A type system of digital institutions

Joshua Tan
University of Oxford
Oxford, UK

Aleksandar Petrov
ETH Zurich
Zurich, Switzerland

Online social platforms are built out of many socio-technical systems—e.g. authentication systems, identity management systems, reputation systems, voting systems, content moderation systems—that communicate with each other through well-defined technical interfaces. In this work, we propose mapping the technical interfaces of these platforms onto a series of open games in order to construct a "type system" of digital institutions, and demonstrate this strategy by mapping a number of such platforms onto game-theoretic models of voting and identity management. We then show how platform designers can use this type system to efficiently partition, analyze, and prototype digital institutions. In particular, designers can use the underlying composition of open games to control the composition of platforms' technical interfaces, allowing them to prototype functional, multi-platform digital institutions on the fly. Finally, we describe work-in-progress to implement this type system within a computational toolkit for building digital institutions.

This extended abstract describes current work-in-progress.

1 Introduction

Institutions are patterns of social behavior. While game theory has had tremendous success in modeling features of economic institutions such as markets and auctions, it has proven too clunky and coarse to model the diversity and complexity of many empirically-observed institutions [18], motivating a range of ethnographic alternatives such as the action situation [17] and the institutional grammar [3].

Recently, compositional game theory has been proposed as a new, computational answer to the challenge of modeling institutional diversity and complexity [8]. In this paper, we take that argument to its logical conclusion: if compositional game theory is useful because it facilitates computational modeling of complex institutions, then it should be useful in directly prototyping and implementing complex *digital* institutions, especially since digital institutions are already implemented through computational models.

Digital institutions are simply institutions that exist online. In particular, they exist within online platforms—these range from marquee social platforms like Facebook, Reddit, or Slack to online social games such as Minecraft or World of Warcraft to blockchains like Bitcoin or Ethereum. These platforms function as both spaces within which people develop their own communities, technology stacks to support those activities, as well as public institutions in their own right. In order to engage, monetize, and govern their users, these platforms have built complex technical systems made up of many subsystems, including ones for identity management, reputation, content moderation, voting, group formation, wallet management, billing, and so on. These systems and subsystems typically communicate through well-defined technical interfaces called application programming interfaces, or APIs. While social institutions are distinct from the technical artifacts that enable them, because human activity on a platform is always channeled through and constrained by these technical interfaces [16], they are often the truest (and most accessible) computational models of a platform's digital institutions.

Submitted to:
ACT 2021

© J. Tan & A. Petrov
This work is licensed under the
Creative Commons Attribution License.

The difference between an institution and a technical platform also hints at our reason for introducing game theory into the picture. That is—why do we need to map these APIs to games in order to build with them when we could more easily model them as functions and/or map them to other APIs? The answer: APIs are already abstractions over function calls, and we are already mapping APIs to APIs—at this end of this note we will describe work-in-progress by a larger collaboration of researchers to build out the lower level of an API-based "governance layer" [19]—but to reason consistently about the behavior of many competing and overlapping services, we need something like a type system to organize their composition. And because these services are fundamentally intended to structure *human* behavior, game theory is a natural choice of abstraction. In particular, we will show in Section 5 how the strategic dependence between systems—separate from the usual dependence relations we analyze between software modules—can help us reason about and design better platforms.

Compositional game theory sits neatly within the intersection of the two requirements of composition and abstraction. By mapping these institutions' APIs to a series of open games, we can deploy and control these software services directly through a game-theoretic interface. Since these services (1) serve an institutional purpose within an commmunity and (2) inherit the well-defined composition of the category of open games, we call our framework a *type system* for digital institutions. We demonstrate this strategy through two cases: a voting type and identity management type.

The rest of this paper will proceed as follows. In Section 2, we will cover some mathematical background on compositional game theory and string diagrams. In Section 3 and Section 4, we demonstrate our proposal by constructing two examples: a Voting type for voting services and an Identity type for identity management services. Finally, we cover how to use the type system in Section 5 before concluding with a description of work-in-progress to implement the type system within a computational toolkit for building digital institutions.

2 Mathematical background

We review the basic definitions and properties of open games (Section 2.1) and an algebraic encoding of string diagrams (Section 2.2) which will be used for defining weak conservative substitution in Section 5.2.

2.1 Open games

Open games were originally developed in [10]. The definitions we use follow [6] and [9].

Definition 2.1 (Open game [9])**.** Let X, S, Y, R be sets. An open game $G\colon (X,S) \to (Y,R)$ is defined to be a 4-tuple $G = (\Sigma_G, \mathbf{P}_G, \mathbf{C}_G, \mathbf{E}_G)$, where

i. Σ_G is a set of *strategy profiles* of G;
ii. $\mathbf{P}_G\colon \Sigma_G \times X \to Y$ is the *play function* of G: it takes a strategy profile $\sigma \in \Sigma_G$ and an observation $x \in X$, and it "chooses" a play *action* $\mathbf{P}_G(\sigma, x) \in Y$;
iii. $\mathbf{C}_G\colon \Sigma_G \times X \times R \to S$ is called the *co-play function* of G: given a strategy profile $\sigma \in \Sigma_G$, an observation $x \in X$ and a received utility $r \in R$ it determines what co-utility $s \in S$ to return to the environment;
iv. $\mathbf{E}_G\colon X \times (Y \to R) \to \mathscr{P}(\Sigma_G)$ is called the *equilibrium function* of G: it gives the subset of equilibrium strategies of G, given an observation $x \in X$ and a *context* function $k\colon Y \to R$ which determines what utility the game receives from the environment for a chosen action.

We will call X the *observation set*, Y the *action set*, R the *utility set*, and S the *co-utility set*.

Open games form a category (Definition A.1), which also has monoidal products, making it a symmetric monoidal category (Lemma A.3). It also has units but does not have unambiguous co-units (Remark A.4), preventing it from being a closed Cartesian category.

Definition 2.2 (Closed game)**.** A closed game is an open game with no "dangling wires", that is, an open game of the form $(1,1) \to (1,1)$, where $1 = \{\star\}$ denotes the singleton set.

Closed games have trivial play and co-play functions ($k_{\text{triv}} : 1 \to 1$). The equilibrium function of a closed game reduces to a unary relation and hence we can talk of a *set of equilibria* $\bar{\mathbf{E}}_G \subseteq \Sigma_G$ of a closed game rather than equilibrium function. That is because the observation set contains only one element and as there is only one possible context function: k_{triv} [9].

