Child-Safe Social Networking: From Social Contract to App (Full Version)

Ehud Shapiro
London School of Economics and Weizmann Institute of Science
UK and Israel

Abstract

The harms to children from centrally-controlled social networks are inherent to their surveillance-capitalism business model. Hence, despite legislation and regulation, parents still cannot exercise their responsibility for their children in the digital realm the way they do in the physical one: to decide whom their child may meet; which gatherings and activities their child may join; which parents, teachers, and coaches to entrust their child with. Ideally, a child-safe social network should rest on a social contract among its participants under which a parent is a consenting party to every act by which their child comes to know another adult or child, gains a friend or joins a group. For example, child-befriending should require the will of both children and a parent of each, and any one of the four should be able to end the friendship.

Here, we present such a child-safe social contract and derive from it a working smartphone app that realises the social network it specifies, following the abstraction cascade programming methodology. The resulting social network is grassroots, implying that (1) it can operate on networked smartphones of parents and children without third party intervention or control; (2) any community may initiate its own child-safe social network; and (3) community networks may coalesce through cross-community befriending.

We prove that in this child-safe social network, parents can exercise their responsibility for their children's safety in the digital realm as they do in the physical one, that the network is grassroots, and that the implementation is correct; we illustrate its operation on a small example, and stochastically simulate the child-safe social graph in a village of families.

CCS Concepts

• **Computer systems organization** → **Peer-to-peer architectures**; • **Theory of computation** → **Distributed computing models**; • **Security and privacy** → *Social aspects of security and privacy*; • **Human-centered computing** → *Social networking sites.*

Keywords

Grassroots Platforms, Grassroots Social Contracts, Volition-Guarded Transactions, Child Safety, Social Networks, GLP

1 Introduction

The problem. Centrally-controlled social networks cause well-documented harms to children [33, 35, 51, 76], inherent to their surveillance-capitalism business model [81]. The parental controls, child-only services, and educational platforms they offer are features the operator runs and can change: even Messenger Kids, which lets parents approve their child's contacts from their own app, is run by Meta, which holds the children's messages, verifies no identity, and sets one set of rules for all [45, 46] (Section 6.1). Decentralised alternatives subvert centralised control but do not address child safety as a design concern (Section 6.1).

TL;DR

Children meet each other on the Web's social platforms, and the companies that own them, not the parents, decide whom a child may meet.
We propose a child-safe social network that is part of a Web owned, operated and governed by the people who use it: it runs on the smartphones of parents and children, under a social contract each community adopts, by which a parent takes part in every act by which their child comes to know someone, befriends someone or joins a group.

Aim. In the physical realm, parents exercise responsibility for their children's safety by deciding whom their child may meet; which gatherings and activities their child may join; which parents, teachers, and coaches to entrust their child with. We aim to offer exactly these same capabilities to parents in the digital realm. The social contract of this paper gives each of them, and Proposition 4.3 proves it.

- *whom their child meets*: a child comes to know an agent[1] only by an act of both and of a parent of the child who knows the agent, two children by an act of both and of a parent of each, the parents knowing each other (GnW:2 and CSSN:2); and two children become friends only by an act of both and of a parent of each, the parents being friends (GnW:2 and CSSN:3);
- *which gatherings their child joins*: a child joins a group only of a friend, by an act of the child, the group leader and a parent of each child among them (GSN:3 and GnW:2);
- *which adults they entrust their child with*: a child and an adult become friends only by an act of both and of a parent of the child, the adult being a friend of the parent (GnW:2 and CSSN:3);

and a parent may end each of the above, unilaterally (GnW:3).

An example. Alice and Bob are friends; Carol is Alice's child and Dave is Bob's. Alice requests that Carol and Dave come to know each other, naming Bob, and Bob, Carol and Dave approve; Alice then requests that they befriend, and the three approve again. Later Bob ends the friendship.

Design. Here we present the design of a grassroots child-safe social network, in which a child's partaking in online activities is subject to a parent being willing.

Grassroots platforms [63, 65] are designed to be owned, operated and governed by their participants. They are person-to-person smartphone apps that realise Proudhon's two-centuries-old concept: a social contract that can have multiple, independent, volitionally-forming instances that may coalesce by mutual consent, possibly

[1] For simplicity, assume an agent is an adult or a child; the notion is expanded below.

into a single global instance [41, 67]. A grassroots platform is constituted from its participants' smartphones and the grassroots app each participant runs, and requires no global resource but the network: no hardwired servers, no global authority, and nobody's permission to start [63, 65]. Any group of parents can form their own child-safe social network at will, which operates solely on the networked smartphones of the parents and children. Independently-formed instances may interconnect—for example, when a parent in one instance befriends a parent in another—and may eventually coalesce into a larger network.

This design differs from existing approaches to child safety online in three structural respects:

(1) **Parental willingness is structural.** In existing platforms, parental controls are features offered by the platform and can be circumvented. Here, parental willingness is a precondition of the befriending transaction itself: the parents are *guards* of the transaction. An agent the network records as a child cannot befriend without a parent of theirs being willing, because the transaction requires it. A child's identity is a ward, operated by the child; the child's key pair is chosen by their creator, a parent, and only an act of the creator ends a guardianship, so no agent recorded as a child can cease to be one by an act of their own.
(2) **Parents are structurally connected.** The requirement that parents be friends before their children can befriend ensures that parental networks form first. Parents who know, trust, and can verify each other's identities can then endow their children with the ability to befriend each other.
(3) **Childhood is declared.** A child enters the contract by the declaration of an adult, their creator — in practice, the creator sets up the app on the child's phone — who may make other adults the child's parents, and stops being a child when the creator declares them of age, the contract's proxy for the legal act of majority. Only an adult can be a parent, and a parent cannot be a child. The contract records the declarations and who made them, and the courts decide whether they are true (Section 4.1).

The contract. We first present the design as a social contract (Section 4.1). It extends two contracts. The first, the grassroots social network [23, 64], rests on a base contract of keys and knowledge — an agent establishes their identity by a key pair, two agents come to know each other by an act of both, and an agent who knows two others may introduce one to the other — and provides for befriending, by two agents who know each other, and ending a friendship, and for messages and groups. The second, the guardians and wards contract, provides for a person creating a ward, of whom they are then the creator and a guardian; for a ward coming to know an agent, and entering a relation with one, by an act of both, of a guardian of the ward, and of a guardian of the other if they are a ward; for ending a relation, by either agent or a guardian; and for the ward's creator retiring the ward. It is shared with a contract, in preparation, whose wards are operated by AI agents answerable to a principal. The contract's own six clauses are for parents and children: an adult creates their child's identity and may make other adults its parents; a child comes to know an agent only by an act of both and of a parent who knows the agent, two children by an act of both and of a parent of each, the parents knowing each other; two children befriend only if their parents in the act are friends, and a child and an adult only if the adult is a friend of the parent; the creator declares the child of age, which ends every guardianship; an adult shall not befriend, nor let their child befriend, an agent they know to be a child with no guardian, and shall report it to the child's parents; and only children may join a group a child leads. The contract is written at the level on which parents and educators discuss these matters, and each community may adopt its own (Section 4.2).

Specification. The clauses are converted from natural language to *speech-act schemas* [23], a formal language for the acts the contract describes, each naming the roles of its participants, which of them must will the act, and its precondition and effect at each role, with the agents in the roles determined upon signature, or, for a role that takes every agent another participant lists, by that participant's state (Appendix B). The protocol realising a contract written in this language is grassroots, and volitionally so, provided the contract is *syntactically grassroots* [23], satisfying: (*i*) *Introduction*, that some act can introduce any two willing parties, following which the state of each stores an identifier of the other; (*ii*) *Provenance*, that the state of a party stores an identifier of a person only by an act with that person, or with a party whose state stores that identifier; and (*iii*) *Volition*, that an act requires the consent of a party in each connected part of the undirected graph formed by identifiers stored by its parties, unless that graph is connected. For a contract with wards, whose acquaintance forbids the ward's mark, an *introduction with wards*, by the acts in which a guardian takes part, stands for (*i*); the conditions for wards are those of a paper in preparation on a contract whose wards are AI agents, stated and proved in Appendix B.4. The schemas compile into *volition-guarded multiagent atomic transactions* [41]: each transaction specifies the initial and final machine states of the participating agents, together with the persons that must be willing—its *guards* (Sections 4.3 and 4.4). Child-befriending is a quaternary transaction guarded by both children and a parent of each, with the children knowing each other and the parents already being friends as preconditions; the children come to know each other by a quaternary transaction with the same guards, the parents knowing each other. A child joining a group of a friend is guarded by the group's leader, the child, and a parent of the child, with an additional parent when the leader is also a child.

Closing a group is one act of its leader and of every member the leader lists, and ends every membership. Group messages reach the members one at a time: a post sends the item to the leader, and the leader delivers it to each member they have not yet delivered it to, each delivery requiring that member's friendship with the leader. A grassroots platform communicates along the edges of its social graph in any case.

One act, all the way down. This paper follows one act of the contract, and its undoing, through every layer of the abstraction cascade of Figure 1 (Section 2), from the legal text to the app. The two show the two ways an act may be guarded: the first takes four parties, each of whom must be willing, with a precondition on two of them that neither child can bring about; the second is the act of a parent alone.

Section 4.1 states the two with the rest of the contract; Example 4.2 presents the transactions they compile to, and Section 4.4 proves their properties; Section 4.6 presents the request and the approvals that carry out the first, the user interface derived from them, and the two on the phone. Their speech-act schemas and the certification of the contract are in Appendix B, and the two-phase commit by which communicating volitional agents carry them out, with its correctness, in Appendix C.3. The other acts of the contract appear in the body only where a theorem ranges over them, and are set out in Appendix C.

Implementation. Grassroots Logic Programs (GLP) [66, 68, 70] is a high-level, typed, multiagent, concurrent programming language, designed for programming grassroots platforms, and volition-guarded GLP (vGLP) [37] extends GLP with volitional procedures, whose goals put a typed question to the person and reduce on the answer. We implement the specification by vGLP's canonical platform of communicating volitional agents, a two-phase commit compiled into vGLP from the contract [37]. Since a volitional reduction requires the person's answer, a correct implementation must elicit it; this is the reason for the user interface, in which an open question is presented as the widget of its interactive type [37]. We exercise the acts in a village scenario of seven agents, and show the platform in the grassroots app (Section 4.6). Stochastic GLP (sGLP) [69], with which the grassroots social graph was simulated at scale, simulates a grassroots platform from the vGLP program that deploys it, with stochastic agents in place of people; we simulate with it the child-safe social graph, in a village of families (Section 5). The correctness of the implementation depends on GLP being secure; a secure implementation of the social graph, in which agents verify each other to execute the agreed code using mutual attestations [13], is ongoing work.

Our results and those we cite. New here are the contract, its speech-act schemas and its transactions; the parents' capabilities (Proposition 4.3); the conditions that certify a contract with wards and with roles that take every agent another participant lists, as closing a group takes every member, with the theorem that the protocol realising such a contract is volitionally grassroots (Theorem B.26), and the contract's certification by them (Proposition B.27), from which the grassroots property of the protocol follows (Proposition 4.4); the argument that, of each act of the contract, one guard's agent knows the agent of every other participant, which brings the platform under vGLP's correctness theorem (Theorem C.12); and the program, the village, the app and the simulation. Cited, and recalled where the paper uses them, are speech-act schemas, their compilation and the conditions for certification [23], the conditions for wards being those of a paper in preparation, volition-guarded transactions and grassroots protocols [41], vGLP's canonical platform, its compilation and their correctness [37], and sGLP's simulation [69].

Paper outline. The rest of the paper is organised as follows. Section 2 presents grassroots social contracts and how one becomes an app, and the grassroots social graph and the grassroots social network, which the contract extends, each with its app. Section 3 presents the wards and guardians contract. Section 4 presents the child-safe social network: its contract and community-specific contracts; its transactions; the parents' capabilities and the grassroots property of the protocol; what the contract enforces, attests and undertakes; and the app, with the vGLP program, the user interface derived from it, a village scenario of seven agents and the platform in the grassroots app. Section 5 presents a stochastic simulation of a child-safe social graph in sGLP. Section 6 discusses related work and Section 7 concludes. Appendix A collects the mathematical foundations; Appendix B presents the speech-act schemas, compiles them into transactions and checks them against the conditions for certification, which it states and proves for contracts with wards; and Appendix C sets out every artefact of the cascade, together with the proofs that range over the whole contract and the correctness of the implementation.

2 Grassroots Social Contracts

A grassroots social contract [8, 23] is a text in natural language, organised in clauses its participants undertake towards one another. Each clause describes an act — who takes it, who must agree to it, and what it changes — or binding the agents without describing one. Acquaintance is an act of two agents, both of whom must agree, as is befriending; ending a friendship is an act of one. The contract defines the platform: there is no server, no operator and no account, only the people who adopted the contract and their smartphones [63, 65]. Each contract is presented in a box, a clause is cited by the contract's name and its number, as GSG:1, and 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 [23] (Section 4.5). A contract also carries *pragmas*, which informs the user-interface design of the app: a line naming a tab of the app and what it lists, and, for each clause, a line naming the person who proposes its act, a button for each act they propose, in brackets, bearing its act's name, and any field they fill, in angle brackets, then an indented line for each other person asked, their name and the question put to them [23]; a line is named by a clause of the contract or of one it extends, as CSSN's Joining and Ending a relation are by GSN:3 and GnW:3. The pragmas of the contracts this paper quotes are in the cited papers as well as in the repository https://github.com/EShapiro2/GLP; those of CSSN end its box (Section 4.1).

Legal Modalities of Contract Clauses [23]

■ **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.

Agents running code that correctly implements the contract cannot breach an enforced clause. Different parts of a clause may carry different modalities, in which case the clause carries the weakest modality among them.

From the contract to the app. The passage from the social contract to a running app goes through the *abstraction cascade* [37] of Figure 1. Each step is an implementation of the layer above it by the layer below, defined in the reference its arrow carries; the rendering of the clauses as speech-act schemas, within the first layer, is the one step left to judgement. Each layer is coded by AI

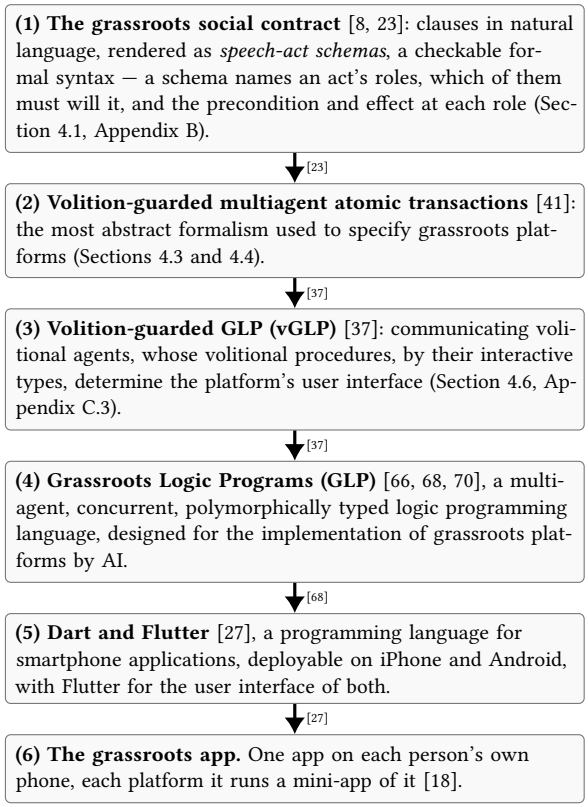


Figure 1: The abstraction cascade, from a social contract to a running platform. Each arrow is an implementation of the abstraction above it by the one below, carrying the reference in which that implementation is defined, and implementations compose, so the app implements every abstraction above it; the rendering of the clauses as speech-act schemas, within the first layer, is the one step left to judgement. A condition on the transactions of (2), politeness, is sufficient for the platform they specify to be a grassroots platform, and conditions on the schemas of (1) are sufficient in turn for politeness. We carry one contract down the whole of it.

from the paper that describes it; a gap found in coding is repaired in the paper first, and the code follows [68].

What a person sees. An act a person must will appears on their phone as a question, with the answers the contract allows: Bob is asked whether to be Alice's friend, and answers Accept or Decline. A person's phone holds their own state and nothing else — their friends, their groups, the messages they sent and received — and draws what it shows from that state, by the widget of each question's type [37]. The phones of two friends talk to each other, and to nobody else [18, 64]. Each screen in this paper is a screenshot of a mini-app, compiled from its social contract down the abstraction cascade [23, 37]. One instance of the grassroots app [18] on an iPhone simulator runs the mini-app of every agent; the agents communicate over channels [66] within it, and the 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.

2.1 Grassroots Keys and Knowledge

A *person* is a legally culpable entity: a natural person of age, also referred to as an *adult*, or a legal person of good standing, such as a business. A person takes the pronoun 'they'. An *agent* is a person or an entity that may act but is not legally culpable, such as a child or a computational (AI) agent, whose identity is a *ward* (Section 3). Alice, Bob, Carol and Dave name distinct agents, and each contract declares what kind of agent each is.

GKK: Grassroots Keys and Knowledge [23]

Alice, Bob and Carol are persons.

1. ○ *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.
2. ■ *Acquaintance.* Alice and Bob may come to know each other by an act of both.
3. ■ *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.

GKK is the base contract, which every grassroots social contract extends [23]; a key pair chosen at random is unique with high probability, and its uniqueness is what the agent undertakes. An agent *knows* another when they hold the other's public key. Acquaintance, an act of both, makes two agents know each other; an introduction, an act of the introducer and the agent introduced to, makes that agent know the one introduced, who takes no part; knowing is never undone.

2.2 The Grassroots Social Graph

GSG: Grassroots Social Graph [23]

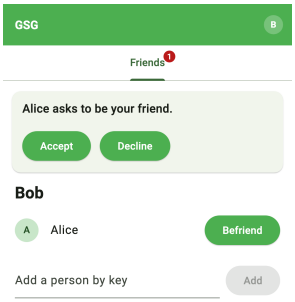
Alice and Bob are persons.

1. ◆ *Friendship.* Alice and Bob, who know each other personally and are not friends, may jointly decide to become friends.
2. ■ *Ending a friendship.* Alice may end their friendship with Bob.

GSG is the grassroots social graph [23, 64], which adds friendship to GKK; every grassroots platform that rests on friendship extends it, and coins extend GKK differently [62]. Two agents befriend only if they know each other. The social graph is stored distributively under the control of the people themselves, with each person storing the local neighbourhood pertaining to them, and no third party having access unless explicitly granted. The GSG app is a mini-app of the grassroots app [18], compiled down the cascade from GSG; Figure 2 shows it realising GSG:1.

2.3 The Grassroots Social Network

A *child* is a natural person not of age. A *computational agent*, or *AI agent*, is a program that holds a key pair of its own and takes acts



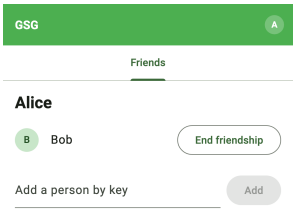
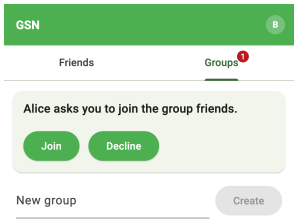


Figure 2: The GSG app: Alice asks to be Bob's friend, and Bob answers Accept or Decline (GSG:1); Alice's friends after Bob accepts.



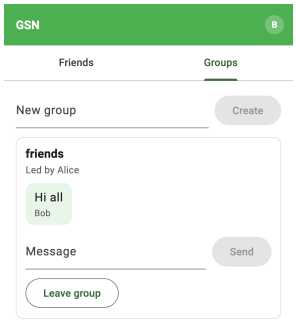


Figure 3: The GSN app, on Bob's phone: Alice asks Bob to join her group *friends*, and Bob answers Join or Decline (GSN:3); the group *friends*, led by Alice (GSN:2), with Bob's message to the group as Alice delivered it to him (GSN:5).

of the contract with it. A child also takes the pronoun 'they', and an AI agent 'it'.

GSN: Grassroots Social Network [23]

Alice and Bob are persons.

1. ■ *Message.* Alice may send a message to their friend Bob.
2. ■ *Group.* Alice may create a group, of which they are the leader and first member.
3. ■ *Joining.* Alice, the leader of a group, may add their friend Bob to it, by an act of both.
4. ■ *Leaving.* Bob, a member of a group, may decide to leave it; when the leader leaves, the group ends. A member of a group is removed from it if they cease to be a friend of its leader.
5. ■ *Group message.* Bob, a member of a group, may send a message to the group, and the leader delivers it to every current member.

GSN is the grassroots social network [23, 64], which adds messages and groups to the social graph. The GSN app is a mini-app of the grassroots app, compiled down the cascade from GSN; Figure 3 shows it realising GSN:2, GSN:3 and GSN:5.

3 The Guardians and Wards

A *relation* is a friendship or the membership of a group.

GnW: Guardians 'n' Wards

Alice and Dave are persons, Bob an agent, and Carol a ward.

1. ■ *Guardianship.* Alice may act to create a *ward*, Carol, to be operated by a non-person agent, and establish Carol's identity by choosing Carol's key pair, and Alice is then the *creator* and a *guardian* of Carol, as well as their friend.
2. ■ *Knowing and relations.* Carol, a ward of Alice, comes to know Bob, and enters a relation with Bob, by an act of both, of Alice, and of Dave, a guardian of Bob, if Bob is a ward, Alice and Dave possibly one person.
3. ■ *Ending a relation.* Bob, who is in a relation with a ward, Carol, may act to end it, and so may Alice, a guardian of Carol.
4. ■ *Retiring.* The creator of Carol may retire Carol, which ends every guardianship of Carol, after which Carol takes no act.

GnW, Guardians 'n' Wards, is a social contract for wards and their guardians, shared by two contracts: CSSN, whose wards are children's identities, operated by the children, and their guardians the parents (Section 4), and a contract in preparation, whose wards are operated by AI agents and the guardian of each is its principal, answerable for it; GnW is that contract's. GnW says what a guardian is party to: the ward's creation, every act by which the ward comes to know an agent or enters a relation, and the ending of the guardianship; who may create a ward and how guardianship ends, each extension says. The two differ there: the creator declares the child of age, and the child goes on as an adult; a principal retires the AI agent, which then takes no act. Answerability for a ward's acts is the AI contract's, not GnW's: a parent consents to a child's friendships and is not answerable for the child's acts.

4 The Child-Safe Social Network

4.1 The Contract

The child-safe social contract, CSSN, extends GSN (Section 2.3), and through it GSG and GKK, and GnW (Section 3), cites their clauses whole, and has six clauses of its own, for parents and children. It covers the standard functions of social networking—messages between friends, and groups—and child safety. Child befriending, by GnW:2 and CSSN:3, and a parent's ending of a child's friendship, by GnW:3, an act and its undoing, are the two this paper follows down every layer.

A child's identity is a *ward*, operated by the child, and a guardian of it is a *parent* of the child. Each box opens with the declaration of its names, a term of the contract from which its schemas and pragmas take the names of their roles.

CSSN: Child-Safe Social Network

Alice and Bob are adults, Carol and Dave children.

1. ◆ *Parents.* Only an adult may create a child's identity, and is then the child's *creator* and a *parent* of the child. Alice, the creator of Carol, may make Bob, whom Alice knows, a parent of Carol, by an act of the three, and may end that parenthood by their own act. Alice declares themselves a parent of Carol only if they are Carol's legal parent or guardian.
2. ■ *Acquaintance of a child.* To come to know Bob, Carol's acting parent Alice must know Bob; to come to know Dave, Carol's acting parent Alice and Dave's acting parent Bob must know each other.
3. ■ *Friendship of a child.* To enter a friendship relation between Carol and Dave, Carol's acting parent Alice and Dave's acting parent Bob must be friends; to enter a friendship relation between Carol and Bob, Carol's acting parent Alice must be a friend of Bob. While Alice is a parent of Carol, neither may end their friendship.
4. ○ *Coming of age.* The creator of Carol may declare Carol of age, which ends every guardianship of Carol, Carol's friendships remaining; the creator shall do so when Carol comes of age, and not before.
5. ◆ *A child with no guardian.* Alice, aware that Dave is a child though Dave's identity is not a ward, shall neither befriend Dave nor let their child befriend Dave, and shall report it to Dave's parents if Alice knows who they are.
6. ■ *A child's group.* Only a child may join a group Carol leads.

