n)$

“Alice may act once to establish their identity” is establish, guarded by Alice and forbidding the *identity* it adds, and every other role requires it. “Alice and Bob may come to know each other by an act of both” is meet, guarded by both, each role adding *knows* of the other; “an act of Alice and Bob can introduce Carol to Bob” is introduce, Carol its agent variable *c*, which Alice must know and Bob comes to know, and “if Alice and Bob know each other” is the *knows* each role requires of the other. “Alice and Bob, who know each other ... and are not friends, may jointly decide” is the two guarding roles of befriend, the *knows* each requires and its forbidden atom, and that they know each other personally describes no act; “by their own act” is the single guarding role of unfriend, which deletes in both roles. “Alice may send a message to their friend Bob” is message: only the sender guards, since only the sender decides; “their friend Bob” is the *friend*(*Bob*) the sender requires; and the message is a speech act signed by the sender (Definition 4), what they received from others signed inside it. Create is GSN:2, join, guarded by both, GSN:3, leave, close and drop GSN:4, and post_own, post and deliver GSN:5. Delivery is not guarded: the leader undertook it on creating the group. Drop is not guarded either: a member who is not a friend of the leader is removed.

The records. The schemas meet the clauses differently. Befriend is the only schema adding a *friend* atom, that atom’s argument is a role, and befriend guards in both, so an agent’s friends are only agents who have willed the friendship, and no act of the contract can put one there against that agent’s will. Unfriend is the only schema deleting a *friend* atom and it deletes in both roles, so a friendship never ends for one agent without the other. A speech act a person signs records nothing against them until it is sent; the record is then the recipient’s *received*(*x*, *Alice*), signed by the sender, what the sender received from others signed inside it. Meet and introduce are the schemas adding a *knows* atom, so a person knows another only by meeting them or through someone who knows them, and befriend requires *knows* of the other at each role, so two persons can befriend only if they know each other. Knowing a person is not knowing them personally, so the schemas cannot stop the personal acquaintance GSG:1 requires being false; instead befriend leaves each of the two on record in the other’s state.

3.4 The Schemas of the Currency

The schemas of GC are GKK’s three and four over one predicate, €, of arity one: €(*u*) in the state of Alice is a coin issued by *u* and held by Alice. The roles are Alice and Bob; every role requires *identity*, shown only in establish.

<i>schema</i>	<i>at Alice</i>	<i>at Bob</i>
$\text{mint}(\text{Alice?}; k) :$	$+k \cdot \mathfrak{C}(\text{Alice})$	
$\text{swap}(\text{Alice?}, \text{Bob?}; u, k, v, k_1) :$	$\text{knows}(\text{Bob}), -k \cdot \mathfrak{C}(u), +k_1 \cdot \mathfrak{C}(v)$	$\text{knows}(\text{Alice}), -k_1 \cdot \mathfrak{C}(v), +k \cdot \mathfrak{C}(u)$
$\text{pay}(\text{Alice?}, \text{Bob}; k) :$	$\text{knows}(\text{Bob}), -k \cdot \mathfrak{C}(\text{Bob})$	$+k \cdot \mathfrak{C}(\text{Bob})$
$\text{redeem}(\text{Alice?}, \text{Bob}; r) :$	$\text{knows}(\text{Bob}), -\mathfrak{C}(\text{Bob}), +\mathfrak{C}(r)$	$-\mathfrak{C}(r), +\mathfrak{C}(\text{Bob})$

“Alice may decide” is the single guarding role of mint, of pay and of redeem; “Alice and Bob may jointly decide” is the two guarding roles of the swap; “Bob-coins they hold” is that pay and redeem require of Alice coins whose argument is Bob, pay k of them and redeem one; “whom they know” and “who know each other” are the *knows* the roles require; “any coin held by Bob” is redeem’s agent variable r . The amounts k and k_1 are multiplicities (Section 3.2), positive whole numbers. The mutual credit line is the swap at a binding sending u to Alice and v to Bob, where each gives coins of its own issue.

The records. Selling at an advertised price is no act of the contract, so no act can breach GC:5 and no run shows whether the issuer honours it; but a mutual credit line leaves a coin of the issuer in another agent’s holding, and that coin, naming the issuer, is the record of the undertaking. Minting records nothing against the issuer, putting the coin in its own holding; the undertaking becomes evidence only when the coin first passes to someone else, as a promissory note does when handed over. A clause that a holding contains a coin only if its issuer willed the act that put it there is breached by pay, which adds $\mathfrak{C}(\text{Bob})$ at Bob and guards only at Alice, and by the swap, which moves a coin between two agents by the will of the two. An issuer is not asked when its coin moves. A coin is therefore money rather than a promise between two people.

3.5 The Conditions Every Act-Schema Contract Meets

Of the eight conditions of Section 1, (1), (2) and (4) are met by every contract written as act schemas, and so is (5), in that nobody is bound who has not adopted the contract, by two properties of the language. Both are about the protocol realising a contract, so both are stated and proved in Section 5, where that protocol is defined, and used here. Conditions (3), (6), (7) and (8) are those of Section 3.6.

The first property is that every participant in an act takes a role in it: an agent’s state changes only under a schema of the contract and only in a role of it, which is Proposition 17.

The second is that the language cannot name an agent. The agent roles R , the agent variables X and the speech-act variables Y are disjoint from Π , and a schema is written over atoms whose arguments are these and name terms built from them, so no schema mentions an agent; and by Definition 1 every agent begins in the same state $s_0 = \emptyset$. Permuting the agents therefore carries the system to itself, which is Proposition 18, and no agent can reach a position another could not, which is Corollary 19.

- (1) (1) *An agreement of man with man.* Only agents hold states, and by Proposition 17 an agent bears an effect of the contract only in a role; by Corollary 19 no agent can reach a position another could not. No schema can require that an agent be a natural person.
- (2) (2) *Reciprocal*, (4) *subject to no external authority.* An agent consents twice and both consents are their own: adopting the contract is consent to its schemas, and willing an act in a guarding role is consent to that act. By Proposition 17 nothing binds an agent outside a role of a schema, so nothing binds them that their adoption did not, and every participant in an act takes a role in it, so no registry, no log and no timing authority takes part in one; by Corollary 19 no agent is privileged by the text. There is nothing else in the model to be subject to. That an agent is bound only by their own promise, condition (3), is Volition, Definition 9; Proudhon’s *reciprocal*

delivery, that each agent has value for value, is a condition on the contract's content and is not carried over.

- (3) (5) *Freely accepted and signed by all the participants*. This is how a contract is entered into rather than what it says. Section 9 says how: each participant signs a one-sided undertaking to keep the contract and presents it to every participant it interacts with, so every participant has signed it and, in an instance of two or more, another participant holds it (Proposition 34); and by Proposition 17 nobody else is bound.

The remaining conditions do not hold for every contract. A contract with no act by which two strangers can come into relation is written as act schemas as readily as one that has such an act, and fails Introduction. Section 3.6 states them as conditions on the schemas.

<i>condition</i>	<i>formal counterpart</i>	<i>established by</i>
(1) an agreement of man with man	every participant takes a role; no privileged agent	Proposition 17, Corollary 19
(2) reciprocal	every participant takes a role	Proposition 17
(3) no obligation but by personal promise	volition, Definition 9	Proposition 27
(4) subject to no external authority	every participant takes a role	Proposition 17
(5) freely accepted and signed by all	the undertaking, Section 9	Proposition 34
(6) of the nature of a contract of exchange	an unobstructed introductory act, Definitions 6, 7	Proposition 26
(7) people free to deal with each other	an unobstructed introductory act	Proposition 26
(8) no external register of people	traceable provenance, Definition 8	Lemma 63

The counterpart of a condition in the table is structural. Proudhon's reciprocal delivery, that each agent has value for value, and exchange in its substantive sense are conditions on a contract's content, met by its clauses rather than by the form of its schemas.

3.6 What We Prove

Conditions (3), (6), (7) and (8) of Section 1 become three conditions on a contract's schemas, and the protocol realising a contract meeting them is grassroots. All three are properties of the text and are decidable, so whether a contract meets them is decided by a checker rather than left to its drafter. This section states them and the theorem; the sections that follow provide what is needed to prove it, and Section 7 proves it.

An agent's state names agents: those they know, their friends, the issuers of the coins they hold, the senders of the messages they hold. A state that names an agent stores an identifier of them, in the words of Section 1, and every condition below is about which agents a state names.

Definition 5 (Records). A local state *records* an agent q if one of its atoms names q .

The state $\{\text{friend}(q)\}$ of p records q . A state holding $\text{received}(x, p)$ records p , who signed x ; an agent signed inside x is no argument of the atom and is not recorded.

A speech act names its signer and nobody else. A state holding one records the signer; an agent inside it, its earlier signers among them, is no argument of the atom carrying it and is not recorded: the chain of signatures inside a speech act is carried rather than required. A contract records, and its schemas can require, only the arguments of atoms and their signers, and an atom has finitely many.

Introduction. Conditions (6) and (7): any two agents must be able to come into relation by an act of the contract willed by both that leaves each recording the other, e.g. meeting.

Definition 6 (Introductory act). A schema τ of C of arity two, guarded in both roles and with no name term, is an *introductory act* if any two agents $p \neq q$ have an *introduction*: an admissible binding β of τ into $\{p, q\}$ that sends π_1 to p and π_2 to q , under which some atom added at π_1 names q and some atom added at π_2 names p .

Meeting has arity two and guards in both roles, and at the binding sending *Alice* to p and *Bob* to q it adds $knows(q)$ at p and $knows(p)$ at q , each naming the other agent. Befriending is an introductory act too, but not an unobstructed one: it requires $knows$ of the other, which no schema of arity one adds.

Having such an act is not enough, because it may be out of reach: it may require an atom an agent cannot obtain, or forbid one that somebody else can put there and leave. The condition is that neither happens.

Definition 7 (Unobstructed). An introductory act τ of C is *unobstructed* if any two agents $p \neq q$ have an introduction β such that, at each role π_i of τ :

- (1) every atom τ requires at π_i is, under β , added by some schema of C of arity one that is guarded in its role, has no name term, requires nothing, and forbids nothing or only that atom, which τ then requires once at π_i , at some admissible binding that sends its role to $\beta(\pi_i)$;
- (2) no listed role of a schema of C adds an atom whose predicate is that of an atom τ forbids at π_i , and every schema of C that, at some binding, adds at $\beta(\pi_i)$ an atom τ forbids at π_i has, at that binding, the other of p and q in a role and adds at $\beta(\pi_i)$ an atom naming them.

Meeting requires *identity* at each role, once, which establish adds, establish having arity one, being guarded in its role, requiring nothing and forbidding only *identity*, so clause 1 holds; and it forbids nothing, so clause 2 asks nothing.

The second clause concerns an atom somebody else can put in an agent's state and leave there. It excepts an act that puts it there while bringing the two agents together: its binding assigns the other agent a role, and the atom it adds names that agent, so every transaction it determines changes the state of both and leaves the first recording the second, and is an interaction between them. Befriending at the binding that swaps its two roles is one, and so is any more guarded act that specialises the introduction, e.g. a befriending that also takes the parents' will. The clause requires only that the act leave one of the two recording the other, which is enough for the act to be an interaction between them, and is all the proof of Proposition 26 uses of it.

Provenance. Condition (8): a record may name an agent its holder has never dealt with: a coin that changed hands, a message delivered to a group by its leader. Such a record must be traceable to someone who was there: either the act that made it had that agent as a participant, or another participant of the act already held a record naming them.

Definition 8 (Traceable provenance). The *base* of an argument of a schema is the argument itself where it is an agent role or an agent variable, and its base where it is a name term. Write Φ_E for the atoms of a schema whose predicate is in E . A set $E \subseteq \Sigma$ has *traceable provenance* in a contract C if for every schema τ of C with roles $\pi_1, \dots, \pi_k$, every i , every $\varphi \in +(i) \cap \Phi_E$ and every argument u of φ that is an agent role, an agent variable or a name term, whose base is not that of π_i ,

the base of u is the base of some π_j or of an argument of some $\psi \in (= (j) \uplus - (j)) \cap \Phi_E$, π_j an agent role,

and every speech-act variable that is an argument of φ or occurs in a name term among its arguments is an argument of such a ψ , occurs in a name term among its arguments, or occurs in no atom required at a role of τ . A predicate has *traceable provenance* in C if it belongs to such a set, and an atom is said to have it when its predicate does.

Delivery (Section 3.3) shows what the condition asks. It adds *received_in*($x, Alice, n$) at *Bob*: *Alice* is a role of it, and x is an argument of *posted*(x, n), which *Alice* requires, so the speech act reaches Bob through an agent that held it already.

A predicate whose atoms are added only with roles among their arguments has traceable provenance. The condition is weaker: an act may give an agent a record of someone they do not yet record, provided another agent to that act records them already.

Volition. Condition (3): the agents to an act fall into parts connected by the records they hold, and the act either has one such part, as unfriending and messaging between friends and paying by a holder of the payee's coins do, or is willed by an agent in each part.

Definition 9 (Volition). The *role graph* of a schema is the undirected graph on its roles with an edge between π_i and π_j when they have the same base or when one of the two requires an atom of traceable provenance an argument of which has the base of the other. A contract C satisfies *volition* if every schema of it has a connected role graph or has a guarding role in every connected component of its role graph.

The role graph of *unfriend* joins its two roles: *Alice* requires *friend*(*Bob*), an atom of traceable provenance with *Bob* among its arguments. Only *Alice* guards unfriending, so that edge is what its role graph rests on.

The role graph is the undirected graph formed by the identifiers the agents store, of Section 1, taken on the roles of one schema. For a schema of two agent roles it is connected exactly when one of them requires such an atom naming the other. Above that arity connectedness is weaker than requiring it of every pair, and Proposition 27 uses it: lying in one instance is an equivalence, so a path through the roles puts all the participants there.

Definition 10 (Syntactically grassroots). A contract is *syntactically grassroots* if it has an unobstructed introductory act, the set of predicates occurring in it has traceable provenance, and it satisfies volition.

The compilation of Section 5 is defined for act schemas, provided they are syntactically grassroots.

THEOREM 11 (GRASSROOTS REALISATION). *The protocol realising a syntactically grassroots social contract is grassroots and volitionally grassroots.*

Sections 4 to 7 prove it.

Whether a contract is syntactically grassroots is decidable, and a checker deciding it accompanies this paper, at <https://github.com/EShapiro2/GLP/tree/main/programs/gsc>. It certifies GSN and GC, with the sets of predicates of traceable provenance that Section 7 states. Everything the three conditions quantify over is finite once the bindings are taken up to renaming: an introductory act is an assignment of a schema's agent variables to its two roles; unobstructed is the matching of one atom against those the finitely many schemas add; volition is a connectivity check on the roles. Traceable provenance is not decided in a single pass: the largest set of predicates having it is reached by dropping any predicate that fails the condition and repeating, which ends because there are finitely many.

Section 4 recalls what *grassroots* means and Section 5 says what the protocol realising a contract is. Section 6 proves the theorem the proof rests on, and Section 7 proves this one.

4 Volition-Guarded Transactions and Grassroots Protocols

This section recalls from [40] what the statements below use, in brief. Nothing in it is ours. Appendix A states the same notions formally, together with the results used in the proofs of Appendix B, so that the paper is self-contained; the proofs of those results are in [40] and are not repeated. Volition-guarded transactions follow multiagent transition systems [51] and multiagent atomic transactions [61]. Throughout, Π is a countable, potentially infinite set of *agents*, an agent being a person operating a machine, and $P \subset \Pi$ ranges over its finite nonempty subsets. For a set S , S^P is the set of total functions from P to S , and c_p is the value of c at p .

A *local-states function* S maps every $P \subset \Pi$ to a set $S(P)$ of local machine states containing the initial state s_0 , with $P \subset P'$ implying $S(P) \subset S(P')$. A *machine transaction* over *participants* Q is a pair $d \rightarrow d' \in (S^Q)^2$ with $d \neq d'$; it is *unary* if $|Q| = 1$. A *volition-guarded transaction* is a pair (t, Q') of a machine transaction t over Q with its *guards* $Q' \subseteq Q$: the agents in Q' must be willing for it to be taken. A *transaction equivalence* $\sim$ relates transactions with the same participants and has countably many classes; an agent wills a class, and taking one member of it fulfils that will. For a set R of volition-guarded transactions over S and $P \subset \Pi$, we write $R(P)$ for those of them whose transactions are over $S(P')$ for some $P' \subseteq P$, and call R *contained* if every transaction of it with participants in P and source over $S(P)$ is in $R(P)$.