2.2 String diagrams

In Section 5 we will need to analyze the actions and utilities that the component games of a closed game produce and observe. Hence, we will need to add *labels* to every composition operator $\fatsemi$ and monoidal product $\otimes$. Processing this algebraically can be quite challenging. By using string diagrams we can do that much more easily in a graphical form: instead of labeling operators, we will just assign labels to edges in the diagram.

As string diagrams are commonly used in category theory we defer their formal definition to Appendix A.2. Intuitively, a string diagram represents morphism variables of a monoidal signature $\mathscr{S}$ as dots and uses wires to designate how the morphisms are combined: wires connecting dots represent morphism composition (terms of type $\sigma_a \fatsemi \sigma_b$), while wires running in parallel represent monoidal product (terms of type $\sigma_a \otimes \sigma_b$). Each wire corresponds to an element of the free algebra generated by the *object variables* of $\mathscr{S}$. This is formally defined in Definitions A.5 and A.6. Every morphism term of a monoidal signature can be represented as a string diagram (Definition A.8) and every diagram in general position (Definition A.9) can be translated back into a morphism term (Definition A.16).

String diagrams in general position allow for a compact combinatorial encoding. The core idea is that a diagram in general position can be split into layers, each one having exactly one morphism variable, and potentially several identity edges. Then every layer can be then described by the number of inputs and outputs of the morphism, as well as how many identity edge pass to its right. We will use the representation proposed in [5] but will augment it with an additional map M that stores the morphism variable at each level. This is illustrated in Figure 1.

Definition 2.3 (Diagram encoding)**.** Given a general-position diagram Γ of a morphsim term generated by the monoidal signature $(\Sigma_0, \Sigma_1, \text{dom}, \text{cod})$, a *diagram encoding* (S, N, L, I, O, M) is defined as:

i. the number of *source edges* $S \in \mathbb{N}_0$, denoting the number of open edges at the top of Γ;
ii. *diagram height* $N \in \mathbb{N}$, denoting the number of morphism variables (ignoring the identity edges, hence equal to the number of layers);
iii. a function $L : [N] \to \mathbb{N}_0$, where $[N] = \{1, \dots, n\}$, which designates the *left offset* of every layer, i.e. how many identity connections pass to the right of the morphism variable at this level;
iv. a function $I : [N] \to \mathbb{N}$, which designates the number of input edges (i.e. edges entering the morphism variable from the top);
v. a function $O : [N] \to \mathbb{N}$, which designates the number of output edges (i.e. edges exiting the morphism variable from the bottom);
vi. a function $M : [N] \to \Sigma_1$, which assigns each layer its morphism variable.

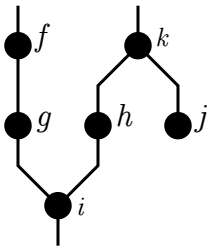
We will sometimes write the functions L, I, O, M as lists where the elements will be ordered as the function evaluated at the list index, e.g. $[L(1), \dots, L(n)]$.

$$((f \fatsemi g) \otimes (k \fatsemi (h \otimes j))) \fatsemi (i \otimes \mathrm{id}_I)$$

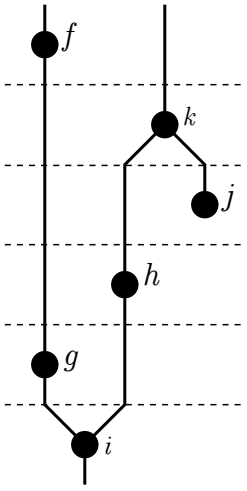
(a) The original morphism term.

$$(f \otimes \mathrm{id}_J) \fatsemi (\mathrm{id}_B \otimes k) \fatsemi (\mathrm{id}_B \otimes \mathrm{id}_D \otimes j) \fatsemi (\mathrm{id}_B \otimes h \otimes \mathrm{id}_I) \fatsemi (g \otimes \mathrm{id}_F \otimes \mathrm{id}_I) \fatsemi (i \otimes \mathrm{id}_I)$$

(b) Translation of the general position diagram (d) into a morphism term.



(c) Diagram of the original morphism term (a).



(d) Diagram in general position (levels separated by dotted lines).

$$\begin{aligned} S &= 2 \\ N &= 6 \\ L &= [\ 0,\ 1,\ 2,\ 1,\ 0,\ 0\ ] \\ I &= [\ 1,\ 1,\ 1,\ 1,\ 1,\ 2\ ] \\ O &= [\ 1,\ 2,\ 1,\ 1,\ 1,\ 1\ ] \\ M &= [\ f,\ k,\ j,\ h,\ g,\ i\ ] \end{aligned}$$

(e) Diagram encoding of (d).

$$\begin{aligned} f &: A \to B \\ g &: B \to C \\ h &: D \to F \\ i &: C \otimes F \to G \\ j &: H \to I \\ k &: J \to D \otimes H \end{aligned}$$

(f) Morphism variables.

Figure 1: Illustration of the translation of a morphism term to a string diagram (from (a) to (c)), of putting a diagram in general position (from (c) to (d)), translation of a diagram into a morphism term (from (d) to (b)), and the encoding of a diagram (from (d) to (e)). The domain and co-domain of the morphism variables are given in (f).

3 Case 1: the Voting type

Voting is the process of aggregating the preferences of several entities into a single preference. It is an extremely common institution in both online and offline communities. In online communities, voting is usually handled by specialized software in form of either a platform feature (e.g. "emoji voting" in Slack), a plugin (e.g. Discourse Polls), or an external service (e.g. SurveyMonkey, ElectionBuddy, or Snapshot). Even mildly-sophisticated voting software has far more configurable parameters than most models that appear in the study of formal voting games [7].

In this example, we map the API endpoints of four popular voting services—Loomio, Discourse Polls, Snapshot, and Pol.is—onto a "prototypical" voting open game. We do not claim that our particular model of voting is standard or canonical. The idea of this example is to (1) illustrate the diversity and complexity of empirically-observed digital institutions in voting and (2) to illustrate how that diversity and complexity can be abstracted and controlled through a common interface.