Pragmas

Children: the children Alice is parenting.
Parents: `Alice [Enter] <child>`
  `Carol: Alice asks to be your parent.`
  `Alice [Make parent] <person>`
  `Bob: Alice asks you to be a parent of Carol.`
  `Carol: Alice asks that Bob be your parent.`
  `Alice [End parent]`
Acquaintance of a child: `Alice [Child acquaint] <person> <person>`
  `Carol: Alice wishes that you and Dave know each other.`
  `Dave: Alice wishes that you and Carol know each other.`
  `Bob: Alice wishes that Carol and your child Dave know each other.`
  `Alice [Child acquaint] <person>`
  `Carol: Alice wishes that you and Bob know each other.`
  `Bob: Alice wishes that you and Carol know each other.`
Friendship of a child: `Alice [Child befriend] <person> <person>`
  `Carol: Alice asks that you and Dave befriend.`
  `Dave: Alice asks that you and Carol befriend.`
  `Bob: Alice asks that Carol and your child Dave befriend.`
  `Alice [Child adult befriend] <person>`
  `Carol: Alice asks that you and Bob befriend.`
  `Bob: Alice asks that you and Carol befriend.`
Coming of age: `Alice [Of age]`
Joining: `Carol [Child join] <person>`
  `Dave: Carol asks you to join their group.`
  `Alice: Carol asks that Dave join their group.`
  `Bob: Carol asks that your child Dave join their group.`
  `Bob [Child join] <person>`
  `Carol: Bob asks you to join their group.`
  `Alice: Bob asks that Carol join their group.`
Ending a relation: `Alice [Child unfriend] [Child leave] [Child close]`
Retiring: `Alice [Retire]`

The box ends with the contract's pragmas, from which the app's screens are compiled with those of GSN (Section 4.6): a tab of a person's children, a button for each act of CSSN a person proposes, and the question each act puts to the persons whose will it needs. Without children, the contract is GSN, every agent being a person. GnW is realised for children by the acts of CSSN: CSSN:1 says who creates a child's identity; CSSN:2 and CSSN:3 strengthen GnW:2, for knowing and for friendships; and CSSN:4 says what act ends the guardianship. Three parts of CSSN describe no act: the second sentence of CSSN:1, that Alice declares themselves a parent of Carol only if they are Carol's legal parent; the obligation that ends CSSN:4, that a parent shall declare the child of age when the child comes of age, and not before; and CSSN:5. Child safety depends on them, and Section 4.5 says what record the contract keeps of each.

Genuine digital identity. A *genuine personal identifier* is one a person declares as theirs, and is genuine if it is the first they declare [60]. A person establishes their identity by choosing their own key pair (GKK:1). A child's identity is established by their creator, a parent, who chooses the child's key pair when they create it and becomes the child's first friend (GnW:1 and CSSN:1); the creator may then make other adults the child's parents. So a child is a child from their first act: no identity that has acted can be created a child, and a child's identity has no friend, group or item but those its creation and its acts as a child have given it.

Why parenthood and majority are declarations. A contract whose realisation is to be a grassroots platform admits no participant of a second kind: no global registry that issues identifiers, no authority that certifies a birth, no global clock that says when a birthday has passed. Whether an adult is the parent of a child, and whether a child has come of age, are therefore facts no act of the contract can determine. The contract records who declared them, and the courts decide whether the declarations are true. In practice the creator sets up the app on the child's phone, to which they hold the sole password, and chooses the child's key pair there, holding both phones and answering on the child's phone for the child; the contract cannot tell who held the phone. A declaration of age is the contract's proxy for the legal act of majority, and the creator's declaration ends every guardianship. A child come of age whose parent does not declare it can buy a new phone and choose their own key pair as an adult; that identity is a second identity of one person, a sybil in the sense of [60], which the contract cannot detect. A child who controls their own phone can likewise choose their own key pair and pass as an adult. The contract cannot tell such a child from an adult; it protects against this case only by CSSN:5, which binds the adults who know the child.

Why the creator ends the guardianship. If a child could end the guardianship, they could end child safety. Guardianship therefore ends only by an act of the child's creator: the ending of the parenthood of a parent they made (CSSN:1), their declaration of age (CSSN:4), which ends every guardianship and leaves the child's friendships in place, or their retiring of the child (GnW:4), after which the child takes no act. Guardianship and friendship come apart: a guardianship ends, and the friendship of the parent and the child stays. The obligation to declare the child of age runs to the creator: they owe the declaration when the child comes of age, and not before, and a creator who withholds it or makes it early answers for it. A refusal is an act not taken, so the contract has no record of it; Section 4.5 classifies the obligation to declare as undertaken, and declaring early as attested.

Speech-act schemas. The clauses are converted to twenty-eight speech-act schemas [23], one per act, each naming the act's roles, which of them must will it, and what it requires, forbids, adds and deletes at each role (Appendix B). The schemas settle, clause by clause, whether an agent can breach the clause and what record of it the contract keeps (Section 4.5), and they compile into the volition-guarded multiagent atomic transactions of Section 4.3 (Appendix B.3). The protocol realising the contract is grassroots, since every child has a person on their record and two persons can always come to know each other (Proposition 4.4). A checker certifies from the schemas the contracts that have an act any two agents can always take [23]; this contract restricts acquaintance to persons and has none, and is certified instead by the conditions for contracts with wards — children here, and AI agents answerable to a principal — and with roles that take every agent another participant lists, as closing a group takes every member (Appendix B.4).

4.2 Community-Specific Social Contracts

The contract is written at the level on which parents and educators discuss how children may meet, play and learn together: who may befriend whom, with whose consent, and who may end it. A community can read it, argue over it, and change it, without reading code, since the rest of the abstraction cascade is derived from it.

A child-safe social contract need not be universal. Communities differ in their values, their cultures and their approaches to education, and each may adopt its own contract and revise it as its children grow and its views change. Each such contract is a separate mini-app of the grassroots app, derived from its own clauses as this one is, and a family may belong to several communities and run the mini-app of each. Four examples, each a change to the clauses above:

(1) Teenagers befriend with the approval of one parent, or of none, from an age their parents declare.
(2) Both parents of each child, where they have two, must approve a child friendship.
(3) A new member of a group receives none of the group's earlier posts.
(4) A teacher whom the parents entrust approves friendships among the children of a class.

The contract presented here makes one choice on each of these points: the approval of one parent of each child, at every age, for whom the child comes to know and whom the child befriends; earlier posts delivered to a new member; and teachers and coaches entrusted through their groups, a child joining a group by GnW:2.

4.3 The Transactions

We assume a potentially infinite set of *agents* Π, 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. 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 4.1 (Machine State, Configuration, Transaction, Volition–Guarded Multiagent Atomic Transaction [41]). 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 $c \to c' \in (S^Q)^2$ 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 [61]—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 are machine transactions that can be carried out only if their guards $Q' \subseteq Q$ are willing. Volition-guarded transactions do not distinguish between agents that initiate a transaction and those willing to participate 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. A volition-guarded transaction whose guard is

empty ($Q' = \emptyset$) is *unguarded* and is enabled whenever its machine precondition holds.

We recall informally the additional mathematical foundations on which this paper is based, included for completeness as Appendix A. A *transaction equivalence* $\sim$ relates transactions with the same participants and has countably many classes; a person wills a class, and taking one member of it fulfils that will [41]. An *agent state* is a pair of a *volitional state*, a set of classes the person 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 a person 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.

We are particularly interested in *grassroots protocols* [41, 63, 67]. A protocol is *oblivious* if any interleaving of correct runs of two disjoint groups is a correct run of the two together, *interactive* if two groups that have not yet interacted can always still do so, and *grassroots* if it is both. It is *volitionally grassroots* if it is oblivious, interactive among the agents that have not left — an agent has left when no transaction they guard can ever again be machine-enabled — and the first interaction of two groups is induced by a transaction guarded by a member of each, so groups become connected only by mutual consent [41]. In a grassroots protocol two instances can always coalesce. The two conditions state formally that grassroots platforms (specified by grassroots protocols) can have multiple instances that can form and operate independently, yet may coalesce into ever-larger instances, possibly (but not necessarily) into a single global platform.

The local state. The contract's speech-act schemas determine the local state of an agent p, all initially empty (Appendix C.2); the transactions below use four of its components: c_p, the friends of p; η_p, a multiset of the agents p knows; κ_p, the children of whom p is a parent; and χ_p, which is 1 if p is a child and 0 otherwise. An agent is a *child* when $\chi_p = 1$ and a *person* otherwise, and $q = p$ may stand for $q \in c_p$ or $q \in \eta_p$, *friend* and *knows* being reflexive. Every transaction requires an established identity at each of its participants and a guard that is not retired, and the transactions below leave these requirements out.

In the transactions that follow, the participants of a transaction are the agents in the extents of the roles of its schema at its binding (Definition B.6): those the binding sends its agent roles to, and, for the closing of a group, every member the leader lists (Appendix C.2); an agent who signed a speech act a transaction carries is not one; those no effect updates are stationary; and of a condition and its mirror, equivalent in every run by Lemma C.1 and Lemma C.2, one side is written. By Lemma C.3, the child of an entering holds nothing and is recorded nowhere before it, so the entering's updates are exact.

Two persons come to know each other by an act of both, and an agent who knows two others may introduce one to the other, by an act of the agent and the other, a person, who then knows the one introduced (GKK:2 and GKK:3); acquaintance makes two agents know each other, an introduction makes one agent know another, and knowing is never undone. Two persons who know each other may befriend (GSG:1). A child comes to know an agent by an act of both and of a parent who knows the agent, and two children by an act of both and of a parent of each, the parents knowing each other (CSSN:2).

An agent may send a friend a message, and the friend then holds its content with the sender (GSN:1).

A group is a set of agents who communicate through its leader, any of whom may create one. Closing a group is one act of the leader and of every member the leader lists, and ends every membership. A post reaches the leader, and the leader delivers it to each member in turn, each delivery having the leader and that member as its participants. A post and a delivery require that the member be the leader's friend, and a member who is not a friend of the leader is removed from the group (GSN:4), so a group's posts travel along the edges of the social graph.

From the contract to transactions. The transactions of Example 4.2 and Appendix C.2 are those the speech-act schemas of the contract (Appendix B.2) determine by the compilation of [23] (Appendix B.3), written in the components of the table above.

Example 4.2 (Child befriending and child unfriending). The two transactions below realise GnW:2 with CSSN:3, and GnW:3, for two children: two children who know each other become friends by an act of both and of a parent of each, the two parents being friends; and a parent of a friend who is a child may end the friendship, the two friends ending it by Unfriend (Appendix C.2). The two children come to know each other by Child acquaint with two children (Appendix C.2), guarded by the same four. The participants of the first are two children r and s, a parent p of r and a parent q of s, and it is guarded by all four; the participants of the second are r, their friend s and a parent p of r, and it is guarded by p. The parents are participants whose states the two transactions leave as they are: they take part by their will. In each transaction, all components not explicitly updated are preserved.

(2) Volition-Guarded Multiagent Atomic Transactions

(1) **Child befriend**(r, s, p, q), where $r \neq s$, $r \in \kappa_p$, $s \in \kappa_q$, $\chi_r = \chi_s = 1$, $\chi_p = \chi_q = 0$, $q \in c_p$ or $q = p$, $s \in \eta_r$, $r \in \eta_s$, and $s \notin c_r$:
$c'_r := c_r \cup \{s\}$ and $c'_s := c_s \cup \{r\}$, guarded by $\{r, s, p, q\}$.

(2) **Child unfriend**(r, s, p), where $r \in \kappa_p$, $\chi_r = 1$, $\chi_p = 0$, $s \in c_r$ and $r \notin \kappa_s$:
$c'_r := c_r \setminus \{s\}$ and $c'_s := c_s \setminus \{r\}$, guarded by p.

4.4 Properties

This subsection proves that the transactions the contract determines give parents, in the digital realm, the three capabilities they have in the physical one — to decide whom their child meets, which gatherings their child joins, and which adults they entrust their

child to — together with the power to end each; and that the protocol is grassroots. The proofs are in Appendix C.2.

Who is a child and who is that child's parent cannot be data of the specification, since a schema names no agent (Definition B.4): they are records that the child's entering creates, and that a declaration of age removes. By Lemma C.3, a child is a child from their first transaction.

A child comes to know an agent by a child acquaintance, which requires the child, the agent and a parent of the child to be willing, and the parent to know the agent, or by a child acquaint with two children, which requires the two children and a parent of each to be willing, and the two parents to know each other (Appendix C.2); knowing is never undone. Child befriending (Example 4.2) requires the two children, who know each other, and one parent of each to be willing, and requires the two named parents to be friends already. Child unfriending is the act of a parent of one of the two friends, the other friend being no parent of the child. The creator of a child may end the guardianship of a parent they made, and their declaration of age, or their retiring of the child, ends every guardianship of the child. The parents' states do not change in child befriending and child unfriending: a parent holds no record of whom their child befriended, and any parent of a child may end any friendship of that child other than one with a parent. When $p = q$ — two children of one parent — the participants of child befriending are $\{r, s, p\}$, and the condition that the two parents be friends holds by $q = p$ (Definition B.7).

A person *consents* to an act when they are a guard of the transaction that carries it out.

Proposition 4.3 (Parents' Capabilities). *In every run of the child-safe social network protocol:*

(1) A child comes to know an agent only with the consent of a parent of theirs, who knows the agent; or, if the agent is a child, with the consent of a parent of each, the two parents knowing each other.
(2) A child befriends an agent only with the consent of a parent of theirs, who is the agent's friend; or, if the agent is a child, with the consent of a parent of each, the two parents being friends.
(3) A child joins a group only with the consent of a parent of theirs, the leader being the child's friend; and of a parent of the leader too, if the leader is a child.
(4) A parent of a child can always end, by their own act, any friendship of the child but one with another parent, and any membership of the child in a group.
(5) A child stops being a child only by an act of their creator.

Clause (5) is why the capabilities are enforced rather than attested: an agent the contract records as a child cannot cease to be one by any act of their own, and so cannot escape CSSN:2, GnW:2 and CSSN:3 without their creator's declaration of age. Whether that agent is in truth a child, and whether the other is in truth their parent, remain out of contract (Section 4.1).

Proposition 4.4 (Grassroots Property). *The child-safe social network protocol (Appendix C.2) is volitionally grassroots.*

Every child has a person on their record, their parent, and two persons can always come to know each other; through their parent a child can come to interact with anyone, by a child acquaintance or a child acquaint with two children. The contract has no act any two agents can always take, and has instead an introduction with wards (Appendix B.4), so the proposition follows from the certification of contracts with wards and with roles that take every agent another participant lists (Theorem B.26).

4.5 What the Contract Enforces, Attests and Undertakes

For each clause the schemas settle whether an agent can breach it and what record of it the contract keeps in another agent's state [23]. A *clause* of a contract 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 the contract *determine* the clause if whether it is kept is a function of the run, and a transaction *may breach* the clause if it occurs in some correct run that breaches it. A transaction is *witnessed against* an agent a if the state of one of its other participants records a after it, and a schema is witnessed against a role if every transaction it determines is witnessed against the agent in that role; a schema adding at one role an atom that names another role is therefore witnessed against that role. Let φ be a clause of the contract, with obligor o where it has one. Then φ is *enforced* if the runs of the protocol realising the contract determine it and every correct run of that protocol keeps it; *attested* if some transaction may breach it and every transaction that may is witnessed against o; and *undertaken* if no transaction may breach it, the runs do not determine it, and some transaction is witnessed against o or o's undertaking is held by another agent. The three are exclusive and are not exhaustive: a clause some transaction may breach without witnessing against its obligor can be breached invisibly, and one the runs neither determine nor record is owed by no one identifiable.

The clauses that describe acts are enforced: each is performed by its schema, and no schema of the contract adds a *knows*, *friend*, *parenting*, *child*, *member* or *identity* atom except those that clause provides for.

CSSN:2 and GnW:2 say that a parent is a participant of every act by which their child comes to know an agent, gains a friend or joins a group, a relation being a friendship or the membership of a group. They are the paper's central claim, and they are enforced. An agent holding *child* has held it since their first act, and gains a *knows* atom only by *enter*, naming the creator who guards it, by *make_parent*, naming the parent the creator makes, or by *child_acquaint* or $child_acquaint_2$, since *acquaint* forbids *child* at both roles and *introduce* at the role of the agent who comes to know; gains a *friend* atom only by *enter*, or by *child_befriend* or *child_adult_befriend*, since *befriend* forbids *child* at both roles; and gains a *member* atom of a group they do not lead only by *child_join* or $child_join_2$, since *join* forbids *child* at the joiner's role. A parent of the child guards each of these.

The second sentence of CSSN:1 — Alice declares themselves a parent of Carol only if they are Carol's legal parent or guardian — is attested. A person may breach it, by entering a child that is not theirs or by becoming a parent of one; the breach is an act of the contract; and *enter* adds at the child's role the atom $creator(p)$, naming the creator, and *make_parent* the atom $knows(q)$, naming the parent made, so they are witnessed against the declaring parent.

Whether the declaration was true is out of contract, and contestable at law.

The obligation of CSSN:4 — the creator shall declare the child of age when the child comes of age, and not before — has two parts. Declaring the child of age before they come of age is attested: a declaration of age may breach it, being taken at a time no run records, since coming of age is not an act of the contract; and it is witnessed against the creator, the child's state recording them after it by the *friend* atom naming them, which *of_age* leaves in place. Declaring the child of age when they come of age is undertaken: the creator breaches it by not acting, so no transaction may breach it; for the same reason the runs do not determine it; and *enter* is witnessed against the creator, since the *creator*(p) atom it adds in the child's state names the person who owes the declaration. This is the price of the ending being the creator's, and the design chooses to pay it: a child able to end the guardianship could leave child safety at will.

CSSN:5 is attested in its prohibition. The transactions that may breach it are a *befriend* of the adult with the child, which leaves the child's state recording the adult, and a *child_adult_befriend* of the adult's own child with the child, guarded by the adult, which leaves their own child's state recording them as a parent; whether the adult knew is out of contract. Its duty to report is kept or breached outside the runs, which neither determine nor record it.

The requirement of GSG:1 that the two persons know each other personally is attested: *befriend* adds at each role a *friend* atom naming the other, so it is witnessed against both [23].

The marks of the boxes of Sections 2 to 4.1 follow: of GKK, GSG and GSN as [23] classes them, GKK:1 undertaken in the uniqueness of the key pair, GSG:1 attested, and the rest enforced; of GnW, all four enforced, as its owner classes them; of CSSN, CSSN:1 and CSSN:5 attested, CSSN:4 undertaken, each by its weakest part, and CSSN:2, CSSN:3 and CSSN:6 enforced.

4.6 The App

System Model. We assume a secure GLP implementation: agents are pairs of a person and a personal machine, persons are identified by self-chosen keypairs, and machines execute the agreed code faithfully, exhibiting crash but not Byzantine failures. A secure implementation of the social graph is ongoing work; recovery from the loss of a person's key or machine, by identity custodians the person designates and by friends as state custodians of the social graph, is the subject of a companion paper [19].

The transactions of Section 4.4 and Appendix C.2 are implemented by vGLP's canonical platform of communicating volitional agents, which carries out each transaction by a two-phase commit coordinated by its initiator [37] (Appendix C.3), compiled into a program in volition-guarded GLP (vGLP) [37]. vGLP extends GLP [66] with volitional procedures, whose goals put a typed question to the person and reduce on the answer. A grassroots platform is realised as such a program, which people load and activate voluntarily on their devices; as noted in Section 1, correctness depends on GLP being secure. The implementation is correct by vGLP's theorem for its canonical platform (Theorem C.12, Appendix C.3): the parents' capabilities of Proposition 4.3 hold in every run of it, and a parent's ending of a relation is carried out in every run that meets the theorem's conditions.

4.6.1 The Platform as a vGLP Program. GLP [66] extends logic programming with paired variables: each writer X has a paired reader $X?$, assigned at most once, on which a reader suspends until the writer is assigned. A channel pairs a read end and a write end; what one agent writes, the paired agent reads. In vGLP a procedure may put a question to the person: a *volitional procedure* is declared `procedure (T)*p(T1, ..., Tn)`, `T` its *interactive type*, and each of its clauses, `(A)*p(S1, ..., Sn) :- Tests | Body`, carries an interactive term `A` matched against the person's answer, so a clause that reads the answer reduces only once the person has given it [37]; the interface is derived from the interactive types, each question rendered by the widget its type declares or, where none is declared, by the default widget built from the type's constructors [37].

The guards of a transaction and the volitional procedures of the program are the same requirement at two levels. A transaction guarded by $\{r, s, p, q\}$ is carried out only when all four are willing; in the program, each of the four answers the question of a volitional procedure of its own agent, and the transaction is committed only once all four have. A guard who does not answer leaves their reader unbound, the reading goal suspends, and the transaction is not carried out.

The program. The program is the canonical platform of the contract, compiled by vGLP's compiler from the contract's speech-act schemas in the term form the checker of [23] reads, with *knows* its introduction predicate, by which an agent introduces to each other, at the level of the machines, every two agents its person knows, and with the introductions by which the initiator of a committed transaction introduces its other participants to each other (Appendix C.3) [37]. Its agent asks its person for their requests, proposes each as an attempt, and asks its person to approve the attempts other agents propose to it.

The program in the grassroots app. The mini-app's entry is declared as the instance, at the mini-app's own root-channel type, of the one entry template the super-app declares [18, 70]. The super-app activates it with two channels, at types the super-app declares: the person channel, on which the program's questions reach the person and the person's input comes back [37], and the channel on which the social graph hands it a conversation with each person its agent knows. The type of the conversation between two executions of the mini-app is the mini-app's own and opaque to the super-app, which holds no declaration of it [70]. The mini-app holds no network entry and calls no predicate that reaches the network or the person, so the program is certified. Every person's mini-app is activated by the one entry: an adult's then establishes their identity on their request, and a child's identity is established by its entering, which its parents request and approve at the setup of the child's phone.

Files. Shared types, the person's among them, are declared in the scope file `self.glp`, visible to all modules; the type constructors `Stream(X)`, `List(X)` and `Channel(In,Out)` are declared in the root `programs/self.glp`, the ancestor scope of every program. Beside the program are the plays, the village among them; a play hands every two agents a conversation, in place of the super-app,

and the agents come to know each other through the platform (Appendix C.3).

The schema the compiler reads. The compiler reads each speech-act schema as a term of the checker of [23]: a schema is its name, the names of its roles as the clause gives them, its variables and its roles, a role is `rs(G, A, D, R, F)`, its guard and the atoms it adds, deletes, requires and forbids, and an atom refers to a role by its index. Child befriending's, as `programs/cssn/CSSN.glp` has it, its roles r, s, p and q in order (Appendix B.2):

```
schema(child_befriend,
 ['Carol', 'Dave', 'Alice', one('Bob', 'Alice')], [],
 [rs(guarded, [atom(friend, [role(2)])], [],
     [atom(identity, []), atom(child, []),
      atom(knows, [role(2)])],
     [atom(friend, [role(2)]), atom(retired, [])]),
  rs(guarded, [atom(friend, [role(1)])], [],
     [atom(identity, []), atom(child, []),
      atom(knows, [role(1)])],
     [atom(friend, [role(1)]), atom(retired, [])]),
  rs(guarded, [], [],
     [atom(identity, []), atom(parenting, [role(1)]),
      atom(friend, [role(4)])],
     [atom(child, []), atom(retired, [])]),
  rs(guarded, [], [],
     [atom(identity, []), atom(parenting, [role(2)]),
      atom(friend, [role(3)])],
     [atom(child, []), atom(retired, [])])])
```

From it the compiler generates the precondition and effect at each role and the question that asks each guard to approve. The record of the run of the cascade on the contract, `cascade/CSSN` in the repository at the tag `cascade/CSSN/2026-10-25-2`, commit `852c3b07`, holds what it generated. The VMAT form, `vmat.tex` (lines 2938–3160), prints the transaction with the four roles named Alice, Bob, Carol and Dave in the order r, s, p, q — the printer's names, the first two the children here — and with c_r for the local state of r, a multiset of atoms; at r it is the effect $c_r' := c_r \uplus \{friend(s)\}$, provided $identity, child, knows(s) \in c_r$ and $\{friend(s), retired\} \cap c_r = \emptyset$, and the transaction is guarded by all four. The platform, `platform.vglp`, holds the schema as the agent reads it, the class `child_befriend(Peer, Peer, Peer, Peer)` in its class type and in its request type, the binding of a request to the four roles, and the question put to each guard: the interactive type `Approval ::= approval(Asking, List(Word), YesNo?)` and the volitional procedure `(Approval)*respond` that puts it (lines 57, 111, 326, 1007 and 1198–1203). Code checks the contract, prints the schemas' notation and runs the cascade whole, and the record's `schemas.txt` (line 90) prints the schema above as `child_befriend(Carol?, Dave?, Alice?, one(Bob, Alice)?)` in the notation of Appendix C.1, the roles named as the clause names them, the two parents possibly one person.