An *agent state* is a pair of a *volitional state*, a set of classes the agent is willing their machine to take part in, and a machine state. A set of volition-guarded transactions over a local-states function induces, for each P , a transition system whose configurations are agent configurations over P and whose transitions are *volition changes*, by which an agent by themselves alters their volitional state, and *volitional machine transactions*, each induced by a volition-guarded transaction whose guards all hold its class. A transaction is *machine-enabled* in a configuration whose machine states restrict on the participants to its source, and *enabled* if in addition every guard holds its class. The family of these systems, one per P , is the *protocol* $\mathcal{F}$ over the set, $\mathcal{F}(P)$ being its member at P ; runs are *safe* when consecutive states form transitions, *live* when no class stays enabled forever without a member being taken, and *correct* when both.

A transition of $\mathcal{F}(Q)$ is an *interaction* between two agents if it changes the state of each and its restriction to one of them is no transition of that agent's own system; it is an interaction between two disjoint sets of agents if it is one between a member of each. In the *interaction graph* of a run, two agents are joined whenever some transition of the run is an interaction between them; an *instance* in the run is a connected component of that graph, and two instances *coalesce* when they come to lie in one component.

A protocol is *oblivious* if any interleaving of correct runs of two disjoint sets of agents is a correct run of the two together, *interactive* if two sets that have not yet interacted can always still do so, and *grassroots* if it is both. An agent has *left* when no transaction they guard can be machine-enabled again. A protocol is *volitionally grassroots* if it is oblivious, two sets each holding an agent that has not left can always still interact, and the first interaction of two sets is induced by a transaction guarded by a member of each, so instances coalesce only by mutual consent; a grassroots protocol meeting the last condition is volitionally grassroots. In a grassroots protocol two instances can always coalesce.

5 Compiling Schemas into Transactions

Act schemas have no operational semantics of their own: a schema of a syntactically grassroots contract means the volition-guarded transactions it compiles to, and those have the semantics of [40]. The compilation is defined for syntactically grassroots contracts, so a contract is checked before it is compiled. S_C is a local-states function in the sense of Appendix A: $s_0 \in S_C(P)$ for every

P , and $P \subset P'$ implies $S_C(P) \subset S_C(P')$, strictly, provided some predicate has positive arity, as one does in every syntactically grassroots contract, its introductory act adding at π_1 an atom naming π_2 : for $q \in P' \setminus P$ and such a predicate, an atom with q among its arguments gives a state in $S_C(P')$ and not in $S_C(P)$.

5.1 The Compilation

Compilation maps an act schema of a syntactically grassroots contract to a volition-guarded multiagent atomic transaction [40].

For multisets $s_1, \dots, s_n$ write $\bigsqcup_j s_j$ for their *join*, in which the multiplicity of each member is the greatest of its multiplicities in the s_j .

Definition 12 (Extent). Let β be a binding for a schema τ and d a machine configuration over a set of agents holding $\beta(\pi)$ for every agent role π of τ . The *extent* in d of an agent role π of τ is $\text{ext}_d(\pi) := \{\beta(\pi)\}$, and of a listed role it is

$$\text{ext}_d(\pi[\rho(*, u_2, \dots, u_r)]) := \{q \in \Pi : \rho(q, \beta(u_2), \dots, \beta(u_r)) \in d_{\beta(\pi)}\}.$$

At a binding sending *Alice* to a and n to v , the extent of the listed role of closing a group in a configuration c is $\{q : \text{listed}(q, v) \in c_a\}$, every member a lists, a among them.

Definition 13 (Compilation). Let τ be a schema with roles $\pi_1, \dots, \pi_k$ and β a binding for it, and let $Q_i := \text{ext}_d(\pi_i)$ for a machine configuration d . A machine transaction $d \rightarrow d'$ over $Q := \bigcup_i Q_i$ is *determined by τ at β* if $\beta(\sigma \cdot y)$ is an argument of no atom of d for every name term $\sigma \cdot y$ of τ that occurs in no atom required at a role, and if for every i and every $p \in Q_i$

$$\beta(=(i) \uplus -(i)) \subseteq d_p, \quad \beta(\neg(i)) \cap d_p = \emptyset,$$

and for every $p \in Q$

$$\begin{aligned} \bigsqcup_{i: p \in Q_i} \beta(+(i)) \cap \bigsqcup_{i: p \in Q_i} \beta(-(i)) &= \emptyset, \\ d'_p &= \left(d_p \setminus \bigsqcup_{i: p \in Q_i} \beta(-(i)) \right) \uplus \bigsqcup_{i: p \in Q_i} \beta(+(i)), \end{aligned}$$

and the volition-guarded transaction it determines is the pair of it with $\{\beta(\pi) : \pi \text{ a guarding role of } \tau\}$. In the notation of [40], with $Q_i := \text{ext}_c(\pi_i)$ and $Q := \bigcup_i Q_i$, τ *compiles to*

$$\begin{aligned} c'_p &:= (c_p \setminus \bigsqcup_{i: p \in Q_i} -(i)) \uplus \bigsqcup_{i: p \in Q_i} +(i) && (p \in Q), \\ \text{provided } =(i) \uplus -(i) &\subseteq c_p && (p \in Q_i, 1 \leq i \leq k), \\ \text{provided } \neg(i) \cap c_p &= \emptyset && (p \in Q_i, 1 \leq i \leq k), \\ \text{provided } \bigsqcup_{i: p \in Q_i} +(i) \cap \bigsqcup_{i: p \in Q_i} -(i) &= \emptyset && (p \in Q), \\ \text{provided } \sigma \cdot y &\text{ is an argument of no atom of } c && (\sigma \cdot y \text{ a name term of } \tau \text{ in no required atom),} \\ &\text{guarded by the guarding roles of } \tau, \end{aligned}$$

which at a binding β denotes the volition-guarded transactions determined by τ at β .

With *Alice* and *Bob* standing for the agents in those roles, *befriend* (Section 3.3) compiles to

$$\begin{aligned} c'_{\text{Alice}} &:= c_{\text{Alice}} \uplus \{\text{friend}(\text{Bob})\}, && c'_{\text{Bob}} := c_{\text{Bob}} \uplus \{\text{friend}(\text{Alice})\}, \\ \text{provided } \text{identity}, \text{ knows}(\text{Bob}) &\in c_{\text{Alice}} \text{ and } \text{identity}, \text{ knows}(\text{Alice}) \in c_{\text{Bob}}, \\ \text{provided } \text{friend}(\text{Bob}) &\notin c_{\text{Alice}} \text{ and } \text{friend}(\text{Alice}) \notin c_{\text{Bob}}, \\ &\text{guarded by } \{\text{Alice}, \text{Bob}\}, \end{aligned}$$

and denotes, for every d with $\text{identity}, \text{knows}(\text{Bob}) \in d_{\text{Alice}}$, $\text{identity}, \text{knows}(\text{Alice}) \in d_{\text{Bob}}$, $\text{friend}(\text{Bob}) \notin d_{\text{Alice}}$ and $\text{friend}(\text{Alice}) \notin d_{\text{Bob}}$, the volition-guarded transaction $(d \rightarrow d', \{\text{Alice}, \text{Bob}\})$ with $d'_{\text{Alice}} = d_{\text{Alice}} \uplus \{\text{friend}(\text{Bob})\}$ and $d'_{\text{Bob}} = d_{\text{Bob}} \uplus \{\text{friend}(\text{Alice})\}$. There is one for every admissible d , since the two may befriend whatever other friends each has, and $\sim$ relates them all: they are one act of befriending, available at different points of a run.

Write L for $\text{ext}_c(\text{Alice}[\text{listed}(*, n)])$. Close (Section 3.3) compiles to

$$\begin{aligned} c'_{\text{Alice}} &:= (c_{\text{Alice}} \setminus \{\text{listed}(\text{Alice}, n), \text{member}(\langle \text{Alice} \rangle \cdot n)\}) \uplus \{\text{closed}(n)\}, \\ c'_q &:= c_q \setminus \{\text{member}(\langle \text{Alice} \rangle \cdot n)\} \quad (q \in L, q \neq \text{Alice}), \\ &\text{provided } \text{listed}(\text{Alice}, n) \in c_{\text{Alice}} \text{ and } \text{member}(\langle \text{Alice} \rangle \cdot n) \in c_q \quad (q \in L), \\ &\text{guarded by } \{\text{Alice}\}. \end{aligned}$$

Alice lists herself, so she is in the extent of the listed role, and the act deletes her membership with her listing. Its participants are Alice and every member she lists, read from her own state, so no member is left out of the act.

The swap (Section 3.4) compiles to

$$\begin{aligned} c'_{\text{Alice}} &:= (c_{\text{Alice}} \setminus k \cdot \Phi(u)) \uplus k_1 \cdot \Phi(v), & c'_{\text{Bob}} &:= (c_{\text{Bob}} \setminus k_1 \cdot \Phi(v)) \uplus k \cdot \Phi(u), \\ &\text{provided } \text{identity}, \text{knows}(\text{Bob}) \in c_{\text{Alice}} \text{ and } k \cdot \Phi(u) \subseteq c_{\text{Alice}}, \\ &\text{provided } \text{identity}, \text{knows}(\text{Alice}) \in c_{\text{Bob}} \text{ and } k_1 \cdot \Phi(v) \subseteq c_{\text{Bob}}, \\ &\text{guarded by } \{\text{Alice}, \text{Bob}\}. \end{aligned}$$

The bindings with $u = v$ determine nothing, by Definition 13: under them each role adds coins it deletes. The swap at $u = \text{Alice}$ and $v = \text{Bob}$ is the mutual credit line.

The equivalence is on machine transactions, so two acts that no configuration distinguishes are one act, and a contract whose act names carry a distinction of their own writes that distinction into the effects. A contract compiles schema by schema. A schema with no guarding role compiles to transactions with an empty guard, which are enabled whenever they are machine-enabled and are then eventually taken: no agent wills such an act.

Definition 14 (Realisation of a contract). Let C be a syntactically grassroots contract. Its *realisation* R_C is the set of volition-guarded transactions determined by the schemas of C at all admissible bindings for them, and $\sim$ is the equivalence on their underlying machine transactions generated by relating two of the same participants when one schema at one admissible binding determines both. The *protocol realising* C is the volitional transactions-based protocol over R_C , S_C and $\sim$.

A set of volition-guarded transactions over S_C with an equivalence $\sim$ is *well formed* if two machine transactions of different classes, each underlying an unguarded transaction of it, have disjoint P -closures for every $P \subset \Pi$, and $\sim$ is a transaction equivalence with countably many classes over any P , which is what Definitions 36 and 43 require.

LEMMA 15 (THE COMPILATION IS WELL DEFINED). R_C with $\sim$ is well formed.

LEMMA 16 (CONTAINMENT). R_C is contained.

5.2 Every Participant Takes a Role

Throughout, C is a syntactically grassroots contract, $\mathcal{F}$ the protocol realising it and $P \subset \Pi$. No act reaches anyone without a role in it.

PROPOSITION 17 (EVERY PARTICIPANT TAKES A ROLE). *Every agent whose machine state a transition of $\mathcal{F}(P)$ changes is in the extent of a role of a schema of C at some binding.*

An agent therefore bears an effect of the contract only under a schema of it, and only in a role. The second property of Section 3.5 is that the schemas single out no agent.

A permutation σ of Π extends to speech acts by $\sigma(x, p) := (\sigma(x), \sigma(p))$, the identity on utterances, to community names by $\sigma(\langle p \rangle) := \langle \sigma(p) \rangle$ and $\sigma(v \cdot y) := \sigma(v) \cdot \sigma(y)$, to atoms by permuting their arguments, and to states, configurations, transactions and runs componentwise.

PROPOSITION 18 (ANONYMITY). *For every permutation σ of Π , $\sigma(R_C) = R_C$, $\sigma(c_0(P)) = c_0(\sigma(P))$, and σ carries the runs of $\mathcal{F}(P)$ to the runs of $\mathcal{F}(\sigma(P))$.*

COROLLARY 19 (NO PRIVILEGED AGENT). *If a configuration is reachable in a run over P , so is its image under every permutation of P .*

Conditions (1), (2), (4) and (5) of Section 1 follow from these two properties, as Section 3.5 sets out. That the protocol realising the contract is grassroots does not follow from them.

6 Polite Sets of Volition-Guarded Transactions

Here we prove that the following condition on a contained set of volition-guarded transactions makes the protocol over it grassroots and volitionally grassroots: (1) any two agents can always come to interact, and (2) no act joins two instances unless each of them holds a guard of it. Section 7 states conditions on the acts of a contract sufficient for its compiled volition-guarded transactions to satisfy it, and GSN and GC (Section 2) satisfy it.

The proofs use two graphs: the interaction graph of Section 4, undirected, whose connected components are the instances; and the directed graph of who holds a record of whom, defined next.

Throughout this section, S is a local-states function, R a set of volition-guarded transactions over S with equivalence $\sim$, and $\mathcal{F}$ the protocol over R , S and $\sim$. The proofs, in Appendix B, use four results of [40] recalled in Appendix A: obliviousness rests on Lemmas 55 and 54 through Proposition 56, and interactivity on Lemma 40; openness yields interactivity and closure yields consent.

Definition 20 (Records). A machine state s records an agent $q \in \Pi$ if $s \notin S(P)$ for every $P \subset \Pi$ with $q \notin P$.

Definition 21 (Recording graph). The recording graph of a machine configuration d is the directed graph on Π with an edge $p \rightarrow q$ whenever d_p records q . The recording graph of an agent configuration is that of its machine states, and the recording graph of a run is the union of the recording graphs of its configurations.

Definition 22 (Mutual recording). A volition-guarded transaction $(d \rightarrow d', \{p, q\})$ over participants $\{p, q\}$, $p \neq q \in \Pi$, is a mutual recording of p and q if it changes the state of each and the recording graph of d' has the edges $p \rightarrow q$ and $q \rightarrow p$.

A state records q when it could not have arisen among agents that leave q out: every set of agents whose states include it contains q . The recording graph of the initial configuration is empty, since $s_0 \in S(P)$ for every P . In GKK (Section 2.1) the edges out of an agent are those they know, in GSG their friends too, and in GC (Section 3.4) the issuers of the coins they hold. A mutual recording is an act of two agents, willed by both, after which there is an edge each way between them: meeting is one, and so is befriending. An edge may point at an agent its source has never transacted with, as when a message is delivered to a group by its leader or a coin changes hands, and nothing below forbids it. The recording graph of a configuration is not monotone along a run, as an act may

delete the atoms that put an edge there; the recording graph of a run and the interaction graph are monotone by construction.

Definition 23 (Open, Closed, Polite). The set R is:

- (1) *open* if for any two agents $p \neq q$ in any $Q \subset \Pi$, every finite safe run of $\mathcal{F}(Q)$ with no interaction between p and q has a finite safe extension, by volition changes and transitions that are not interactions between p and q , at whose end some transaction of $R(Q)$ whose induced transitions are interactions between p and q is machine-enabled;
- (2) *closed* if for every $Q \subset \Pi$, every transaction of R enabled at the end of a finite safe run of $\mathcal{F}(Q)$ has all its participants in one instance in that run or a guard in every instance in that run holding one of them;
- (3) *polite* if it is open and closed.

Openness says that any two agents can always be brought, by zero or more acts that are not between the two, to a configuration at which an act coupling the two can be taken if all its guards are willing. It does not require that no one else take part in the act: a mutual recording is one, and so is any act that leaves one of them holding a record of the other, whoever else takes part. Of two agents that have already taken part in one act nothing is required: they have interacted, and being interactive requires no more. Closure says that an act whose participants lie in two or more instances is enabled only where each of those instances holds a guard of it, and an act no one wills only where they already lie in one instance.

THEOREM 24 (POLITE). *The protocol over a contained, polite set of volition-guarded transactions is grassroots and volitionally grassroots.*

Politeness is a condition on the transactions of a protocol. Being grassroots is a property of its runs. The theorem states that the first is sufficient for the second. Section 7 proves the step before it, from the acts of a contract to the transactions realising them.

7 Conditions Sufficient for Openness and Closure

We are now ready to prove Theorem 11, restated. Openness and closure are conditions on runs; the three conditions are on the schemas, and the proof has two halves: openness follows from the unobstructed introductory act, and closure from volition and traceable provenance. The proofs are in Appendix B. Throughout this section C is a syntactically grassroots contract and $\mathcal{F}$ the protocol realising it.

THEOREM 11 (GRASSROOTS REALISATION). *The protocol realising a syntactically grassroots social contract is grassroots and volitionally grassroots.*

LEMMA 25 (RECORDS). *On local states, Definitions 20 and 5 agree.*

PROPOSITION 26 (OPENNESS). R_C is open.

PROPOSITION 27 (CLOSURE). R_C is closed.