3.1 The APIs

Loomio is an popular, open-source platform for community decision-making that exposes a wide variety of API endpoints for public consumption. It is organized around a forum-like core (accessed through a

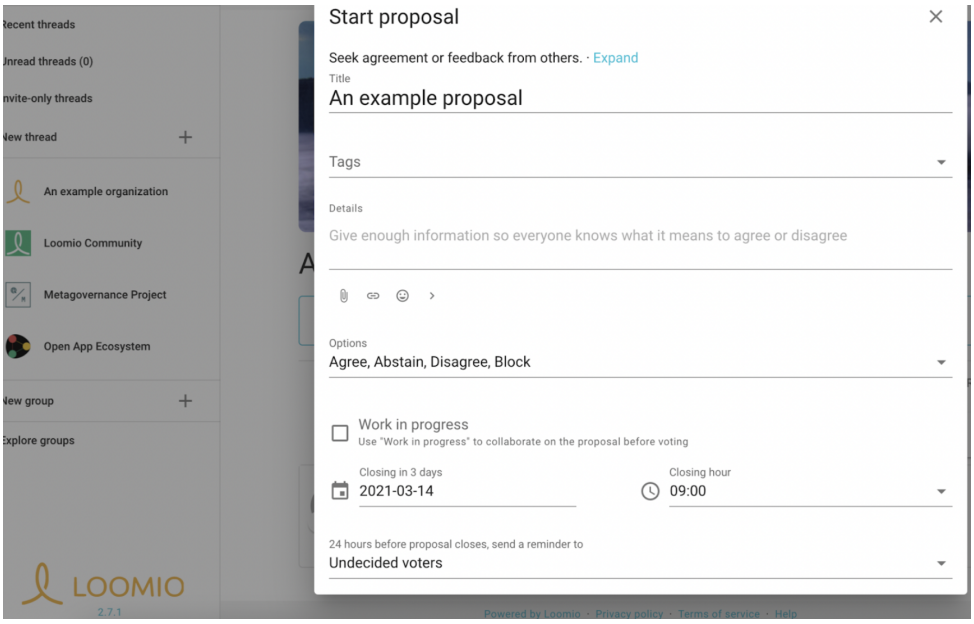


Figure 2: The Loomio voting interface.

'discussions' endpoint) but also features a powerful vote tool for organizing a wide variety of community votes.

Snapshot is a relatively new polling service geared toward blockchain projects that has drastically streamlined the vote-setup process into a minimal set of parameters: subject, start date, end date. Voting eligibility is based on token holdings (i.e. you can vote on Ethereum polls only if you own ETH).

Pol.is is a consensus-driven service that consists entirely of votes; each "post" by a user is immediately published as a "agree, disagree, or pass" vote. It is designed to foster consensus and to counteract polarization in forums with more unstructured forms of discourse.

Finally, Discourse Polls is a polling service built into the open-source forum software Discourse, similar though more basic than Loomio, which is activated via a Markdown-like script directly within a Discourse post, rather than through an API or a user interface.

```
curl -X POST -d 'title=hello world&api_key=undefined'
      http://localhost:3000/api/b1/discussions
```

Figure 3: A typical API call to Loomio, in this case using their "b1 discussions" endpoint.

3.2 The model

Definition 3.1 (Voting game)**.** A *voting game* is a tuple $V = \langle O, L_v, L_r, C, W, a \rangle$ with the following properties:

- a finite set of *options* $O = \{O_1, \ldots, O_n\}$;
- a set of *vote-labels* L_v;
- a set of *choices* $C \subseteq L_v^{|O|}$;
- a set of *result-labels* L_r;
- a set of *voter weights* W;

- an *aggregation function*

$$a : T \to L_r^{|O|},$$

where T is the set of all possible vote tables:

$$T = \bigcup_{m=1}^{\infty} (W \times C)^m.$$

The corresponding open game is the covariant lifting of a in **Game**, in other words the open game

$$G_V : (T,1) \to \left(L_r^{|O|},1\right),$$

where $1 = \{\star\}$ is the singleton set, the strategy profile is $\Sigma_{G_V} = 1$, the play function is $\mathbf{P}_{G_V}(\star,t) = a(t)$, the coplay function is $\mathbf{C}_{G_V}(\star,t,\star) = \star$, and the best response function is $\mathbf{B}_{G_V}(t,\cdot) = \langle\star,\star\rangle$.

In the above, we assume that there is an externally provided finite set of *voting options O* which is the same for all identities and known by them, i.e. everybody knows what they are voting on. We further assume that all voters use the same system and have the same set of choices that they can cast. We do not model the content of the options or their strategic significance within the voting structure. Since voting services support a range of voting protocols and aggregation functions (e.g. just Loomio supports simple majority, ranked choice, and Condorcet), the model above supports settings where the voter can cast a choice consisting consisting of a single option, several options, an assignment of real-valued weights to all options, or any arbitrary labeling of the options. Similarly, various types of outcomes must also be supported as some protocols provide a single winner (e.g. in plurality voting), some provide a distribution over winners (e.g. in proportional representation), while others have highly structured outputs (e.g. Pol.is, which provides percentages of voters who agreed, disagreed, passed, and didn't vote for all options). One important consideration is the system by which a voting system determines who can vote. We defer this question to Section 4.

Finally, Table 3.2 shows how the voting services described in Section 3.1 can be represented in this form. Note that many potential configuration options of the voting protocol itself are accounted for in the definition of the aggregation function a.

To be clear, the model above is built for demonstration purposes only. It has many limitations. For example, it does not model deliberation time and voting windows, even though those parameters can have strategic implications. It also not model the possibility of voters adding their own options in the course of the voting process, which is a frequent option in voting services—though we believe that this option has limited strategic implications and, if necessary, can be modelled as by a pre-voting option suggestion process. Finally, the model does not consider the option to reveal intermediate results or the possibility of allowing voters to change of their votes after they have been cast.