Child befriending in vGLP. Here are illustrative fragments of the program, in which `...` elides alternatives and arguments. A person's own question is a menu, as in sGLP's social graph [69]: it shows the person the persons their agent knows, and takes one request, an act of the contract with the persons it names:

```
Person  ::= Key.
Class   ::= child_meet_2(Person, Person, Person, Person)
          ; child_befriend(Person, Person, Person, Person)
          ; child_unfriend(Person, Person, Person) ; ... .
Request ::= propose(Class) ; ... .
Menu    ::= menu(List(Person), Request?).

%% Show the person the persons they know, and hand
%% them the request.
procedure (Menu)*ask(List(Person)?, Request).
(menu(Ks?, R))*ask(Ks, R?).
```

An attempt another agent proposes is put to the person for approval, showing the attempt — its initiator, two numbers by which concurrent attempts are ordered, the act and the persons it names — with the question the contract puts to the person's role, and taking yes or no:

```
Attempt  ::= attempt(Person, Integer, Integer, Class,
                     Names).
Asking   ::= Attempt ; hello(Person, Proposal).
YesNo    ::= yes ; no.
Approval ::= approval(Asking, List(Word), YesNo?).

%% Show the person an attempt with its question,
%% and hand them yes or no.
procedure (Approval)*respond(Attempt?, List(Word)?,
                             ...).
(approval(A?, Ws?, Y))*respond(A, Ws, ...) :- ... .
```

Both interactive types are in writer mode. `ask` writes the agents the agent knows and hands the person the request, which stays open while messages come and go; each request the person makes is followed by a new menu. Alice's request `propose(child_befriend(Carol, Dave, Alice, Bob))` names her child, Bob's child, herself and her friend Bob, in the order of the roles r, s, p and q of the schema; Alice's agent knows Dave's because Bob's agent introduced them, Bob knowing both, and Carol and Dave know each other by the child acquaintance Alice requested before, with Bob (CSSN:2). `respond` writes the attempt and hands the person yes or no, and the agent votes with the answer. The four guards of *child_befriend* are thus one request and three approvals, each a volitional reduction of its own agent, and the two-phase commit carries the transaction out on a yes from all three (Appendix C.3). The code that proposes, votes and commits is the compiled canonical platform, proved correct in [37], and is not shown.

4.6.2 The User Interface. Since a volitional reduction requires the person's answer, a correct implementation must elicit it; this is the reason for the user interface [37]. An open question is presented as the widget of its interactive type, which shows what the program writes for the person and forms the person's input into a term of the type. By the default widgets, which draw a question whose type declares no widget, a tuple is a form headed by its functor, with one field per argument; a union of constants is a row of buttons; any other union is a menu, one item per alternative, each drawn by its widget; and a value the program writes is shown and not edited [37]. The mini-app declares, for its two questions, the two widgets the grassroots app has for every canonical platform [18, 37]: an approval is drawn as one sentence for the attempt's class, "Alice asks that their child Carol and your child Dave befriend.", and two buttons labelled for it, Approve? Yes and No; and a request as a sentence the person completes by choosing a person into each of its places, "Ask that Carol and Dave befriend, with Bob.", and sends, the persons the agent knows listed above it, each with what the person's state says of them. The words in which each act of

the contract is put to a person are the one thing written for the platform's screen.

The child-safe platform's constructs follow from its acts.

- **Acquaintance** has two guards and two constructs: a person's request to acquaint, naming the other, and the approval asked of the other; **an introduction** likewise, the introducer's request, naming the two, and the approval asked of the agent introduced to. **A child's acquaintance** has three constructs, or four for two children, as a child's befriending has: a parent's request, naming the agents, and an approval asked of each other guard.
- **An identity** is established at the setup of a phone: an adult's by their own request, and a child's by the request of their creator to enter them, which the child's agent asks its person to approve — the creator, holding both phones, answers on the child's phone before handing it over. **Making a parent** is the creator's request, naming the adult and the child, which the adult and the child are asked to approve. **A declaration of age**, the ending of a parenthood the creator made, and **retiring** a child are the creator's requests, and nobody is asked. The obligation to make the declaration when the child comes of age, and not before, is the creator's, not the protocol's (Section 4.5).
- **Child befriending** has four guards and therefore four constructs: a parent's request, naming the four, and an approval asked of each of the other three. The four are the four roles of *child_befriend*; where one parent has both children, the two parent roles are one guard, asked once, and there are three constructs. **Child-adult befriending** has three: the request of one of them, and an approval asked of each of the other two.
- **Child group joining** has three constructs, or four when the leader is a child: the leader's invitation, and an approval asked of the child and of a parent of each child among them.
- **Ending** a friendship or a membership, and closing a group, are guarded by one person, so each is a request of a person who may end it, and nobody is asked. A parent's closing of a group their child leads is proposed by the child's agent, which holds the list of members, on the parent's request.

An approval, once given, holds until its attempt closes, and cannot be withdrawn [37]; a guard who has not answered may answer no.

A menu shows the agents its agent knows, each with what the person's state says of them: a parent's shows the agents they know, their friends and their children among them, and the agents their agent was introduced to at the level of the machines, their children's friends and the leaders of their children's groups among them (Appendix C.3), so a parent can name any of them in a request, though the contract records none of the children's friends at the parent. The menu offers every request the person's own state allows, and an attempt that another participant's precondition refuses is shown to the person as refused: an adult's row for a child the adult's state does not record as one offers Acquaint, which the child's agent refuses. A child's menu offers introducing, which the contract allows a child, the agent introduced to being a person, and offers neither acquaintance nor adding a person by key: a child comes to know an agent only by a child acquaintance a parent requests (CSSN:2), and the child's agent discovers only its parents' agents, at the setup of the child's phone (Appendix C.3). A person's pending asks are the approvals they owe. A request the person makes while their agent is busy with another attempt is held and taken when the agent is free, and the person sees each request they made until it is fulfilled or refused, so a person who makes two requests one after the other — Bob's two introductions in the village — sees the first taken before the second is [37].

4.6.3 Village Demonstration. To demonstrate that the acts compose into a functioning social network, we run a village of seven agents: Alice, Bob, Grace and Frank, and Carol, Dave and Eve. The scenario runs eight acts:

(1) **Coming into the contract.** Alice, Bob, Grace and Frank establish their identities and come to know each other: Alice and Bob, Bob and Frank, and Bob and Grace. Bob enters Dave and Eve, and makes Grace a parent of each, Grace and the child approving; Alice enters Carol. The children's phones are set up by their creators, who become their first friends.
(2) **Adult friendships.** Alice befriends Bob and sends Bob a message; Bob befriends Frank; Bob introduces Frank to Alice, an act of Bob and Alice, and Alice to Frank, an act of Bob and Frank, and Alice befriends Frank.
(3) **Child friendships.** Alice requests that Carol and Dave come to know each other, naming Bob, and Bob, Carol and Dave approve — a child acquaint with two children, four guards; then that they befriend, and the three approve again — a child befriending, four guards. Bob requests that Dave and Eve come to know each other, then befriend: the sibling case, where the two named parents are one person and the preconditions that the parents know each other and are friends hold trivially. Alice requests that Carol and Eve come to know each other, then befriend, naming Bob.
(4) **Child-adult friendships.** Bob requests that Dave and his friend Frank come to know each other, then befriend, and Alice that Carol and Frank come to know each other, then befriend — each a child acquaintance and then a child-adult befriending, guarded by the child, the adult and the parent.
(5) **Child-managed group.** Carol creates a group `kids_chat` and invites her friends Dave and Eve; since Carol is a child, each invitation requires the leader, the joiner, and a parent of each to be willing. Carol, Dave and Eve exchange messages, each posted to Carol and delivered by her to the other members. Eve then leaves the group.
(6) **Adult-managed group.** Frank creates a study group and invites his friends Dave and Carol, each with the approval of the child's parent. Frank, Dave and Carol chat in the group.
(7) **Parents ending.** Grace ends Dave's friendship with Carol, which Bob had approved, and Carol's agent drops Dave from `kids_chat`, Dave being no longer her friend (GSN:4); Bob ends Dave's membership of the study group; and Alice closes `kids_chat`.
(8) **Coming of age.** Bob, Eve's creator, declares Eve of age, which ends Grace's guardianship of Eve too. Eve is thereafter an adult: she comes to know Frank and befriends him, with no parent among the guards; Bob and Grace are no longer her parents, and Eve and Bob remain friends.

The village runs twenty-three of the twenty-eight transactions of Appendix C.2, all but Unfriend, Close group, Join group, End parent and Retire. All seven agents run concurrently as GLP processes, communicating by typed message streams; each person is scripted, reading the draws and granting on the constructs (Section 4.6.4). Figure 4 shows the seven persons' screens. The full implementation — the program, its compilation and the village scenario — is available at https://github.com/EShapiro2/GLP/tree/main/programs/cssn. Section 5 simulates the child-safe social graph in a village of ten families.

4.6.4 In the Grassroots App. One Dart bridge draws the constructs of any compiled vGLP program — the requests and the approvals its construct processes put to the person — and the lists the program writes to its person, each by the widget its type declares or by the default widget of the type, and returns the person's raw input as grants on the person channel; everything crossing to Dart is a ground term or a construct identifier, and the answers and the channels stay on the GLP side [37]. A new mini-app adds to the construct family the words of its acts and nothing else: it gains its screen, an icon and its alerts from its types. The deployed build implements the social graph, one-to-one conversations and grassroots bonds, as a native app on a physical smartphone [37].

The child-safe platform runs as a certified mini-app of the grassroots super-app [18], under which two participants become friends and a message passes; the super-app knows nothing of the relation between a parent and their child, it being clauses and guards in the mini-app's own program. It introduces no new construct: its requests and approvals are those of Section 4.6.2. The parent's approval is asked in the parent's own app, on the parent's own phone, and the child's act does not occur until the parent answers it.

Figure 5 shows the mini-app in a scenario of four people. Alice enters Carol, and Bob enters Dave, each the creator of their child; Alice and Bob come to know each other and befriend; Alice requests that Carol and Dave come to know each other, naming Bob, and Bob, Carol and Dave approve; Alice requests that Carol and Dave befriend, naming Bob, and the three approve again; Carol ends the friendship with Dave; and Bob declares Dave of age. The other three persons are scripted, and read the draws and grant on the constructs [37]. The bridge draws the constructs the compiled program's dispatcher draws, an approval and a request by the two widgets the mini-app declares, in the words of its acts (Section 4.6.2), and everything else by the default widget of its interactive type, and returns the person's input as grants.

Use of Large Language Models. The AI disclosure of Section 7 states how each artefact was made.

5 Simulating the Child-Safe Social Graph

A village of seven people exercises the acts of the contract; how a population of parents and children fares under it needs many people. A grassroots platform ought to be evaluated at scale before it is deployed, and enlisting the many people needed to do so is a contradiction in terms [69]. Stochastic GLP (sGLP) [69] simulates a grassroots platform from the vGLP program that deploys it, with stochastic agents in place of people: each question the program puts to its person is answered by a *stochastic person*, an sGLP program typed by the dual of the question's interactive type, which answers at the rate its profile declares with an answer its profile draws. Every run of a vGLP program in which stochastic persons replace the people is a run of the vGLP program [69], so a simulation shows a behaviour of the program the phone runs. sGLP simulated the grassroots social graph with a population of men and women of four profiles [69]; here we simulate the child-safe social graph in a village of families, and ask three questions: how many of the parents' requests for a child friendship commit, and how long those that do take; how many approvals a parent answers each month; and how far the children's friendship graph lags behind the parents' in becoming one connected component.

The program. The program simulated is the child-safe mini-app of Section 4.6.1, without the super-app. Each question it puts to its person is answered by that person's stochastic person, which sees what the question shows and what the program has shown the person [69]: the menu shows the persons the agent knows, an approval shows the attempt, and the program writes its person their state, their friends and their children among them (Section 4.6.1). The persons take the acts of the child-safe social graph but the declarations of age: establishing an identity, acquaintance and introduction, a child's entering with their parents, befriending and ending a friendship, a child's acquaintance with a child or an adult, child befriending and child-adult befriending, and a parent's ending of a child's friendship (Examples C.5, C.6 and C.9).

The population. A *census* draws the persons of a run from its seed [69]: a village of ten families; a family has two parents 70 times in 100 and one otherwise, and one, two or three children, equally likely; the village of the runs has 17 parents and 20 children. A person has a *role*, parent or child, and attributes: their village and family, and the persons they know at the start, from outside the app — a parent knows the parents of their village and their children, and a child their parents; a parent's attributes carry the table of the families besides, which is how a parent knows whose child a child is. Knowing by the contract (GKK:2 and GKK:3) is what the run forms and reports: a person acquaints, in the app, with the persons they know at the start, one per menu, each after an hour on average, the other approving the acquaintance at once; a parent introduces a friend to another; and a child comes to know an agent by a child acquaintance a parent requests (CSSN:2). A person's first acts are their answers to their first menus [69], each after an hour on average: an adult establishes their identity, acquaints with the persons they know at the start, and the first parent of a family enters the children and makes the second, whom they have met, a parent of each; the children remain children for the run.

The profiles. A dimension of profiles is of one role [69]: a parent has two and a child one.

- A parent's *approach* is *local* or *open*, drawn half and half, and answers the menu. Once their children are entered, a parent answers it 670 times a year on average, about twice a day: 45 times in 67 with an offer of friendship to a parent they know who is not yet a friend, a local parent to a parent of their own village 95 times in 100 and an open one 60 times in 100, uniformly from the list; 6 times in 67 introducing a friend of theirs to another, both adults, drawn uniformly; 6 times in 67 with a request that a child of theirs and a child of a friend, whom they know, come to know each other, the child, the friend and the friend's child



Figure 4: The village scenario of Section 4.6.3, as the play drew it. Each panel is one person's screen over the run: what the person requested and approved, and what the program wrote to them. The agents run concurrently as GLP processes, so a panel may show the requests and approvals of two acts interleaved.

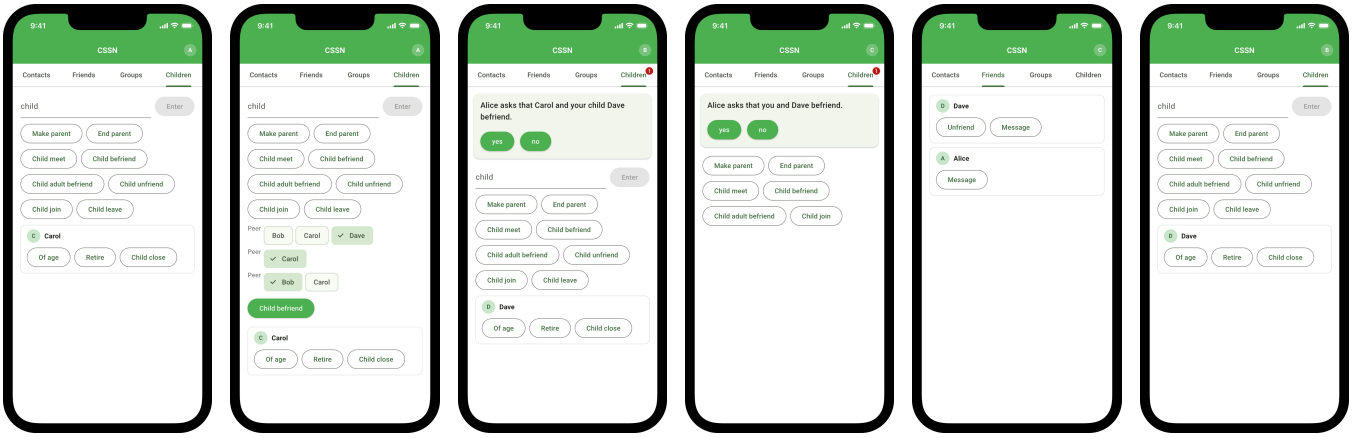


Figure 5: The child-safe mini-app in the grassroots app, on the iPhone simulator, in the scenario of Section 4.6.4, one person's screen each: (1) Alice's Children tab, after she and Bob come to know each other and befriend: the field to enter a child, the buttons of the acts that name two or more, and her child Carol, with the acts that name only Carol; (2) her request that Carol and Dave befriend, after the one that they come to know each other, Carol, Dave and Bob picked into its places; (3) Bob asked to approve it, with yes and no; (4) Carol asked to approve it; (5) Carol's Friends, Dave, with Unfriend and Message, and Alice, her creator and first friend, with Message, whom she may not unfriend; (6) Bob's Children tab after Carol ends the friendship, with Of age beside Dave, Bob being Dave's creator. Every word on a screen is the contract's, but yes and no.

drawn uniformly; 6 times in 67 with a request that a child of theirs and a child the agent knows befriend, which fails where the two children do not know each other, a parent holding no record of whom their child knows; once in 67 with a request that a child of theirs and a friend of theirs come to know each other, and once in 67 that they befriend; once in 67 with the ending of a friendship of a child of theirs with a child the agent knows, which fails where the two are not friends, a parent holding no record of their child's friends; and once in 67 with the ending of a friendship of their own, drawn uniformly from their friends who are not their children.

- A parent's *policy* is *strict* or *permissive*, and answers an approval, after two hours on average: an acquaintance offered by a person known at the start, and an introduction, are approved at once; an offer of friendship is approved 90 times in 100; an acquaintance, a befriending or a joining of a child, their own child's or another's with them, 20 times in 100 by a strict parent and 80 by a permissive one; the entering of their child with the other parent at once.
- A child's *response* is *choosy* or *keen*, and answers both questions: asked to approve an acquaintance, a befriending or a joining, a choosy child approves 50 times in 100 and a keen one 90, after two hours on average, and their entering at once; and ten times a year on average the child ends a friendship, drawn uniformly from their friends who are not their parents.

The census, the profiles and the run declaration are the contract's sGLP source, `programs/cssn/CSSN_sglp.glp` in the repository, over the platform's interface. The local parent's menu, the strict parent's approval and the run declaration, in sGLP's surface syntax [69], . . . eliding arguments:

```
person local.
Menu =::= local_menu.
local_menu(menu(Me, Ks, R?), person(_, _, As), S) :-
    seen(L),
    cs_parent_menu(95, Me?, Ks?, As?, L?, S?, slot(R)).
```

```
cs_parent_act(K, ..., slot(W?)) :- K? =< 45 |
    cs_offer(..., slot(W)).
cs_parent_act(K, ..., slot(W?)) :- K? > 45, K? =< 51 |
    cs_introduce(..., slot(W)).
cs_parent_act(K, ..., slot(W?)) :- K? > 51, K? =< 57 |
    cs_cm_request(..., slot(W)).
cs_parent_act(K, ..., slot(W?)) :- K? > 57, K? =< 63 |
    cs_child_request(..., slot(W)).
cs_parent_act(64, ..., slot(W?)) :-
    cs_child_meet_request(..., slot(W)).
cs_parent_act(65, ..., slot(W?)) :-
    cs_child_adult_request(..., slot(W)).
cs_parent_act(66, ..., slot(W?)) :-
    cs_end_child_friendship(..., slot(W)).
cs_parent_act(67, ..., slot(W?)) :-
    cs_end_friendship(..., slot(W)).

person strict.
Approval =::= strict_approval.
strict_approval(approval(hello(_, _), _, A?), _, _) :-
    cs_say(slot(A), yes).
strict_approval(approval(attempt(_, _, _,
      make_parent(_, _, _), _), _, A?), _, _) :-
    cs_say(slot(A), yes).
strict_approval(approval(attempt(_, _, _,
      acquaint(_, _), _), _, A?), _, _) :-
    cs_say(slot(A), yes).
strict_approval(approval(attempt(_, _, _,
      introduce(_, _, _), _), _, A?), _, _) :-
    cs_say(slot(A), yes).
strict_approval(approval(attempt(_, _, _, C, _), _, A?),
                _, S) :-
    otherwise |
    cs_parent_answer(20, C?, S?, slot(A)) @ 12/day.

run cs_village1
    [ approach of parent ~ (local : 0.5 ; open : 0.5),
      policy of parent ~ (strict : 1.0 ; permissive : 0.0),
      response of child ~ (choosy : 1.0 ; keen : 0.0) ]
    until 1 year seed 1.
```

`seen(L)` gives the person procedure what the person has been shown, the last first, and `cs_parent_menu` reads from it the agent's state, establishes the identity, makes known the persons known at the start one by one, enters the children, each after an hour on average, and then draws `K` in 1 to 67 and the act by it, at the rates above; `cs_parent_answer` draws yes or no by the class of the attempt.

The runs. Each run is one simulated year of the village, from seed 1. Policy and response make four mixes — all parents strict or all permissive, all children choosy or all keen — and each mix is one run; a parent's approach is drawn half and half in each.

Table 1: The runs by the mix of policy and response, one village of 17 parents and 20 children for one year: the child friendships at the end of the year; the parents' requests for a child friendship that committed, of those made; the days from a request to its commit, mean and most; the approvals a parent answers a month, months 2 to 12; and the connected components of the children's friendship graph at the end of the year. The parents' graph is one component by the end of month 1 in every run.

policy	response	friendships	committed	days	approvals	components
strict	choosy	0	0 of 105	—	2.1	20
strict	keen	0	0 of 105	—	2.1	20
permissive	choosy	1	2 of 104	0.5, 0.7	2.2	19
permissive	keen	8	9 of 92	2.6, 4.1	2.0	12

What is measured. From the log of a run, which records each answer, and each element the person was shown, with its simulated time [69], from a monitor that checks, at the commit of each act a parent guards, its guards' approvals and its precondition, and from the agents' records at its end:

(1) the acts a parent guards — a child's friendships, joins and their endings, an entering and a declaration of age — that the monitor finds committed without the approval of each of their guards, or with their precondition failing: the exceptions to Proposition 4.3;
(2) the fraction of a parent's requests for a child friendship that commit, the time from such a request to its commit, and the child friendships at the end of each month;
(3) the approvals each parent answers each month;
(4) the month by whose end the parents' friendship graph, and the children's, is one connected component, and the graph of knowing, by the contract, at the end of the run;
(5) the wall-clock time and the memory of a run.

Results. The monitor found no exception to Proposition 4.3: in each of the four runs, every act a parent guards was committed with each of its guards approving and its precondition holding, as Corollary C.13 requires of the implementation. Table 1 shows the runs by the mix. The parents befriend quickly: the 17 parents are one component by the end of the first month in every run, and all 136 pairs are friends by the end of the year. The children befriend slowly: strict parents' children gain no friendship in the year, and permissive parents' keen children eight, in months 4, 6, 8, 10 and 12, the twenty children ending the year in twelve components. Two things account for it. A child friendship takes the yes of four persons, and at the rates of the profiles the four say yes together 1 time in 100 for strict parents and choosy children and 52 times in 100 for permissive parents and keen children; and a parent's request for a friendship of their child with a child the parent knows is refused by the child's agent where the two children do not know each other, which the parent cannot see, since the contract records nothing of a child's acquaintances at the parent: of the 92 requests of the permissive and keen run, 9 committed, 12 were declined by a guard, and the rest were refused so. A parent answers about 17 approvals in the first month, the acquaintances and friendships of the start, and about 2 a month after it. Each run of 37 agents for a simulated year took 64 to 70 minutes and at most 433 MB.

6 Related Work

6.1 Child Safety in Social Networks

A substantial empirical literature documents the harms to children from commercial social networks [33, 35, 51, 76, 80], attributing them to the design choices flowing from the surveillance-capitalism business model [81]. Governments have responded with legislation—COPPA and its 2025 update [21, 77], the proposed US Kids Online Safety Act [78], the EU Digital Services Act [14], the UK Online Safety Act [54], Australia's under-16 ban [53]—but legislation regulates platform behaviour without changing the underlying architecture. Major platforms offer parental controls [3, 28, 49], dedicated child-only services [26, 48], and institutional educational platforms [25, 50]: operator-controlled features, routinely circumvented by self-declared ages [75], which leave the platform in possession of the child's data and in control of the child's experience. Akter, Park and Wisniewski argue for community-based approaches to adolescent online safety beyond parental control [2]; a community's contract can grant teenagers such autonomy (Section 4.2).

Messenger Kids [48] comes closest to the capabilities of Section 1. Parents approve their child's contacts from a dashboard in their own Facebook app, may let the child add and remove contacts under their supervision, and may approve a teacher or coach who connects children in a group, only children whose parents approved that same adult [45, 46]. Table 2 compares it with CSSN. The two differ in who verifies identities, who runs the service and who sets the rules. Meta verifies no identity, and takes action on fake Facebook accounts by the billion — 1.2 billion in January to March 2026 [47]; it runs the service, holds every message, and sets one set of rules for all. In CSSN the social graph verifies identities: two persons befriend only if they know each other personally (GSG:1), which the contract attests [60]; the families' phones run it; and each community sets its own contract.

Decentralised alternatives to centrally-controlled social networks have been explored for two decades: federated architectures such as Mastodon [59] and Matrix [74], the AT Protocol underlying Bluesky [38], relay-based protocols such as Nostr [52] and Farcaster [20], peer-to-peer and distributed social networks [7, 11, 12, 32, 36], blockchain-based protocols such as DSNP [56, 58], and peer-to-peer mesh protocols such as Secure Scuttlebutt [73]. While these proposals subvert centralised control, they still require members to surrender personal information to third parties—federated-server operators, DHT agents, or blockchain validators—and none addresses child safety as a design concern: any child with a keypair can participate.