PROOF OF THEOREM 11. Let C be syntactically grassroots. By Proposition 26 R_C is open and by Proposition 27 it is closed, so it is polite; by Lemma 16 it is contained; and by Theorem 24 the protocol over it is grassroots and volitionally grassroots. $\square$

Openness and closure. Openness lets any agent start an instance, and without permission. A contract that gives one agent of each instance a standing of their own is not open: two instances

with no such agent can never coalesce. Federated architectures fail this condition, a server being such an agent [40].

Closure leaves an instance requiring no global resource and no central authority: a registry that issues identifiers, a log that orders acts, or an authority that timestamps them takes part in a transaction without a role in it. Every act of a grassroots social contract is between its agents, so no third party takes part in one. A smart contract that depends on a fact outside the blockchain obtains it from an *oracle*, a third party whose report the contract cannot check [5]; here a fact about the world enters as an agent's signed assertion. A smart contract requires a consensus protocol; a contract realised as volition-guarded transactions requires none where the agents transact between themselves [31, 39], and where a collective decision is called for it constitutes its own consensus among the agents [35].

7.1 The Social Network Certified

Meet, which has no name term, is an introductory act: for any two agents, at the binding sending Alice and Bob to them, every transaction it determines adds *knows*(Bob) at Alice and *knows*(Alice) at Bob, changing both states, and it is guarded in both roles, so each is a mutual recording of the two. It is unobstructed: it requires *identity* at each role, once, which establish adds, establish having arity one and no name term, being guarded in its role, requiring nothing and forbidding only *identity*; and it forbids nothing. Meet and introduce are the schemas adding a *knows* atom: meet adds at each role an atom naming the other role, and introduce adds *knows*(c) at Bob, c an argument of *knows*(c), which Alice requires; so *knows* has traceable provenance, and so has *friend*, befriend being the only schema adding a *friend* atom and that atom's argument a role. So has the set of all ten predicates of the contract: establish adds *identity*, which has no arguments; message adds *received*(x, Alice) at Bob, whose arguments are the role Alice and the speech-act variable x, which occurs in no atom required at a role; and the schemas of a group meet the condition, as the paragraph below shows.

Volition holds, by Definition 9: meet and befriend are guarded in both their roles; establish has arity one; introduce's role Alice requires *knows*(Bob), and the role Alice of unfriend and of message requires *friend*(Bob), each an atom of traceable provenance naming the other role; and the schemas of a group meet it, as the paragraph below shows.

The schemas of a group. Create adds atoms whose arguments are its role, the name $\langle \text{Alice} \rangle \cdot n$ based on it, and a speech-act variable it requires nowhere; join and close add atoms whose arguments are roles and that name; post_own and post add *posted*(x, n) at the leader, x required nowhere; deliver adds *delivered*(x, Bob, n) at the leader and *received_in*(x, Alice, n) at the member, whose arguments are roles, n and x, which the *posted*(x, n) required at the leader carries; and leave and drop add nothing, so the six predicates of a group have traceable provenance, each traced through an agent role. Join is guarded in both its roles; create and post_own have arity one; the two roles of close have the base Alice; and the role Alice of leave, drop and deliver requires *listed*(Bob, n) and the role Bob of post requires *member*($\langle \text{Alice} \rangle \cdot n$), each an atom of traceable provenance an argument of which has the base of the other role, so volition holds of them. Meet forbids nothing, so clause 2 of Definition 7 asks nothing of the listed role of close.

The contract is therefore syntactically grassroots, and by Theorem 11 the protocol realising it is grassroots and volitionally grassroots, so the platform realising the contract is a grassroots platform.

The instances. The introductory act is meeting, and a *knows* atom was added by a meet, a mutual recording and so an interaction by Lemma 61, or by an introduce, an interaction of the two it joins, the one introduced being known to the introducer already; every *friend* atom was added

by a befriend, a mutual recording. Every interaction of this contract between two agents requires or adds an atom in the state of one of them naming the other, so it puts an edge of the recording graph between them; and by Lemma 63, every edge of the recording graph joins two agents in one instance. The instances of a run are therefore the connected components of its recording graph, taken undirected. A message adds an edge from the recipient to the sender, a participant; a delivery adds one from the member to the leader, a participant, and one to the member who posted the item, who signed it, reached through the leader (Lemma 63). A coin is different: it names its issuer, who may never have transacted with its holder (Section 7.2).

7.2 The Currency Certified

Meet is GC's introductory act, unobstructed, as in GKK (Section 7.1); the swap is an introductory act too, at the binding sending u to Alice and v to Bob leaving each holding a coin the other issued, the mutual credit line, but not an unobstructed one, requiring at each role *knows* of the other and a coin, which no schema of arity one that requires nothing adds, mint requiring *identity*.

$\mathbb{C}$ has traceable provenance, as *identity* and *knows* have in GKK. Mint adds $\mathbb{C}(\text{Alice})$ at Alice and pay adds $\mathbb{C}(\text{Bob})$ at Bob, each naming the role that holds it; the swap adds in each role a coin the other role requires; and redeem adds $\mathbb{C}(r)$ at Alice, which role Bob requires, and $\mathbb{C}(\text{Bob})$ at Bob, naming its own role. Volition holds, by Definition 9: mint has arity one, the swap is guarded in both its roles, and pay and redeem are guarded at Alice only and require there *knows*(*Bob*) and $\mathbb{C}(\text{Bob})$, atoms of traceable provenance having Bob among their arguments. The contract is therefore syntactically grassroots, and by Theorem 11 the protocol realising it is grassroots and volitionally grassroots, so the platform realising the contract is a grassroots platform.

The origin of a record. A coin names its issuer, who is not a participant of the payment that moves it, so a edge may point at an agent its source has never transacted with, where a *friend* edge can only have been added by the two agents it joins. Traceable provenance covers it: an edge is added either to a participant of the act that adds it or to an agent another participant of that act records already, so no act ever leaves an edge whose ends lie in different instances, which Lemma 63 proves by induction on the run. For coins the chain is the passage of the coin from its issuer, and conservation of money [40] is the special case, here an invariant of the schemas rather than one proved of the runs.

8 Legal Modalities of Contract Clauses

The schemas also determine, for each clause, whether an agent can breach it and what record of it the contract leaves in another agent's state: its *legal modality*, its *modality* for short. The two contracts show it. A clause may be one no act of the contract can breach, one an agent may breach with the contract recording the breach, or one whose performance lies outside the contract and of which it records only the undertaking. The question is about every clause, not only about those that describe no act. The three are what a realisation does with a clause, and are not kinds of obligation: a friendship kept on paper can be entered against an agent's will, and under the schemas of Section 3.3 it cannot.

Throughout this section C is a syntactically grassroots contract. An agent joins the platform realising C by signing an *undertaking* to keep C towards all its other participants, which they present to every participant they interact with (Section 9).

Definition 28 (Clause). A *clause* of a contract C is a proposition about the conduct of its agents, and is *kept* or *breached* by that conduct; an agent the clause names as answerable is its *obligor*. The runs of the protocol realising C *determine* the clause if whether it is kept is a function of the run. A

transaction of R_C *may breach* the clause if it is taken in some correct run and taking it breaches the clause.

Definition 29 (Witnessed). A volition-guarded transaction $(d \rightarrow d', Q')$ over participants Q is *witnessed against* $a \in \Pi$ if $d'_b \setminus d_b$ holds an atom naming a for some $b \in Q$ with $b \neq a$. A schema is *witnessed against* an agent role if every transaction it determines is witnessed against the agent in that role.

That atom is one the act put in the state of another agent, and by Lemma 25 an edge $b \rightarrow a$ of the recording graph of d' that d did not have. A record serves as evidence when the act put it there, it names the agent answerable, and someone else holds it.

Definition 30 (Enforced, Attested, Undertaken). Let φ be a clause of C , with obligor o where it has one. Then φ is

- (1) *enforced* by C if the runs of the protocol realising C determine it and every correct run of that protocol keeps it;
- (2) *attested* by C if some transaction of R_C may breach it and every transaction that may is witnessed against o ;
- (3) *undertaken* by C if no transaction of R_C may breach it, the runs do not determine it, and some transaction of R_C is witnessed against o or o 's undertaking is held by another agent.

The three are exclusive and not exhaustive. Two kinds of clause have none. One that some transaction may breach without witnessing against the obligor can be breached invisibly. One that the runs neither determine nor record is owed by no one identifiable. A realisation does nothing with either.

A schema *names* its agent role π_j at another agent role π_i if an atom added at π_i names π_j .

PROPOSITION 31 (WITNESSING). *A schema is witnessed against every agent role it names at another agent role.*

A predicate e is *guarded* in C if whenever a schema of C adds an e -atom, the atom names only roles of that schema that will the act. The *consent clause* for e says that an agent's state holds an e -atom only if those the atom names willed the act that added it.

PROPOSITION 32 (ENFORCEMENT BY GUARDING). *If e is guarded in C , the consent clause for e is enforced by C .*

The clauses of the contracts. GKK:2, GKK:3 and GSG:2 are *enforced*, by the schemas requiring what the clauses require: a person knows another only by meeting them or through someone who knows them. GSG:1 is *attested*: befriending someone one does not know personally is an act, and by Proposition 31 it is witnessed against both persons, while no act can breach the rest of the clause, friendship being guarded (Proposition 32) and befriending requiring that each knows the other. GSN:1 to GSN:3 and the leaving of GSN:4 permit acts and oblige nothing, and the delivery of GSN:5 and the removal of a member under GSN:4 are unguarded and so taken in every correct run in which they stay enabled: the runs determine GSN:1 to GSN:5 and every correct run keeps them, so they are *enforced*. GKK:1 is *undertaken*: no act can breach the establishing of an identity, and being accountable for what is signed with a key is not an act of the contract, so the runs do not determine the clause, and the agent's undertaking of Section 9, which another agent holds, is its record.

The clauses of GC. GC:1 to GC:4 permit acts and oblige nothing, and the knowing GC:2 to GC:4 require the schemas require, so the runs determine them and every correct run keeps them: they are *enforced*. GC:5 is *undertaken*, a mutual credit line adding $\pounds(\text{Bob})$ at Alice and $\pounds(\text{Alice})$ at Bob

and so being witnessed against both of them. The clause that a coin enters a holding only by its issuer's will has no modality, being breachable by pay without pay naming the payer.

Performance outside the system. GC:5 and GSG:1 rest on what is done outside the schemas, since selling at an advertised price is no act of the contract and neither is knowing a person personally, and they have different modalities, the first *undertaken* and the second *attested* (Section 8). The three modalities therefore do not divide by whether performance is digital.

Attestation and undertaking. The two require the same thing of the schemas, that the act naming the obligor leave the record in another agent's hands, and differ only in whether the breach is itself an act of the contract. The evidence depends on the schemas and the modality on the clause. A performance promised for later is never attested, because failing to perform is not an act; a warranty given in the course of an act is attested exactly when that act is witnessed against the agent giving it.

Witnessing therefore depends on whether an effect names the agent performing the act, and not only the agents the atom concerns. Unfriend adds no atom, so no clause against ending a friendship at will can be attested under GSG; pay adds a coin naming its issuer and not its payer, so no clause against the payer can be attested under GC. A contract whose breaches are to be attested must have acts whose effects name their author. Signing provides it: an act taken as a digital speech act, an utterance signed with the key of the agent taking it, leaves an atom naming that agent, and the signed record is non-repudiable [7, 24].

9 Signing the Contract

A person joins a global platform by signing an End-User Licence Agreement with its operator. An agent joins a grassroots platform by signing a one-sided *undertaking* to keep its grassroots social contract towards all its other participants, and presents it to every participant they interact with; the code realising the contract, or the platform hosting it, enforces this, refusing an interaction with a participant that has not presented one. The undertaking is a speech act, signed with the key of the agent giving it and non-repudiable [7, 24]. An agent signs by themselves, before their first interaction; where a participant invites them, the signed invitation names the contract and the invitee, and the invitee presents their undertaking first to the inviter.

Presentation is not part of the protocol: the code carries the undertaking in the messages of an interaction, and it leaves no atom in a state. Throughout this section C is a syntactically grassroots contract, $\mathcal{F}$ the protocol realising it and $Q \subset \Pi$; write $p \triangleright q$, for $p, q \in Q$, when p has presented its undertaking to q .

Definition 33 (Presenting run). A run of $\mathcal{F}(Q)$ with a relation $\triangleright$ on Q is *presenting* if $p \triangleright q$ and $q \triangleright p$ for every interaction of the run between p and q .

The code realising C , or the platform hosting it, keeps every run presenting.

PROPOSITION 34 (EVERY PARTICIPANT HAS PRESENTED ITS UNDERTAKING). *In a presenting run, every agent of an instance of two or more agents has presented their undertaking to another agent of that instance.*

By Section 8 a record is evidence when it names the agent answerable and is held by someone else; the undertaking names its signer, and by the proposition another participant holds it. Condition (5) of Section 1 is therefore enforced by the code.

The code. Enforced clauses are enforced by the code the agents run: a breach of one is impossible with a correct realisation of the contract (Section 8), so a breach requires an agent to run other code. The code is warranted by its *creator*, a person, natural or legal, and by its *compiler*. Each warranty

is signed and carried with the code: the creator's, that the code realises the contract, carrying the contract's schemas and the checker's verdict that they are syntactically grassroots; the compiler's, that the compiled code is the creator's. Neither need have adopted the contract, so the warranties are not clauses of the contract and have no modality under Definition 30; a breach of either is provable from the warranted code and the contract by anyone holding both. In their undertaking an agent also promises to run the warranted code. This clause is *undertaken*: no act of the contract can breach it, no run shows whether an agent keeps it, and the undertaking, in another agent's hands, is its record. Where the platform hosting the code attests it, every interaction opening with proof that both agents run the warranted code, as in the Grassroots Super-App (in preparation), the clause is *enforced* by the platform.

Electronic signatures. An undertaking signed with an agent's key is an electronic signature. In the United States a signature, contract or record may not be denied legal effect, validity or enforceability solely because it is in electronic form [70]; in the European Union an electronic signature shall not be denied legal effect and admissibility as evidence solely on the grounds that it is in electronic form [14]; and in the United Kingdom an electronic signature, and its certification, are admissible in evidence on the authenticity and integrity of the data signed [69]. These statutes settle the form; whether an undertaking is enforceable is a question for the ordinary law of contract: the agents of a grassroots social contract freely accept and sign it, as condition (5) requires, and its realisation keeps signed records of the acts that follow.

10 Related Work

Each paragraph below is expanded in Appendix D.

Contracts as formal objects. The formal study of contracts in this field is dominated by the contract-as-automaton family – contract automata [1], a loan agreement as a finite automaton [17], the L4 language [71], timed contract automata [8], and encodings as defeasible deontic rules [28] and the comparison of imperative with declarative smart contracts [27]. All fix the parties when the contract is formed, keep one contract state, and assume an alphabet of events, a clock or a ledger, where our conditions quantify over the family of systems a contract induces on every finite set of people, need a local state per agent, and forbid a participant without a role in the act. Symboleo [45, 64] and Stipula [10] are the nearest to ours and make the same three commitments; Catala [41] does the corresponding work for statute. Befriending cannot be written in Stipula, which allows joint consent only in the constructor. We claim no new language: the conditions are on the four things a schema carries, which Stipula's functions and Symboleo's obligations also carry, given a semantics in which each agent holds its own state. BSPL [66] specifies an interaction protocol by message schemas between roles, with no global state: the local state of a role is the history of its messages, and a role may send a message once it knows the information the message requires, as an act schema requires atoms at a role; a message passes between two roles, where an act may take any number of roles and the will of each.

Norms and normative systems. Contracts are represented as defeasible rules with deontic operators, against which compliance is checked [26], and normative multi-agent systems concern how norms are operationalised and brought into conflict [4]; a guard [40] is a condition of enactment rather than a deontic operator. A social commitment [65] is of a debtor to a creditor in a context group, whose norms settle disputes between the two; an undertaken clause of Section 8 is a commitment of its obligor to every other participant, with no group above the agents, and another agent holds its record.

Juridical acts and declarative power. What we call an act is the *result-declaration* of Sartor [49], modelled by Hage [32], and a declarative power exercised jointly is a contract [21, 22]. There a

proclamation has effect only if an institution provides for it [18, 21, 30, 34]; a grassroots social contract has no institution, so an act's effect is the change in the agents' own states, and its guard takes the place of authorisation. An electronic institution [13] specifies the scenes in which agents in roles exchange speech acts, with institutional roles that external agents cannot play, and the institution keeps each agent's obligations; a grassroots social contract has no institutional role, and the records of its acts are in the agents' own states.

Standard-form contracting. The End-User Licence Agreement is a *contract of adhesion*, treated by Radin [47], Kim [37] and Ben-Shahar and Schneider [2]; a grassroots social contract has no drafting party, so the questions of that literature do not arise.