4 Case 2: the Identity type

Many mechanisms for governing voting come down to ways of limiting who is eligible to vote. This effort depends on having an identity service that can authenticate the voter and associate them to a known identity. On an online platform, many such voting processes are mediated through digital identity management providers, whether that be a private institutional database (e.g. a university's list of registered students), a popular commercial service such as Google or Facebook, or a professional service such as

Platform	Configuration	O	L_v	C	L_r	W	a
Loomio	poll_type=poll, hide_results_until_closed=true, anonymous=true, voter_can_add_options=false, options=[Plan A, Plan B, Plan C]	Plan A, Plan B, Plan C	$\mathbb{B}$	$\{(\mathtt{T},\mathtt{F},\mathtt{F}), (\mathtt{F},\mathtt{T},\mathtt{F}), (\mathtt{F},\mathtt{F},\mathtt{T})\}$	$\mathbb{B}$	??	argmax
Snapshot	poll_type=dot_vote, hide_results_until_closed=true, anonymous=true, voter_can_add_options=false, options=[Plan A, Plan B, Plan C], total_dots=10	Plan A, Plan B, Plan C	$\{0,\ldots,10\}$	$\{(0,0,0), (1,0,0), (0,1,0), \ldots, (0,10,0), (0,0,10)\}$	$\mathbb{N}_0$	??	sum
Pol.is	statements = [We need more A, Current state of B is good, Neither A nor B is necessary - C should be a priority]	{We need more A, Current state of B is good, Neither A nor B is necessary - C should be a priority}	{Agree, Disagree, Pass, Didn't vote}	{ (Agree Agree, Didn't vote), (Disagree, Pass, Agree), …}	$\mathbb{R}^4_{0,1}$	??	calc_percent

Table 1: Mapping API endpoints to components of a voting game from Definition 3.1.

Auth0. Identity management is a critical part of almost every online platform, and it is particular important when one platform is adding some sort of value-added feature (e.g. a voting service) to another platform.

From the perspective of the human end-user, an identity management service is really two functions. The first is the authentication service which, given an authentication input from the user (such as a user name and a password), matches the input against an internal database and returns the identity information to another service. There is no strategic component to this technical process, per se, though authentication details can be stolen or counterfeited. The second function is a registration service to create an account. Many game-theoretic models of identity focus on understanding attacks on this registration process, e.g. classic sybil attacks [2, 4].

4.1 The APIs

There are several different models for identity management on the internet: isolated models (service provider is identity provider), centralized models (many services, one identity provider, e.g. OAuth or Facebook Auth), user-centric models (many services), and a federated model linked by a trusted network (think services like Shibboleth) [15]. In this example, we consider three examples of identity management services: OAuth , Auth0, and BrightID.

- **OAuth** is not really an full-fledged identity management provider so much as an authentication service. However, it is extremely widespread on the Internet and can be used to perform "pseudo-authentication", where a user of one site is forwarded to another website where they login, and then the second site sends a "login key" to the first site.
- **Auth0** is a popular, standard authentication service and identity management service used by a wide variety of commercial platforms.
- **BrightID** is a social identity management service that uses your authenticated identities across multiple social and web services to establish sybil-resistant proofs of unique identity.

4.2 The model

This section is currently in progress.

5 Using the type system

The ease of creating online governance platforms results in the existence of many comparable solutions. For example, one can use Google Calendar, or Doodle, or Calendly to schedule meetings, or Slack, Discourse, or Zulip for facilitating community discussions. They all solve a similar problem but have different approaches, and each might have some features or configuration options that the other platforms do not. Hence, replacing one such platform with another might not be trivial. In this section we study the implications of such substitutions, in particular the conditions under which one can replace platforms without affecting the behavior of the users of the system. We will call such substitutions *conservative substitutions*.

Consider the case of a community with a governance system composed of several platforms. Suppose that the community wishes to change its meeting scheduler from one provider to another. We wish to study how one could perform such a substitution without affecting the strategic properties of the overall game, as well as what requirements must be met for such a strategy-preserving substitution to be possible. Our practical motivation: being able to compose "types" of digital institutions. We want to give designers the ability to composing these systems abstractly, using "types", i.e. a Voting type, an Identity type. But in order to interact with the underlying platform, we need to be able to understand the possible variations between these digital platforms and their strategic implications.

5.1 Strong conservative substitution

As we model the different platforms as open games, evaluating the implications of replacing one platform provider with another boils down to the problem of studying the change of the equilibrium strategies of a composite game when we substitute one of its component games with another. In particular, we are interested in the case where such substitution does not change the equilibrium function of the composite game. The equilibrium function of a component game is determined by the context in which it is put in, i.e. the other open games it is composed with. Therefore, we can study two different flavours of this substitution problem. In this section we consider *strong substitution*, which ensures that two games can replace one another as part of any composite game without affecting the overall equilibrium function. In the next section, we will introduce *weak substitution*, which is restricted only to one particular composite game in which the substitution does not affect the overall equilibrium function.

Definition 5.1 (Strong conservative substitution)**.** An open game $G : (X,S) \to (Y,R)$ is *strongly conservatively substitutable* (or, for short, *strongly substitutable*) by an open game $G' : (X,S) \to (Y,R)$ if, for all $x \in X$ and all $k\colon Y \to R$ it holds that:

$$S_G(x,k) = S_{G'}(x,k), \tag{1}$$

where

$$S_G(x,k) = \{\langle \mathbf{P}_G(\sigma,x), \mathbf{C}_G(\sigma,x,k(\mathbf{P}_G(\sigma,x)))\rangle \mid \sigma \in \mathbf{E}_G(x,k)\}.$$

We denote G being strongly substitutable by G' as $G \simeq G'$.

Then given an aribtrary composite game containing component game G, $G \simeq G'$ implies that if we substitute G with another game G' that has the same domain and co-domain, then the possible actions and coutilities that the other games would "see" from it would be exactly the same, regardless of what observations and contexts they provide it with. In other words, there is no way for the other games to determine whether they interact with the original game or with its substitution. We formalize this

with Theorem 5.11. Note that we do not require that the sets of strategy profiles should be the same. Furthermore, we can have $G \simeq G'$ even if $|\mathbf{E}_G| \neq |\mathbf{E}_{G'}|$.

Lemma 5.2. *The strong substitution relation $\simeq$ is an equivalence relation.*

Remark 5.3. The strong substitution relation $\simeq$ should not be confused with the equivalence relation $\sim$ defined in [6, Definition 10]. While similar they are not the same: $G \sim G'$ implies $G \simeq G'$ but the reverse does not necessairly hold.

Lemma 5.4. *Composition and monoidal product preserve the strong substitution property.*

Though tempting, we cannot replace the equality in Equation (1) by a subset or a superset relation to obtain an order on $\mathrm{Hom}_{\mathbf{Game}}((X,S),(Y,R))$. Doing so could either remove some of the original equilibria or introduce new ones, resulting in a potentially drastic change of the set of equilibrium strategies of a composite game could also. Hence, we cannot have an order on the strongly substitutable games.