The shared structural failure is twofold. Centralised and federated platforms remain *autocratic*: an operator—a corporation or a federation administrator—retains authority over participants. Blockchain-based platforms are *plutocratic*: control follows capital—validator stake, mining power, governance tokens. Neither admits the third structural option pursued here, *egalitarian* control by participants themselves, under which parental responsibility can be exercised digitally as it is in the physical realm.

Table 2: Messenger Kids [45, 46] and CSSN compared.

	Messenger Kids	CSSN
who verifies identities	nobody	the social graph: people befriend people they know
who runs the service	Meta	the families' phones
who holds the messages	Meta	the sender and the recipients
who sets the rules	Meta, one set for all	each community, its own contract
how a child befriends	a parent approves each contact, or lets the child add contacts and may override	both children and a parent of each will it, the parents being friends
how adults connect children	an adult the parents approve puts their children in a group	a child joins a group of an adult friend, a parent approving
what parents see	whom the child chats with, recent photos and videos, and every message on request	the acts they are asked to approve; none of the child's messages
how a parent ends a connection	removes a contact from the dashboard	ends the friendship or the membership, or closes the child's group

6.2 Formal Foundations

Atomic transactions in distributed computing. Atomic transactions have been investigated early in distributed computing, mostly in the context of database systems [39, 42, 43]. Most research since and until today focuses on their efficient and robust implementation [6, 10]. The integration of atomic transactions in programming languages has also been explored [5]. In terms of formal models of concurrency, the extension of CCS with atomic transactions has been investigated [1, 16, 17].

Multiagent transition systems and grassroots protocols. The formal framework employed here builds on multiagent transition systems [61] and their extension with atomic transactions [67]. Previous work [41] extends this framework with guard annotations (Definition 4.1), transaction equivalence, and liveness. Here we apply volition-guarded transactions to specify a child-safe social network.

Digital social contracts. The notion of a digital social contract was introduced by Cardelli et al. [8] as the digital incarnation of Proudhon's social contracts [57]. Grassroots platforms [63, 67] realise digital social contracts as programs that people load and activate voluntarily on their smartphones.

Contract specification languages. The formal study of contracts is dominated by the contract-as-automaton family — contract automata [4], a loan agreement as a finite automaton [22], the L4 language [79], timed contract automata [9], encodings as defeasible deontic rules [30], and the comparison of imperative with declarative smart contracts [29]. 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 the conditions on speech-act schemas [23] quantify over the family of systems a contract induces on every finite set of people, need a local state per party, and forbid a participant that is not a party. Symboleo [55, 71] and Stipula [15] are the nearest and make the same three commitments; Catala [44] does the corresponding work for statute. Speech-act schemas [23] are the language used here, and the checker that decides the conditions on them is that paper's. Child befriending is not directly expressible in Stipula: it takes the will of both parties, it is available repeatedly between any two of them, and Stipula's joint consent is only at contract formation, so each child friendship would be the formation of a new contract.

6.3 Grassroots Platforms

Grassroots platforms [63] have been instantiated across a range of application domains, including grassroots social networks [64], grassroots cryptocurrencies [40, 62], and grassroots federations [34, 72]. Grassroots Logic Programs (GLP) [66] serves as the implementation language; the child-safe social network presented here is implemented in GLP (Section 4.6).

6.4 Secure Scuttlebutt

The protocol closest to grassroots social network in spirit and goal is Secure Scuttlebutt (SSB) [73], a peer-to-peer protocol and self-hosted social media platform in which each participating computer acts as a relay. SSB offers single-authored, authenticated, append-only logs, which relays replicate along a social graph: a person follows the identities whose logs they want, and their relay replicates those logs and, transitively, those of friends of friends up to a configurable distance, while block messages limit the replication [73].

SSB and grassroots social network differ in several respects. In SSB a person follows another without the other's consent, and following may be one-sided; in a grassroots social network friendship is mutual, taking the will of both, and in CSSN a child's friendship takes the will of a parent as well. SSB relays replicate logs over TCP connections, with highly available relays at published static IP addresses [73]; grassroots dissemination is connectionless and geared for peers with dynamically-changing IP addresses communicating via unreliable messages, namely smartphones.

7 Conclusion

We have carried a child-safe social network from a grassroots social contract to an app. The contract's clauses (Section 4.1) become speech-act schemas (Appendix B), the schemas compile into volition-guarded multiagent atomic transactions (Sections 4.3 and 4.4), and the transactions are implemented by vGLP's canonical platform, compiled into a volition-guarded GLP program whose volitional procedures determine the user interface (Section 4.6). Only the first step is left to judgement. The platform runs as a mini-app of the grassroots app, and, simulated in sGLP from the same program

in a village of ten families for a year, its parents befriend within a month and its children slowly, a child friendship taking the yes of four persons and a parent's request failing where the children do not know each other, which the parent cannot see.

The contract gives parents in the digital realm the three capabilities they have in the physical one: to decide whom their child meets — whom the child comes to know, and whom the child befriends — which gatherings their child joins, and which adults they entrust their child to; and a parent may end each friendship and membership by their own act. These are preconditions and guards of the acts themselves, enforced by the contract rather than by an operator's policy (Proposition 4.3), and every child's genuine digital identity is established by their parents, so a child is a child from their first act. The protocol is grassroots, every child having a person on their record and two persons always being able to come to know each other, and the contract meets the conditions that certify a contract with wards (Proposition 4.4), and its implementation is correct by vGLP's theorem (Theorem C.12). The contract is written at the level on which parents and educators discuss these matters, and each community may adopt and revise its own.

Several questions remain open. First, custodial arrangements change, and the contract provides for a creator making and unmaking a child's other parents and declaring the child of age, but not for the transfer of the creator's part to another parent. Second, a secure GLP runtime is required to fulfil the implementation assumption that agents execute the agreed code; it is in development as a companion effort. Third, a child come of age whose parent does not declare it can establish a second identity on a new phone; mutual sureties among the owners of genuine personal identifiers [60] bound such sybils, and are not part of this contract.

AI Disclosure. The authors used Anthropic's Claude models throughout, under their direction and review, and made all design decisions. The clauses of the contract are the authors', drafted with Claude. The speech-act schemas were written by Claude from the clauses and checked by the checker of [23], itself written with Claude. The program is not written by hand: vGLP's compiler [37], written with Claude, compiles it from the schemas. The plays of the village and the profiles of the simulation are written by Claude. The proofs and the prose were drafted with Claude. The authors verified the correctness and originality of all content, including all references, and responsibility for all content rests with them.

References

[1] Lucia Acciai, Michele Boreale, and Silvano Dal Zilio. 2007. A concurrent calculus with atomic transactions. In *European Symposium on Programming*. Springer, 48–63.

[2] Mamtaj Akter, Jinkyung Katie Park, and Pamela J. Wisniewski. 2025. Moving Beyond Parental Control toward Community-based Approaches to Adolescent Online Safety. Position paper, ACM CHI 2025 Mobile Technology and Teens Workshop, arXiv:2503.22995. https://arxiv.org/abs/2503.22995

[3] Apple. 2024. Use Screen Time to manage your child's iPhone or iPad. https://support.apple.com/en-us/108806. Accessed 2026.

[4] Shaun Azzopardi, Gordon J. Pace, Fernando Schapachnik, and Gerardo Schneider. 2016. Contract automata. *Artificial Intelligence and Law* 24, 3 (2016), 203–243. https://doi.org/10.1007/s10506-016-9185-2

[5] Johannes Borgström, Karthikeyan Bhargavan, and Andrew D Gordon. 2009. A compositional theory for STM Haskell. In *Proceedings of the 2nd ACM SIGPLAN Symposium on Haskell*. 69–80.

[6] Manuel Bravo and Alexey Gotsman. 2019. Reconfigurable atomic transaction commit. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. 399–408.

[7] Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. 2009. PeerSoN: P2P social networking: early experiences and insights. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*. 46–52. https://doi.org/10.1145/1578002.1578010

[8] Luca Cardelli et al. 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

[9] 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. https://doi.org/10.3233/FAIA220460

[10] Gregory Chockler and Alexey Gotsman. 2021. Multi-shot distributed transaction commit. *Distributed Computing* 34 (2021), 301–318.

[11] Gregory Chockler, Roie Melamed, Yoav Tock, and Roman Vitenberg. 2007. Constructing scalable overlays for pub-sub with many topics. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*. 109–118.

[12] Gregory Chockler, Roie Melamed, Yoav Tock, and Roman Vitenberg. 2007. Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication. In *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*. 14–25.

[13] George Coker et al. 2011. Principles of Remote Attestation. *International Journal of Information Security* 10, 2 (2011), 63–81. https://doi.org/10.1007/s10207-011-0124-7

[14] European Commission. 2020. The Digital Services Act: Ensuring a safe and accountable online environment. *The Digital Services Act: Ensuring a safe and accountable online environment* (2020). https://digital-strategy.ec.europa.eu/en/policies/digital-services-act

[15] 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.

[16] Edsko de Vries, Vasileios Koutavas, and Matthew Hennessy. 2010. Communicating transactions. In *International Conference on Concurrency Theory*. Springer, 569–583.

[17] Edsko De Vries, Vasileios Koutavas, and Matthew Hennessy. 2010. Liveness of communicating transactions. In *Asian Symposium on Programming Languages and Systems*. Springer, 392–407.

[18] Ohad Eitan, Idit Keidar, and Ehud Shapiro. 2026. The Grassroots Super-App: A Secure Operating System for Grassroots Mini-Apps. arXiv:XXX:XXXXX. arXiv:XXX:XXXXX [cs.DC]

[19] 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.

[20] Farcaster. 2024. Farcaster Protocol Specification. https://github.com/farcasterxyz/protocol. Accessed 2026.

[21] Federal Trade Commission. 2025. Children's Online Privacy Protection Rule: Final Amendments. Federal Register, Vol. 90, No. 77, April 22, 2025. https://www.ftc.gov/legal-library/browse/federal-register-notices/16-cfr-part-312-childrens-online-privacy-protection-rule

[22] 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. https://doi.org/10.1007/s10506-021-09300-9

[23] James Golike, Andy Lewis-Pye, and Ehud Shapiro. 2026. Formalising and Implementing Grassroots Social Contracts: From Legal Text to Grassroots Platforms (Full Version). *arXiv:2609.06231* (2026).

[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. 2014. Google Classroom. https://edu.google.com/classroom. Accessed 2026.

[26] Google. 2015. YouTube Kids. https://www.youtubekids.com. Accessed 2026.

[27] Google. 2024. Dart Programming Language. https://dart.dev/

[28] Google LLC. 2021. Family Link: Parental Controls and Supervised Experiences. https://families.google.com/familylink/

[29] Guido Governatori et al. 2018. On legal contracts, imperative and declarative smart contracts, and blockchain systems. *Artificial Intelligence and Law* 26, 4 (2018), 377–409. https://doi.org/10.1007/s10506-018-9223-3

[30] 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.

[31] James N. Gray. 1978. Notes on Data Base Operating Systems. In *Operating Systems: An Advanced Course*. Lecture Notes in Computer Science, Vol. 60. Springer, 393–481.

[32] Barbara Guidi, Marco Conti, Andrea Passarella, and Laura Ricci. 2018. Managing social contents in decentralized online social networks: a survey. *Online Social Networks and Media* 7 (2018), 12–29. https://doi.org/10.1016/j.osnem.2018.07.001

[33] Jonathan Haidt. 2024. *The Anxious Generation: How the Great Rewiring of Childhood Is Causing an Epidemic of Mental Illness*. Penguin Press. https://www.penguinrandomhouse.com/books/729231/the-anxious-generation-by-jonathan-haidt/

[34] Daniel Halpern, Ariel D Procaccia, Ehud Shapiro, and Nimrod Talmon. 2024. Federated Assemblies. *Proc AAAI 2025; arXiv preprint arXiv:2405.19129* (2024). https://arxiv.org/abs/2405.19129

[35] Frances Haugen. 2021. Statement of Frances Haugen, October 4, 2021. *Testimony before the United States Senate Committee on Commerce, Science, and Transportation, Subcommittee on Consumer Protection, Product Safety, and Data Security* (2021). https://www.commerce.senate.gov/wp-content/uploads/meetings/A04578DB-ADD9-41A7-BF1C-A22703DAEA02/Frances%20Haugen%20Written%20Testimony.pdf

[36] Le Jiang and Xinglin Zhang. 2019. BCOSN: A blockchain-based decentralized online social network. *IEEE Transactions on Computational Social Systems* 6, 6 (2019), 1454–1466.

[37] 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

[38] Martin Kleppmann et al. 2024. Bluesky and the AT protocol: Usable decentralized social media. *arXiv preprint arXiv:2402.03239* (2024). https://arxiv.org/abs/2402.03239

[39] Butler W Lampson. 1981. Chapter 11. atomic transactions. In *Distributed Systems—Architecture and Implementation: an Advanced Course*. Springer, 246–265.

[40] 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

[41] 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

[42] Nancy Lynch, Michael Merritt, William Weihl, and Alan Fekete. 1988. A theory of atomic transactions. In *ICDT'88: 2nd International Conference on Database Theory Bruges, Belgium, August 31–September 2, 1988 Proceedings 2*. Springer, 41–71.

[43] Nancy A Lynch and Michael Merritt. 1993. *Atomic transactions: in concurrent and distributed systems*. Morgan Kaufmann Publishers Inc.

[44] 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). https://doi.org/10.1145/3473582

[45] Meta. 2020. Expanding Messenger Kids to More Countries and Adding New Features. https://about.fb.com/news/2020/04/expanding-messenger-kids/. Published 22 April 2020; accessed 16 October 2026.

[46] Meta. 2020. Giving Parents Even More Control in Messenger Kids. https://about.fb.com/news/2020/02/messenger-kids-controls/. Published 4 February 2020; accessed 16 October 2026.

[47] Meta. 2026. Community Standards Enforcement Report: Fake Accounts. https://transparency.meta.com/reports/community-standards-enforcement/fake-accounts/facebook/. Data for January–March 2026; accessed 16 October 2026.

[48] Meta Platforms. 2017. Messenger Kids. https://messengerkids.com. Accessed 2026.

[49] Meta Platforms, Inc. 2022. Introducing Family Center and Parental Supervision Tools on Instagram and in VR. https://about.fb.com/news/2022/03/parental-supervision-tools-instagram-vr/

[50] Microsoft. 2017. Microsoft Teams for Education. https://www.microsoft.com/en-us/education/products/teams. Accessed 2026.

[51] Vivek H. Murthy. 2023. Social Media and Youth Mental Health: The U.S. Surgeon General's Advisory. Office of the Surgeon General. https://www.hhs.gov/surgeongeneral/priorities/youth-mental-health/social-media/index.html

[52] Nostr Protocol. 2024. NIP-01: Basic Protocol Flow Description. https://github.com/nostr-protocol/nips/blob/master/01.md. Accessed 2026.

[53] Parliament of Australia. 2024. Online Safety Amendment (Social Media Minimum Age) Act 2024. https://www.legislation.gov.au/C2024A00127/asmade Banning children under 16 from social media platforms.

[54] Parliament of the United Kingdom. 2023. Online Safety Act 2023. c. 50. https://www.legislation.gov.uk/ukpga/2023/50/contents Royal Assent October 26, 2023.

[55] Alireza Parvizimosaed et al. 2022. Specification and Analysis of Legal Contracts with Symboleo. *Software and Systems Modeling* 21 (2022). https://doi.org/10.1007/s10270-022-01053-6

[56] Project Liberty. 2021. Decentralized Social Networking Protocol (DSNP) Specification. https://dsnp.org/. Version 1.0.

[57] Pierre-Joseph Proudhon and John Beverley Robinson. 2004. *General idea of the revolution in the nineteenth century*. Courier Corporation. Original work published 1851.

[58] Mohsin Ur Rahman, Barbara Guidi, and Fabrizio Baiardi. 2020. Blockchain-based access control management for decentralized online social networks. *J. Parallel and Distrib. Comput.* 144 (2020), 41–54.

[59] Aravindh Raman et al. 2019. Challenges in the decentralised web: The mastodon case. In *Proceedings of the internet measurement conference*. 217–229. https://doi.org/10.1145/3355369.3355572

[60] Gal Shahaf, Ehud Shapiro, and Nimrod Talmon. 2020. Genuine Personal Identifiers and Mutual Sureties for Sybil-Resilient Community Growth. In *International Conference on Social Informatics*. Springer, Springer, EU, 320–332. https://doi.org/10.1007/978-3-030-60975-7_24

[61] Ehud Shapiro. 2021. Multiagent Transition Systems for Composing Fault-Resilient Protocol Stacks. *arXiv preprint arXiv:2112.13650* (2021). https://arxiv.org/abs/2112.13650

[62] Ehud Shapiro. 2022. Grassroots Currencies: Turning Mutual Trust into Liquidity. *arXiv preprint arXiv:2202.05619* (2022). https://arxiv.org/abs/2202.05619

[63] 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

[64] Ehud Shapiro. 2023. Grassroots Social Networking: Serverless, Permissionless Protocols for Twitter/LinkedIn/WhatsApp. In *OASIS '23* (Rome, Italy). Association for Computing Machinery. https://doi.org/10.1145/3599696.3612898

[65] 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

[66] 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

[67] 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. https://doi.org/10.1145/3772290.3772309 arXiv preprint arXiv:2502.11299.

[68] 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

[69] Ehud Shapiro. 2026. sGLP: Simulating People with Stochastic Agents. arXiv:2610.14738. arXiv:2610.14738 [cs.MA]

[70] Ehud Shapiro. 2026. Types for Grassroots Logic Programs (Full version). *arXiv preprint arXiv:2601.17957* (2026). https://arxiv.org/abs/2601.17957

[71] Sepehr Sharifi et al. 2020. Symboleo: Towards a Specification Language for Legal Contracts. In *IEEE 28th International Requirements Engineering Conference (RE)*. IEEE, 364–369. https://doi.org/10.1109/RE48521.2020.00049

[72] Nimrod Talmon and Ehud Shapiro. 2025. Grassroots Federation: Fair Democratic Governance at Scale. *arXiv preprint arXiv:2505.02208; also Proc. of AAMAS'26* (2025). https://arxiv.org/abs/2505.02208

[73] 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. https://doi.org/10.1145/3357150.3357396

[74] The Matrix.org Foundation. 2024. Matrix Specification. https://spec.matrix.org/.

[75] Thorn. 2022. Emerging Online Trends in Child Sexual Exploitation 2022: Thorn Research Report. https://www.thorn.org/research/

[76] Jean M. Twenge, Thomas E. Joiner, Megan L. Rogers, and Gabrielle N. Martin. 2018. Increases in Depressive Symptoms, Suicide-Related Outcomes, and Suicide Rates Among U.S. Adolescents After 2010 and Links to Increased New Media Screen Time. *Clinical Psychological Science* 6, 1 (2018), 3–17. https://doi.org/10.1177/2167702617723376

[77] United States Congress. 1998. Children's Online Privacy Protection Act (COPPA). 15 U.S.C. §§ 6501–6506. Enacted October 21, 1998.

[78] United States Senate. 2024. Kids Online Safety Act (KOSA), S. 1409. Passed the U.S. Senate July 30, 2024.

[79] 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. https://doi.org/10.3233/FAIA230954

[80] Georgia Wells, Jeff Horwitz, and Deepa Seetharaman. 2021. Facebook Knows Instagram Is Toxic for Teen Girls, Company Documents Show. *The Wall Street Journal* (Sept. 2021).

[81] Shoshana Zuboff. 2019. *The age of surveillance capitalism: The fight for a human future at the new frontier of power*. Public Affairs, US. https://www.hachettebookgroup.com/titles/shoshana-zuboff/the-age-of-surveillance-capitalism/9781610395694/

A Volition-Guarded Multiagent Atomic Transactions

This appendix recalls, in the words of [41], the mathematical foundations on which the paper is based; the proofs of the results recalled are in that paper.

Each agent consists of a *person* and a *machine* (e.g., smartphone) operated by the person. A *machine transaction* specifies what machines do; a *volition guard* specifies which persons must be willing for the transaction to occur.

We assume a potentially infinite set of *agents* Π, but consider only finite subsets of it, so we assume any particular set of agents $P \subset \Pi$ to be nonempty and finite. We use $\subset$ to denote the strict subset relation and $\subseteq$ when equality is also possible. 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 $p \in P$.

Machine states, configurations, machine transactions, and volition-guarded transactions are as in Definition 4.1 (Section 4.3), which also recalls the reading of guards and of the unguarded case.

Distinct machine transactions can represent "the same action" in different configurations; this is captured with an equivalence relation on machine transactions.

Definition A.1 (Transaction Equivalence [41]). 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 $\sim$.

For example, all befriend(p, q) transactions—differing only in the configurations in which they occur—form an equivalence class. Next is the mathematical machinery allowing people to express their volitions regarding the classes of machine transactions their machines may participate in.

Definition A.2 (Agent State and Configuration [41]). 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 $c^v_p \subseteq (T/\sim)_p$ for every $p \in P$, where $(T/\sim)_p$ denotes the classes in $T/\sim$ in which p is a participant; we write c^v_p for the volitional state and c^m_p for the machine state of agent p in c.

Definition A.3 (Volitional Transaction [41]). 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 $c \to c'$ of agent configurations over $\{p\}, S, T$, and $\sim$ such that $c^v_p, {c'}^v_p \subseteq (T/\sim)_p$ and $c^v_p \neq {c'}^v_p$, and $c^m_p = {c'}^m_p$.

(2) A *volitional machine transaction* induced by a volition-guarded transaction (t, Q'), for some $t = (d \to d') \in T$ over $Q' \subseteq Q \subseteq P$, is a pair $c \to c'$ where $c \neq c'$ are agent configurations over P, S, T, and $\sim$ such that $[t] \in c^v_q$ for every $q \in Q'$; $c^m_p = d_p$ and ${c'}^m_p = d'_p$ for every $p \in Q$; $c^m_p = {c'}^m_p$ for every $p \in P \setminus Q$; and ${c'}^v_p = c^v_p \setminus \{[t]\}$ 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 $\lfloor t \rfloor$ is removed from every agent's volitional state: the will is fulfilled by any equivalent transaction. A person 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 person's not willing a transaction has, so far, no consequence: no volitional transaction is enabled by the absence of a will. An implementation of volition-guarded transactions by communicating agents [37] needs one, so that a machine asked for its person's will answers either way; it is the following, by which the machine answers for a person who does not will an act: in machine state s, if p wills it s' is taken, else s''.

Definition A.4 (Volition-Guarded If-Then-Else Transaction, Then-Branch, Else-Branch [41]). Given machine states S, a *volition-guarded if-then-else transaction* over a single participant $p \in \Pi$ is a triple $s \to s'\,;\, s''$ *guarded by* p, with $s, s', s'' \in S$, $s \neq s'$ and $s \neq s''$; the unary machine transaction $s \to s'$ is its *then-branch* and $s \to s''$ its *else-branch*.

Definition A.5 (Else Transaction [41]). Given agents P, states S, machine transactions T over P and S, and equivalence $\sim$ on T, an *else transaction* induced by a volition-guarded if-then-else transaction $s \to s'\,;\, s''$ guarded by $p \in P$, with $s \to s' \in T$, is a pair $c \to c'$ of agent configurations over P, S, T, and $\sim$ such that $\lfloor s \to s' \rfloor \notin c_p^v$; $c_p^m = s$ and $c'_p{}^m = s''$; $c_q^m = c'_q{}^m$ for every $q \in P \setminus \{p\}$; and $c'_q{}^v = c_q^v$ for every $q \in P$.

The then-branch, taken when p wills it, is the unary volition-guarded transaction $(s \to s', \{p\})$ and induces volitional machine transactions per Definition A.3; a unary volition-guarded transaction is thus the special case with no else-branch. The else transaction is taken when p does not will the then-branch; it fulfils no will and removes no class. The two are placed in one class below, so that liveness obliges the machine to answer either way; the obligation is the machine's, and it has no time bound. Only a unary transaction has an else-branch: a binary one would let an agent that never wills it trigger it.

Induced transition systems. The following is a simplified variation, sufficient for the purpose of this work, on the foundations introduced in [61].