Terms a machine can evaluate. Surden [67], smart legal contracts [9] and enforceable security policies [50] each divide a contract's terms in two by what a realisation can do with them; Section 8 divides them in three, and the third class, a breach recorded against its obligor, is the one a remedy needs. Compliance monitors receive inputs from trusted observers [6, 42]; a monitor is a participant without a role in the act, and closure forbids one.

Code as a regulator. Lessig [38] argued that architecture regulates as code, and the digital social contract [7] and smart contracts [11] make the contract a program; a grassroots social contract is not code, its conditions being checked on the text.

Self-governance. Ostrom [44] and commons scholarship [3, 33] concern communities governing a resource without an owner or a state, and platform regulation [15] binds an operator; a grassroots social contract governs a relationship, and a platform constituted from its members' own devices [53, 55, 56] has no operator.

Grassroots systems. Grassroots protocols are due to [53] and were recast with volition-guarded transactions in [40]; platforms have been specified for social networking [54] and for personal cryptocurrencies [39, 52, 60]. Secure Scuttlebutt [68] is a peer-to-peer social network in which each person chooses their own key pair, signs each message of their own append-only log, and replicates the logs of those they follow, with no central authority; following takes no consent of the one followed, where befriending under GSG:1 takes the will of both. Added here are politeness, the legal text with decidable conditions on it, and what a realisation does with a clause.

Neighbouring instruments. A licence [20] presupposes an entitlement, bylaws an association, a code of conduct and terms of membership whoever runs the forum; a grassroots social contract presupposes none. A Ricardian contract [29] is a document an issuer signs and offers to the holders of an instrument, readable by people and parsable by programs, whose hash every transaction record carries; a grassroots social contract has no issuer, and each participant signs an undertaking to all the others (Section 9).

11 Conclusion

Proudhon's conditions on a social contract, with two more, formalised as conditions on its acts, give it the property he envisioned for it: any two agents may come into relation without anyone's leave, two instances formed independently can later become one, and every participant in an act takes a role in it. A grassroots social contract is one meeting the conditions, and its realisation has all three.

Such a contract has two artefacts, a legal text and a protocol realising it, and the grassroots literature has had only the second. This paper translates the first into the second. A clause of the text describes an act, and the acts become act schemas; the schemas compile into volition-guarded transactions over a local-states function in which a state records exactly the agents its atoms name; and three conditions on the schemas make a contract *syntactically grassroots*, which is enough

for the protocol realising it to be grassroots and volitionally grassroots. The compilation and the conditions are algorithmic and the conditions are decidable, so a contract can be certified from its text, and the only step left to judgement is the first, rendering a text as schemas.

Act schemas have listed roles: the participants of a listed role are the agents one agent's state lists, as a group's leader lists its members, so closing a group is one act of the leader and of every member.

The same schemas determine what a realisation does with a clause, whether or not the clause describes an act: leave it unbreachable, record its breach against the agent in breach, or record the undertaking and who gave it. This depends on one thing: whether the act that names the agent answerable leaves the record in another agent's hands. Attestation and undertaking both require that of the schemas, and differ only in whether the breach is itself an act of the contract.

Both contracts are certified from their schemas. Who holds a record of whom is a directed graph, and an instance is a connected component of the undirected graph of interactions. In the social graph GSG the two coincide: befriending is the introductory act, and a friend edge can only have been added by the two agents it joins. A coin can be handed on and a message delivered to a group, so an edge may point at an agent its source has never transacted with; an edge is added either to a participant of the act that adds it or to an agent another participant of that act records already, and that condition on the schemas is enough for no edge ever to join two instances. For coins it is conservation of money.

The conditions also leave no participant without a role. Openness lets any agent start an instance, and without permission. Closure leaves an instance requiring no registry that issues identifiers, no log that orders acts, and no authority that timestamps them, each of which would take part in an act without a role in it. A community organised in this way requires no permission to form, no authority to operate, and no proprietor to govern it.

Three directions remain. The conditions are sufficient and not necessary, and a characterisation would say which contracts are grassroots. Which further invariants of a contract follow from its schemas, as conservation of money does, is open. And the four things a schema carries are present, under other names, in the contract specification languages of Section 10; defining a semantics for one of them in which each agent holds their own state would let the conditions be checked on contracts already written in it.

References

- [1] Shaun Azzopardi, Gordon J. Pace, Fernando Schapachnik, and Gerardo Schneider. 2016. Contract automata. *Artificial Intelligence and Law* 24, 3 (2016), 203–243. doi:10.1007/s10506-016-9185-2
- [2] Omri Ben-Shahar and Carl E. Schneider. 2014. *More Than You Wanted to Know: The Failure of Mandated Disclosure*. Princeton University Press, Princeton, NJ.
- [3] Yochai Benkler. 2006. *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. Yale University Press, New Haven, CT.
- [4] Guido Boella, Leendert van der Torre, and Harko Verhagen. 2006. Introduction to Normative Multiagent Systems. *Computational & Mathematical Organization Theory* 12, 2 (2006), 71–79. doi:10.1007/s10588-006-9537-7
- [5] Giulio Caldarelli. 2020. Understanding the Blockchain Oracle Problem: A Call for Action. *Information* 11, 11 (2020), 509. doi:10.3390/info11110509
- [6] Maria-Emilia Cambronero, Luis Llana, and Gordon J. Pace. 2017. Timed Contract Compliance Under Event Timing Uncertainty. In *Legal Knowledge and Information Systems: JURIX 2017 (Frontiers in Artificial Intelligence and Applications, Vol. 302)*. IOS Press, 33–38.
- [7] Luca Cardelli, Liav Orgad, Gal Shahaf, Ehud Shapiro, and Nimrod Talmon. 2020. Digital social contracts: A foundation for an egalitarian and just digital society. In *CEUR Proceedings of the First International Forum on Digital and Democracy*, Vol. 2781. CEUR-WS, 51–60. <https://ceur-ws.org/Vol-2781/paper5.pdf>
- [8] Stefan Chircop, Gordon J. Pace, and Gerardo Schneider. 2022. An Automata-Based Formalism for Normative Documents with Real-Time. In *Legal Knowledge and Information Systems: JURIX 2022 (Frontiers in Artificial Intelligence and Applications, Vol. 362)*. IOS Press, 158–163. doi:10.3233/FAIA220460

- [9] Christopher D. Clack, Vikram A. Bakshi, and Lee Braine. 2016. Smart Contract Templates: Foundations, Design Landscape and Research Directions. *arXiv:1608.00771* <https://arxiv.org/abs/1608.00771>
- [10] Silvia Crafa, Cosimo Laneve, and Giovanni Sartor. 2022. Pacta sunt servanda: Legal Contracts in Stipula. *Science of Computer Programming* (2022). <https://arxiv.org/abs/2110.11069> *arXiv:2110.11069*.
- [11] Primavera De Filippi, Chris Wray, and Giovanni Sileno. 2021. Smart contracts. *Internet Policy Review* 10, 2 (2021). doi:10.14763/2021.2.1549
- [12] Ohad Eitan, Idit Keidar, and Ehud Shapiro. 2026. Securing People and their Machines Against Major Faults (Full Version). *Proceedings of The 28th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2026)*, to appear (2026). <https://arxiv.org/abs/2607.02304> *arXiv:2607.02304*.
- [13] Marc Esteve, Juan A. Rodríguez-Aguilar, Carles Sierra, Pere Garcia, and Josep Lluís Arcos. 2001. On the Formal Specification of Electronic Institutions. In *Agent-Mediated Electronic Commerce (AMEC) (LNCS, Vol. 1991)*. Springer, 126–147.
- [14] European Parliament and Council. 2014. Regulation (EU) No 910/2014 on Electronic Identification and Trust Services for Electronic Transactions in the Internal Market (eIDAS). *Official Journal of the European Union*, L 257.
- [15] European Parliament and Council of the European Union. 2016. Regulation (EU) 2016/679 (General Data Protection Regulation). OJ L 119, 4.5.2016. Recital 26: anonymous information is information which does not relate to an identified or identifiable natural person; Art. 4(1) defines personal data..
- [16] Richard E. Fikes and Nils J. Nilsson. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2 (1971), 189–208.
- [17] Mark D. Flood and Oliver R. Goodenough. 2022. Contract as automaton: representing a simple financial agreement in computational form. *Artificial Intelligence and Law* 30, 3 (2022), 391–416. doi:10.1007/s10506-021-09300-9
- [18] Nicoletta Fornara, Francesco Viganò, Mario Verdicchio, and Marco Colombetti. 2008. Artificial institutions: a model of institutional reality for open multiagent systems. *Artificial Intelligence and Law* 16, 1 (2008), 89–105. doi:10.1007/s10506-007-9055-z
- [19] Martin Fowler. 2025. Understanding Spec-Driven-Development: Kiro, spec-kit, and Tessel. *MartinFowler.com* (15 October 2025). <https://martinfowler.com/articles/exploring-gen-ai/sdd-3-tools.html>
- [20] Free Software Foundation. 2007. GNU General Public License, Version 3. <https://www.gnu.org/licenses/gpl-3.0.html>.
- [21] Jonathan Gelati, Guido Governatori, Antonino Rotolo, and Giovanni Sartor. 2002. Declarative Power, Representation, and Mandate. A Formal Analysis. In *Legal Knowledge and Information Systems: JURIX 2002 (Frontiers in Artificial Intelligence and Applications)*, Trevor Bench-Capon, Aspasia Daskalopulu, and Radboud Winkels (Eds.). IOS Press, 41–52.
- [22] Jonathan Gelati, Antonino Rotolo, Giovanni Sartor, and Guido Governatori. 2004. Normative autonomy and normative co-ordination: Declarative power, representation, and mandate. *Artificial Intelligence and Law* 12, 1 (2004), 53–81. doi:10.1007/s10506-004-1922-2
- [23] GitHub. 2025. Spec Kit: Toolkit for Spec-Driven Development. GitHub repository. Available at <https://github.com/github/spec-kit>.
- [24] James Golike and Ehud Shapiro. 2026. Digital Speech Acts Retain Control of Copyright with People, Not Platforms. *arXiv preprint arXiv:2606.19263* (2026). <https://arxiv.org/abs/2606.19263>
- [25] Google. 2024. Dart Programming Language. <https://dart.dev/>
- [26] Guido Governatori. 2018. Practical Normative Reasoning with Defeasible Deontic Logic. In *Reasoning Web. Learning, Uncertainty, Streaming, and Scalability*. Lecture Notes in Computer Science, Vol. 11078. Springer, 1–25. doi:10.1007/978-3-030-00338-8_1
- [27] Guido Governatori, Florian Idelberger, Zoran Milosevic, Regis Riveret, Giovanni Sartor, and Xiwei Xu. 2018. On legal contracts, imperative and declarative smart contracts, and blockchain systems. *Artificial Intelligence and Law* 26, 4 (2018), 377–409. doi:10.1007/s10506-018-9223-3
- [28] Guido Governatori and Antonino Rotolo. 2004. Modelling Contracts Using RuleML. In *Legal Knowledge and Information Systems: JURIX 2004 (Frontiers in Artificial Intelligence and Applications)*, Thomas F. Gordon (Ed.). IOS Press, 141–150.
- [29] Ian Grigg. 2004. The Ricardian Contract. In *First IEEE Workshop on Electronic Contracting*. IEEE.
- [30] Davide Grossi, John-Jules Ch. Meyer, and Frank Dignum. 2005. Modal Logic Investigations in the Semantics of Counts-as. In *Proceedings of the 10th International Conference on Artificial Intelligence and Law (ICAIL '05)*. ACM, 1–9. doi:10.1145/1165485.1165487
- [31] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi. 2019. The consensus number of a cryptocurrency. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. 307–316. doi:10.1145/3293611.3331589
- [32] Jaap Hage. 2011. A model of juridical acts: part 1: the world of law. *Artificial Intelligence and Law* 19, 1 (2011), 23–48. doi:10.1007/s10506-011-9105-4

- [33] Charlotte Hess and Elinor Ostrom (Eds.). 2007. *Understanding Knowledge as a Commons: From Theory to Practice*. MIT Press, Cambridge, MA.
- [34] Andrew J. I. Jones and Marek Sergot. 1996. A Formal Characterisation of Institutionalised Power. *Logic Journal of the IGPL* 4, 3 (1996), 427–443. doi:10.1093/jigpal/4.3.427
- [35] Idit Keidar, Andrew Lewis-Pye, and Ehud Shapiro. 2025. Constitutional Consensus. *arXiv preprint arXiv:2505.19216* (2025). <https://arxiv.org/abs/2505.19216>
- [36] Idit Keidar and Ehud Shapiro. 2026. Volition Elicitation: Operational Semantics for People and Their Machines. arXiv:2607.14138. arXiv:2607.14138 [cs.DC] <https://arxiv.org/abs/2607.14138>
- [37] Nancy S. Kim. 2013. *Wrap Contracts: Foundations and Ramifications*. Oxford University Press, New York.
- [38] Lawrence Lessig. 1999. *Code and Other Laws of Cyberspace*. Basic Books, New York.
- [39] Andrew Lewis-Pye, Oded Naor, and Ehud Shapiro. 2023. Grassroots Flash: A Payment System for Grassroots Cryptocurrencies. *arXiv preprint arXiv:2309.13191* (2023). <https://arxiv.org/abs/2309.13191>
- [40] Andy Lewis-Pye and Ehud Shapiro. 2026. Volition-Guarded Multiagent Atomic Transactions: Describing People and their Machines. *arXiv preprint arXiv:2604.25596* (2026). <https://arxiv.org/abs/2604.25596>
- [41] Denis Merigoux, Nicolas Chataing, and Jonathan Protzenko. 2021. Catala: A Programming Language for the Law. *Proceedings of the ACM on Programming Languages* 5, ICFP (2021). doi:10.1145/3473582
- [42] Sanjay Modgil, Nir Oren, Noura Faci, Felipe Meneguzzi, Simon Miles, and Michael Luck. 2015. Monitoring compliance with E-contracts and norms. *Artificial Intelligence and Law* 23, 2 (2015), 161–196. doi:10.1007/s10506-015-9167-9
- [43] Ugo Montanari and Francesca Rossi. 1995. Contextual nets. *Acta Informatica* 32, 6 (1995), 545–596.
- [44] Elinor Ostrom. 1990. *Governing the Commons: The Evolution of Institutions for Collective Action*. Cambridge University Press, Cambridge.
- [45] Alireza Parvizimosaed, Sepehr Sharifi, Daniel Amyot, Luigi Logrippo, Marco Roveri, Aidin Rasti, Ali Roudak, and John Mylopoulos. 2022. Specification and Analysis of Legal Contracts with Symboleo. *Software and Systems Modeling* 21 (2022). doi:10.1007/s10270-022-01053-6
- [46] Pierre-Joseph Proudhon and John Beverley Robinson. 2004. *General idea of the revolution in the nineteenth century*. Courier Corporation. Original work published 1851.
- [47] Margaret Jane Radin. 2013. *Boilerplate: The Fine Print, Vanishing Rights, and the Rule of Law*. Princeton University Press, Princeton, NJ.
- [48] Ranjan Sapkota, Konstantinos I. Roumeliotis, and Manoj Karkee. 2025. Vibe Coding vs. Agentic Coding: Fundamentals and Practical Implications of Agentic AI. <https://arxiv.org/abs/2505.19443> arXiv:2505.19443.
- [49] Giovanni Sartor. 2006. Fundamental legal concepts: A formal and teleological characterisation. *Artificial Intelligence and Law* 14, 1 (2006), 101–142. doi:10.1007/s10506-006-9009-x
- [50] Fred B. Schneider. 2000. Enforceable Security Policies. *ACM Transactions on Information and System Security* 3, 1 (2000), 30–50.
- [51] Ehud Shapiro. 2021. Multiagent Transition Systems for Composing Fault-Resilient Protocol Stacks. *arXiv preprint arXiv:2112.13650* (2021). <https://arxiv.org/abs/2112.13650>
- [52] Ehud Shapiro. 2022. Grassroots Currencies: Turning Mutual Trust into Liquidity. *arXiv preprint arXiv:2202.05619* (2022). <https://arxiv.org/abs/2202.05619>
- [53] Ehud Shapiro. 2023. Grassroots Distributed Systems: Concept, Examples, Implementation and Applications (Brief Announcement). In *37th International Symposium on Distributed Computing (DISC 2023)*. (Extended version: arXiv:2301.04391). LIPICs, Italy, 47:1–47:7. <https://arxiv.org/abs/2301.04391>
- [54] Ehud Shapiro. 2023. Grassroots Social Networking: Serverless, Permissionless Protocols for Twitter/LinkedIn/WhatsApp. In *OASIS '23* (Rome, Italy). Association for Computing Machinery. doi:10.1145/3599696.3612898
- [55] Ehud Shapiro. 2024. A Grassroots Architecture to Supplant Global Digital Platforms by a Global Digital Democracy. arXiv:2404.13468, *Proceedings of DAWO'24* (2024). <https://arxiv.org/abs/2404.13468>
- [56] Ehud Shapiro. 2025. Characterising Global Platforms: Centralised, Decentralised, Federated, and Grassroots. *arXiv preprint arXiv:2511.03286* (2025). <https://arxiv.org/abs/2511.03286>
- [57] Ehud Shapiro. 2025. GLP: A Grassroots, Multiagent, Concurrent, Logic Programming Language for AI (full version). *arXiv preprint arXiv:2510.15747, Summary in Proc. of ICLP'26, EPTCS 450*, pp. 119–133 (2025). <https://arxiv.org/abs/2510.15747>
- [58] Ehud Shapiro. 2026. Child-Safe Social Networking: From a Grassroots Social Contract to an App. *In preparation* (2026).
- [59] Ehud Shapiro. 2026. Corraling AI Risks with Grassroots Social Contracts. *In preparation* (2026).
- [60] Ehud Shapiro. 2026. Grassroots Bonds: Financing by the People, for the People. arXiv:2603.13671. arXiv:2603.13671 <https://arxiv.org/abs/2603.13671>
- [61] Ehud Shapiro. 2026. Grassroots Platforms with Atomic Transactions: Social Graphs, Cryptocurrencies, and Democratic Federations. In *Proceedings of the 27th International Conference on Distributed Computing and Networking*. 71–81. doi:10.1145/3772290.3772309 arXiv preprint arXiv:2502.11299.