5.2 Weak conservative substitution

Definition 5.1 is a very restrictive condition: we ask for two open games to produce behavior under *all* possible integrations within a composite game. However, in practice, a governance system developer would be interested only about the effect of the substitution within the context of their own governance system. For this reason, we introduce the notion of *weak conservative substitution*, which is context-dependent: the substitution of G with G' preserves the equilibrium strategies of a *particular closed game* rather than *any open game*. We then show that strong substitution implies weak substitution, as one would indeed expect (Theorem 5.11).

We extend the definition of diagram encoding (Definition 2.3) to also contain the specific values assigned to observations, actions, utilities, and co-utilities. Note that, in general, for a diagram generated by an arbitrary monoidal signature it is not obvious what the edge labels should be. For a diagram generated by a signature of **Game**, however, we know that edges correspond to objects in **Game** which are pairs of sets. Hence, we label each edge with a pair of elements from its corresponding pair of sets. An example of an open game and its representation as a labeled open game diagram can be seen in Figure 4.

Definition 5.5 (Labeled open-game diagram encoding)**.** Let $\mathscr{S} = (\Sigma_0, \Sigma_1, \mathrm{dom}, \mathrm{cod})$ be a monoidal signature with Σ_0 a collection of pairs of sets, Σ_1 being open games, and dom and cod respecting the domain and codomain of the open games. Given a diagram $\Gamma = (S,N,L,I,O,M)$ of a morphism term generated by $\mathscr{S}$, a *labeling* for Γ is defined as a function $A : [N] \to \bigcup_{m \in M} \mathrm{cod}(m)$ such that $A(i) \in \mathrm{cod}(M(i)), \forall i \in [N]$, which assigns each output edge a pair of elements from the observation and co-utility sets of its domain.

Remark 5.6. We do not provide labels for the source edges. While they can be easily added to the definition, we will predominantly work with closed games which have trivial labels for their source edges, hence we will be omitting them to keep the presentation succinct.

Definition 5.7 (Equilibrium labels of a diagram)**.** Given a diagram Γ of a closed game and its encoding (S,N,L,I,O,M), we call *equilibrium labels of* Γ the set $\mathscr{E}(\Gamma)$ of all possible equilibrium labelings for this diagram:

$$\mathscr{E}(\Gamma) = \left\{ A \in \left(\bigcup_{m \in M} \mathrm{dom}(\Gamma) \right)^{[N]} \;\middle|\; \Gamma \text{ is equilibrium-labeled with } A \right\}.$$

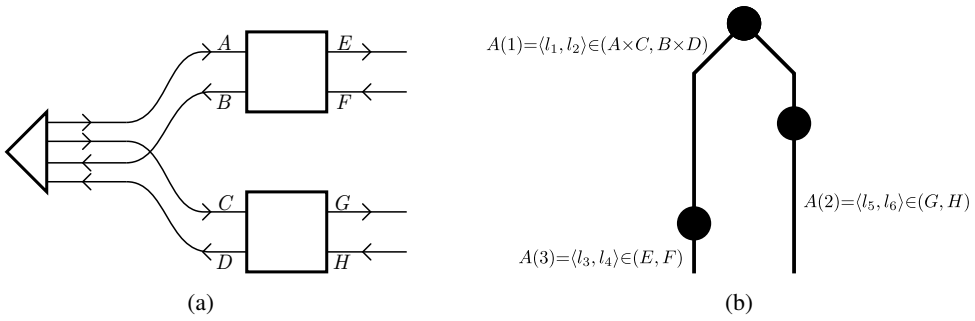


Figure 4: An open game (a) together with its representation as a labeled open game diagram encoding (b).

Definition 5.8 (Substitution of a morphism variable)**.** Let $\mathscr{S} = (\Sigma_0, \Sigma_1, \mathrm{dom}, \mathrm{cod})$ be a monoidal signature of open games. Given a diagram encoding (S,N,L,I,O,M) of a closed game Γ generated by $\mathscr{S}$, a morphism term $m \in M, m : (X,S) \to (Y,R)$ that appears at level i and another morphism term $m' \in \Sigma_1, m' : (X,S) \to (Y,R)$, possibly not in M, we can substitute m with m' in level i of Γ to obtain a new diagram $\Gamma[i \to m']$. Formally, $\Gamma[i \to m'] = (S,N,L,I,O,M')$ with:

$$M'(j) = \begin{cases} M(i) & \text{if } j \neq i, \\ m' & \text{if } j = i. \end{cases}$$

Remark 5.9. The same definition can be extended to labeled diagrams because if A is a valid label assignment for Γ, then it is a valid label assignment for $\Gamma[i \to m']$ as well. However, the equilibrium strategies of Γ and $\Gamma[i \to m']$ can be different: hence the need for the following definition.

Definition 5.10 (Weak conservative substitution)**.** Let $\mathscr{S} = (\Sigma_0, \Sigma_1, \mathrm{dom}, \mathrm{cod})$ be a monoidal signature of open games. Given a diagram encoding (S,N,L,I,O,M) of a closed game Γ generated by $\mathscr{S}$, a morphism term $m \in M, m : (X,S) \to (Y,R)$ that appears at level i, and another morphism term $m' \in \Sigma_1, m' : (X,S) \to (Y,R)$, possibly not in M, the substitution of m with m' at level i of Γ is called *weak* if $\mathscr{E}(\Gamma) = \mathscr{E}(\Gamma[i \to m'])$.

We can now show that strong conservative substitution implies weak conservative substitution.

Theorem 5.11. *Let $\mathscr{S} = (\Sigma_0, \Sigma_1, \mathrm{dom}, \mathrm{cod})$ be a monoidal signature of open games. Given a diagram encoding (S,N,L,I,O,M) of a closed game Γ generated by $\mathscr{S}$, a morphism term $m \in M, m : (X,S) \to (Y,R)$ that appears at level i and another morphism term $m' \in \Sigma_1, m' : (X,S) \to (Y,R)$, such that m is strongly substitutable by m' (Definition 5.1), the substitution of m with m' at level j of Γ is weak (Definition 5.10).*

We defer the proof of Theorem 5.11 to the Appendix.