Definition A.6 (Transition System, Computation, Run, Safe, Live, Correct [41]). 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 $t = s \to s'$.
(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 $r = s_1, s_2, \cdots$; 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 $r = s_1, s_2, \ldots$ is *safe*, also written $r \subseteq T$, if $s_i \to s_{i+1} \in T$ for every two consecutive states; and $s \xrightarrow{*} s' \subseteq T$ denotes the existence of a safe computation from s to s' (empty if $s = s'$). A class $\lfloor t \rfloor \in T/\!\sim$ is *enabled* in a state s if $s \to s' \in \lfloor t \rfloor$ for some $s' \in S$. A run r is *live* if no class $\lfloor t \rfloor \in T/\!\sim$ is enabled in every state of some suffix of r with no member of $\lfloor t \rfloor$ 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 at all: 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. A run with uncountably many classes enabled in every state of a suffix is not live.

Lemma A.7 (Extension [41]). *Every finite safe run of a transition system is a prefix of a correct run of it.*

Definition A.8 (Multiagent Transition System [41]). 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 (Definition 4.1). 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 A.9 (Transaction Closure [41]). Let $P \subset \Pi$, S a set of machine states, and $C := S^P$. For any transition or transaction $t = c \to c'$, we write $t_q := c_q \to c'_q$ and say p is *stationary* in t if $c_p = c'_p$. For a machine transaction $t = (c \to c')$ 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.

Namely, the closure over $P \supseteq Q$ of a machine transaction t over Q includes all transitions t' over P in which members of Q do the same in t and in t', and the rest remain in their current (arbitrary) state. The closure likewise carries any relation on transactions to one on the induced transitions. 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 (Definition A.10).

A set R of volition-guarded transactions with an equivalence $\sim$ on their underlying machine transactions induces a volitional multiagent transition system:

Definition A.10 (Volitional Multiagent Transition System [41]). Given agents $P \subset \Pi$, machine states S with initial state s_0, a set R of volition-guarded transactions and volition-guarded if-then-else transactions—a volition-guarded if-then-else transaction $s \to s'\,;\, s''$ guarded by p standing in R for its then-branch $(s \to s', \{p\})$ together with its else-branch—such that every $(t, Q') \in R$ has the participants of t contained in P, every guard p of a volition-guarded if-then-else transaction is in P; and an equivalence $\sim$ on the set $T_R := \{t : (t, Q') \in R \text{ for some } Q'\}$ of underlying machine transactions (a total equivalence, hence in particular a partial equivalence in the sense of Definition A.6), such that two machine transactions of different classes, each the machine transaction of an unguarded transaction in R or an else-branch, have disjoint P-closures, no else-branch is in T_R and volition-guarded if-then-else transactions with the same else-branch have $\sim$-equivalent then-branches, 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 $c_0{}_p^v = \emptyset$ and $c_0{}_p^m = s_0$ for every $p \in P$;
(3) T_V consists of all transitions $e \to e' \in (\mathcal{A}^P)^2$ of one of three 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 $r \in P \setminus \{p\}$; (*ii*) a *volitional machine transaction* induced by some volition-guarded transaction $(t, Q') \in R$ per Definition A.3(2); or (*iii*) an *else transaction* induced by some volition-guarded if-then-else transaction in R per Definition A.5;
(4) $\sim_V$ is the restriction of $\sim{\uparrow}P$ (Definition A.9) to the volitional machine transactions and else transactions, relating two of them whenever their inducing machine transactions are $\sim$-equivalent, where the inducing machine transaction of an else transaction is the then-branch $s \to s'$ of its volition-guarded if-then-else transaction, and leaves every volition-change transition outside its domain—so volition changes belong to no class and carry no liveness obligation.

Every transition of T_V other than a volition change 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 $\lfloor t \rfloor$ from the volitional state of each guard, which held it; a step of another class removes another class, and an else transaction removes none, so neither is that step, and two guarded transactions of different classes may share their P-closures. Two unguarded transactions of different classes are kept apart by the condition on P-closures. The two conditions on else-branches give each else transaction one class: an else-branch that is not a transaction of R is induced only by if-then-else transactions, and those sharing it have $\sim$-equivalent then-branches. The set R of volition-guarded transactions that specifies a platform thus determines the volitional machine transitions of the induced volitional multiagent transition system, while volition-change transitions are freely available to every agent independent of R.

Definition A.11 (Enabled [41]). Given a set of volition-guarded transactions, each (t, Q') with $t = d \to d'$ a machine transaction over some $Q' \subseteq Q \subseteq P$ and S, 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 $\lfloor t \rfloor \in c_q^v$ for every $q \in Q'$. The else-branch of a volition-guarded if-then-else transaction $s \to s'\,;\, s''$ guarded by p is *enabled* in c if $c_p^m = s$ and $\lfloor s \to s' \rfloor \notin c_p^v$. An equivalence class $\lfloor t \rfloor$ is *enabled* in c if some volition-guarded transaction (t', Q') with $t' \in \lfloor t \rfloor$ is enabled in c, or the else-branch of some volition-guarded if-then-else transaction with then-branch $t' \in \lfloor t \rfloor$ is enabled in c.

A volition-guarded transaction with an empty guard ($Q' = \emptyset$) requires no volitions and is enabled whenever its machine precondition is met. The class of the then-branch of a volition-guarded if-then-else transaction guarded by p is thus enabled whenever p's machine is in its source state, whether or not p wills it. For the volitional multiagent transition system above, a class of volitional machine transactions and else transactions is enabled at a configuration in the sense of Definition A.6 exactly when some volition-guarded transaction or else-branch inducing it is enabled in the above sense; this determines which runs are live and correct per Definition A.6. Volition-change transitions belong to no class and so impose no liveness obligation; personal choices remain free.

Protocols and grassroots protocols. 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 an arbitrary 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 A.12 (Protocol [41]). 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 $p \in P$.

Definition A.13 (Interleaving [41]). Let $P, P' \subset \Pi$ be disjoint nonempty sets of agents, $r = c_0, c_1, \ldots$ a run of $\mathcal{F}(P)$, and $r' = d_0, d_1, \ldots$ a run of $\mathcal{F}(P')$. An *interleaving* of r and r' is a sequence $e_0, e_1, \ldots$ 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).

Definition A.14 (Interaction, Interactive Run, First Interaction [41]). Let $Q \subset \Pi$. For a configuration c over Q and $R \subseteq Q$, write $c|_R$ for the restriction of c to R. A transition $e \to e'$ 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 $e|_{\{x\}} \to e'|_{\{x\}}$ is not a transition of $\mathcal{F}(\{x\})$ or $e|_{\{y\}} \to e'|_{\{y\}}$ 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 A.15 (Oblivious, Interactive, Grassroots [41]). 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 groups coexist without interference: any interleaving of a correct run of one with a correct run of the other is a correct run of the combined system. Being interactive means two groups that have not yet interacted can still do so: the combined system has a correct continuation in which a single transaction engages agents of both groups. Agents that have taken part in a common interaction are joined; an instance is a maximal set of agents joined directly or indirectly, and two instances coalesce when they come to be one.

Definition A.16 (Interaction Graph, Instance, Coalescence [41]). 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 $\hat{r}$.

Definition A.17 (Transactions Over a Local-State Function [41]). 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 A.18 (Volitional Transactions-Based Protocol [41]). 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 A.10). 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 A.12 requires—and $c_0(P)$ has p-component $(\emptyset, s_0)$ for every $p \in P$.

The obliviousness results rest on the following invariant: in a group's own system an agent wills only transactions internal to the group, so the guard of a transaction that reaches another group cannot will it while the group runs alone, and the transaction is never enabled in an interleaving.

Lemma A.19 (Volitional Containment [41]). *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 $p \in P$: $c.p \subseteq T_{R(P)}/\sim$, where $T_{R(P)} := \{t : (t, Q') \in R(P) \text{ for some } Q'\}$.*

Lemma A.20 (Interleaving Safety [41]). *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 A.21 ([41]). *A volitional transactions-based protocol 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 $\mathcal{F}(P')$.*

Corollary A.22 (Guarded Obliviousness [41]). *A volitional transactions-based protocol over a set of volition-guarded transactions R is oblivious if every volition-guarded transaction $(t, Q') \in R$ whose machine transaction t has two or more participants has a nonempty guard, $Q' \neq \emptyset$.*

A person may die or become incapacitated, and an AI agent operating a machine may be retired; the agent has then left, and its will never again has effect. Leaving is made explicit, and interactivity is required only of groups each holding an agent that has not left.

Definition A.23 (Left [41]). 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 A.24 (Interactive Among the Agents That Have Not Left [41]). 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 $\mathcal{F}(P \cup P')$ a member may will a transaction reaching the other group, which a group's own run forbids; the volitional notion constrains the resulting coupling:

Definition A.25 (Volitionally Grassroots [41]). 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 $Q' \cap P' \neq \emptyset$.

Two groups thus become connected only by an act a person of each wills. An else transaction (Definition A.5) is over a single participant and so is never an interaction; a binary transaction with an else-branch would let an agent of another group, who never wills it, trigger it, and could be a first interaction willed by neither, which is why Definition A.4 admits only unary ones.

B Speech-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. Each act an agent takes is a digital speech act [24], a speech act performed digitally by signing a text with their private key, and speech-act schemas [23] are a formal syntax for those acts. We recall the language, write the contract of Section 4.1 in it, compile it into transactions, and check it against the conditions on which a checker certifies a contract with wards grassroots.

B.1 The Language

Write Π for a 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 $\text{signer}(x, p) := p$ [8, 24]; a speech act signed again carries its earlier signers inside it, and $D_P := \{d \in D : \text{signer}(d) \in P\}$ for $P \subseteq \Pi$. Write Σ for a set of *predicate symbols*, *predicates* for short, and $\alpha : \Sigma \to \mathbb{N}$ for their arities; a predicate of arity n applied to n arguments is an atom, a syntactic object with no truth value.

We recall the language of [23, Section 3 and Appendix D], specialised to the roles and the names the contract uses.

Definition B.1 (Community name [23]). The set $\mathcal{N}$ of *community names* is disjoint from $\Pi \cup D$ and is generated freely by an injection $p \mapsto \langle p \rangle$ of Π into it and an injection $(\nu, y) \mapsto \nu \cdot y$ of $\mathcal{N} \times D$ into it with disjoint images. The *people* of a name are $\text{ppl}(\langle p \rangle) := \{p\}$ and $\text{ppl}(\nu \cdot y) := \text{ppl}(\nu) \cup \{\text{signer}(y)\}$.

A group is named by the community name $\langle a \rangle \cdot n$ of its leader a and a speech act n that a signs [23], and $\mathcal{N}_P := \{\nu \in \mathcal{N} : \text{ppl}(\nu) \subseteq P\}$ for $P \subseteq \Pi$. For a set W, the *atoms over W* are the $e(u_1, \ldots, u_{\alpha(e)})$ with $e \in \Sigma$ and $u_i \in W$. 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, name terms and signed terms. An atom of a state *names* $q \in \Pi$ if q is an argument of it, or the signer of an argument of it, or a member of $\text{ppl}(\nu)$ for a name ν among its arguments [23]; an atom of a schema names the agent roles and variables among its arguments, the role of a signed term among them, and the base of a name term among them. 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 $S_C(P)$ is the set of finite multisets over the atoms over $P \cup D_P \cup \mathcal{N}_P$. 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.

Definition B.2 (Name term [23]). A *name term* is $\langle u \rangle$ for $u \in R \cup X$, or $\sigma \cdot y$ for a name term σ and $y \in Y$. Its *base* is u, and the base of σ, respectively.

Definition B.3 (Role [23]). A *role* is an agent role $\pi \in R$, or a *listed role* $\pi[\rho(*, u_2, \ldots, u_r)]$ of an agent role π, a predicate ρ of arity $r \geq 1$ and arguments $u_2, \ldots, u_r$ of the schema. The *base* of an agent role is itself, and of a listed role $\pi[\rho(*, u_2, \ldots, u_r)]$ the agent role π.

Definition B.4 (Speech-act schema [23]). An *speech-act schema* is written

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

with τ its name, $\pi_1, \ldots, \pi_k$ distinct roles, $k \geq 1$ its *arity*, $x_1, \ldots, x_m \in X \cup Y$ distinct, its *variables*, and each E_i a finite multiset of *marked atoms*, an atom over the agent roles, the agent variables, the speech-act variables, the name terms and the *signed terms* of τ, the signed terms being $\text{sgn}(y, \pi)$ for $y \in Y$ and π an agent role of τ, the speech act y signed by the agent in π, carrying at most one of the marks $-, +, \neg$. For each role the marks divide E_i into the finite multisets $+(i), -(i), \neg(i)$ and $=(i)$ of its atoms marked $+$, marked $-$, marked $\neg$ and unmarked; the atoms π_i *requires* are $=(i) \uplus -(i)$; $+(i) \cap -(i) = \emptyset$, a signed term occurs only in an atom marked $+$, and its speech-act variable occurs in an atom its role requires. An agent role may carry the mark ?, and the *guarding roles* of τ are those carrying it.

Definition B.5 (Binding [23]). A *binding* β for a schema τ maps the agent roles and agent variables of τ to Π, sending distinct agent roles to distinct agents, except two that the schema declares possibly one person, which it sends to one agent only if neither of them adds or deletes an atom naming the other, and Y to D; it is extended to name terms by $\beta(\langle u \rangle) := \langle \beta(u) \rangle$ and $\beta(\sigma \cdot y) := \beta(\sigma) \cdot \beta(y)$, to signed terms by $\beta(\text{sgn}(y, \pi)) := (\beta(y), \beta(\pi))$, and to atoms and to multisets of them componentwise.

Definition B.6 (Extent [23]). Let β be a binding for a schema τ and d a machine configuration over $Q \subset \Pi$. The *extent* $\text{ext}_d(\pi)$ in d of a role π of τ is $\text{ext}_d(\pi_i) := \{\beta(\pi_i)\}$ for an agent role π_i, and

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

for a listed role $\pi[\rho(*, u_2, \ldots, u_r)]$.

Two agent roles sent to one agent are carried out by that agent together, as Definition B.7 states; a schema declares the two it allows this of, as the clause does, "Alice and Bob, possibly one person", and here the two parent roles of a child acquaint with two children, a child befriending and a child join with a child leader are so declared, by GnW:2. A role that adds or deletes an atom naming another cannot share its agent, so no relation the contract records is reflexive. A contract may declare a predicate of arity one *reflexive*: every state is then taken to hold the atom of that predicate naming its holder, and no schema adds or deletes, at a role, the atom of it naming that role.

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 [23]. Three consequences of the definitions are used throughout. An act changes the state of each of its participants but a guard whose role adds and deletes nothing, who takes part by their will, as a parent does in their child's befriending (Definition B.7). A listed role takes every agent its base lists, so an act may have every member of a group as its participants, as closing a group does. A guard is a role, so an agent whose will an act requires is a participant of the act.

A local state is a multiset, so a predicate of arity zero carries a count: an agent holds as many copies of it as acts have added and not deleted. A state holding a speech act records its signer; an agent named inside the utterance is not an argument of the atom carrying it and is not recorded.

B.2 The Schemas of the Contract

Sixteen predicates. *identity*, of arity zero: in the state of an agent who has established an identity. *knows*, of arity one and reflexive: *knows*(q) in the state of p says that p holds q's public key, once for each act by which p came to know q; it is never deleted. *child*, of arity zero: in the state of a child, so an agent who holds none is a person. *retired*, of arity zero: in the state of a child their creator retired. *creator* and *created*, of arity one: *creator*(p) in the state of r is p's declaration, held by r, that p created r, and *created*(r) in the state of p is p's own record of it. *parenting*, of arity one: *parenting*(r) in the state of p says that p is a parent of r, so a person is a parent exactly when they hold one; and *made*, of arity two: *made*(q, r) in the state of the creator of r says that they made q a parent of r. *friend*, of arity one and reflexive: *friend*(q) in the state of p says that q is a friend of p, and every agent counts as their own friend. *member*, of arity one, its argument a

group's community name: $member(\langle a\rangle \cdot n)$ in the state of p says that p is a member of the group $\langle a\rangle \cdot n$, led by a; and *listed*, of arity two, the second argument the speech act n that names the group: $listed(q, n)$ in the state of that leader says that q is on their member list for n. *closed*, of arity one, its argument the speech act that names a group: $closed(n)$ in the state of a leader a says that their group $\langle a\rangle \cdot n$ has ended. And for messages, *received* and *posted* of arity two and *delivered* and *received_in* of arity three: $received(x, p)$ in the state of q is a content x, a speech act, that q received from their friend p; $posted(x, n)$ in the state of a leader says that x was posted to their group named n; $delivered(x, q, n)$ says that the leader has delivered x to the member q in that group; and $received_in(x, a, n)$ in the state of q says that q received x in the group named n that a leads.

The contract is twenty-eight schemas over these predicates: the fourteen schemas of GSN [23] — GKK's three, GSG's two, and the nine of messages and groups — with *child* forbidden at both roles of *acquaint*, of *befriend* and of *join*, and at the role of the agent who comes to know in *introduce*, and *creator* and *parenting* of the other at the first role of *unfriend*, and fourteen schemas for children, which realise GnW and CSSN:2. They are set out in Appendix C.1, which explains clause by clause how they render the text and reads the two this paper follows, GnW:2 with CSSN:3, and GnW:3, role by role. Every role of every schema requires *identity*, except the role of *establish* and the child's role of *enter*, which forbid it and add it; and every guarding role, *establish*'s included, forbids *retired*. The tables leave these out.

The records. *acquaint*, *introduce*, *enter*, *make_parent*, *child_acquaint* and $child_acquaint_2$ are the only schemas adding *knows*, and no schema deletes it; *acquaint* forbids *child* at both roles and *introduce* at the role of the agent who comes to know, so a child comes to know an agent only by their entering, which adds their creator, by *make_parent*, which adds the parent made and which the creator guards, or by *child_acquaint* or $child_acquaint_2$, which a parent guards; and *befriend*, *child_befriend* and *child_adult_befriend* require *knows* of the other at both roles, as the entering adds it, so friends know each other. *establish* and *enter* are the only schemas adding *identity*, and each forbids it where it adds it, so an agent establishes an identity once, a child's identity is established by their entering, and an agent who has established an identity is never entered as a child. Every other role of every schema requires *identity*, so an agent who has none takes part in no act but these two, and holds no atom; and since every atom a schema adds names its roles, or a speech act that an atom it requires names, no state records an agent before their identity is established. So a child is a child from their first act, and a child is never a parent: *enter* is the only schema adding *child*, *creator* and *created*, and with *make_parent* the only ones adding *parenting*, each forbidding *child* at the parent's role; *of_age* and *retire*, guarded by the creator, are the only schemas deleting *child*, and with them every *parenting* of the child, at the creator and at every parent the creator made, and *end_parent*, guarded by the creator, deletes the *parenting* of a parent they made; an agent so made a person can never be entered as a child again. *unfriend* forbids at its first role *creator* and *parenting* of the other, and *child_unfriend* forbids *parenting* of the child at the friend's role, so no act ends the friendship of a child and a parent while the parent is the child's parent. *retire*, the only schema adding *retired*, and every guarding role forbids *retired*, so a retired child takes no act. *enter*, *befriend*, *child_befriend* and *child_adult_befriend* are the only schemas adding a *friend* atom, the arguments of those atoms are roles, and each is guarded by the roles that gain them, so an agent's friends are only agents who willed the friendship; no schema of a guardianship deletes one, so a child's friendships outlast its guardianship. A parent's state carries nothing of their child's friendships and groups: the child's schemas read *parenting* at the parent and add and delete nothing there, so any parent of a child may end any membership of that child, and any friendship of that child other than one with a parent, whether or not they took part in its making. *message* and *deliver* are the only schemas adding *received* and *received_in*, at the recipient, naming the friend the content came from or the group it was delivered in, so an agent holds a content only from a friend or from a group they are a member of.

B.3 From Schemas to Transactions

Speech-act schemas have no operational semantics of their own: a schema of a certified contract means the volition-guarded transactions it compiles to, and those have the semantics of [41]. 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; for a configuration d and an agent p, $\hat{d}_p$ is d_p together with the atom naming p of every reflexive predicate.

Definition B.7 (Compilation [23]). Let τ be a schema with roles $\pi_1, \dots, \pi_k$ and β a binding for it, and let $Q_i := \mathrm{ext}_d(\pi_i)$ for a machine configuration d. A machine transaction $d \to d'$ 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 $p \in Q_i$

$$\beta(=(i) \uplus -(i)) \subseteq \hat{d}_p, \qquad \beta(\neg(i)) \cap \hat{d}_p = \emptyset,$$

and for every $p \in Q$

$$\bigsqcup_{i:\, p \in Q_i} \beta(+(i)) \cap \bigsqcup_{i:\, p \in Q_i} \beta(-(i)) = \emptyset,$$

$$d'_p = \Big(d_p \setminus \bigsqcup_{i:\, p \in Q_i} \beta(-(i))\Big) \uplus \bigsqcup_{i:\, p \in Q_i} \beta(+(i)),$$

with $d'_p \neq d_p$ unless every role whose extent holds p is a guarding agent role. The volition-guarded transaction it determines is the pair of it with $\{\beta(\pi) : \pi$ a guarding role of $\tau\}$, where every speech-act variable y of τ occurring in an atom or name term added at a role and in none required at a role has $\beta(y)$ signed by a member of that set.

Definition B.8 (Realisation of a contract [23]). A *contract* C is a finite set of speech-act schemas over Σ. Its *realisation* R_C is the set of volition-guarded transactions determined by the schemas of C at all bindings for them, and $\sim$ is the equivalence on their underlying machine transactions generated by relating two of the same participants that one schema at one binding determines. The *protocol realising* C is the volitional transactions-based protocol over R_C, S_C and $\sim$.

R_C with $\sim$ is well formed, by the lemma of [23] that the compilation is well defined.

A schema at a binding determines a machine transaction for every configuration its precondition admits, and those of one set of participants are a part of one class of $\sim$, one act available at different points of a run; a binding of *close* or of *child_close* determines a part for each list of members, and bindings that determine a common machine transaction put their parts in one class [37]. A schema with one guarding role determines, at the binding that swaps two roles, the same machine transaction under the other guard, which is how "either of them may end it" comes out of a schema guarded in one role. A guard whose role adds and deletes nothing is a participant whose state the transaction leaves as it is; two such transactions of different classes may have a transition in common, and a step of each removes its own class from the will of its guards (Definition A.10).

B.4 Certification

Three conditions on the schemas of a contract are sufficient for the protocol realising it to be volitionally grassroots [23]: an unobstructed introductory act, traceable provenance and volition. A contract with wards has no unobstructed introductory act, since its acquaintance forbids the ward's mark. For such a contract a condition in place of the first, an introduction with wards, with the other two is sufficient (Theorem B.23); and for one whose roles are agent roles and listed roles based on them, as this contract's closing of a group is, the three together with conditions on its listed roles are (Theorem B.26). The conditions for wards, with the definitions that state them, are those of a paper in preparation on a contract whose wards are AI agents, which shares GnW with this one; this paper rests on them, so they are stated and proved here, and composed with the conditions of [23] for contracts with listed roles. All 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.

Definition B.9 (Records [23]). A machine state s *records* an agent $q \in \Pi$ if $s \notin S_C(P)$ for every $P \subset \Pi$ with $q \notin P$.

A local state records q exactly when one of its atoms names q (Lemma Records of [23]).

Definition B.10 (Introductory act [23]). A schema τ of C of arity two, guarded in both roles, is an *introductory act* if any two agents $p \neq q$ have an *introduction*: a 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.

Definition B.11 (Traceable provenance [23]). The *base* of an argument is the argument itself where it is an agent role, an agent variable or a speech-act variable, and its base as a name term otherwise. Write Φ_E for the atoms of a schema whose predicate is in E; a signed term $\mathrm{sgn}(y, \pi)$ among the arguments of an atom counts here as the two arguments y and π. 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 outside Y whose base is not that of π_i, the base of u is the base of some π_j, or is the base of some argument of some

$$\psi \in (=(j) \uplus -(j)) \cap \Phi_E, \quad \text{some } j,$$

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 the arguments of such a ψ, 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.

Definition B.12 (Volition [23]). 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.

For a schema whose roles are agent roles, two roles have the same base only when they are one role, and for one of arity two the role graph is connected exactly when one of its two roles requires such an atom naming the other. Above that arity connectedness is weaker than asking it of every pair. A listed role and its base are joined by an edge.

A *mark* is a predicate of arity zero, and its atom is written as the mark. Below, M is a set of marks of C, and $\hat{d}_p$ is as in Appendix B.3.

Definition B.13 (Guardianship). A *guardianship* for M in C is (m, ι, g, w, v), with $m \in M$ and ι marks and g, w and v predicates of arity one, v possibly w, such that:

(1) no schema of C deletes ι, and no schema adds two different marks in M at one role;
(2) a schema that adds m at a role adds ι there and forbids ι there;
(3) a schema adds, and deletes, m at a role as many times as it adds, and deletes, atoms of g there;
(4) an atom $g(u)$ a schema adds at a role ρ has u a role $\sigma \neq \rho$ of the schema at which the schema requires ι, forbids every mark in M and adds $w(\rho)$; and a schema deletes $g(u)$ at ρ if and only if it deletes $w(\rho)$ at the role u;
(5) a schema that adds $w(\rho)$ at a role adds $v(\rho)$ there; a schema that adds $v(\rho)$ at a role requires ι there and forbids every mark in M there; and a schema that deletes $v(\rho)$ at a role deletes or forbids $w(\rho)$ there.