- [62] Ehud Shapiro. 2026. Implementing Grassroots Logic Programs with Multiagent Transition Systems and AI (Full Version). *arXiv:2602.06934. Summary to appear in Proc. of LOPSTR+PPDP’26* (2026). <https://arxiv.org/abs/2602.06934>
- [63] Ehud Shapiro. 2026. Types for Grassroots Logic Programs (Full version). *arXiv preprint arXiv:2601.17957* (2026). <https://arxiv.org/abs/2601.17957>
- [64] Sepehr Sharifi, Alireza Parvizimosaed, Daniel Amyot, Luigi Logrippo, and John Mylopoulos. 2020. Symboleo: Towards a Specification Language for Legal Contracts. In *IEEE 28th International Requirements Engineering Conference (RE)*. IEEE, 364–369. doi:10.1109/RE48521.2020.00049
- [65] Munindar P. Singh. 1999. An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts. *Artificial Intelligence and Law* 7 (1999), 97–113.
- [66] Munindar P. Singh. 2011. Information-Driven Interaction-Oriented Programming: BSPL, the Blindingly Simple Protocol Language. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*. International Foundation for Autonomous Agents and Multiagent Systems, Taipei, Taiwan.
- [67] Harry Surden. 2012. Computable Contracts. *UC Davis Law Review* 46 (2012), 629.
- [68] Dominic Tarr, Erick Lavoie, Aljoscha Meyer, and Christian Tschudin. 2019. Secure scuttlebutt: An identity-centric protocol for subjective and decentralized applications. In *Proceedings of the 6th ACM conference on information-centric networking*. 1–11. doi:10.1145/3357150.3357396
- [69] United Kingdom Parliament. 2000. Electronic Communications Act 2000. c. 7, s. 7, as amended by the Electronic Identification and Trust Services for Electronic Transactions Regulations 2016.
- [70] United States Congress. 2000. Electronic Signatures in Global and National Commerce Act. Public Law 106–229, 114 Stat. 464.
- [71] Seng Joe Watt, Oliver Goodenough, and Meng Weng Wong. 2023. Deontics and Time in Contracts: An Executable Semantics for the L4 DSL. In *Legal Knowledge and Information Systems: JURIX 2023 (Frontiers in Artificial Intelligence and Applications)*. IOS Press, 119–124. doi:10.3233/FAIA230954

A The Framework of Volition-Guarded Multiagent Atomic Transactions

This appendix states formally what Section 4 recalls in brief, so that the statements and proofs of Section 6 can be read without recourse to another paper. Everything in it is from [40], and the proofs of its results are there and are not repeated. It leaves out the volition-guarded if-then-else transactions of [40], which compiled schemas do not produce.

Earlier work introduced multiagent transition systems [51], grassroots protocols and platforms [53], and their definition via multiagent atomic transactions [61], which describe the behaviour of machines but not of the people operating them. A volition-guarded transaction is a machine transaction guarded by the volitions of some, all, or none of the agents whose machines participate; an agent’s volitional state is a set of equivalence classes of machine transactions they are willing their machine to take part in. The social graph illustrates the two extremes: befriending is guarded by both agents, while unfriending is guarded by either. An agent may freely change their volitional state; an equivalence class is removed from every agent’s volitional state when a machine transaction in that class is taken, the will having been fulfilled.

A.1 Agents, Machines and Volition-Guarded Transactions

We assume a countable, potentially infinite set of *agents* Π , an agent being a person operating a machine, but consider only finite subsets of it, so when referring to a particular set of agents $P \subset \Pi$ we assume P to be nonempty and finite. We use $\subset$ to denote the strict subset relation and $\subseteq$ when equality is also possible, and use $p \neq q \in P$ as a shorthand for $p \in P \wedge q \in P \wedge p \neq q$. As standard, we use S^P to denote the set of all total functions from P to S , and if $c \in S^P$ we use c_p to denote the value of c at $p \in P$.

Definition 35 (Machine State, Configuration, Transaction, Volition-Guarded Transaction [40]). Given an arbitrary set S of *machine states*, with a designated *initial state* $s_0 \in S$, and agents $Q \subset \Pi$, a *machine configuration* over Q is a member of S^Q , and a *machine transaction* over participants Q is a pair $c \rightarrow c' \in (S^Q)^2$ such that $c \neq c'$; it is *unary* if $|Q| = 1$. Given such a machine transaction t , a

volition-guarded multiagent atomic transaction over t —henceforth, *volition-guarded transaction*—is a pair (t, Q') where $Q' \subseteq Q$ are its *guards*.

Machine transactions are atomic and asynchronous [51]: they can be carried out by their participants at any time, regardless of the states of non-participants. Participants include both active agents, whose state changes, and stationary agents, whose state is a precondition but does not change. Volition-guarded transactions can be carried out only if their guards $Q' \subseteq Q$ are willing, and do not distinguish between agents that initiate a transaction and those willing to take part in it. When we say a transaction is guarded by $\{p, q\}$, both must be willing; when we say it is guarded by either p or q , we mean there are two volition-guarded transactions over the same machine transaction, $(t, \{p\})$ and $(t, \{q\})$, so that either agent's volition suffices.

Distinct machine transactions can represent the same act in different configurations; a transaction equivalence relates them.

Definition 36 (Transaction Equivalence [40]). Given a set of machine transactions R , a *transaction equivalence* is an equivalence relation $\sim$ on R such that $t \sim t'$ implies t and t' have the same participants, and $R/\sim$ is countable. We write $[t]$ for the equivalence class of t under $\sim$.

All befriend transactions of p and q , differing only in the configurations in which they occur, form one equivalence class. An agent wills a class, and taking one member of it fulfils that will.

Definition 37 (Agent State and Configuration [40]). Given agents P , states S with initial state s_0 , a set of machine transactions T each over its own participants $Q \subseteq P$ and S , and equivalence $\sim$ on T , an *agent state* is a pair $(V, m) \in \mathcal{A} = (2^{T/\sim} \times S)$ where V is its *volitional state* and $m \in S$ its *machine state*. The *initial agent state* is $(\emptyset, s_0)$. An *agent configuration* c over P, S, T , and $\sim$ is a member $c \in \mathcal{A}^P$ in which $c_p^v \subseteq (T/\sim)_p$ for every $p \in P$, where $(T/\sim)_p$ denotes the classes in $T/\sim$ in which p is a participant; we write c_p^v for the volitional state and c_p^m for the machine state of agent p in c .

Definition 38 (Volitional Transaction [40]). Given agents P , states S , machine transactions T over P and S , and equivalence $\sim$ on T :

- (1) A *volition change of agent* $p \in P$ is a pair $c \rightarrow c'$ of agent configurations over $\{p\}, S, T$, and $\sim$ such that $c_p^v, c_p'^v \subseteq (T/\sim)_p$ and $c_p^v \neq c_p'^v$, and $c_p^m = c_p'^m$.
- (2) A *volitional machine transaction* induced by a volition-guarded transaction (t, Q') , for some $t = (d \rightarrow d') \in T$ over $Q \subseteq P$ with guards $Q' \subseteq Q$, is a pair $c \rightarrow c'$ where $c \neq c'$ are agent configurations over P, S, T , and $\sim$ such that $[t] \in c_q^v$ for every $q \in Q'$; $c_p^m = d_p$ and $c_p'^m = d_p'$ for every $p \in Q$; $c_p^m = c_p'^m$ for every $p \in P \setminus Q$; and $c_p'^v = c_p^v \setminus \{[t]\}$ for every $p \in P$.
- (3) A *volitional transaction* is a volition change or a volitional machine transaction.

When a volitional machine transaction induced by (t, Q') is taken, the class $[t]$ is removed from every agent's volitional state: the will is fulfilled by any equivalent transaction. An agent may independently change their volitional state via volition changes, which may add or remove classes; beyond these, the framework removes a class from c_p^v only upon fulfilment.

A.2 Transition Systems and the Induced Volitional System

Definition 39 (Transition System, Computation, Run, Safe, Live, Correct [40]). A *transition system* is a tuple $TS = (S, s_0, T, \sim)$, where:

- (1) S is an arbitrary non-empty set, referred to as the set of *states*.
- (2) Some $s_0 \in S$ is the designated *initial state*.
- (3) $T \subseteq S^2$ is a set of *correct transitions* over S , where each transition $t \in T$ is a pair (s, s') of non-identical states $s \neq s' \in S$, also written as $t = s \rightarrow s'$.

- (4) $\sim$ is a *partial equivalence relation* on T : a symmetric and transitive relation on T , not necessarily reflexive. Its domain $\{t \in T : t \sim t\}$ is partitioned into countably many *liveness classes* $T/\sim$; a transition outside the domain belongs to no class.

A *computation* of TS is a nonempty, potentially infinite sequence of states $r = s_1, s_2, \dots$; it is a *run* of TS if $s_1 = s_0$. A *prefix* of a computation is a finite initial segment of it. A computation $r = s_1, s_2, \dots$ is *safe*, also written $r \subseteq T$, if $s_i \rightarrow s_{i+1} \in T$ for every two consecutive states. A class $[t] \in T/\sim$ is *enabled* in a state s if $s \rightarrow s' \in [t]$ for some $s' \in S$. A run r is *live* if no class $[t] \in T/\sim$ is enabled in every state of some suffix of r with no member of $[t]$ occurring in the suffix. A run is *correct* if it is safe and live.

A partial equivalence, rather than a total one, is used so that some transitions may carry no liveness requirement: only transitions in the domain of $\sim$ form classes and thereby incur a liveness obligation, while transitions outside the domain, belonging to no class, may occur in a correct run but are never required to.

LEMMA 40 (EXTENSION [40]). *Every finite safe run of a transition system is a prefix of a correct run of it.*

Definition 41 (Multiagent Transition System [40]). Given agents $P \subset \Pi$ and an arbitrary set S of states with a designated initial state $s_0 \in S$, a *multiagent transition system* over P and S is a transition system $TS = (C, c_0, T, \sim)$ with configurations $C := S^P$, initial configuration $c_0 := \{s_0\}^P$, transitions $T \subseteq C^2$ a set of transactions over P and S , and $\sim$ a partial equivalence on T .

Rather than specifying a multiagent transition system over a set of agents P directly, it is specified via machine transactions. A machine transaction over $Q \subseteq P$ defines a set of multiagent transitions over P in which all members of $P \setminus Q$ are stationary.

Definition 42 (Transaction Closure [40]). Let $P \subset \Pi$, S a set of machine states, and $C := S^P$. For any transition or transaction $t = c \rightarrow c'$, we write $t_q := c_q \rightarrow c'_q$ and say p is *stationary* in t if $c_p = c'_p$. For a machine transaction $t = (c \rightarrow c')$ over S with participants Q , the *P -closure* of t , $t \uparrow P$, is the set of transitions over P and S defined by:

$$t \uparrow P := \begin{cases} \{t' \in C^2 : \forall q \in Q. (t_q = t'_q) \wedge \forall p \in P \setminus Q. (p \text{ is stationary in } t')\} & \text{if } Q \subseteq P \\ \emptyset & \text{otherwise} \end{cases}$$

If R is a set of machine transactions, each $t \in R$ over some Q and S , then the *P -closure* of R , $R \uparrow P$, is the set of transitions over P and S defined by $R \uparrow P := \bigcup_{t \in R} t \uparrow P$. Given a relation $\sim$ on R , its *P -closure* $\sim \uparrow P$ is the relation on $R \uparrow P$ with $\hat{t} (\sim \uparrow P) \hat{t}'$ iff $\hat{t} \in t \uparrow P$ and $\hat{t}' \in t' \uparrow P$ for some $t \sim t'$. A transition over P is *unary* if it lies in the P -closure of a unary transaction.

If distinct transactions in R have disjoint P -closures, as when every participant of every transaction in R changes state, then each transition in $R \uparrow P$ has a unique inducing transaction, $\sim \uparrow P$ relates two transitions exactly when their inducing transactions are $\sim$ -related, and $\sim \uparrow P$ is a partial equivalence whenever $\sim$ is. For volition-guarded transactions, disjointness is required only of unguarded ones of different classes (Definition 43).

Definition 43 (Volitional Multiagent Transition System [40]). Given agents $P \subset \Pi$, machine states S with initial state s_0 , a set R of volition-guarded transactions such that every $(t, Q') \in R$ has the participants of t contained in P , and an equivalence $\sim$ on the set $T_R := \{t : (t, Q') \in R \text{ for some } Q'\}$ of underlying machine transactions such that two of them of different classes, each underlying an unguarded transaction of R , have disjoint P -closures, the *volitional multiagent transition system induced by* $(S, R, \sim)$ *over* P is the multiagent transition system $(\mathcal{A}^P, c_0, T_V, \sim_V)$ where:

- (1) $\mathcal{A} := 2^{T_R/\sim} \times S$ is the *agent state space*;
- (2) $c_0 \in \mathcal{A}^P$ is the *initial agent configuration*, with $c_{0p}^v = \emptyset$ and $c_{0p}^m = s_0$ for every $p \in P$;
- (3) T_V consists of all transitions $e \rightarrow e' \in (\mathcal{A}^P)^2$ of one of two forms: (i) a *volition change* of some $p \in P - e_p^v, e_p'^v \subseteq (T_R/\sim)_p$ and $e_p^v \neq e_p'^v, e_p^m = e_p'^m$, and $e_r = e_r'$ for every $r \in P \setminus \{p\}$; or (ii) a *volitional machine transaction* induced by some volition-guarded transaction $(t, Q') \in R$ per Definition 38(2);
- (4) $\sim_V$ is the restriction of $\sim \uparrow P$ (Definition 42) to the volitional machine transactions, relating two of them whenever their inducing machine transactions are $\sim$ -equivalent, and leaves every volition-change transition outside its domain, so volition changes belong to no class and carry no liveness obligation.

Every volitional machine transaction has $\sim$ -equivalent inducing machine transactions, so $\sim_V$ is a partial equivalence. A step induced by a guarded transaction (t, Q') , $Q' \neq \emptyset$, removes $[t]$ from the volitional state of each guard, which held it, and a step of another class removes another class, so two guarded transactions of different classes may share their P -closures; two unguarded transactions of different classes have no transition in common, by the condition on P -closures.

Definition 44 (Enabled [40]). Given a set of volition-guarded transactions, each (t, Q') with $t = d \rightarrow d'$ a machine transaction over some $Q \subseteq P$ and S with guards $Q' \subseteq Q$, and an equivalence $\sim$ on machine transactions: the volition-guarded transaction (t, Q') is *machine-enabled* in agent configuration c over P if $c_p^m = d_p$ for every $p \in Q$, and *enabled* if in addition $[t] \in c_q^v$ for every $q \in Q'$. An equivalence class $[t]$ is *enabled* in c if some volition-guarded transaction (t', Q') with $t' \in [t]$ is enabled in c .