Remark 5.12. The opposite does not generally hold. Strong substitution (Definition 5.1) is a stronger condition than weak substitution (Definition 5.10). Strong substitution requires that two open games have the exact same equilibrium actions and co-utilities for any observation and context. Weak substitution however, requires that they are the same only for the particular observation and context that the games encounter in the closed game in which they are incorporated. That is we can speak of the games G and G' being (always) strongly substitutable, but weak substitution makes sense only given a diagram encoding of a closed game Γ that has G (or G') as the morphism term on some level i. Still, weak substitution is a stronger condition than an arbitrary substitution $\Gamma[i \to m']$.

Lemma 5.13 presents even further evidence that strong substitution implies weak substitution.

Lemma 5.13. *Given a diagram Γ with encoding (S,N,L,I,O,M) and some $i \in [N]$ there is an equivalence relation over the set U of games that can be substituted at level i in Γ, i.e.*

$$U = \mathrm{Hom}_{\mathbf{Game}}(\mathrm{dom}(M(i)), \mathrm{cod}(M(i))).$$

In particular, we can define the equivalence relation $\approx_i^\Gamma \subseteq U \times U$ as:

$$(m,m') \in \approx_i^\Gamma \iff \mathscr{E}(\Gamma[i \to m]) = \mathscr{E}(\Gamma[i \to m']).$$

Furthermore, if m and m' are strongly substitutable ($m \sim m'$) then $m \approx_i^\Gamma m'$. Hence, $\approx_i^\Gamma$ is a coarser equivalence relation than $\sim$ and every equivalence class of $\sim$ is a subset of an equivalence class of $\approx_i^\Gamma$.

6 Discussion

We have described some of the basic structure and examples of a type system for digital institutions built on compositional game theory. However, there are a raft of practical obstacles to deploying this type system on top of a realistic set of APIs. For example, any practical type system for composing APIs must live on top of an existing API service integrator, since different kinds of APIs (e.g. REST, GraphQL, SOAP, etc.) do not natively speak to each other. There are also questions around the synchronicity of communication, type mismatches, and other questions about API contracts. To implement institutions "on the fly", a type system is necessary but not sufficient; we need to integrate it into a practical toolkit for building online platforms and digital institutions. That work is currently taking place as part of a large research collaboration called the Metagovernance Project.

Acknowledgements

The authors would like to thank Jules Hedges, Philipp Zahn, and Seth Frey for helpful comments in the development of this project.

References

[1] Krzysztof Bar, Aleks Kissinger & Jamie Vicary: *Globular: an online proof assistant for higher-dimensional rewriting*, p. Issue 1 ; 18605974. doi:10.23638/LMCS-14(1:8)2018. Available at `https://lmcs.episciences.org/4223`. Medium: PDF Publisher: Episciences.org.

[2] William Casey, Ansgar Kellner, Parisa Memarmoshrefi, Jose Andre Morales & Bud Mishra: *Deception, identity, and security: the game theory of sybil attacks* 62(1), pp. 85–93. doi:10.1145/3190836. Available at `https://dl.acm.org/doi/10.1145/3190836`.

[3] Sue E. S. Crawford & Elinor Ostrom: *A Grammar of Institutions* 89(3), pp. 582–600. doi:10.2307/2082975. Available at `https://www.cambridge.org/core/product/identifier/S0003055400097173/type/journal_article`.

[4] Vincent P Crawford: *Deceived while Rational? Game-theoretic Models of Deception and Gullibility*, p. 63.

[5] Antonin Delpeuch & Jamie Vicary: *Normalization for planar string diagrams and a quadratic equivalence algorithm*. Available at `http://arxiv.org/abs/1804.07832`.

[6] Neil Ghani, Jules Hedges, Viktor Winschel & Philipp Zahn: *Compositional game theory*. Available at `http://arxiv.org/abs/1603.04641`.

[7] Jac C. Heckelman & Nicholas R. Miller: *Handbook of Social Choice and Voting*. Edward Elgar Publishing. Google-Books-ID: KrckCwAAQBAJ.

[8] Jules Hedges, Seth Frey, Joshua Z. Tan & Philipp Zahn: *From games to institutions*.

[9] Jules Hedges, Evguenia Shprits, Viktor Winschel & Philipp Zahn: *Compositionality and String Diagrams for Game Theory*. Available at `http://arxiv.org/abs/1604.06061`.

[10] Julian Hedges: *Towards compositional game theory*, p. 143.

[11] Günter Hotz: *Eine Algebraisierung des Syntheseproblems von Schaltkreisen I* 1(3), pp. 185–205.

[12] Günter Hotz: *Eine Algebraisierung des Syntheseproblems von Schaltkreisen II* 1(4), pp. 209–231.

[13] André Joyal & Ross Street: *The geometry of tensor calculus, I* 88(1), pp. 55–112. doi:10.1016/0001-8708(91)90003-P. Available at `https://www.sciencedirect.com/science/article/pii/000187089190003P`.

[14] André Joyal & Ross Street: *Planar diagrams and tensor algebra*. Available at `http://maths.mq.edu.au/~street/PlanarDiags.pdf`.

[15] Hasnae L'Amrani, Badr Eddine Berroukech, Younes El Bouzekri El Idrissi & Rachida Ajhoun: *Identity management systems: Laws of identity for models [7] evaluation*. In: *2016 4th IEEE International Colloquium on Information Science and Technology (CiSt)*, IEEE, pp. 736–740, doi:10.1109/CIST.2016.7804984. Available at `http://ieeexplore.ieee.org/document/7804984/`.

[16] Lawrence Lessig: *Code: Version 2.0*, second edition. Basic Books.

[17] Elinor Ostrom: *Governing the Commons: The Evolution of Institutions for Collective Action*, 1st edition edition. Cambridge University Press.

[18] Elinor Ostrom: *Understanding institutional diversity*. Princeton paperbacks, Princeton University Press. OCLC: ocm57207709.

[19] Nathan Schneider, Primavera De Filippi, Seth Frey, Joshua Z. Tan & Amy X. Zhang: *Modular Politics: Toward a Governance Layer for Online Communities* 5, pp. 1–26. doi:10.1145/3449090. Available at `http://arxiv.org/abs/2005.13701`.