Definition B.14 (Leaving mark). Let C have a guardianship for every mark in M. A mark $\ell \notin M$ is a *leaving mark* of C if:

(1) every guarding role of every schema of C forbids ℓ, and no schema of C deletes ℓ;
(2) a schema that adds ℓ at a role requires a mark in M there;
(3) a schema that adds, at a role, an atom $g(\sigma)$ of the guardianship of a mark in M forbids ℓ at σ.

Retirement under GnW:4 is written with a leaving mark: the act that ends a ward's last guardianship adds it at the ward.

Definition B.15 (Unobstructed but for marks). An introductory act τ of C that adds one atom at each role, of a predicate e of arity one, is *unobstructed but for* M if at each role π_i of τ:

(1) every atom τ requires at π_i, other than an atom of a reflexive predicate naming π_i, is, up to renaming the role, the one atom added by some schema of C of arity one that is guarded in its role, requires nothing and forbids nothing but that atom and the leaving marks of C;
(2) every atom τ forbids at π_i is a mark in M or a leaving mark of C.

Definition B.16 (Relay act). Let C have a guardianship for every mark in M and τ be an introductory act of C unobstructed but for M, adding atoms of e. A schema χ of C is a *relay act* with *related roles* $\rho_1 \neq \rho_2$, roles of it guarded in χ at each of which χ adds an atom of e naming the other, if:

(1) each related role ρ is a *ward role* of some $m \in M$, requiring m, with exactly one other role σ of χ, its *guardian role*, requiring $v(\rho)$, the v of the guardianship of m, a guardian of the ward (GnW:2), or a *free role*, forbidding every mark in M; every role of χ is a related role or a guardian role; and χ adds and deletes at a guardian role no atom naming another guardian role;
(2) every atom χ requires at a role is an atom of a reflexive predicate naming the role; or, up to renaming the role, the one atom of a schema of arity one as in clause 1 of Definition B.15; or, at

a ward role, its m or $g(\sigma)$; or, at a ward role of m, an atom of a predicate h whose arguments are distinct speech-act variables of χ occurring in no other atom χ requires but its record below, where every schema of C that adds m at a role adds there an atom of h whose arguments are speech-act variables or signed terms and adds, at the role its g-atom names, its *record*, an atom of a predicate h' whose arguments are the two roles and the arguments of the h-atom, and no schema of C deletes an atom of h or of h'; or, at the guardian role σ of a ward role ρ, the atom $v(\rho)$ or $w(\rho)$ of that guardianship, or the record, the atom of h' whose arguments are ρ, σ and the arguments of the h-atom ρ requires; or, at a guardian role or a free role, $e(\kappa)$ for another guardian role or free role κ, with e reflexive where both are guardian roles;

(3) every atom χ forbids at a role is a mark in M or a leaving mark of C.

Definition B.17 (Introduction with wards). C has an *introduction with wards* if it has a nonempty set M of marks, a guardianship for each mark in M, an introductory act unobstructed but for M, and, for every $m, m' \in M$, a relay act with a ward role of m and a free role and a relay act with a ward role of m and a ward role of m'.

The compilation of Appendix B.3 uses nothing of the introductory act, and the realisation R_C of a contract C with an introduction with wards, traceable provenance and volition is taken by Definition B.8. In the lemmas, the proposition and the theorems that follow, C is such a contract and $\mathcal{F}$ the protocol realising it.

Lemma B.18 (Wards and guardians). *Let C have a guardianship for every mark in M and (m, ι, g, w, v) be one of them. In every configuration of a run of the protocol realising C, an agent holds m as many times as they hold atoms of g; an agent holding $g(a)$ holds ι, while a holds w of them and ι and no mark in M; an agent holding $w(x)$ holds $v(x)$; and an agent holding $v(x)$ holds ι and no mark in M.*

Proof. By induction on the run. The initial state is empty. A transition is induced by a transaction determined by a schema at a binding, and changes a state only by the atoms the schema adds and deletes there. Clause 3 keeps the counts of m and of g-atoms equal at every role. If the transition adds $g(a)$ at p then by clause 3 it adds m at p, by clause 2 it adds ι at p, and by clause 4 its schema has a at a role requiring ι and forbidding every mark in M and adds $w(p)$ at a: so after it p holds ι, and a holds $w(p)$, ι and no mark in M. These persist: ι is never deleted (clause 1); a mark in M is added at a role only where ι is forbidden (clause 2 of its guardianship), so an agent holding ι never gains one; and $w(p)$ is deleted at a only when $g(a)$ is deleted at p (clause 4). By clause 5, a transition adding $w(x)$ at an agent adds $v(x)$ there, and one deleting $v(x)$ there deletes $w(x)$ or finds it absent, so an agent holding $w(x)$ holds $v(x)$; and a transition adding $v(x)$ at an agent finds ι there and no mark in M, which persist as above. □

Lemma B.19 (Leaving). *Let C have a guardianship for every mark in M and ℓ be a leaving mark of C. In every configuration of a run of the protocol realising C, an agent holding ℓ has left; and if an agent holds $g(a)$ for the guardianship (m, ι, g, w, v) of a mark in M, then a does not hold ℓ.*

Proof. No schema deletes ℓ (clause 1), so an agent holding ℓ holds it in every later configuration. A transaction (t, Q') with that agent in Q' sends them a guarding role, which forbids ℓ (clause 1), so no such transaction is machine-enabled in a later configuration, and the agent has left (Definition A.23). When an atom $g(a)$ is added at an agent, a does not hold ℓ (clause 3), and the act adds none at a, whose role forbids every mark in M (Definition B.13, clause 4; clause 2); and while an agent holds $g(a)$, a holds no mark in M (Lemma B.18), so no transition adds ℓ at a (clause 2). □

Definition B.20 (Open among the agents that have not left). A set R of volition-guarded transactions is *open among the agents that have not left* 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, in whose last configuration neither p nor q has left, 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.

Proposition B.21 (Openness with wards). *R_C is open among the agents that have not left, and open if C has no leaving mark.*

Proof. Let $p \neq q$ in Q, let r be a finite safe run of $\mathcal{F}(Q)$ with no interaction between p and q, and let c be its last configuration, at which, if C has a leaving mark, neither p nor q has left; by Lemma B.19, neither holds a leaving mark at c. Call $x \in Q$ a *ward* at c if its state holds a mark in M, which by Lemma B.18 is one mark m and comes with $g(a)$ for some a, its *guardian*, who holds $w(x)$ and ι and no mark; otherwise x is *free*. Two facts. First, the transition that added $g(a)$ at a ward x is determined by a schema with x and a in roles, changes both states and leaves x recording a (Definition B.13, clause 4), so it was an interaction between x and a by Lemma Records of [23] and Definition A.14; hence the guardian of p, if p is a ward, is not q, and that of q is not p. Second, for free $x \neq y$ in Q holding no leaving mark, the introductory act τ at the introduction of x and y can be made machine-enabled by unary transitions at x and at y: an atom τ requires that is absent is the one atom of a schema of arity one requiring nothing and forbidding nothing but that atom, which is machine-enabled, and adding it adds nothing τ forbids, as a schema of arity one requiring nothing adds no mark in M, such a mark being added only together with a g-atom naming another role (Definition B.13, clauses 3 and 4), and no leaving mark, which is added only where a mark in M is required (Definition B.14, clause 2); and what τ forbids is a mark in M, absent at the free x and y, or a leaving mark, absent at x and y. Taking τ there, after volition changes of x and y, leaves x holding $e(y)$ and y holding $e(x)$, and is an interaction between x and y and no others.

If p and q are free, the second fact at $x := p$ and $y := q$ gives an extension by unary transitions after which τ at their introduction is machine-enabled, and every transaction it determines is a mutual recording of p and q, as in the proof of Proposition Openness of [23], so its transitions are interactions between p and q by Lemma A Mutual Recording is an Interaction of [23].

Otherwise let χ be the relay act for the statuses of p and q, bound with its ward roles to those of p and q that are wards, its free role to the free one, and the guardian role of each ward role to that ward's guardian. The related roles are bound to distinct agents, and two guardian roles are bound to one person only where p and q share a guardian, which Definition B.5 permits as the guardian roles add and delete nothing naming each other. Extend r as follows, by transitions none of which is an interaction between p and q: for every atom χ requires at a role that is the one atom of a schema of arity one and is absent at the agent bound there, the unary transition adding it, machine-enabled as above; and for every $e(\kappa)$ required at a guardian role or free role bound to x, with κ bound to $y \neq x$, where x does not hold $e(y)$, the unary transitions and the transaction of τ at the introduction of x and y by the second fact, x and y being free by Lemma B.18, holding no leaving mark, as p and q hold none and a guardian holds none by Lemma B.19, and $\{x, y\} \neq \{p, q\}$, since a guardian is bound only where its ward is one of p and q, and is not the other by the first fact. At the end χ is machine-enabled: a reflexive atom naming its holder is in $\hat{d}$; the unary atoms are present; a ward role bound to p has m and $g(a)$ at p; a guardian role bound to a has v of its ward at a, which holds w of it, by Lemma B.18; $e(\kappa)$ at a role bound to x with κ bound to y holds by the extension where $y \neq x$ and by reflexivity where $y = x$; and a mark in M that χ forbids is absent at the free agents and at the guardians, by Lemma B.18, and a leaving mark is absent at p, at q and at the guardians, by Lemma B.19. Every transaction χ determines there leaves p holding an atom naming q and q one naming p, changing both states, so by Lemma Records of [23] each records the other after it, the transition restricted to p is no transition of $\mathcal{F}(\{p\})$, and it is an interaction between p and q by Definition A.14. □

Theorem B.22 (Polite among the agents that have not left). *The protocol over a contained set of volition-guarded transactions that is closed and open among the agents that have not left is volitionally grassroots.*

Proof. Let R be such a set and $\mathcal{F}$ the protocol over it. The parts of the proof of Theorem Polite of [23] headed *Oblivious* and *Volitionally grassroots* use containment and closure and not openness, so $\mathcal{F}$ is oblivious, and the first interaction of two disjoint sets of agents in a safe run is induced by a transaction guarded by a member of each. Let $Q \subset \Pi$, let $P, P' \subseteq Q$ be disjoint and nonempty, and let $\hat{r}$ be a prefix of a correct run of $\mathcal{F}(Q)$ containing no interaction between P and P', in whose last configuration $p \in P$ and $q \in P'$ have not left. By Definition B.20 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 the extension is an interaction between P and P', the extension contains one. Otherwise extend it by a volition change of each agent of Q', where needed, so that $\lfloor t \rfloor$ lies in the volitional state of each, and by the volitional machine transaction that (t, Q') then induces, an interaction between p and q and hence between P and P'. Either way 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 A.7 it is a prefix of a correct run of $\mathcal{F}(Q)$. So $\mathcal{F}$ is interactive among the agents that have not left, and volitionally grassroots by Definition A.25. □

Theorem B.23 (Grassroots realisation with wards). *The protocol realising a contract with an introduction with wards, traceable provenance and volition is volitionally grassroots.*

Proof. By Proposition B.21 R_C is open among the agents that have not left. The proof of Proposition Closure of [23] uses traceable provenance and volition and nothing of the introductory act, so R_C is closed, and by Lemma Containment of [23], whose proof uses traceable provenance only, it is contained. By Theorem B.22 the protocol over it is volitionally grassroots. □

A contract with wards may also be agent-named [23], as this one is, whose groups are closed by the listed role of their members; its certification composes the two.

Definition B.24 (Agent-named [23]). A contract is *agent-named* if every role of each of its schemas is an agent role or a listed role, and every name term of each of its schemas has an agent role of that schema as its base.

Definition B.25 (Syntactically grassroots with wards). An agent-named contract C with an introduction with wards is *syntactically grassroots with wards* if:

(1) its introductory act, its relay acts and the schemas of arity one of clause 1 of Definition B.15 have agent roles and no name term;
(2) no listed role of a schema of C adds or deletes a mark in M, a leaving mark of C, an atom of the predicate the introductory act adds, or an atom of g or of w for the guardianship (m, ι, g, w, v) of a mark in M;
(3) the set of predicates occurring in C has traceable provenance, each atom ψ there required at an agent role;
(4) every schema of C has a connected role graph or has a guarding agent role in every connected component of its role graph.

Theorem B.26 (Grassroots realisation with wards and listed roles). *The protocol realising an agent-named contract syntactically grassroots with wards is volitionally grassroots.*

Proof. R_C with $\sim$ is well formed, by the lemma of [23] that the compilation of an agent-named contract is well defined. The proofs of Lemma No edge crosses instances and Proposition Closure for agent-named contracts of [23] use traceable provenance and volition, clauses 3 and 4 of Definition B.25, and nothing else of it, so R_C is closed, and by Lemma Containment for agent-named contracts of [23], whose proof uses clause 3 only, it is contained. The proofs of Lemmas B.18 and B.19 hold for C: by clause 2 a transition adds and deletes marks in M, leaving marks and atoms of g and of w only at the agents in agent roles, no schema deletes ι, and every guarding role forbids a leaving mark. The proof of Proposition B.21 holds for C: the schemas it uses have agent roles and no name term, by clause 1, so the transactions they determine have the participants their bindings name and are determined at every configuration that meets their requirements; and a transition that adds an atom of g is determined by a schema with the agents that atom concerns in agent roles, by clause 2, and changes the state of every participant, as in the proof of Lemma A transaction joins its participants of [23]. So R_C is open among the agents that have not left, and by Theorem B.22 the protocol over it is volitionally grassroots. □

Proposition B.27 (Certification). *The contract of Section 4.1 is agent-named and syntactically grassroots with wards.*

The proof walks all twenty-eight schemas and is given in Appendix C.1, beside them. The one mark is *child*, its guardianship (*child*, *identity*, *creator*, *created*, *parenting*), the introductory act *acquaint*, which adds *knows*, and the relay acts *child_acquaint* and *child_acquaint*$_2$: a child comes to know an agent through a parent, by acts the parent guards. The one leaving mark is *retired*, and a retired child has left (Lemma B.19). The listed roles are those of *close* and *child_close*, every member the leader lists, each deleting the membership of its members and nothing else, and those of *of_age* and *retire*, every parent the creator made, each deleting their *parenting* of the child and nothing else.

C The Abstraction Cascade, Artefact by Artefact

Figure 1 names six layers. This appendix has one subsection per layer. Each shows the artefact this paper produces there, says which artefact it was derived from and by what, and where the artefact is code points into the repository instead. We write A1 to A6 for the artefacts, the numbering being the figure's; the first layer has two parts, the clauses and their speech-act schemas, and the third two, the communicating volitional agents and the vGLP program.

The text shown here and the text in the body are the same: each artefact is a file that both places input, so the two cannot disagree.

C.1 A1: The Social Contract and Its Speech-Act Schemas

Five contracts, set out in full in Sections 2 to 4.1: GKK, GSG and GSN, of [23], GnW, of the AI-agents contract in preparation, and CSSN, which extends GSN and GnW. The clauses are not derived: they are drafted, and the rest of the abstraction cascade is derived from them. GKK is the base contract, identity, acquaintance and introduction; GSG adds friendship, and GSN messages and groups. GnW is shared with the AI-agents contract and is realised by CSSN's schemas for children, whose *child_unfriend*, *child_leave* and *child_close* render GnW:3 directly. CSSN:1 to CSSN:6 are new here.

The speech-act schemas. Twenty-eight schemas over sixteen predicates, Appendix B.2: the fourteen schemas of GSN [23], with *child* forbidden at both roles of *acquaint*, of *befriend* and of *join*, and at the role of the agent who comes to know in *introduce*, and *creator* and *parenting* of the other at the first role of *unfriend*, and fourteen for children. Converted from the clauses by judgement: this is the one step of the abstraction cascade that is not mechanical.

The checker of [23] decides the conditions of Appendix B.4; this contract meets those for a contract with wards (Proposition B.27).

The schemas are in the two tables at the top of the pages that follow, of the persons and of the groups and the messages, the first column of each naming the clause the schema renders.
Nothing else is required, forbidden, added or deleted, but *identity*, which every role requires except the role of *establish* and the child's role of *enter*, which forbid it and add it; and *retired*, which every guarding role forbids. Roles are listed in the order of the schema's parameters, one line each; after the semicolon come the variables, c an agent variable, the agent introduced, and x and n speech-act variables, a content and a group's name. *unfriend* is guarded at one role, and at the binding that swaps its roles is the other's: that is how either friend may end a friendship (GSG:2 and GnW:3). *child_unfriend*, *child_leave* and *child_close* are guarded by the parent, and *end_parent*, *of_age* and *retire* by the creator. The parent's role adds and deletes nothing in *child_acquaint*, $child_acquaint_2$, *child_befriend*, *child_adult_befriend*, *child_unfriend*, *child_join*, $child_join_2$, *child_leave* and *child_close*: the parent takes part by their will and their record, and gains no record of the acquaintance, the friendship or the membership. *acquaint* and *introduce* are GKK's, with *child* forbidden at both roles of the first and at the role the second introduces, so a child comes to know an agent only by *child_acquaint* or $child_acquaint_2$ (GnW:2 and CSSN:2), whose parent's role requires *knows* of the adult, or of the other parent; *knows* is a count, no schema forbidding or deleting it. *make_parent* is the creator's making of an adult they know a parent of their child, an act of the three, the creator answering on the child's phone at its setup; *end_parent* is the creator's ending of that parenthood; and *of_age* and *retire*, the creator's declaration of age and retiring of the child (GnW:4), each take the listed role of every parent the creator made, ending every guardianship of the child in one act, as closing a group ends every membership, and leave the child's friendships in place. *unfriend* forbids *creator* and *parenting* of the other at its first role, and *child_unfriend parenting* of the child at the friend's role, so no act ends the friendship of a parent and their child while they are its parent. The creator keeps $made(q, r)$ after a declaration of age, an entry that no act reads once $created(r)$ is gone. $child_join_2$ is GnW:2 for a child joining the group of a child, and *child_join* for a child joining the group of an adult. A member leaves by *leave* (GSN:4), a member who is no longer the leader's friend is dropped by *drop* (GSN:4), and a parent of a child member ends the membership by *child_leave*, realising GnW:3; the leader's own membership ends by *close*, the leader's leaving of GSN:4, or by *child_close* of a parent of a child leader, realising GnW:3; each takes the listed role of every member the leader lists, the leader among them, ends every membership in one act, and ends the group: its name cannot be used again, since *create* forbids $closed(n)$, and its leader neither admits, posts nor delivers on it, since each of those requires $member(\langle a \rangle \cdot n)$ at the leader. The leader keeps $listed(q, n)$ for the other members, an entry that no act reads once the group is closed. *deliver* sends a post from the leader to one member, and with *drop* is a schema of the contract no agent guards, which is a design decision: an unguarded act is owed, so a member is owed each post of the group not yet delivered to them, the group's earlier posts among them, while they are a member and the leader's friend and the group is open, and cannot decline one; and a member who is not the leader's friend is dropped in every correct run in which *drop* stays enabled (GSN:4). The will is given at the join, which a child and a parent of them guard: joining a group entrusts the child to its leader, as a parent entrusts a child to a coach, and a parent who no longer trusts the group ends the membership. A leader is on their own member list from the moment they create the group, so their own posting is *post_own*, of arity one, which adds the *posted* atom that *deliver* requires and nothing else; the leader then delivers to the others. *posted* ties a content to the group it was posted to, and *delivered* keeps a member from being sent the same post twice. The sender of a message, and a member who posts, add nothing: the record is the recipient's.

The two acts this paper follows. Read *child_befriend* role by role. The four roles are the two children r and s, a parent p of r and a parent q of s, and all four carry ?: that is "by an act of both, of Alice, and of Dave, a guardian of Bob, if Bob is a ward" (GnW:2), Alice being a guardian of the ward Carol. At r the atom *child* is required, so the agent there is a child; $friend(s)$ is forbidden, since a child who is already a friend cannot "enter" the friendship (CSSN:3); and the same atom is added, as befriending does. At s it is the mirror image. At p the atom $parenting(r)$ is required, p's record that they are a parent of r, and $friend(q)$, which is "Carol's acting parent Alice and Dave's acting parent Bob must be friends"; at q, the mirror again. The parents add and delete nothing: each takes part because the act requires their record and their will, and they gain no record of the friendship they approve.

child_unfriend is the parent's way of undoing it, and only the parent's role carries ?: "and so may Alice, a guardian of Carol" (GnW:3). Its roles are the child r, the friend s, and a parent p of r. At r the atom *child* is required and $friend(s)$ deleted; at s, $friend(r)$ is deleted; and at p, $parenting(r)$ is required, and nothing is added or deleted. The two friends end the friendship by *unfriend*, as any two friends do (GSG:2 and GnW:3). The two schemas differ in what satisfies volition of them (Definition B.12): *child_befriend* is guarded by all four roles, which suffices; *child_unfriend* is guarded by one role, so its role graph must be connected — p requires $parenting(r)$ and r requires $friend(s)$, giving the path $p - r - s$.

How the schemas render the clauses. That a child's identity is a ward and a guardian of it a parent, the definition before the box of Section 4.1, is the *child* that *enter* adds at the child's role, and *of_age* and *retire* delete; an agent holding none is a person, and a parent is a person holding *parenting* atoms. "Alice may act to establish their identity" (GKK:1) is *establish*, which adds *identity*; and "Only an adult may create a child's identity, and is then the child's creator and a parent of the child" (CSSN:1), choosing the child's key pair and becoming the child's friend (GnW:1), is *enter*, which forbids *identity* at the child's role and adds it there with *child* and the *creator* and *friend* atoms naming the creator. The child's role is guarded, and the creator performs its part in setting up the child's phone; the creator must be an adult, $\neg child$ at their role. "Alice, the creator of Carol, may make Bob, whom Alice knows, a parent of Carol, by an act of the three, and may end that parenthood by their own act" is *make_parent*, guarded by all three roles, and *end_parent*, guarded by the creator. "The creator of Carol may declare Carol of age, which ends every guardianship of Carol, Carol's friendships remaining" (CSSN:4) is *of_age*, and the retiring of GnW:4 is *retire*, which adds the *retired* that every guarding role forbids; each deletes *parenting* of the child at the creator and, through its listed role, at every parent the creator made. "Alice and Bob", two persons (GKK:2 and GSG:1), is the $\neg child$ forbidden at both roles of *acquaint* and of *befriend*, and at the role of the agent who comes to know in *introduce* (GKK:3); "Carol's acting parent Alice must know Bob" (CSSN:2) is *child_acquaint*, whose parent's role requires *knows* of the adult, and "Carol's acting parent Alice and Dave's acting parent Bob must know each other" the *knows* of the other parent required at each parent's role of $child_acquaint_2$; "who know each other" (GSG:1) is the *knows* of the other required at both roles of *befriend*, *child_befriend* and *child_adult_befriend*; and the $\neg child$ forbidden at the joiner's role of *join* leaves a child to join a group by GnW:2, through *child_join* or $child_join_2$. "Only a child may join a group Carol leads" (CSSN:6) is the $\neg child$ forbidden at the leader's role of *join*, so a group a child leads admits members only by $child_join_2$. Each child role requires *child* and each parent role forbids it, so no binding sends one person to a child role and to a parent's or an adult's; the two parent roles of $child_acquaint_2$, of *child_befriend*, and of $child_join_2$, are declared possibly one person (GnW:2) — the sibling case, *knows* and *friend* being reflexive. "Carol's acting parent Alice and Dave's acting parent Bob must be friends" is the $friend(q)$ required of p in *child_befriend*, and "Carol's acting parent Alice must be a friend of Bob" the $friend(a)$ required of p in *child_adult_befriend* (CSSN:3); "enter" is the forbidden *friend* atom, since an agent who is already a friend cannot enter the friendship; and "by an act of both, of Alice, and of Dave, a guardian of Bob, if Bob is a ward" (GnW:2) is that all roles are guarding. "The leader of a group may add a friend to it" (GSN:3) is the $friend(a)$ required at the child's role of *child_join*, and $friend(r)$ of $child_join_2$. A parent adds and deletes nothing when their child gains or loses a friend or a group; a group's leader gains the entry on their member list.

Proposition B.27 (Certification). *The contract of Section 4.1 is agent-named and syntactically grassroots with wards.*

Proof. *Agent-named* (Definition B.24). Every role of every schema is an agent role, but for the listed roles $a[\,listed(*, n)\,]$ of *close*, $r[\,listed(*, n)\,]$ of *child_close*, and $p[\,made(*, r)\,]$ of *of_age* and *retire*, and every name term is $\langle a \rangle \cdot n$ or $\langle r \rangle \cdot n$, its base the leader's agent role of its schema.