A volition-guarded transaction with an empty guard requires no volitions and is enabled whenever its machine precondition is met. A class of volitional machine transactions is enabled at a configuration in the sense of Definition 39 exactly when some volition-guarded transaction inducing it is enabled in the above sense, and this determines which runs are live and correct. Volition changes belong to no class and so impose no liveness obligation; personal choices remain free.

A.3 Protocols, Grassroots Protocols and Coalescence

A protocol is a family of multiagent transition systems, one for each set of agents $P \subset \Pi$, which share an underlying set of local machine states $\mathcal{S}$ with a designated initial state s_0 . A *local-states function* maps every set of agents $P \subset \Pi$ to a set of local machine states $S(P) \subset \mathcal{S}$ that includes s_0 and satisfies $P \subset P' \subset \Pi \implies S(P) \subset S(P')$.

Definition 45 (Protocol [40]). A protocol $\mathcal{F}$ over a local-states function S is a family of multiagent transition systems that has exactly one transition system $\mathcal{F}(P) = (C(P), c_0(P), T(P), \sim(P))$ for every $P \subset \Pi$, with *agent states* $\mathcal{A}(P)$, configurations $C(P) := \mathcal{A}(P)^P$, initial configuration $c_0(P) \in C(P)$, and partial equivalence $\sim(P)$ on $T(P)$ determined by the protocol, such that $P \subseteq P' \subset \Pi$ implies $\mathcal{A}(P) \subseteq \mathcal{A}(P')$ and $c_0(P)_p = c_0(P')_p$ for every $p \in P$.

In a grassroots protocol two disjoint sets of agents can each operate independently — their interleaved correct runs are correct runs of the combined system — yet the combined system offers behaviours that neither set could produce by itself.

Definition 46 (Interleaving [40]). Let $P, P' \subset \Pi$ be disjoint nonempty sets of agents, $r = c_0, c_1, \dots$ a run of $\mathcal{F}(P)$, and $r' = d_0, d_1, \dots$ a run of $\mathcal{F}(P')$. An *interleaving* of r and r' is a sequence $e_0, e_1, \dots$ of configurations in $C(P \cup P')$ for which there exist non-decreasing sequences of indices $(i_k)_{k \geq 0}$ and $(j_k)_{k \geq 0}$ with $i_0 = j_0 = 0$ such that for every $k \geq 0$:

- (1) $(e_k)_p = (c_{i_k})_p$ for every $p \in P$,

- (2) $(e_k)_q = (d_{j_k})_q$ for every $q \in P'$,
- (3) if e_{k+1} exists, then exactly one of: (i) $i_{k+1} = i_k + 1$ and $j_{k+1} = j_k$, a P -step, or (ii) $i_{k+1} = i_k$ and $j_{k+1} = j_k + 1$, a P' -step.

Moreover, if r is finite of length n then $i_k = n$ for some k , and if r is infinite then for every $m \geq 0$ there is a k with $i_k = m$; likewise for r' and (j_k) .

An interleaving is well defined: by Definition 45, $\mathcal{A}(P) \subseteq \mathcal{A}(P \cup P')$ and $\mathcal{A}(P') \subseteq \mathcal{A}(P \cup P')$, so each e_k , with p -components in $\mathcal{A}(P)$ and q -components in $\mathcal{A}(P')$, is a configuration in $C(P \cup P')$. Also $e_0 = c_0(P \cup P')$, by the agreement of initial configurations across $\mathcal{F}(P)$, $\mathcal{F}(P')$ and $\mathcal{F}(P \cup P')$.

Definition 47 (Interaction, Interactive Run, First Interaction [40]). Let $Q \subset \Pi$. For a configuration c over Q and $Q' \subseteq Q$, write $c|_{Q'}$ for the restriction of c to Q' . A transition $e \rightarrow e'$ of $\mathcal{F}(Q)$ is an *interaction between* $x \neq y \in Q$ if $e|_{\{x\}} \neq e'|_{\{x\}}$ and $e|_{\{y\}} \neq e'|_{\{y\}}$, and $e|_{\{x\}} \rightarrow e'|_{\{x\}}$ is not a transition of $\mathcal{F}(\{x\})$ or $e|_{\{y\}} \rightarrow e'|_{\{y\}}$ is not a transition of $\mathcal{F}(\{y\})$. It is an *interaction between* disjoint nonempty $P, P' \subseteq Q$ if it is an interaction between some $x \in P$ and some $y \in P'$. A run $\hat{r}$ of $\mathcal{F}(Q)$ is *interactive between* P and P' if some transition of $\hat{r}$ is an interaction between them; the *first interaction* of P and P' in $\hat{r}$ is the first such transition.

Definition 48 (Oblivious, Interactive, Grassroots [40]). A protocol $\mathcal{F}$ is:

- (1) *oblivious* if for every disjoint nonempty $P, P' \subset \Pi$, every interleaving of a correct run of $\mathcal{F}(P)$ and a correct run of $\mathcal{F}(P')$ is a correct run of $\mathcal{F}(P \cup P')$.
- (2) *interactive* if for every $Q \subset \Pi$ and disjoint nonempty $P, P' \subseteq Q$, every prefix of a correct run of $\mathcal{F}(Q)$ containing no interaction between P and P' is a prefix of a correct run of $\mathcal{F}(Q)$ that contains one.
- (3) *grassroots* if it is oblivious and interactive.

Being oblivious means two disjoint sets of agents coexist without interference. Being interactive means two sets that have not yet interacted can still do so, and the condition is required at every such point and not at the initial configuration only, so a protocol whose sets of agents can reach a state from which they can never couple is not interactive.

Definition 49 (Interaction Graph, Instance, Coalescence [40]). Let $\mathcal{F}$ be a protocol, $Q \subset \Pi$, and r a prefix of a run of $\mathcal{F}(Q)$. The *interaction graph* of r has vertices Q and an edge between $a \neq b \in Q$ whenever some transition of r is an interaction between a and b . A nonempty $P \subseteq Q$ is an *instance* in r if it is a connected component of the interaction graph of r . Disjoint nonempty $P, P' \subseteq Q$ *coalesce* in a run $\hat{r}$ of $\mathcal{F}(Q)$ if they are instances in some prefix of $\hat{r}$ and lie in one component of the interaction graph of a longer prefix of $\hat{r}$.

PROPOSITION 50 (COALESCENCE [40]). *Let $\mathcal{F}$ be a grassroots protocol, $Q \subset \Pi$, and $P, P' \subseteq Q$ disjoint instances in a prefix r of a correct run of $\mathcal{F}(Q)$. Then r is a prefix of a correct run of $\mathcal{F}(Q)$ in which P and P' coalesce.*

A.4 Volitional Transactions-Based Protocols

A volitional transactions-based protocol assigns to each set of agents the volitional multiagent transition system induced by its volition-guarded transactions.

Definition 51 (Transactions Over a Local-States Function [40]). Let S be a local-states function. A set of transactions R is *over* S if every transaction $t \in R$ is a multiagent transition over Q and $S(P')$ for some $Q \subseteq P' \subset \Pi$. Given such a set R and $P \subset \Pi$, $R(P) := \{t \in R : t \text{ is over } Q \text{ and } S(P'), Q \subseteq P' \subseteq P\}$.

Definition 52 (Volitional Transactions-Based Protocol [40]). Let S be a local-states function and R a set of volition-guarded transactions over S with equivalence $\sim$. The *protocol* $\mathcal{F}$ over R , S , and $\sim$ assigns to each set of agents $P \subset \Pi$ the volitional multiagent transition system $\mathcal{F}(P)$ induced by $(S(P), R(P), \sim)$ over P (Definition 43). In particular, $C(P) = \mathcal{A}(P)^P$ with agent state space $\mathcal{A}(P) := 2^{T_{R(P)}/\sim} \times S(P)$, monotone in P as Definition 45 requires, and $c_0(P)$ has p -component $(\emptyset, s_0)$ for every $p \in P$.

Throughout, $\sim$ is an equivalence on T_R , the machine transactions underlying the whole set R ; for $T' \subseteq T_R$ we write $T'/\sim$ for the set of $\sim$ -classes with a representative in T' , and $[t]$ for the class of t in $T_R/\sim$. Hence $P \subseteq P'$ implies $T_{R(P)}/\sim \subseteq T_{R(P')}/\sim$.

Definition 53 (Contained [40]). A set R of volition-guarded transactions over a local-states function S is *contained* if for every $P \subset \Pi$, every $(t, Q') \in R$ with $t = c \rightarrow c'$ over participants $Q \subseteq P$ and $c \in S(P)^Q$ is in $R(P)$.

The three results below are used in the proofs of Section 6. The first is the invariant that in $\mathcal{F}(P)$ an agent will only transactions over agents of P , so the guard of a transaction that reaches outside P cannot will it while P runs by itself.

LEMMA 54 (VOLITIONAL CONTAINMENT [40]). *Let $\mathcal{F}$ be a volitional transactions-based protocol over a set of volition-guarded transactions R with equivalence $\sim$. For every $P \subset \Pi$, every configuration c of $\mathcal{F}(P)$, and every $p \in P$: $c_p^0 \subseteq T_{R(P)}/\sim$, where $T_{R(P)} := \{t : (t, Q') \in R(P) \text{ for some } Q'\}$.*

LEMMA 55 (INTERLEAVING SAFETY [40]). *Let $\mathcal{F}$ be a volitional transactions-based protocol and $P, P' \subset \Pi$ disjoint and nonempty. Every finite prefix of an interleaving of a correct run of $\mathcal{F}(P)$ and a correct run of $\mathcal{F}(P')$ is a finite safe run of $\mathcal{F}(P \cup P')$.*

PROPOSITION 56 ([40]). *A volitional transactions-based protocol over a contained set of volition-guarded transactions is oblivious provided that for every disjoint nonempty $P, P' \subset \Pi$, no equivalence class whose transactions have participants spanning both P and P' is ever enabled in any interleaving of correct runs of $\mathcal{F}(P)$ and $\mathcal{F}(P')$.*

A person may die or become incapacitated, and an AI agent operating a machine may be retired; their will then has no further effect. An agent has left when no act they guard can be enabled again.

Definition 57 (Left [40]). Let $\mathcal{F}$ be a volitional transactions-based protocol over R , $Q \subset \Pi$, c a configuration of $\mathcal{F}(Q)$, and $x \in Q$. Then x *has left* in c if no $(t, Q') \in R(Q)$ with $x \in Q'$ is machine-enabled in any configuration c' of $\mathcal{F}(Q)$ with $c \xrightarrow{*} c' \subseteq T(Q)$.

Definition 58 (Interactive Among the Agents That Have Not Left [40]). A volitional transactions-based protocol $\mathcal{F}$ is *interactive among the agents that have not left* if for every $Q \subset \Pi$ and disjoint nonempty $P, P' \subseteq Q$, every prefix of a correct run of $\mathcal{F}(Q)$ containing no interaction between P and P' , in whose last configuration some agent of P and some agent of P' have not left, is a prefix of a correct run of $\mathcal{F}(Q)$ that contains one.

An interactive protocol is interactive among the agents that have not left, the latter requiring a continuation of fewer prefixes.

In a run of $\mathcal{F}(P \cup P')$ an agent of P may will a transaction reaching P' , which a run of $\mathcal{F}(P)$ forbids; the volitional notion constrains the resulting coupling.

Definition 59 (Volitionally Grassroots [40]). A volitional transactions-based protocol $\mathcal{F}$ is *volitionally grassroots* if it is oblivious, interactive among the agents that have not left, and, for every disjoint nonempty $P, P' \subset \Pi$ and every safe run of $\mathcal{F}(P \cup P')$ interactive between P and P' , the first interaction of P and P' is induced by a volition-guarded transaction (t, Q') with $Q' \cap P \neq \emptyset$ and $Q' \cap P' \neq \emptyset$.

A grassroots protocol meeting the condition on first interactions is volitionally grassroots.

B Proofs

The statements of Sections 5, 6, 7, 8 and 9 are restated and proved here in order, with the three lemmas and the one definition used only in these proofs. Throughout, C is a syntactically grassroots contract, $\mathcal{F}$ the protocol realising it and $P \subset \Pi$.

B.1 Compiling Schemas into Transactions

LEMMA 15 (THE COMPILATION IS WELL DEFINED). *R_C with $\sim$ is well formed.*

PROOF. Let $d \rightarrow d'$ be determined by τ at β over Q , and let P' be Q together with the agents β assigns to the agent variables of τ , the agents among the arguments of the atoms of d , the signers of the speech acts among them and the agents of the community names among them, and the signers of the speech acts β assigns. Every atom of every d'_p is an atom of d_p or the image under β of an atom added at a role, and the arguments of either lie in $P' \cup D_{P'} \cup \mathcal{N}_{P'}$, so $d \rightarrow d'$ is a machine transaction over Q and $S_C(P')$.

Let a machine transaction underlie an unguarded transaction of R_C , determined by a schema with no guarding role. By Definitions 3 and 13 the state of every participant of it changes and no other state changes, so its participants are exactly the agents that are not stationary in any transition of its P -closure. A transition lying in the closures of two such machine transactions therefore forces the two to have the same participants and the same restriction to them, hence to be equal, so two of them of different classes have disjoint closures, which is what Definition 43 requires of R_C .

Two machine transactions related by the relation generating $\sim$ have the same participants by Definition 14, and so do two related by $\sim$. A class over P is a union of sets of transactions each determined by a schema at a binding sending its agent roles and agent variables into P and its speech-act variables into D ; a contract has finitely many schemas, each has finitely many such bindings of its agent roles and agent variables, and D is countable, Δ and Π being countable, so the classes over P are countably many, as Definition 36 requires. $\square$

LEMMA 16 (CONTAINMENT). *R_C is contained.*

PROOF. Let $(t, Q') \in R_C$, with $t = d \rightarrow d'$ determined by a schema τ at an admissible binding β over participants $Q \subseteq P$ and $d \in S_C(P)^Q$. An atom of d'_p , $p \in Q$, is an atom of d_p or $\beta(\varphi)$ for an atom φ added at a role, and it suffices that every agent $\beta(\varphi)$ names is in P (Definition 1). Such an agent is $\beta(u)$ for u the base of an argument of φ that is an agent role, an agent variable or a name term, or the signer of $\beta(y)$ for a speech-act variable y that is an argument of φ or occurs in a name term among its arguments. By Definition 8, the set of predicates of C having traceable provenance, u is the base of a role of τ , and then $\beta(u)$ is the agent in an agent role of τ , an agent of Q ; or u is the base of an argument of an atom ψ required at an agent role π_j , and then $\beta(\psi)$ is an atom of $d_{\beta(\pi_j)} \in S_C(P)$ naming $\beta(u)$, so $\beta(u) \in P$. Likewise y is an argument of such a ψ or occurs in a name term among its arguments, and then the signer of $\beta(y)$ is in P , or y occurs in no atom required at a role, and then $\beta(y)$ is signed by the agent in a guarding role (Definition 4), an agent of Q . So $d'_p \in S_C(P)$ for every $p \in Q$, and $(t, Q') \in R_C(P)$. $\square$

The proof of Lemma 16 uses traceable provenance and no other condition of Definition 10.