A Appendix

A.1 Open games

Definition A.1 (The **Game** category)**.** There is a category denoted **Game** with objects are pairs of sets and hom-sets $\mathrm{Hom}((X,S),(Y,R))$ being open games $(X,S)\to(Y,R)$. The identity morphism for an object (X,S) is the open game $\mathrm{id}_{(X,S)}:(X,S)\to(X,S)$ defined as $(\Sigma_{\mathrm{id}_{(X,S)}},\mathbf{P}_{\mathrm{id}_{(X,S)}},\mathbf{C}_{\mathrm{id}_{(X,S)}},\mathbf{E}_{\mathrm{id}_{(X,S)}})$ where

i. the set of strategy profiles is $\Sigma_{\mathrm{id}_{(X,S)}}=1$;
ii. the play function is $\mathbf{P}_{\mathrm{id}_{(X,S)}}(\star,x)=x$;
iii. the co-play function is $\mathbf{C}_{\mathrm{id}_{(X,S)}}(\star,x,s)=s$;
iv. the equilibrium function is $\mathbf{E}_{\mathrm{id}_{(X,S)}}(x,k)=1$,

with $1=\{\star\}$ denoting the singleton set. Morphism composition for a pair of open games $G:(X,S)\to(Y,R)$ and $H:(Y,R)\to(Z,Q)$ is the open game $G\fatsemi H:(X,S)\to(Y,R)$ defined as $(\Sigma_{G\fatsemi H},\mathbf{P}_{G\fatsemi H},\mathbf{C}_{G\fatsemi H},\mathbf{E}_{G\fatsemi H})$ where

i. the set of strategy profiles is $\Sigma_{G\fatsemi H}=\Sigma_G\times\Sigma_H$;
ii. the play function is $\mathbf{P}_{G\fatsemi H}((\sigma,\tau),x)=\mathbf{P}_H(\tau,\mathbf{P}_G(\sigma,x))$;
iii. the co-play function is $\mathbf{C}_{G\fatsemi H}((\sigma,\tau),x,q)=\mathbf{C}_G(\sigma,x,\mathbf{C}_H(\tau,\mathbf{P}_G(\sigma,x),q))$;
iv. the equilibrium function is

$$\mathbf{E}_{G\fatsemi H}(x,k)=\left\{(\sigma_G,\sigma_H)\in\Sigma_G\times\Sigma_H\mid\sigma_G\in\mathbf{E}_G(x,k')\wedge\sigma_H\in\mathbf{E}_H(\mathbf{P}_G(\sigma_G,x),k)\right\},$$

with k' is defined as

$$\begin{aligned} k' : Y &\to S \\ y &\mapsto \mathbf{C}_H(\sigma_H, y, k(\mathbf{P}_H(\sigma_H, y))). \end{aligned}$$

Remark A.2. Strictly speaking, the morphisms of **Game** are equivalence classes of open games but we will ignore this detail and will consider them to be games. This (non-)issue is discussed in [6].

Lemma A.3 (**Game** is a symmetric monoidal category)**.** *The monoidal product on objects is defined as* $(X_1, S_1) \otimes (X_2, S_2) = (X_1 \times X_2, S_1 \times S_2)$. *The monoidal unit is the object* $I = (1, 1)$. *The monoidal product on two open games (morphisms)* $G : (X_1, S_1) \to (Y_1, R_1)$ *and* $H : (X_2, S_2) \to (Y_2, R_2)$ *is the open game* $G \otimes H : (X_1 \times X_2, S_1 \times S_2) \to (Y_1 \times Y_2, R_1 \times R_2)$ *defined as*

i. the set of strategy profiles is $\Sigma_{G\otimes H} = \Sigma_G \times \Sigma_H$*;*
ii. the play function is $\mathbf{P}_{G\otimes H}((\sigma_G, \sigma_H), (x_G, x_H)) = (\mathbf{P}_G(\sigma_G, x_G), \mathbf{P}_H(\sigma_H, x_H))$*;*
iii. the co-play function is $\mathbf{C}_{G\otimes H}((\sigma_G, \sigma_H), (x_G, x_H), (r_G, r_H)) = (\mathbf{C}_G(\sigma_G, x_G, r_G), \mathbf{C}_H(\sigma_H, x_H, r_H))$*;*
iv. the equilibrium function is

$$\mathbf{E}_{G\otimes H}((x_G, x_H), k) = \{(\sigma_G, \sigma_H) \in \Sigma_G \times \Sigma_H \mid \sigma_G \in \mathbf{E}_G(x_G, k_G) \wedge \sigma_H \in \mathbf{E}_H(x_H, k_H)\},$$

with

$$\begin{aligned} k_G : Y_1 &\to R_1 \\ y_1 &\mapsto \pi_1(k(y_1, \mathbf{P}_H(\sigma_H, x_H))), \end{aligned}$$

and

$$\begin{aligned} k_H : Y_2 &\to R_2 \\ y_2 &\mapsto \pi_2(k(\mathbf{P}_G(\sigma_G, x_G), y_2)), \end{aligned}$$

where π_1 *and* π_2 *are the first and the second projections of the Cartesian product.*

Proofs of the unitality and associativity properties of **Game**, as well as the natural isomorphisms required for its monoidality can be found in [6, 10].

Remark A.4. **Game** also has co-units $(X, X) \to I$ which have the trivial strategy profile, trivial play function, and trivial equilibrium function. The co-play function of the co-units is $\mathbf{C}(\star, x, k_{\text{triv}}) = x$, with k_{triv} being the trivial function $1 \to 1$. The co-units are the tool providing "immediate feedback" to an open game: when composed with an open game, a co-unit directly feeds the game's action as its utility.

However, units do not generally exist. We cannot return a received utility as an action because the play function does not depend on the utility. In fact, if we have an open game $I \to (X, X)$ with the trivial strategy profile 1, there would be only one action $x \in X$ that this game could possibly take.

A.2 String diagrams

A string diagram is a graphical representation of a term representing morphism composition and monoidal products in a monoidal category. They were proposed in [11, 12] and [14, 13]. Due to their intuitive properties and strong correspondence to terms representing morphism, string diagrams are widely used for visual representation and theorem proving for monoidal categories. String diagrams have even been used to build the graphical proof assistant Globular [1]. As **Game** is a symmetric monoidal category, the theory of string diagrams can be applied directly.

As every open game is a morphism in **Game** and every morphism has at least one corresponding morphism term, there is a string diagram representing every open game. Developing the corresponding graphical language has indeed been a big part of the development of open games and compositional game theory [6, 9, 10]. The original intention of representing open games as string diagrams was to have an intuitive and graphical representation of the process of building up more complex open games from simpler components. However, in the development of weak conserving substitution we need to study what happens on the interfaces between the open games, so we want to expose the distinct games and the connections between them rather than collapse them into a single big open game. String diagrams and their diagram encodings are a good structure to study such effects, hence we will make an extensive use of them.