Introduction with wards, with $M = \{child\}$. (*child*, *identity*, *creator*, *created*, *parenting*) is a guardianship (Definition B.13). Clause 1: no schema deletes *identity*, and *child* is the one mark of M. Clause 2: *enter*, the only schema adding *child*, adds *identity* at the child's role and forbids it there. Clause 3: *enter* adds *child* and a *creator* atom once at the child's role, *of_age* and *retire* delete one of each, and no other schema adds or deletes either. Clause 4: the $creator(p)$ that *enter* adds at the child's role names the creator's role, at which *identity* is required, *child* forbidden and $created(r)$ added; and *of_age* and *retire* delete $creator(p)$ at the child's role exactly when they delete $created(r)$ at p. Clause 5: *enter*, the only schema adding $created(r)$, adds $parenting(r)$ with it; *enter* and *make_parent*, the only schemas adding $parenting(r)$, require *identity* and forbid *child* at that role; and *of_age* and *retire* delete $parenting(r)$ at the creator's role, deleting $created(r)$ there, and at their listed role and *end_parent* at the parent's role, forbidding $created(r)$ there. *retired* is a leaving mark (Definition B.14). Clause 1: every guarding role of every schema forbids it, *establish*'s included, and no schema deletes it. Clause 2: *retire*, the only schema adding it, adds it at the child's role, which requires *child*, a mark in M. Clause 3: *enter*, the only schema adding a *creator* atom, adds it naming the creator's role, which is guarding and so forbids *retired*. *acquaint* is an introductory act (Definition B.10): of arity two, guarded in both roles, adding *knows* of the other at each, so any two agents have an introduction; and it is unobstructed but for M (Definition B.15). Clause 1: at each role it requires *identity*, the one atom *establish* adds, which is of arity one, guarded, requires nothing and forbids nothing but that atom and *retired*, a leaving mark. Clause 2: it forbids *child*, a mark of M, and *retired*, a leaving mark. *child_acquaint* is a relay act (Definition B.16) with related roles r, a ward role of guardian role p, and a, a free role, each adding *knows* of the other, and the guardian role p: r requires *child*; a forbids *child*; p requires $parenting(r)$ and $knows(a)$ and forbids *child*; the schema forbids *retired* at all three roles and nothing else, and adds and deletes nothing at p. $child_acquaint_2$ is a relay act with two ward roles, r and s, of guardian roles p and q, each adding *knows* of the other, each guardian role requiring *parenting* of its ward and *knows* of the other guardian role, *knows* being reflexive, and forbidding *child*; the schema forbids *retired* at all four roles and nothing else, and adds and deletes nothing at p and q. Every other atom these schemas require is *identity*.

Clause 1 of Definition B.25. *acquaint*, *child_acquaint*, $child_acquaint_2$ and *establish* have agent roles and no name term.

Clause 2. The listed role of *close* deletes $member(\langle a \rangle \cdot n)$, that of *child_close* $member(\langle r \rangle \cdot n)$, and those of *of_age* and *retire* $parenting(r)$, and none adds or deletes anything else, so none adds or deletes *child*, *retired*, a *knows* atom, or a *creator* or *created* atom.

Clause 3, traceable provenance of the set of all sixteen predicates, each atom ψ required at an agent role. Every atom a schema adds has its agent arguments among the schema's agent roles, or, for the $knows(c)$ that *introduce* adds at the role of the agent who comes to know, an agent variable that is an argument of the $knows(c)$ required at its first role; and the base of every name term among its arguments is an agent role. Every speech-act variable among the arguments of an added atom, or in a name term among them, is an argument of an atom required at an agent role, or occurs in a name term among its arguments, or occurs in no atom required at a role: the x of the $received(x, p)$ that *message* adds, the x of the $posted(x, n)$ that *post_own* and *post* add, and the n of the $member(\langle a \rangle \cdot n)$ and $listed(a, n)$ that *create* adds occur in none; the n of each $closed(n)$ is an argument of the *listed* atom that the same agent role deletes, and so requires; and every other

GKK:1	$establish(p?)$:	$\neg identity,\ +identity$
GKK:2	$acquaint(p?, q?)$:	$\neg child,\ +knows(q)$
		$\neg child,\ +knows(p)$
GKK:3	$introduce(p?, q?; c)$:	$knows(q),\ knows(c)$
		$\neg child,\ knows(p),\ +knows(c)$
CSSN:1	$enter(p?, r?)$:	$\neg child,\ +created(r),\ +parenting(r),\ +friend(r),\ +knows(r)$
		$\neg identity,\ +identity,\ +child,\ +creator(p),\ +friend(p),\ +knows(p)$
CSSN:1	$make_parent(p?, q?, r?)$:	$\neg child,\ created(r),\ knows(q),\ \neg made(q, r),\ +made(q, r)$
		$\neg child,\ knows(p),\ \neg parenting(r),\ +parenting(r),\ +knows(r)$
		$child,\ creator(p),\ +knows(q)$
CSSN:1	$end_parent(p?, q; r)$:	$created(r),\ -made(q, r)$
		$\neg created(r),\ -parenting(r)$
CSSN:4	$of_age(p?, r, p[made(*, r)])$:	$-created(r),\ -parenting(r)$
		$-child,\ -creator(p)$
		$\neg created(r),\ -parenting(r)$
GnW:4	$retire(p?, r, p[made(*, r)])$:	$-created(r),\ -parenting(r)$
		$-child,\ -creator(p),\ +retired$
		$\neg created(r),\ -parenting(r)$
GSG:1	$befriend(p?, q?)$:	$\neg child,\ knows(q),\ \neg friend(q),\ +friend(q)$
		$\neg child,\ knows(p),\ \neg friend(p),\ +friend(p)$
GSG:2	$unfriend(p?, q)$:	$\neg creator(q),\ \neg parenting(q),\ -friend(q)$
		$-friend(p)$
CSSN:2	$child_acquaint(r?, a?, p?)$:	$child,\ +knows(a)$
		$\neg child,\ +knows(r)$
		$\neg child,\ parenting(r),\ knows(a)$
CSSN:2	$child_acquaint_2(r?, s?, p?, q?)$:	$child,\ +knows(s)$
		$child,\ +knows(r)$
		$\neg child,\ parenting(r),\ knows(q)$
		$\neg child,\ parenting(s),\ knows(p)$
CSSN:3	$child_befriend(r?, s?, p?, q?)$:	$child,\ knows(s),\ \neg friend(s),\ +friend(s)$
		$child,\ knows(r),\ \neg friend(r),\ +friend(r)$
		$\neg child,\ parenting(r),\ friend(q)$
		$\neg child,\ parenting(s),\ friend(p)$
CSSN:3	$child_adult_befriend(r?, a?, p?)$:	$child,\ knows(a),\ \neg friend(a),\ +friend(a)$
		$\neg child,\ knows(r),\ \neg friend(r),\ +friend(r)$
		$\neg child,\ parenting(r),\ friend(a)$
GnW:3	$child_unfriend(r, s, p?)$:	$child,\ -friend(s)$
		$\neg parenting(r),\ -friend(r)$
		$\neg child,\ parenting(r)$

GSN:2	$create(a?; n)$:	$\neg member(\langle a\rangle \cdot n),\ \neg closed(n),\ +member(\langle a\rangle \cdot n),\ +listed(a, n)$
GSN:3	$join(a?, q?; n)$:	$\neg child,\ member(\langle a\rangle \cdot n),\ friend(q),\ +listed(q, n)$
		$\neg child,\ \neg member(\langle a\rangle \cdot n),\ +member(\langle a\rangle \cdot n)$
GnW:2	$child_join(a?, s?, p?; n)$:	$\neg child,\ member(\langle a\rangle \cdot n),\ friend(s),\ +listed(s, n)$
		$child,\ \neg member(\langle a\rangle \cdot n),\ +member(\langle a\rangle \cdot n)$
		$\neg child,\ parenting(s)$
GnW:2	$child_join_2(r?, s?, p?, q?; n)$:	$child,\ member(\langle r\rangle \cdot n),\ friend(s),\ +listed(s, n)$
		$child,\ \neg member(\langle r\rangle \cdot n),\ +member(\langle r\rangle \cdot n)$
		$\neg child,\ parenting(r)$
		$\neg child,\ parenting(s)$
GSN:4	$leave(a, q?; n)$:	$-listed(q, n) \qquad -member(\langle a\rangle \cdot n)$
GSN:4	$close(a?, a[listed(*, n)]; n)$:	$-listed(a, n),\ +closed(n)$
		$-member(\langle a\rangle \cdot n)$
GSN:4	$drop(a, q; n)$:	$\neg friend(q),\ -listed(q, n) \qquad -member(\langle a\rangle \cdot n)$
GnW:3	$child_leave(a, s, p?; n)$:	$-listed(s, n)$
		$child,\ -member(\langle a\rangle \cdot n)$
		$\neg child,\ parenting(s)$
GnW:3	$child_close(r, p?, r[listed(*, n)]; n)$:	$child,\ -listed(r, n),\ +closed(n)$
		$\neg child,\ parenting(r)$
		$-member(\langle r\rangle \cdot n)$
GSN:1	$message(p?, q; x)$:	$friend(q)$
		$+received(x, p)$
GSN:5	$post_own(a?; x, n)$:	$member(\langle a\rangle \cdot n),\ \neg posted(x, n),\ +posted(x, n)$
GSN:5	$post(p?, a; x, n)$:	$member(\langle a\rangle \cdot n)$
		$member(\langle a\rangle \cdot n),\ listed(p, n),\ friend(p),\ \neg posted(x, n),\ +posted(x, n)$
GSN:5	$deliver(a, q; x, n)$:	$member(\langle a\rangle \cdot n),\ posted(x, n),\ listed(q, n),\ friend(q),$
		$\neg delivered(x, q, n),\ +delivered(x, q, n)$
		$member(\langle a\rangle \cdot n),\ +received_in(x, a, n)$

is an argument of a *listed* or *posted* atom, or in the name term of a *member* atom, required at an agent role.

Clause 4. *establish*, *create* and *post_own* have arity one, so their role graphs are connected, and the two roles of *close* have the base a, so its role graph is connected. *acquaint*, *introduce*, *enter*, *make_parent*, *befriend*, *child_acquaint*, $child_acquaint_2$, *child_befriend*, *child_adult_befriend*, *join*, *child_join* and $child_join_2$ are guarded by all their roles, each an agent role, so they have a guarding agent role in every connected component of their role graphs. The role graph of each of the rest is connected: in *of_age* and *retire*, p requires $created(r)$, and the listed role has the base p, giving $r - p - p[made(*, r)]$; in *end_parent*, p requires $made(q, r)$; in *unfriend*, p requires $friend(q)$; in *child_unfriend*, p requires $parenting(r)$ and r requires $friend(s)$, giving $p - r - s$; in *child_close*, p requires $parenting(r)$, and the listed role has the base r, giving $p - r - r[listed(*, n)]$; in *leave*, the leader requires $listed(q, n)$; in *child_leave*, the leader requires $listed(s, n)$ and the parent requires $parenting(s)$, giving $a - s - p$; in *message* the recipient requires $friend(p)$, and in *post* the leader requires $listed(p, n)$; and in *deliver* and *drop* the leader requires $listed(q, n)$. □

C.2 A2: The Volition-Guarded Transactions, from A1

Twenty-eight transactions in six groups, written in the local-state components below. Derived from the speech-act schemas of A1 by the compilation of Appendix B.3, term by term: each schema at each binding determines the transactions written below. Every transaction requires $\iota \geq 1$ at each of its participants, except Establish, and Enter at the child, which require $\iota = 0$ there. Each is determined by its schema at a binding (Definitions B.5 and B.7): its participants are the agents in the extents of the schema's roles (Definition B.6), those the binding sends its agent roles to and, in the closing of a group, every member the leader lists; an agent who signed a speech act it carries is not one; and two agent roles sent to one agent are carried out by that agent together, with the join of their effects, as by the parent of two children in child befriending, one participant and one guard.

The correspondence is one transaction per schema, twenty-eight of each, and one schema per act rather than per clause; each transaction carries the name of the schema it comes from. CSSN:1, GnW:3, GSN:4 and GSN:5 determine three each; GnW:2, CSSN:2 and CSSN:3 two each; and CSSN:4, GnW:4, GKK:1, GKK:2, GKK:3, GSG:1, GSG:2, GSN:1, GSN:2 and GSN:3 one each.

The local state. The contract's speech-act schemas (Appendix B.2) determine the local state of an agent p, each component standing for the atoms of a predicate (Lemma C.4), and all initially empty or zero:

	component	stands for
identity, knowing and friends	$\iota_p \in \{0,1\}$, whether p has established an identity	*identity*
	η_p, a multiset of the agents p knows	*knows*
	$c_p \subseteq P$, the friends of p	*friend*
guardianship	$\chi_p \in \{0,1\}$, whether p is a child	*child*
	$\omega_p \subseteq P$, the creator of p, if p is a child	*creator*
	$\xi_p, \kappa_p \subseteq P$, the children p created and the children of whom p is a parent	*created*, *parenting*
	$\varphi_p \subseteq P^2$, the pairs (q,r) of a parent q that p made of a child r	*made*
	$\theta_p \in \{0,1\}$, whether p is retired	*retired*
groups	$\mu_p \subseteq \mathcal{G}$, the groups p is a member of	*member*
	$\lambda^g_p \subseteq P$, the member list of a group g that p leads	*listed*
	$\zeta_p \subseteq \mathcal{G}$, the groups p led and closed	*closed*
	π^g_p, the contents posted to a group g that p leads, and δ^g_p, the pairs (x,q) of a content delivered there and the member it was delivered to	*posted*, *delivered*
messages	m_p, a multiset of the pairs (x,q) of a content p received and the friend q it came from	*received*
	v^g_p, the contents delivered to p in a group g	*received_in*

A group is named by the community name $g = \langle a \rangle \cdot n$ of its leader a and a speech act n that a signs (Appendix B.1); we write $ld(g) = a$ and let $\mathcal{G}$ denote the set of these names. A content x is a speech act too. The atom of the reflexive *friend*, or *knows*, naming p itself is written as the alternative $q = p$ of a condition $q \in c_p$, or $q \in \eta_p$. An agent is a *child* when $\chi_p = 1$ and a *person* otherwise, the *creator* of a child r when $r \in \xi_p$, and a *parent* of r when $r \in \kappa_p$. Every transaction requires $\iota \geq 1$ at each of its participants, except Establish, and Enter at the child, which require $\iota = 0$ there, and $\theta = 0$ at each of its guards; the transactions below leave these requirements out.

The lemmas on the local state, with their proofs.

Lemma C.1 (Mutuality [41]). *In any run of the grassroots social graph, $q \in c_p \iff p \in c_q$ for all agents $p, q \in P$.*

Lemma C.2 (Parenting and Membership Mutuality). *In any run of the child-safe social network, $r \in \xi_p \iff p \in \omega_r$, and $r \in \xi_p$ implies $r \in \kappa_p$, for all $p, r \in P$, and $g \in \mu_q \iff q \in \lambda^g_{ld(g)}$ for all $q \in P$ and $g \in \mathcal{G}$ with $g \notin \zeta_{ld(g)}$.*

Proof. The entering, the declaration of age and the retiring update ξ_p, ω_r and κ_p together, the making and the ending of a parent touch κ_q of a parent q with $r \notin \xi_q$; the creations, the joins, the leaves and the drops update μ and λ together; a closing of g removes g from μ_q for every $q \in \lambda^g_{ld(g)}$ and adds g to $\zeta_{ld(g)}$ (below); and no other transaction touches them. □

Lemma C.3 (Fresh Identity). *In any run of the child-safe social network, an agent r with $\iota_r = 0$ has the initial local state, and no local state records r.*

Proof. Every transaction but Establish and the enterings requires $\iota \geq 1$ at all its participants, and Establish and the enterings require $\iota_r = 0$ and leave $\iota_r = 1$ at the participant r whose identity they establish, so every transaction r takes part in leaves $\iota_r \geq 1$, and an agent with $\iota_r = 0$ has taken part in none. Every agent a transaction adds to a local state is one of its participants, or the signer of a speech act it carries, who signed it and so had an identity. □

Lemma C.4 (Representation). *In every local state of a run of the protocol realising the contract, every atom of friend, creator, created, parenting, made, member, listed, closed, posted, delivered and received_in, and identity, child and retired, has multiplicity at most one, and a child holds one creator atom.*

Proof. By induction on the run. The initial state is empty, and a transition changes a local state only by the atoms its schema adds and deletes there. Every schema adds an atom of these predicates only where the atom is absent. By what the schema forbids: *establish* and *enter* forbid the *identity* they add; *make_parent* forbids the *made* and *parenting* atoms it adds; *befriend*, *child_befriend* and *child_adult_befriend* forbid the *friend* atom they add; *create*, *join*, *child_join* and *child_join*$_2$ forbid the *member*$(\langle a \rangle \cdot n)$ they add at a role and add *listed* of that role at the leader a, who holds no *closed*(n), *create* forbidding it and the others requiring *member*$(\langle a \rangle \cdot n)$ there, and so holds *listed*(q,n) exactly when q holds *member*$(\langle a \rangle \cdot n)$ (Lemma C.2); *close* and *child_close* delete *listed*(a,n) at the leader as they add *closed*(n) there, and *create*, the only schema adding *listed*(a,n) at the leader, forbids *closed*(n); *post_own* and *post* forbid the *posted* atom they add; and *deliver* forbids the *delivered*(x,q,n) atom it adds at the leader, adds *received_in*(x,a,n) at q with it, and no schema deletes *delivered*. Because the atom is at, or names, an agent with no identity, who holds no atom and whom no state records (Lemma C.3): *enter*, the only schema adding *child*, *creator* and *created*, adds them at the child, or at the creator naming the child, with the *parenting* of the creator; *of_age* and *retire* delete them, and the child's identity being established, no later schema adds them there. *retired* is added only by *retire*, which deletes the child's *child* atom, and no schema adds a *child* atom at an agent with an identity, so it is added at most once. □

So the components of the table above are those of the local state: each set component is the set of the arguments of the atoms of its predicate, θ_p is the multiplicity of *retired*, χ_p that of *child*, and ι_p, η_p and m_p are the multiplicity, or the multiset, of the atoms of *identity*, *knows* and *received*.

Example C.5 (Keys and Knowledge). (1) **Establish**(p), where $\iota_p = 0$:
$\iota'_p := 1$, guarded by p.

(2) **Acquaint**(p,q), where $p \neq q$ and $\chi_p = \chi_q = 0$:
$\eta'_p := \eta_p \uplus \{q\}$ and $\eta'_q := \eta_q \uplus \{p\}$, guarded by $\{p,q\}$.

(3) **Introduce**(p,q,c), where $p \neq q$, $q \in \eta_p$, $c \in \eta_p$, $p \in \eta_q$ and $\chi_q = 0$:
$\eta'_q := \eta_q \uplus \{c\}$, guarded by $\{p,q\}$.

In each transaction, all components not explicitly updated are preserved.

A person establishes their own identity. Two persons come to know each other by an act of both, and a person who knows two agents introduces one to the other by an act of the person and the other, a person, who then knows the one introduced, the one introduced taking no part and coming to know nobody; a child comes to know an agent by the transactions of Example C.9. Knowing is never undone, and is counted: an agent holds q in η_p once for each act by which they came to know q.

Example C.6 (The Grassroots Social Graph). (1) **Befriend**(p,q), where $p \neq q$, $\chi_p = \chi_q = 0$, $q \in \eta_p$, $p \in \eta_q$ and $q \notin c_p$:
$c'_p := c_p \cup \{q\}$ and $c'_q := c_q \cup \{p\}$, guarded by $\{p,q\}$.

(2) **Unfriend**(p,q), where $q \in c_p$, $q \notin \kappa_p \cup \omega_p$ and $p \notin \kappa_q \cup \omega_q$:
$c'_p := c_p \setminus \{q\}$ and $c'_q := c_q \setminus \{p\}$, guarded by either p or q.

Befriending requires both agents to be willing, both to be persons, and each to know the other; unfriending can be initiated by either. The condition is the ¬*child* that *befriend* forbids at both roles, and $\chi_p = 0$ is what holding no *child* atom comes to.

Transaction equivalence. The equivalence $\sim$ is that of Definition B.8, generated by relating two transactions of the same participants when one schema at one binding determines both. The transactions one schema determines at one binding, with one set of participants, are a *part* of their class [37], its members differing only in the configuration in which they occur, and bindings that determine a common machine transaction put their parts in one class. All establish(p) transactions form one class; all acquaint(p,q) transactions form one class, with the binding that swaps the roles, and all introduce(p,q,c) transactions one class; all befriend(p,q) transactions form one class, with the binding that swaps the roles; unfriend(p,q) at its two bindings is one class of two parts, guarded by p and by q.

Example C.7 (Messages). (1) **Message**(p,q,x), where $p \neq q$ and $q \in c_p$:
$m'_q := m_q \uplus \{(x,p)\}$, guarded by p.

A message is guarded by the sender, and it is the sender's holding a *friend* atom naming the recipient that licenses it: an agent sends a message only to a friend. The recipient holds the content with the sender.

Example C.8 (Groups). (1) **Create group**(g), where $a := ld(g)$, $g \notin \mu_a$ and $g \notin \zeta_a$:
$\mu'_a := \mu_a \cup \{g\}$ and $\lambda'^g_a := \{a\}$, guarded by a.

(2) **Close group**(g), where $a := ld(g)$ and $g \in \mu_a$, over the participants λ^g_a:
$\mu'_q := \mu_q \setminus \{g\}$ for every $q \in \lambda^g_a$, $\lambda'^g_a := \lambda^g_a \setminus \{a\}$ and $\zeta'_a := \zeta_a \cup \{g\}$, guarded by a.

(3) **Join group**(g,q), where $a := ld(g) \neq q$, $g \in \mu_a$, $q \in c_a$, $\chi_a = \chi_q = 0$ and $g \notin \mu_q$:
$\mu'_q := \mu_q \cup \{g\}$ and $\lambda'^g_a := \lambda^g_a \cup \{q\}$, guarded by $\{a,q\}$.

(4) **Leave group**(g,q), where $a := ld(g) \neq q$ and $g \in \mu_q$:
$\mu'_q := \mu_q \setminus \{g\}$ and $\lambda'^g_a := \lambda^g_a \setminus \{q\}$, guarded by q.

(5) **Drop from group**(g,q), where $a := ld(g) \neq q$, $g \in \mu_q$ and $q \notin c_a$:
$\mu'_q := \mu_q \setminus \{g\}$ and $\lambda'^g_a := \lambda^g_a \setminus \{q\}$, unguarded.

(6) **Post to group**(g,p,x), where $a := ld(g) \neq p$, $g \in \mu_p$, $g \in \mu_a$, $p \in \lambda^g_a$, $p \in c_a$ and $x \notin \pi^g_a$:
$\pi'^g_a := \pi^g_a \cup \{x\}$, guarded by p.

(7) **Post to own group**(g,x), where $a := ld(g)$, $g \in \mu_a$ and $x \notin \pi^g_a$:
$\pi'^g_a := \pi^g_a \cup \{x\}$, guarded by a.

(8) **Deliver to member**(g,q,x), where $a := ld(g) \neq q$, $x \in \pi^g_a$, $g \in \mu_a$, $q \in \lambda^g_a$, $q \in c_a$, $g \in \mu_q$ and $(x,q) \notin \delta^g_a$:
$\delta'^g_a := \delta^g_a \cup \{(x,q)\}$ and $v'^g_q := v^g_q \cup \{x\}$, unguarded.

In each transaction, all components not explicitly updated are preserved.

Delivery is the dissemination step, its participants the leader and one member, and no agent guards it or the dropping of a member: while the leader holds a post in an open group and a member who is their friend is on their list, the member is owed it, unless it was delivered to them before, the posts made before they joined among them; and a member who is not the leader's friend is dropped in every correct run in which the drop stays enabled. A content is delivered in a group only if it was posted to that group, and the leader's record of what they have delivered there keeps a member from receiving the same post twice, also when they leave the group and rejoin it; the member holds the content with the group. Closing a group has as its participants every member the leader lists,

the leader among them, and ends every membership in one act. A closed group admits, carries and delivers nothing, and its name is not used again.

Transaction equivalence. For each g, all create-group(g) transactions form one class, and so do all close-group(g) transactions with one list of members; for each (g,q), all join-group and all leave-group transactions form one class each; for each (g,q), all drop transactions form one class; for each (g,p,x), all post transactions form one class; for each (g,x), all post-to-own-group transactions form one class; and for each (g,q,x), all delivery transactions form one class.