LEMMA 60 (EVERY PARTICIPANT CHANGES STATE). *Every transition of $\mathcal{F}(P)$ induced by a transaction of R_C changes the state of every participant of that transaction.*

PROOF. Let the transaction be (t, Q') , with t determined by a schema τ at a binding β , and let p be a participant of t , so p is in the extent of a role of τ (Definition 13). If one such role is not

guarding, as no listed role is, the machine state of p changes by Definitions 3 and 13. Otherwise $p \in Q'$, and the transition is taken where (t, Q') is enabled, so $[t]$ is in the volitional state of p there, and the transition removes it (Definition 38). $\square$

PROPOSITION 17 (EVERY PARTICIPANT TAKES A ROLE). *Every agent whose machine state a transition of $\mathcal{F}(P)$ changes is in the extent of a role of a schema of C at some binding.*

PROOF. A volition change leaves every machine state unchanged (Definition 38), so a transition changing one is a volitional machine transaction induced by some $(t, Q') \in R_C(P)$, and by that definition it leaves the machine state of every agent outside the participants of t unchanged. By Definition 14 t is determined by a schema τ of C at a binding β , and by Definition 13 the participants of t are the agents in the extents of the roles of τ . $\square$

PROPOSITION 18 (ANONYMITY). *For every permutation σ of Π , $\sigma(R_C) = R_C$, $\sigma(c_0(P)) = c_0(\sigma(P))$, and σ carries the runs of $\mathcal{F}(P)$ to the runs of $\mathcal{F}(\sigma(P))$.*

PROOF. σ maps $\Sigma[P \cup D_P \cup \mathcal{N}_P]$ bijectively onto $\Sigma[\sigma(P) \cup D_{\sigma(P)} \cup \mathcal{N}_{\sigma(P)}]$, carrying signers and the agents of names with it, hence $S_C(P)$ onto $S_C(\sigma(P))$, and fixes $s_0 = \emptyset$; so $\sigma(c_0(P)) = c_0(\sigma(P))$. Let t be determined by a schema τ at a binding β . Then $\sigma \circ \beta$ is again a binding for τ , sending two agent roles to one agent exactly when β does and leaving Y mapped into D , and it is admissible, σ carrying the signer of $\beta(y)$ to the agent in the same guarding role under $\sigma \circ \beta$. Each clause of Definition 13 is an inclusion, a disjointness or an identity between images under β and the local states of d and d' , σ carries the extent of a role at β in d to its extent at $\sigma \circ \beta$ in $\sigma(d)$ and a name in no atom of d to one in no atom of $\sigma(d)$, and σ is a bijection on atoms, so applying it to both sides of each shows $\sigma(t)$ determined by τ at $\sigma \circ \beta$. The guarding roles of τ do not depend on the binding, so σ carries the volition-guarded transaction at β to the one at $\sigma \circ \beta$; by Definition 14 $\sigma(R_C) \subseteq R_C$, and applying σ^{-1} proves the reverse inclusion. Two members of R_C are $\sim$ -related exactly when one schema at one binding determines both, a condition σ preserves, and $R_C(P)$ is carried to $R_C(\sigma(P))$ because σ carries $S_C(P')$ to $S_C(\sigma(P'))$ for every $P' \subseteq P$. The volitional construction and the P -closure are defined from $R_C(P)$, $\sim$ and $c_0(P)$ only, so σ carries $\mathcal{F}(P)$ to $\mathcal{F}(\sigma(P))$ and runs to runs. $\square$

COROLLARY 19 (NO PRIVILEGED AGENT). *If a configuration is reachable in a run over P , so is its image under every permutation of P .*

PROOF. Extend the permutation of P by the identity on $\Pi \setminus P$ and apply Proposition 18, which carries the run to a run over P ending in the image configuration. $\square$

B.2 Polite Sets of Volition-Guarded Transactions

Throughout this subsection, S is a local-states function, R a set of volition-guarded transactions over S with equivalence $\sim$, and $\mathcal{F}$ the protocol over R , S and $\sim$, as in Section 6.

Let $Q \subset \Pi$.

LEMMA 61 (A MUTUAL RECORDING IS AN INTERACTION). *Every transition of $\mathcal{F}(Q)$ induced by a mutual recording of p and q in $R(Q)$ is an interaction between p and q .*

By Definition 47 it is then an interaction between any two disjoint sets of agents one of which holds p and the other q .

PROOF. The machine states of p and of q both change, by Definition 22. Since d'_p records q , $d'_p \notin S(P)$ for every P with $q \notin P$, and in particular $d'_p \notin S(\{p\})$; the machine states of every configuration of $\mathcal{F}(\{p\})$ lie in $S(\{p\})$ by Definition 52, so the restriction of the transition to p is

no transition of $\mathcal{F}(\{p\})$. By Definition 47 the transition is an interaction between p and q , hence between any two disjoint sets of agents one of which holds p and the other q . $\square$

THEOREM 24 (POLITE). *The protocol over a contained, polite set of volition-guarded transactions is grassroots and volitionally grassroots.*

PROOF. Let R be polite. One fact is used throughout: an agent whose state a volitional machine transaction changes is a participant of the transaction inducing it, as only a participant changes machine state and only a participant holds its class in their volitional state (Definitions 37 and 38).

Oblivious. Let $P, P' \subset \Pi$ be disjoint and nonempty, r a correct run of $\mathcal{F}(P)$, r' a correct run of $\mathcal{F}(P')$, and $e = e_0, e_1, \dots$ an interleaving of r and r' , with index sequences (i_k) and (j_k) . By Lemma 55, every finite prefix of e is a finite safe run of $\mathcal{F}(P \cup P')$. Every transition of e is a P -step or a P' -step and so alters the state of agents of one of the two sets only, so no transition of e is an interaction between an agent of P and an agent of P' , and no instance in a prefix of e holds agents of both.

R is contained, and we verify the other hypothesis of Proposition 56. Let $[t]$ be a class of $T_{R(P \cup P')}/\sim$ whose transactions have participants Q meeting both P and P' —the same Q for every member of the class, by Definition 36—let $(t'', Q'') \in R$ with $t'' \in [t]$, and take $a \in Q \cap P$. Suppose (t'', Q'') is enabled at some e_k . Its participants Q meet both sets and no instance in the prefix of e ending at e_k holds agents of both, so they do not all lie in one instance in that prefix; by closure the instance holding a there holds some $b \in Q''$, and $b \in P$ as that instance holds no agent of P' . Since Q meets P' , $t'' \notin R(P)$, so $[t] \notin T_{R(P)}/\sim$; and by Lemma 54 applied to r , $e_{k_b}^v = (c_{i_k})_b^v \subseteq T_{R(P)}/\sim$. Hence $[t] \notin e_{k_b}^v$ and the guard on b fails, so (t'', Q'') is not enabled at e_k —a contradiction.

So no class whose transactions have participants in both sets is ever enabled in e , and by Proposition 56, $\mathcal{F}$ is oblivious.

Interactive. Let $Q \subset \Pi$, let $P, P' \subseteq Q$ be disjoint and nonempty, let $\hat{r}$ be a prefix of a correct run of $\mathcal{F}(Q)$ containing no interaction between P and P' , and take $p \in P$ and $q \in P'$. An interaction between p and q is one between P and P' by Definition 47, so $\hat{r}$ contains none, and by openness some finite safe extension of $\hat{r}$ by volition changes and transitions that are not interactions between p and q ends where some $(t, Q') \in R(Q)$ whose induced transitions are interactions between p and q is machine-enabled. If a transition of that extension is an interaction between P and P' , the extension is a finite safe run having $\hat{r}$ as a prefix and containing one, and by Lemma 40 a prefix of a correct run of $\mathcal{F}(Q)$; so suppose none is.

Extend further by a volition change of each agent of Q' , where needed, so that $[t]$ lies in the volitional state of each; these leave every machine state unchanged, so (t, Q') is now enabled. Extend by the volitional machine transaction it induces, an interaction between p and q and hence between P and P' . The result is a finite safe run of $\mathcal{F}(Q)$ having $\hat{r}$ as a prefix and containing an interaction between P and P' , and by Lemma 40 it is a prefix of a correct run of $\mathcal{F}(Q)$. Hence $\mathcal{F}$ is interactive, and with obliviousness, grassroots.

Volitionally grassroots. Let $P, P' \subset \Pi$ be disjoint and nonempty, let $\hat{r}$ be a safe run of $\mathcal{F}(P \cup P')$ interactive between P and P' , and let $e \rightarrow e'$ be its first interaction. It changes the state of an agent of P and of an agent of P' , and a volition change alters one agent only, so it is a volitional machine transaction induced by some $(t, Q') \in R(P \cup P')$ whose participants Q meet both sets; take $a \in Q \cap P$ and $b \in Q \cap P'$.

The participants of t do not all lie in one instance in the prefix ρ of $\hat{r}$ ending at e : were they, a and b would lie in one connected component of the interaction graph of ρ ; every agent of ρ is in $P \cup P'$, so a path from a to b in that component has an edge between some $x \in P$ and some $y \in P'$, and the transition of ρ that puts that edge there is an interaction between x and y , hence between

P and P' , and it precedes $e \rightarrow e'$ —contradicting that $e \rightarrow e'$ is the first. The same argument shows that every instance in ρ lies wholly in P or wholly in P' , so the instance of a and the instance of b are distinct. As (t, Q') is enabled at e , by closure each of the two holds an agent of Q' , the first in P and the second in P' , so Q' meets both. $\mathcal{F}$ is oblivious and interactive, so it is grassroots, and interactive among the agents that have not left, so it is volitionally grassroots (Definition 59). $\square$

B.3 Conditions Sufficient for Openness and Closure

LEMMA 25 (RECORDS). *On local states, Definitions 20 and 5 agree.*

PROOF. Suppose an atom of s names q , as Definition 5 requires, and let $P \subset \Pi$ with $q \notin P$. That atom has an argument in $\Pi \setminus P$, a speech act among its arguments whose signer is in $\Pi \setminus P$, or a community name among its arguments outside N_P , so $s \notin S_C(P)$ by Definition 1. As P was arbitrary, s records q . Suppose no atom of s names q , and let P be the agents among the arguments of the atoms of s together with the signers of the speech acts among them, the agents of the community names among them and some agent other than q , which exists as Π has at least two agents. Then P is nonempty, $q \notin P$ and $s \in S_C(P)$, so s does not record q . $\square$

The recording graph of a configuration of the protocol realising C therefore has an edge $p \rightarrow q$ exactly when some atom of the state of p names q . Its edges are classified by the predicate of the atom.

Definition 62 (*E-recording graph*). Let E be a set of predicates. The *E-recording graph* of a configuration is the spanning subgraph of its recording graph whose edges $p \rightarrow q$ are those for which the state of p contains an atom with predicate in E that names q ; the *E-recording graph* of a run is the union of those of its configurations.

PROPOSITION 26 (OPENNESS). R_C is open.

PROOF. Let τ be an unobstructed introductory act of C , let $p \neq q$ in Q , let r be a finite safe run of the protocol over Q containing no interaction between p and q , and let β be an introduction of p and q meeting Definition 7. Extend r , for each role π_i of τ and each $\varphi \in \neg(i) \uplus \neg(i)$, by a volition change of $\beta(\pi_i)$ and a unary transition adding $\beta(\varphi)$ there, induced by a schema of arity one of clause 1 at an admissible binding sending its role to $\beta(\pi_i)$ and its agent variables into Q , which that agent takes by itself; where that schema forbids $\beta(\varphi)$ and the state of $\beta(\pi_i)$ holds it already, τ requires it once there, and the transition is skipped. Each transition taken is machine-enabled, the schema requiring nothing, forbidding nothing or only $\beta(\varphi)$, which is then absent, and having no name term, so no name need be fresh. Such a schema deletes nothing, so the extension is finite and safe and every atom τ requires at β is present at its last configuration. A schema of arity one has neither of p and q in a role other than the one sent to $\beta(\pi_i)$, so by clause 2 of Definition 7 no atom it adds at $\beta(\pi_i)$ is one τ forbids at π_i : the extension adds no forbidden atom, and no atom τ requires at β is one it forbids.

Let $\varphi \in \neg(i)$ and suppose the state of $\beta(\pi_i)$ there contains $\beta(\varphi)$. The initial state is empty and the extension added no forbidden atom, so some transition of r added it; that transition is induced by a transaction of R_C determined by a schema τ' at a binding β' with $\beta(\varphi) \in \beta'(+ (j))$ for a role whose extent holds $\beta(\pi_i)$; by clause 2 of Definition 7 it is no listed role, so it is an agent role sent to $\beta(\pi_i)$. By that clause β' sends a role of τ' to the other of p and q , and $\beta'(+ (j))$ names that agent. Both are then participants, so by Lemma 60 the state of each changes, and the state of $\beta(\pi_i)$ afterwards records the other, so it lies in no $S(P)$ omitting them and the transition restricted to $\beta(\pi_i)$ is no transition of that agent's own transition system: the transition is an interaction between p and q , which r does not contain. So no forbidden atom is present, and τ , having no name term, is

machine-enabled at β at the last configuration of the extension, and every transaction it determines there leaves p holding an atom naming q and q one naming p , changing both states, and is guarded by both; by Lemma 25 each records the other, so it is a mutual recording of p and q in $R_C(Q)$, β mapping into $\{p, q\} \subseteq Q$, and by Lemma 61 the transitions it induces are interactions between p and q . $\square$

Closure requires that the participants of an unguarded act already lie in one instance. Where an atom can be added only by an act of the two agents it relates, that follows directly. An edge may also come from a transfer: a coin passes from hand to hand, and the agent it points at is reached through the agents that carried it rather than by a single act. The lemma below covers both: no act ever leaves an edge whose ends lie in different instances.

Let E have traceable provenance in C , $Q \subset \Pi$, and r a finite safe run of $\mathcal{F}(Q)$.

LEMMA 63 (NO EDGE CROSSES INSTANCES). *Every edge of the E -recording graph of r joins two agents that lie in one instance in r .*

PROOF. The E -recording graph of r is the union of those of its configurations, and every configuration of r ends a prefix of r ; the interaction graph of a prefix of r is a subgraph of that of r , so two agents in one instance in a prefix are in one instance in r . It suffices to treat an edge $p \rightarrow q$ of the E -recording graph of the last configuration. The claim is trivial for $q = p$, and we argue by induction on the length of r . The initial state is empty, so a run of one configuration holds it vacuously. Let r end with the transition $e \rightarrow c$, and let the state of p at c contain an atom with predicate in E naming $q \neq p$. If the state of p at e contains such an atom, the induction hypothesis applied to the prefix of r ending at e gives the claim. Otherwise $e \rightarrow c$ added it, so it is a volitional machine transaction induced by a transaction determined by a schema τ at a binding β , and the atom is $\beta(\varphi)$ for an atom φ added at a role π_i , with predicate in E , where p is in the extent of π_i , and $q = \beta(s)$ for s the base of an argument of φ that is an agent role, an agent variable or a name term, or q is the signer of $\beta(y)$ for a speech-act variable y that is an argument of φ or occurs in a name term among its arguments.

By Lemma 60 the state of every participant changes, and by Lemma 25 the state of p at c records q , so it lies in no $S_C(P)$ with $q \notin P$ and the restriction of the transition to p is no transition of $\mathcal{F}(\{p\})$. By Definition 47 the transition is therefore an interaction between p and every other participant.

By Definition 8 one of two cases holds, in the first case of s and in the second of y . If s is the base of a role of τ , an agent role whose agent is in its own extent, or y occurs in no atom required at a role and so by Definition 4 has $\beta(y)$ signed by the agent in a guarding role, then q is a participant, so p and q are joined by an edge of the interaction graph of r . Otherwise s is the base of an argument of an atom ψ required at an agent role π_j , with predicate in E , or y is an argument of such an atom or occurs in a name term among its arguments; put $p' := \beta(\pi_j)$. By Definition 13 $\beta(\psi)$ is an atom of the state of p' at e , its predicate is in E and it names q , so by the induction hypothesis p' and q lie in one instance in the prefix ending at e , hence in r . If $p' \neq p$ then p' is a participant, so p and p' are joined by an edge of the interaction graph of r . Either way p and q lie in one instance in r . $\square$

PROPOSITION 27 (CLOSURE). *R_C is closed.*

PROOF. Let (t, Q') of R_C be enabled at the last configuration c of a finite safe run r of the protocol, determined by a schema τ at a binding β , and let $Q_i := \text{ext}_c(\pi_i)$, so that the participants of t are $\bigcup_i Q_i$ (Definition 13). Let E be the set of predicates occurring in C , which has traceable provenance.

First, let $\pi_i = \pi_k[\rho(*, u_2, \dots, u_r)]$ be a listed role and $p \in Q_i$ (Definition 12). $\beta(\pi_k)$ holds $\rho(p, \beta(u_2), \dots, \beta(u_r))$ at c , an atom naming p , so $p = \beta(\pi_k)$ or the E -recording graph of r has the edge $\beta(\pi_k) \rightarrow p$, and p and $\beta(\pi_k)$ lie in one instance in r by Lemma 63. So the participants in the roles of one base lie in one instance in r .

Second, let π_i require an atom ψ of traceable provenance an argument of which has the base π_l of a role π_j . That argument is π_l or a name term with base π_l , so $\beta(\psi)$ names $\beta(\pi_l)$, and every $p \in Q_i$ holds it, so $p = \beta(\pi_l)$ or p and $\beta(\pi_l)$ lie in one instance in r by Lemma 63. By the first step, the participants in π_i and in π_j lie in one instance in r .