Example C.9 (The Child-Safe Social Graph). (1) **Enter**(p,r), where $\chi_p = 0$ and $\iota_r = 0$:
$\iota'_r := 1$, $\chi'_r := 1$, $\omega'_r := \{p\}$, $\eta'_r := \{p\}$, $c'_r := \{p\}$, $\xi'_p := \xi_p \cup \{r\}$, $\kappa'_p := \kappa_p \cup \{r\}$, $\eta'_p := \eta_p \uplus \{r\}$ and $c'_p := c_p \cup \{r\}$, guarded by $\{p,r\}$.

(2) **Make parent**(p,q,r), where $r \in \xi_p$, $\chi_p = \chi_q = 0$, $\chi_r = 1$, $q \in \eta_p$, $p \in \eta_q$, $r \notin \kappa_q$ and $(q,r) \notin \varphi_p$:
$\varphi'_p := \varphi_p \cup \{(q,r)\}$, $\kappa'_q := \kappa_q \cup \{r\}$, $\eta'_q := \eta_q \uplus \{r\}$ and $\eta'_r := \eta_r \uplus \{q\}$, guarded by $\{p,q,r\}$.

(3) **End parent**(p,q,r), where $r \in \xi_p$, $(q,r) \in \varphi_p$ and $r \notin \xi_q$:
$\varphi'_p := \varphi_p \setminus \{(q,r)\}$ and $\kappa'_q := \kappa_q \setminus \{r\}$, guarded by p.

(4) **Declare of age**(p,r), where $r \in \xi_p$ and $\chi_r = 1$, over the participants $\{p,r\} \cup \{q : (q,r) \in \varphi_p\}$:
$\xi'_p := \xi_p \setminus \{r\}$, $\kappa'_p := \kappa_p \setminus \{r\}$, $\kappa'_q := \kappa_q \setminus \{r\}$ for every $(q,r) \in \varphi_p$, $\chi'_r := 0$ and $\omega'_r := \emptyset$, guarded by p.

(5) **Retire**(p,r), where $r \in \xi_p$ and $\chi_r = 1$, over the participants $\{p,r\} \cup \{q : (q,r) \in \varphi_p\}$:
as Declare of age(p,r), and $\theta'_r := 1$, guarded by p.

(6) **Child acquaint**(r,a,p), where $r \neq a$, $r \in \kappa_p$, $\chi_r = 1$, $\chi_a = \chi_p = 0$, and $a \in \eta_p$ or $a = p$:
$\eta'_r := \eta_r \uplus \{a\}$ and $\eta'_a := \eta_a \uplus \{r\}$, guarded by $\{r,a,p\}$.

(7) **Child acquaint with two children**(r,s,p,q), where $r \neq s$, $r \in \kappa_p$, $s \in \kappa_q$, $\chi_r = \chi_s = 1$, $\chi_p = \chi_q = 0$, $q \in \eta_p$ or $q = p$, and $p \in \eta_q$ or $p = q$:
$\eta'_r := \eta_r \uplus \{s\}$ and $\eta'_s := \eta_s \uplus \{r\}$, guarded by $\{r,s,p,q\}$.

(8) **Child befriend**(r,s,p,q), where $r \neq s$, $r \in \kappa_p$, $s \in \kappa_q$, $\chi_r = \chi_s = 1$, $\chi_p = \chi_q = 0$, $q \in c_p$ or $q = p$, $s \in \eta_r$, $r \in \eta_s$, and $s \notin c_r$:
$c'_r := c_r \cup \{s\}$ and $c'_s := c_s \cup \{r\}$, guarded by $\{r,s,p,q\}$.

(9) **Child-adult befriend**(r,a,p), where $r \neq a$, $r \in \kappa_p$, $\chi_r = 1$, $\chi_a = \chi_p = 0$, $a \in c_p$ or $a = p$, $a \in \eta_r$, $r \in \eta_a$, and $a \notin c_r$:
$c'_r := c_r \cup \{a\}$ and $c'_a := c_a \cup \{r\}$, guarded by $\{r,a,p\}$.

(10) **Child unfriend**(r,s,p), where $r \in \kappa_p$, $\chi_r = 1$, $\chi_p = 0$, $s \in c_r$ and $r \notin \kappa_s$:
$c'_r := c_r \setminus \{s\}$ and $c'_s := c_s \setminus \{r\}$, guarded by p.
In each transaction, all components not explicitly updated are preserved.

Entering is an act of the child's creator and of the child's role, whose part the creator performs in setting up the child's phone, holding both phones; it requires the creator to be a person and the child's identity to be new, and makes the creator the child's first acquaintance and friend. Making a parent is an act of the creator, the adult and the child, the adult and the child coming to know each other; ending a parenthood is the creator's act. A child acquaintance is an act of the child, an adult and a parent of the child who knows the adult, and a child acquaint with two children of the two and a parent of each, the parents knowing each other; the parents' states do not change, and the two come to know each other. A declaration of age is the act of the creator, and ends the guardianship of every parent of the child, the creator's and those the creator made; the child's friendships remain. Retiring the child does the same and leaves the child retired, and a retired agent guards no transaction.

C.2.1 Child-Safe Social Network. Creating a group, posting and delivering are unchanged from Example C.8. What changes for children is joining, leaving and closing: a child joins a group led by their friend, with a parent of each child among them willing; a parent of a child member may end the membership; and a parent of a child leader may close the group, which is their only act on a group their child leads.

Example C.10 (The Child-Safe Social Network). (1) **Child join, adult leader**(g,s,p), where $a := ld(g) \neq s$, $\chi_a = \chi_p = 0$, $\chi_s = 1$, $g \in \mu_a$, $s \in c_a$, $s \in \kappa_p$ and $g \notin \mu_s$:
$\mu'_s := \mu_s \cup \{g\}$ and $\lambda'^g_a := \lambda^g_a \cup \{s\}$, guarded by $\{a,s,p\}$.

(2) **Child join, child leader**(g,s,p,q), where $r := ld(g) \neq s$, $g \in \mu_r$, $s \in c_r$, $r \in \kappa_p$, $s \in \kappa_q$, $\chi_r = \chi_s = 1$, $\chi_p = \chi_q = 0$ and $g \notin \mu_s$:
$\mu'_s := \mu_s \cup \{g\}$ and $\lambda'^g_r := \lambda^g_r \cup \{s\}$, guarded by $\{r,s,p,q\}$.

(3) **Child leave**(g,s,p), where $a := ld(g) \neq s$, $g \in \mu_s$, $s \in \kappa_p$, $\chi_s = 1$ and $\chi_p = 0$:
$\mu'_s := \mu_s \setminus \{g\}$ and $\lambda'^g_a := \lambda^g_a \setminus \{s\}$, guarded by p.

(4) **Child close**(g,p), where $r := ld(g)$, $g \in \mu_r$, $r \in \kappa_p$, $\chi_r = 1$ and $\chi_p = 0$, over the participants $\{p\} \cup \lambda^g_r$:
$\mu'_q := \mu_q \setminus \{g\}$ for every $q \in \lambda^g_r$, $\lambda'^g_r := \lambda^g_r \setminus \{r\}$ and $\zeta'_r := \zeta_r \cup \{g\}$, guarded by p.
In each transaction, all components not explicitly updated are preserved.

Transaction equivalence. As above, the transactions one schema determines at one binding are a part of their class. All child-befriend(r,s,p,q) transactions, with the same four participants, form one class, and a child befriend naming another parent of either child is another class; similarly for child acquaint with two children(r,s,p,q), child acquaint(r,a,p), child-adult befriend(r,a,p) and child-unfriend(r,s,p). The parents being stationary, child-befriend(r,s,p,q) and child-befriend(r,s,p',q) change the same states, and a step of each fulfils the will for its own class (Definition A.10). Where two persons are each a parent of both children, the bindings of child acquaint with two children, of child befriend and of child join, child leader, that exchange them determine the same transactions, with the same guards, and are two parts of one class. Each establishing, each entering, each making and each ending of a parent, at its binding, forms one class, and so does each declaration of age and each retiring with one list of the parents the creator made, and so does each child join and child leave, and each child close with one list of members. The join of Example C.8 at (g,q) is a class of a different schema, as is its leave.

The two properties. They are presented in Section 4.4; each proof ranges over every transaction above.

Proposition 4.3 (Parents' Capabilities). *In every run of the child-safe social network protocol:*

(1) A child comes to know an agent only with the consent of a parent of theirs, who knows the agent; or, if the agent is a child, with the consent of a parent of each, the two parents knowing each other.
(2) A child befriends an agent only with the consent of a parent of theirs, who is the agent's friend; or, if the agent is a child, with the consent of a parent of each, the two parents being friends.
(3) A child joins a group only with the consent of a parent of theirs, the leader being the child's friend; and of a parent of the leader too, if the leader is a child.
(4) A parent of a child can always end, by their own act, any friendship of the child but one with another parent, and any membership of the child in a group.
(5) A child stops being a child only by an act of their creator.

Proof. Let r be an agent with $\chi_r = 1$, a child, in a configuration reached by a run, and call p a parent of r when $r \in \kappa_p$. First, r has been a child since their identity was established. Only Enter sets χ_r, and it requires $\iota_r = 0$, so a child was entered before taking part in any other transaction; and a declaration of age or a retiring, which sets $\chi_r = 0$, cannot be followed by another entering of r. So every transaction r took part in since their entering found $\chi_r = 1$.

(1) The transactions adding to η_r are acquaint, introduce, enter, make parent, child acquaint and child acquaint with two children. Acquaint requires $\chi_r = 0$ at both participants, and introduce at the agent who comes to know, so neither adds at r. Enter adds the creator, who guards it, *knows* being reflexive. Make parent(p,q,r) adds the adult q, and is guarded by the creator p, a parent of r, who knows q. Child acquaint(r,a,p) is guarded by p, a parent of r, and requires $a \in \eta_p$ or $a = p$; child acquaint with two children(r,s,p,q) is guarded by p, a parent of r, and q, a parent of s, and requires $q \in \eta_p$ or $q = p$.

(2) The transactions adding to c_r are befriend, enter, child befriend and child-adult befriend. Befriend requires $\chi_r = 0$ and is excluded. Enter adds the creator, who guards it, *friend* being reflexive. Child-adult befriend(r,a,p) is guarded by p, a parent of r, and requires $a \in c_p$ or $a = p$; child befriend(r,s,p,q) is guarded by p, a parent of r, and q, a parent of s, and requires $q \in c_p$ or $q = p$.

(3) The transactions adding to μ_r a group g that r does not lead are join group, which requires $\chi_r = 0$ and is excluded, and the two child joins: child join, adult leader, is guarded by a parent of r and requires $r \in c_{ld(g)}$; child join, child leader, is guarded in addition by a parent of the leader.

(4) The preconditions of child unfriend, of the child leave and of the child close are $r \in \kappa_p$ with $s \in c_r$ and $r \notin \kappa_s$, with $g \in \mu_r$ and $ld(g) \neq r$, and with $g \in \mu_r$ and $ld(g) = r$, respectively, the conditions at the leader and at the members of g holding by Lemma C.2, as g is open, and each is guarded by p; so each is machine-enabled whenever the friendship or the membership holds.

(5) Declare of age and Retire are the only transactions setting χ_r to 0, each guarded by the creator p of r, with $r \in \xi_p$; and r guards neither. □

Proposition 4.4 (Grassroots Property). *The child-safe social network protocol (Appendix C.2) is volitionally grassroots.*

Proof. The contract is agent-named and syntactically grassroots with wards (Proposition B.27), so the protocol realising it is volitionally grassroots by Theorem B.26. □

C.3 A3: Volition-Guarded GLP, from A2

A volition-guarded transaction changes the machine states of its participants at once, while a communicating volitional agent changes only its own state and reaches another only by a message [37]. *Communicating volitional agents* (CVA) [37] are an implementation-ready restriction of volition-guarded transactions, whose transactions are of four forms: *discover*, by which two agents come to know each other, an act both persons will; *introduce*, by which two agents that a common peer introduced come to know each other, with no person asked; *communicate*, which moves a message from one agent's outbox to another's inbox, an agent addressing only agents it knows; and the unary *app* transactions by which a platform is specified. Knowing, at this level, is which machines can send messages to each other; the platform records whom a person knows by the contract, and who is a friend, a parent or a child. Every CVA protocol is volitionally grassroots [37].

The canonical platform. Volition-guarded transactions are implemented by communicating volitional agents, for every local specification, by vGLP's *canonical CVA platform*: two-phase commit coordinated by the agent that initiates the transaction [31, 37], with the person's will as the vote. The initiator sends the transaction, naming a part of its class, to every other participant; a participant whose approval the part needs checks its own precondition in it and asks its person, and every other participant votes by its machine; each votes yes or no to the initiator, which commits on a yes from every participant and aborts on a no, and tells every participant. A participant that has voted yes holds the attempt prepared until the outcome, and takes part in no other attempt meanwhile, so the preconditions it voted on hold when the transaction is carried out. The realisation of this contract is local: each schema at each binding determines a part of its class for each list of the agents its listed role takes, with the persons in its guarding roles as the part's guard set, and each role's precondition and effect are on the state of each agent in its extent, the base of a listed role holding exactly that list (Definition B.7) [37]. The parts of *child_close* have an *opener*, the child leader r, whose state lists the members: the parent's agent sends r's agent a request, which is the parent's approval, and r's agent proposes the closing with its list [37].

Lemma C.11 (Applicability). *The realisation of the contract is an effective local volition-guarded specification [37].*

Proof. A contract written in speech-act schemas is a local specification, each schema at each binding determining a part of its class for each list of the people of its listed roles, with the persons in its guarding roles as its guard set; and it is effective where every variable of a schema that occurs in an atom it requires or forbids also occurs in an atom it deletes or adds, the atoms each unguarded schema requires at its first role name every other role and every variable of its deleted and added atoms, and the listed roles of a schema have one base, which is its only guarding role or, if it is not a guarding role, its first role, the opener of the guarded parts of the schema [37]. The agents of the contract's schemas are their roles, and their variables are speech-act variables: x and n occur in an atom of *received*, *received_in*, *posted*, *delivered*, *member*, *listed* or *closed* that each schema using them adds or deletes. The unguarded schemas are *deliver* and *drop*, and at their first role, the leader's, they require *listed*(q,n), and *deliver posted*(x,n), which name q, n and x. The schemas with a listed role are *close*, whose only guarding role is a, the base of its listed role, and *child_close*, the base of whose listed role is its first role r, which is not a guarding role and opens its guarded

parts, and the guarded parts of a class of *child_close* are all determined by *child_close*, with the one opener r. □

Theorem C.12 (Correctness of the Implementation). *The canonical CVA platform of the contract is a correct implementation of the child-safe social network protocol, and is complete for its strongly fair runs: every connected, unobstructed, steady and strongly fair run of the platform is mapped to a correct run of the protocol, and every strongly fair correct run of the protocol is the image of a run of the platform.*

Proof. By Lemma C.11 the realisation of the contract is a local volition-guarded specification, and the theorem CVA Implementation of [37] states exactly this of the canonical CVA platform, with an introduction rule, of every such specification. □

Corollary C.13 (Safety of the Implementation). *Every configuration reached by a run of the canonical CVA platform of the contract satisfies Proposition 4.3 under the mapping of Theorem C.12.*

Proof. The safety part of the proof of the theorem CVA Implementation [37] shows that the mapping takes every safe run of the platform to a safe run of the protocol, whether or not the run is connected, unobstructed, steady or strongly fair. So a configuration reached by a run of the platform is mapped to one reached by a run of the protocol, which satisfies Proposition 4.3. □

Safety and liveness. Theorem C.12 states what vGLP proves of its canonical platform [37], applied to the transactions of Appendix C.2: every connected, unobstructed, steady and strongly fair run of the implementation is mapped to a correct run of the specification, and every strongly fair correct run of the specification is the image of a run of the implementation. Corollary C.13 carries the parents' capabilities of Proposition 4.3 to every configuration the implementation reaches. A run is connected if the agents of two persons who will to befriend each other know each other, and the agent of each parent of a child member of a group knows the agent of its leader (below); the compiled agent secures strong fairness [37]; an unobstructed run is one in which an agent that is again and again ready to propose an act it wills is again and again free of other attempts and of waiting starts, and a steady run is one with no class, willed and not taken, whose part that holds with all its guards for it changes without end. The program is the compilation of the canonical platform, which [37] proves correct, and its correctness depends on GLP being secure. The parents' capabilities are safety properties, and hold in every run (Corollary C.13); a parent's ending of a friendship is a liveness property, and is carried out in every run that meets the conditions of Theorem C.12.

Knowing every participant. The runs Theorem C.12 covers are *connected*: of each part of a class that holds, with all its guards for the class, from some point on, its opener, or one of its guards, knows every other participant [37]. Agents come to know each other by Discover, which both persons will, or by an introduction from an agent that knows both, and knowing is never undone [37]. The canonical platform introduces agents in two ways [37]. By the contract's introduction rule, with *knows* its introduction predicate, an agent introduces to each other every two agents its person knows by the contract; so an agent knows, at the level of the machines, every agent its person knows and every agent known to an agent that knows it, and the contract's acquaintances and introductions (GKK:2, GKK:3 and CSSN:2), which record knowing at the level of the contract, are proposed between machines that know each other already, two persons' by Discover. By **Commit**, the initiator of an attempt introduces to each other its other participants, so the participants of a committed transaction know each other, two children who come to know each other among them. The request of an entering sends the child's phone a hello carrying it, and the creator, holding both phones at the setup, answers yes there, which is the child's approval [37]; a parent the creator makes and the child come to know each other by the making.

In the acts of GSN the participants know each other by the contract, or did when a member joined the group, but in acquaintance, whose two agents know each other by Discover or through a common acquaintance; and of an introduction the introducer knows both. Of each child act, one guard knows every other participant:

- of a making of a parent, the creator, who knows the adult and the child; of the ending of a parenthood, the creator, who knows the parent made; and of a declaration of age and a retiring, the creator, who knows the child and every parent they made;
- of a child acquaintance, the parent p of r, who knows r, by the entering or the making, and a; and of a child acquaint with two children, the parent p of r: p knows r and q, and q, who knows p and s, introduced s to it;
- of a child befriending, the parent p of r: p knows r and q, and q, who knows p and s, introduced s to it;
- of a child-adult befriending, the parent p, who knows r and a;
- of a child unfriending, its guard p: p knows r, and r, who knows p and s, introduced s to it;
- of a child join with an adult leader, the child, who knows the leader and the parent; and with a child leader r, the joining child s, who knows r and q, and to whom r, who knows p and s, introduced p;
- of a child close, its opener r, who knows p, and every member of whose group joined as its friend; and of a child leave, its guard p, who knows s, once s has introduced the leader to it, as its introduction rule does while s knows the leader, and as the commit of s's join did where p guarded the join.

So a run is connected if the agents of two persons who will to befriend each other know each other, and the agent of each parent of a child member of a group knows the agent of its leader. Below, the canonical platform is instantiated for the two transactions of Example 4.2.

An *attempt* at a transaction names a part of the transaction's class and is proposed by an *initiator*, a guard of that part, or its *opener* where no person guards it, the leader for a delivery or a drop; its guard set is the part's, and the other members of its guard set are *asked* [37]. Each participant's precondition and effect in the part are its own. For the part that *child_befriend* determines at the binding (r, s, p, q): at r, the precondition $\chi_r = 1$, $s \in \eta_r$ and $s \notin c_r$ and the effect $c_r' := c_r \cup \{s\}$; at s, $\chi_s = 1$, $r \in \eta_s$ and $r \notin c_s$, and $c_s' := c_s \cup \{r\}$; at p, $\chi_p = 0$, $r \in \kappa_p$, and $q \in c_p$ or $q = p$, and no effect; at q, $\chi_q = 0$, $s \in \kappa_q$, and $p \in c_q$ or $p = q$, and no effect. Its guard set is $\{r, s, p, q\}$, so an attempt by p, which names all four — Alice names her child Carol, her friend Bob and Bob's child Dave — asks the other three. The attempt runs through the transactions of the canonical platform [37] as follows.

(1) **Propose**, by p and guarded by it, provided its precondition holds, it holds no attempt prepared, and no start or question waits at it: record its own yes, hold the attempt prepared, and tell start to r, s and q.
(2) **Start**, at each of r, s and q, unguarded, on the start: vote no to p if its own precondition fails, and otherwise record the attempt as a question pending, each of the three being asked.
(3) **Answer**, at each of the three and guarded by it: record the answer yes; its else-branch, taken when the person does not will the then-branch, records no.
(4) **Vote**, at each of the three, unguarded: on the answer no, vote no to p; on yes, once it holds no other attempt prepared, vote yes if its precondition still holds and no otherwise. A yes holds the attempt prepared until the outcome.
(5) **Tally**, at p, unguarded, on each vote: record it, and on a no release the attempt, close it and tell abort. **Commit**, at p, unguarded, once a yes of each of r, s and q is recorded: release, close, tell commit, and introduce to each other every two of r, s and q not yet introduced. **Outcome**, at each of the three, unguarded, on it: apply its own effect if the outcome is commit, release and close.

The transaction is carried out, at all four at once, by the **Tally** that records the last yes at p [37]. While q holds the attempt prepared it takes part in no other attempt, so p and q cannot unfriend between q's yes and the commit, and neither parent's declaration of age can intervene. Where $p = q$, two children of one parent, the participants are three (Definition B.7).

For the part that *child_unfriend* determines at (r, s, p), whose guard set is $\{p\}$, the initiator is p and nobody is asked: at **Start** the other two participants vote by their machines, once they hold no other attempt prepared, yes if their own precondition holds — at r, $\chi_r = 1$ and $s \in c_r$; at s, $r \in c_s$ and $r \notin \kappa_s$ — and no otherwise, and **Answer** and **Vote** do not arise. p knows r, its friend, and s, whom r introduced to it.

The instantiation omits the platform's **Wound** and **Yield**, by which concurrent attempts wait in order of priority and an attempt blocked by one of lower priority has it aborted, each instantiation above running one attempt; the other twenty-six transactions pass through the same construction, the child acquaint with two children with the same four guards as the child befriending.

Child befriending in the village. The correspondence between the specification and the implementation, through child befriending in the village of Section 4.6.3.

The specification. Child befriend (Example 4.2) is guarded by both children r, s and a parent p of r and a parent q of s, with p and q friends and r and s knowing each other as preconditions, and r, s not yet friends.

The protocol. Alice's agent proposes the attempt *child_befriend*(Carol, Dave, Alice, Bob) to the agents of Carol, Dave and Bob, all of which it knows: Carol and Bob are Alice's friends, and Bob's agent introduced Dave's to it, Bob knowing both Alice and Dave; Carol and Dave know each other by the child acquaintance Alice requested before, with Bob. Each of the three checks its own part of the precondition and asks its person; on a yes it votes yes and holds the attempt prepared, taking part in no other attempt until the outcome; on a no, or a precondition that fails, it votes no. Alice's agent commits on three yeses, tells the three, and each applies its own part: Carol and Dave add each other as friends, and Bob's and Alice's states do not change. The steps are those above.

Each guard's willingness in the specification is one volitional reduction in the implementation: Alice's request, and the approvals of Bob, Carol and Dave. If any of the four does not act, the commit does not happen and the friend sets are not updated. While Bob's agent holds the attempt prepared, Bob cannot unfriend Alice, nor declare Dave of age, so the preconditions on which the three voted hold when the transaction is carried out.

The vGLP program. The canonical CVA platform of the contract, compiled into vGLP by the compiler of [37] from the contract's speech-act schemas in the term form the checker of [23] reads; the compilation is that paper's Definition "Compilation of the Canonical Platform", proved correct by its theorem of that name. Code is not reproduced here: it is at https://github.com/EShapiro2/GLP/tree/main/programs/cssn, which holds a superset of what this paper needs, beside the plays.

The volitional procedures of the program, by their interactive types, determine the interface (Section 4.6.2), so the user interface is not a further artefact: it is read off this one.

C.4 A4: The GLP Program, from A3

The canonical compilation of A3 onto GLP [37], produced by the compiler of that paper, at the pointer of A3: each goal of a volitional procedure first reduces with an asking clause, which sends an ask to the dispatcher and receives its end of its interactive variable, and a clause that reads the answer reduces only once the person has given it. A4 rather than A3 is what the engine runs, and the compilation is what that paper proves correct. The dispatcher and the construct processes belong to the vGLP implementation rather than to this platform.

C.5 A5: Dart and Flutter, from A4

The GLP engine and the one Dart bridge that draws the constructs of any compiled vGLP program and returns the person's input as grants [37, 68]. Derived from A4 by that implementation, and shared: this platform adds to the construct family the words of its acts and nothing else, and runs as a mini-app of the grassroots super-app [18] rather than as a separate app.

C.6 A6: The App, from A5

The child-safe mini-app of the grassroots super-app [18]. Derived from A5 by loading A4 into it. Figure 5 shows it on the iPhone simulator, and Figure 4 the village of Section 4.6.3 on its concurrent agents; neither is repeated here.