Instances are the connected components of the interaction graph of r , so lying in one instance is an equivalence, and by Definition 9 the participants in the roles of one connected component of the role graph of τ lie in one instance in r . If the role graph is connected, all participants of t lie in one instance. Otherwise each connected component of it has a guarding role, by Definition 9, whose agent is a guard of (t, Q') and lies in the instance of the participants of that component; every instance holding a participant holds the participants of a component, and so holds a guard. $\square$

B.4 Clauses and Their Modalities

PROPOSITION 31 (WITNESSING). *A schema is witnessed against every agent role it names at another agent role.*

PROOF. Let τ name π_j at π_i by an atom φ added at π_i , let β be a binding for τ , and put $a := \beta(\pi_j)$ and $b := \beta(\pi_i)$, which differ by Definition 4, π_i adding an atom naming π_j . Let $d \rightarrow d'$ be determined by τ at β . By Definition 13 d'_b is d_b less the atoms deleted at the roles sent to b and plus those added there, the two being disjoint, so $d'_b \setminus d_b$ is the join of the atoms added at those roles and holds $\beta(\varphi)$, an atom naming a . $\square$

PROPOSITION 32 (ENFORCEMENT BY GUARDING). *If e is guarded in C , the consent clause for e is enforced by C .*

PROOF. Let r be a finite safe run of the protocol realising C and let an e -atom occur in the state of an agent at some configuration of r . The initial state is empty, so some transition of r added it, and by Definitions 14 and 13 that atom is $\beta(\varphi)$ for a binding β of a schema τ of C and some $\varphi \in +(i)$ with predicate e . By hypothesis every role φ names is a guarding role of τ , so every agent the atom names is β of a guarding role and hence in the guard of the transaction taken; by Definition 44 each held its class in its volitional state at that point, which is to say willed the act. $\square$

B.5 Signing the Contract

PROPOSITION 34 (EVERY PARTICIPANT HAS PRESENTED ITS UNDERTAKING). *In a presenting run, every agent of an instance of two or more agents has presented their undertaking to another agent of that instance.*

PROOF. An instance is a connected component of the interaction graph of the run (Definition 49), so an agent of an instance of two or more agents has an edge to another agent q of it, put there by an interaction between the two; the run is presenting, so the agent has presented their undertaking to q . $\square$

C Values in Act Schemas

Act schemas allow values of any type (Section 3.2). This appendix defines them and extends states and bindings to them; the compilation and the conditions then apply as stated.

Definition 64 (Schema with values). Fix a countable set V of values, disjoint from Π , D and $\mathcal{N}$, and a set Z of value variables, disjoint from R , X and Y . A schema with values is written

$$\tau(\pi_1, \dots, \pi_k; x_1, \dots, x_m) \mid \gamma : E_1 \cdots E_k$$

as in Definition 3, except that $x_1, \dots, x_m \in X \cup Y \cup Z$; an argument of an atom may be a value or a value variable among $x_1, \dots, x_m$; γ , its constraint, is a conjunction of linear equalities and

inequalities over its value variables and values; and an atom marked $+$ or $-$ may carry a *multiplicity* κ , a positive whole number or a value variable among $x_1, \dots, x_m$, written $\kappa \cdot \varphi$.

Definition 65 (Binding with values). A binding β for a schema with values τ is a binding for τ (Definition 4) that also maps each value variable of τ to a value, under which γ holds and every multiplicity of τ is a positive whole number; it maps a value to itself, and $\kappa \cdot \varphi$ to the multiset holding $\beta(\varphi)$ with multiplicity $\beta(\kappa)$.

A ground atom may have values among its arguments: $S_C(P)$ is the set of finite multisets over $\Sigma[P \cup D_P \cup N_P \cup V]$, and a permutation of Π is the identity on values. Which types the values have — numbers, strings, dates — is the realisation's; the contract is parametric in them. A multiplicity counts copies of an atom, so it is a positive whole number in every contract: paying zero coins is no act. In the examples of this paper coins are whole, an amount a multiplicity: GC's payment pays k coins, $-k \cdot \mathfrak{c}(\text{Bob})$ at Alice and $+k \cdot \mathfrak{c}(\text{Bob})$ at Bob; the bonds of [60] number dates by days, the successor of a date being the next day. A binding chooses its values, so an introduction (Definition 6) chooses its own.

Definition 13 applies to a schema with values as written, the multisets of a role at a binding carrying each atom as many times as its multiplicity says. Definition 3 requires that a role that is not guarding add or delete an atom, and a multiplicity is at least one, so this holds at every binding. A value is no agent role, no agent variable and no name term, so it has no base, Definitions 8 and 9 do not concern it, and no state records an agent by it (Definition 5). The conditions of Section 3.6 therefore read a schema with values as they read one without, they stay decidable, the constraints being linear, and Theorem 11 and its proof stand.

GB's redemption of a matured bond, as its owner's copy has it, is the schema with values

$\text{redeem_bc}(\text{Alice?}, \text{Bob}; d, r, t) \mid d \leq t :$

$\text{identity}, \text{knows}(\text{Bob}), -\text{bond}(\text{Bob}, d), +\mathfrak{c}(r) \quad \text{identity}, \text{date}(t), -\mathfrak{c}(r), +\text{bond}(\text{Bob}, d),$

where $\text{bond}(\text{Bob}, d)$ in the state of Alice is a bond issued by Bob and maturing at d , $\text{date}(t)$ in the state of Bob is Bob's date, and the constraint says that the bond is mature. It compiles to

$c'_{\text{Alice}} := (c_{\text{Alice}} \setminus \{\text{bond}(\text{Bob}, d)\}) \uplus \{\mathfrak{c}(r)\}, \quad c'_{\text{Bob}} := (c_{\text{Bob}} \setminus \{\mathfrak{c}(r)\}) \uplus \{\text{bond}(\text{Bob}, d)\},$

provided $\text{identity}, \text{knows}(\text{Bob}), \text{bond}(\text{Bob}, d) \in c_{\text{Alice}}$ and $\text{identity}, \text{date}(t), \mathfrak{c}(r) \in c_{\text{Bob}}$,

provided $d \leq t$,

guarded by $\{\text{Alice}\}$

A binding chooses d and t , and binds the schema only if $d \leq t$; Bob's role, which is not guarding, reads Bob's date and changes state.

D Related Work, Expanded

The paragraphs of Section 10, expanded.

Contracts as formal objects. The formal study of contracts in this field is dominated by the contract-as-automaton family. *Contract automata* [1] give the deontic modalities an operational semantics over the acts of named interactive parties; Flood and Goodenough [17] render a loan agreement as a deterministic finite automaton whose states are performing, delinquent and default and whose transitions are drawn from an alphabet of events; the L4 domain-specific language [71] has an executable semantics in which deontic clauses with deadlines become states and transitions; and timed contract automata [8] add quantitative time. Rule-based encodings represent contract clauses as defeasible deontic rules with violation and reparation chains [28], and imperative and declarative smart contracts running on a distributed ledger have been compared [27].

Three commitments are common to all of these, and this paper makes none of them. The parties are fixed when the contract is formed, so the object of study is one instance among a known set of parties, where our conditions quantify over the family of systems a contract induces on every finite set of people. There is one contract state, the automaton's state, the rule base's facts or the ledger, where the definition of *records*, and therefore of what it is for two agents to be connected, needs a local state per agent and a local-states function monotone in the set of people. And each requires something closure forbids: an alphabet of events every party is assumed to observe alike, a clock, or a ledger.

Two languages outside this venue are the nearest to ours. Symboleo [45, 64] specifies a legal contract as a set of obligations and powers over an ontology of roles, assets and events, with a semantics of logical axioms on statecharts, and properties verified in temporal logic; its purpose is monitoring, and roles are assigned to parties during each contract execution. Stipula [10] is a domain-specific language with legal constructs, agreement, permissions, obligations and violations, whose contracts are state machines over parties, fields and linear assets; its *agreement* operator is the contract's constructor, fixing the full list of parties, and thereafter each function is invoked by one named party. Catala [41] does the corresponding work for statute rather than contract, encoding the base-case and exception structure of legislation in default logic. Both contract languages make the three commitments above: a Stipula configuration carries a global clock, and Symboleo presupposes a monitor, which is a participant that is not a party.

A concrete case is befriending, which cannot be written in Stipula: it is an act that two agents must both will, available repeatedly between any two of them, and Stipula allows joint consent only in the constructor, so each befriending would be the formation of a new contract rather than a clause of a standing one.

We claim no new language. The conditions of Section 7 are on the four things a schema carries. Stipula's function declarations name an invoking party and a precondition, and Symboleo attributes obligations to a debtor and a creditor, so the four could be extracted from either given a semantics in which each party holds its own state.

BSPL [66] is the protocol language of multiagent systems nearest to ours. A protocol is a set of message schemas between roles, each parameter adorned in or out, and a role may send a message once it knows the bindings of the message's in parameters and does not know those of its out parameters. There is no global state: the local state of a role is the history of the messages it sent and received, and the enactments of a protocol are its tuples of parameter bindings, a key keeping them unique. The meaning of a message, the commitments it creates among them, is a layer above. An act schema is the counterpart of a message schema in which an act may take any number of roles, each consenting by its guard, and changes the local state of each. BSPL's well-formedness conditions make a protocol enactable. The conditions of Section 7 concern the family of systems a contract induces on every finite set of people, and the coalescence of two of them.

Norms and normative systems. The formal study of what a contract obliges uses deontic and defeasible logic: contracts are represented as defeasible rules with deontic operators and compliance is checked against them [26]. Normative multi-agent systems concern how norms are operationalised, detected, and brought into conflict [4]. A volition guard [40] is a condition of enactment rather than a deontic operator: an act does not occur unless the people it obliges are willing. Where a clause of the contract describes no act the protocol carries out, Section 8 says what the realisation does with it: record its breach, or record the undertaking and the party who gave it.

Social commitments [65] give the normative concepts one form, a commitment $C(x, y, G, p)$ of a debtor x to a creditor y in a context group G to bring about p , the context group providing the norms by which disputes between the two are settled. A commitment is created, satisfied, cancelled,

released, delegated and assigned, and obligations, taboos, conventions and the Hohfeldian relations are kinds of commitment. The three modalities of Section 8 classify what a realisation does with a clause, and are not kinds of obligation. An undertaken clause is a commitment in this sense of its obligor to every other participant, and another agent holds its record; a grassroots social contract has no context group above the agents.

Juridical acts and declarative power. What we call an act is Sartor's *result-declaration*, one of "acts intended to produce legal determinations" [49], and the fullest model of it is Hage's [32], which analyses juridical acts as intentional changes in a world of law furnished with entities, facts and rules. A multi-party act willed by its parties is not new either. Declarative power is "the capacity of the power-holder of creating normative positions, involving other agents, simply by 'proclaiming' such positions" [22], and its authors observe that a declarative power exercised jointly by several parties, with the consent of each, is a contract [21].

We differ from that tradition on where the effect of an act comes from. There a proclamation has effect only if an institution provides for it: "when an agent j proclaims A , j brings it about that A only if the concerned institution s provides for this result" [21]; in norm-governed institutions "designated agents are *empowered* to create particular kinds of states of affairs" [34]; counts-as statements "hold only with respect to a context" [30]; and an artificial institution "presupposes an agreement on an unambiguous definition of a set of concepts" [18]. A grassroots social contract has no such institution, being subject to no external authority. The effect of an act is therefore the change it makes in the agents' own states, and the condition of its effectiveness is their will rather than a conferred power. An act schema is a result-declaration with the institution taken out: its precondition is on each participant's own local state rather than on an institutionally conferred power, and its guard takes the place of authorisation.

An *electronic institution* [13] is specified by a performative structure, a network of scenes connected by transitions, each scene a protocol over roles whose arcs are labelled with illocution schemes; by institutional roles, which external agents cannot play; by a role hierarchy and a separation of duties among roles; and by normative rules that infer an agent's obligations from its illocutions. Every agent adopts a role, every interaction takes place in a scene, an institutional agent at each transition brokers agents between scenes, and the institution keeps for each agent the obligations its acts incurred. A grassroots social contract has no institutional role, and the records of its acts are kept in the agents' own states.

Standard-form contracting. A grassroots social contract has no drafting party, so there is no party whose terms are policed and no party from whom disclosure is required; the defects the literature finds in the arrangement between a person and a proprietor are absent because the proprietor is absent.

The End-User Licence Agreement is a *contract of adhesion*. Radin [47] treats mass-market boilerplate as the deletion of rights that the legal system otherwise confers, and as a matter for the rule of law rather than for consent only. Kim [37] treats the digital instances of the form — the terms accepted by clicking, browsing, or installing — and what assent means when the act of assent is a condition of access. Ben-Shahar and Schneider [2] address the remedy most often proposed, mandated disclosure, and find that it fails: the terms are not read, and requiring more of them is not read either.

This literature takes the bilateral form as given; its questions are what may be done within the form: whether assent was real, which terms a court should refuse, what a drafter must disclose.

Terms a machine can evaluate. Surden [67] separates the terms of a contract a machine can evaluate from those it cannot; work on smart legal contracts separates the operational parts of an

agreement, which code executes, from the rest, which stays in prose [9]; and in security, Schneider [50] characterises the policies an execution monitor can enforce. Each divides a contract's terms in two, by what a realisation can do with them. Section 8 divides clauses in three, distinguishing a clause no act of the contract can breach from a clause an agent may breach while the contract records the breach against them. A two-way division has no room for the second, and it is the class the law needs: a remedy requires a provable breach and an identifiable obligor.

In the monitoring literature, compliance is computed by monitors that “receive inputs from *trusted* observers” [42], and where the timestamps cannot be trusted the monitor is made robust rather than removed: compliance “is typically computed with respect to timed event traces with event timestamps assumed to be perfect”, and the remedy offered is a semantics for compliance when they are not [6]. A monitor is a participant without a role in the act, and closure forbids one. Here the record is held by the parties themselves, and what makes it evidence is Definition 29: it names the party answerable and it lies in someone else's hands.

Code as a regulator. Lessig [38] argued that architecture regulates conduct as law, markets and norms do, and that in the digital realm the architecture is code. A grassroots social contract is not code: the conditions of Section 7 concern who may take part in an act and whose will is required for it, rather than what the code makes an agent do, and they are checked on the text rather than on the code. The digital social contract [7] identifies the contract with code: the contract is a program, and a party can behave only according to it; the smart contract literature [11] does the same on a blockchain.

Self-governance without an external authority. Ostrom [44] documented communities that devise, adopt and enforce their own rules over shared resources, without an owner and without a state administering them, and identified the conditions under which such arrangements endure. A grassroots social contract makes the same claim in the digital realm, that a community constitutes itself, and openness is its formal counterpart: any two people can come to record each other by acts of their own, so any person may start an instance and instances coexist. Commons scholarship concerns how a resource is governed in common, for physical resources [44] and for knowledge and networked production [3, 33]; a grassroots social contract governs a relationship rather than a resource, and applies where nothing is held in common.

Platform regulation binds the operator: the General Data Protection Regulation [15] imposes duties on the party that processes personal data, and so presupposes it. Where a community's platform is constituted from its members' own devices [53, 55, 56], there is no operator to bind.

Grassroots systems. The notion of a grassroots protocol is due to [53] and was recast in terms of interleaving and interaction in [40], which also introduced volition-guarded transactions and volitionally grassroots protocols. Grassroots platforms have been specified for social networking [54] and for personal cryptocurrencies [39, 52, 60]. Secure Scuttlebutt [68] is a peer-to-peer social network, started in 2014. Each person is identified by a key pair they generate themselves, so no authority introduces people to the system; each signs every message of their own append-only log; and a person's device replicates the logs of the people they follow, and of those these follow, up to a set distance, so data flows along the social graph, with no consensus and no central directory. Following is a message in the follower's log and takes no consent of the one followed, where befriending under GSG:1 takes the will of both. Its messages are signed and cannot be repudiated, as GSN's messages, signed by their senders, cannot. Three things are added here. The first is politeness, a condition on a protocol's transactions, which Theorem 24 proves sufficient for the protocol to be grassroots, so no proof about runs is needed. The second is the step before the specification: the legal text, and decidable conditions on it under which politeness is established from the acts rather

than from the protocol. The third is what a realisation does with a clause of the contract, whether or not the contract provides an act for it, which depends on the same schemas.

Neighbouring instruments. Four instruments resemble a grassroots social contract, and each differs from it in what it does rather than in how it is drafted. A *licence* states what one party may do with something another party is entitled to: the General Public License [20] presupposes the copyright it licenses, grants freedoms on conditions, and binds a taker because they would otherwise infringe. A grassroots social contract licenses nothing, and its obligations are owed by each agent to every other and derive from no agent's prior entitlement, so a person who has never held any right that the others could infringe is bound as the rest are. *Bylaws* bind the members of an association symmetrically and are adopted on joining, which makes them the closest of the four, but they presuppose an association: an entity with legal personality, organs that act for it, and a membership register. A *code of conduct* states standards of behaviour, and its sanction is exclusion administered by whoever runs the forum, which is the party a grassroots social contract does not have; *terms of membership* presuppose the same party, in its capacity as the one who admits.

A *Ricardian contract* [29] is a single document by which an issuer of a financial instrument offers a contract to its holders: readable by people and parsable by programs, signed by the issuer with the keys that verify the signature included, and identified by its hash, which every transaction record of the instrument carries, so neither party can change the contract unnoticed. It binds an issuer to the holders of one instrument. Under GC (Section 2.4) every person issues their own coins, and each participant signs an undertaking to keep the contract towards all the others (Section 9).