Definition A.5 (Monoidal signature)**.** Given a set Σ_0 of *object variables*, let $\mathrm{Mon}(\Sigma_0)$ be the free $(\otimes, I)$-algebra generated by Σ_0 (the set of terms built by successive application of $\otimes$ to the object variables and the monoidal unit I). A *monoidal signature* is a set Σ_0 of object variables, a set Σ_1 of *morphism variables*, and a pair of functions $\mathrm{dom}, \mathrm{cod} : \Sigma_1 \to \mathrm{Mon}(\Sigma_0)$.

Definition A.6 (Morphism terms)**.** We recursively define the *morphism terms* generated by a monoidal signature $\mathscr{S} = (\Sigma_0, \Sigma_1, \mathrm{dom}, \mathrm{cod})$ as:

i. all morphism variables in Σ_1 (called *atoms*);
ii. pairs of the type $\sigma_a \otimes \sigma_b$, where σ_a and σ_b are morphism variables;
iii. pairs of the type $\sigma_a \fatsemi \sigma_b$, where σ_a and σ_b are morphism variables such that $\mathrm{cod}(\sigma_a) = \mathrm{dom}(\sigma_b)$.

We abuse the notation by using dom and cod to be also maps from the morphism terms to their domain and codomain (as used in iii. above). The domain and codomain of morphism terms of the type σ, where σ is a morphism variable in Σ_1, are defined respectively as $\mathrm{dom}(\sigma)$ and $\mathrm{cod}(\sigma)$. The domain and codomain of morphism terms of the type $\sigma_a \otimes \sigma_b$ are recursively defined respectively as $\mathrm{dom}(\sigma_a) \otimes \mathrm{dom}(\sigma_b)$ and $\mathrm{cod}(\sigma_a) \otimes \mathrm{cod}(\sigma_b)$. The domain and codomain of morphism terms of the type $\sigma_a \fatsemi \sigma_b$ are recursively defined respectively as $\mathrm{dom}(\sigma_a)$ and $\mathrm{cod}(\sigma_b)$.

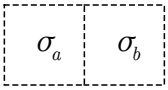
A monoidal signature is a monoidal category without composition or identity morphisms. Hence, if we add an identity morphism variable for every object variable, and we recursively add a morphism variable for every pair of morphism variables $\sigma_a, \sigma_b \in \Sigma_0$ such that $\mathrm{cod}(\sigma_a) = \mathrm{dom}(\sigma_b)$, together with the necessary isomorphims, we end up with a category: the free category generated by the signature $\mathscr{S}$. Given a category $\mathbf{C}$, we can also take a monoidal signature such that the free category generated by this signature is $\mathbf{C}$ itself. We would call this signature a *generating signature for* $\mathbf{C}$.

Remark A.7. Let $\mathscr{S} = (\Sigma_0, \Sigma_1, \mathrm{dom}, \mathrm{cod})$ be a monoidal signature with Σ_0 a collection of pairs of sets, Σ_1 being open games, and dom and cod respecting the domain and codomain of the open games. If σ is morphism term generated by $\mathscr{S}$, it corresponds to an open game. Then, in the diagram of σ, the dots will denote atomic open games and the edges will denote pairs of sets. An edge then represents connecting the action and utility ports of the dot at its top end to the observation and co-utility ports of the dot at its bottom end. An edge with no dot (identity) corresponds to the identity game defined in Definition A.1. Omitted edges correspond to trivial observations/actions because the identity object in **Game** is the singleton set 1. As **Game** is a symmetric monoidal category, it also has braiding morphisms. We can represent them in a diagram as simple wire crossings.

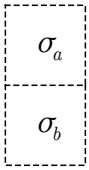
Definition A.8 (Translation of a morphism term to a string diagram [5])**.** Given a monoidal signature $\mathscr{S} = (\Sigma_0, \Sigma_1, \mathrm{dom}, \mathrm{cod})$ and a morphism term σ generated by $\mathscr{S}$ we define its corresponding string diagram recursively as:

i. if σ is an atom, then its diagram is:

ii. if T is a pair of the type $\sigma_a \otimes \sigma_b$, then its diagram is:



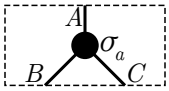
iii. if T is a pair of the type $\sigma_a \fatsemi \sigma_b$, then its diagram is:

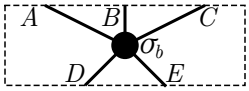


Every edge in the diagram has an associated object term from $\mathscr{S}$ which is determined by the domain and co-domain of the corresponding morphism variable. If σ has $\mathrm{dom}(\sigma) = A$ and $\mathrm{cod}(\sigma) = B$, then the we can label the edges as:

Note that, due to the requirement for endpoint-matching of the terms generated by $\fatsemi$ in Definition A.6, the assignment of object variables to the edges of the diagram is well-behaved.

If the domain or the co-domain of the morphism variable is composite (i.e. generated by applying $\otimes$ to two or more elements of Σ_0, we expose all object terms as individual edges. For example, if we have $\mathrm{dom}(\sigma_a) = A$, $\mathrm{cod}(\sigma_a) = B \otimes C$, $\mathrm{dom}(\sigma_b) = A \otimes B \otimes C$, and $\mathrm{cod}(\sigma_b) = D \otimes E$ we would have the following diagrams corresponding to σ_a and σ_b:





In order to keep the presentation clean we will usually omit the explicit edge labels.

When composing the diagrams of terms with $\fatsemi$ it might be necessary to move the connectors horizontally in order to make them connect. This can be done freely, as long as they are not changing their relative positions (e.g. in the right-hand figure above A must be to the left of B which is to the left of C and similarly for D should be on the left of E).

If σ is an atom and corresponds to an identity morphism variable (and if the concept of identity makes sense for Σ_1) we will use the simplified diagram:

Finally, if the domain or the co-domain of an atom σ is the unit element I we can omit the corresponding edge:

Definition A.9 (Diagram in a general position [5])**.** A string diagram is in general position when none of its vertices share the same height.

Every string diagram Γ can be topologically deformed in a diagram Γ' that is in general position [5]. Intuitively, by topological deformation we mean that vertices and connections can be moved without disconnecting them and crossing them over. Formal definitions can be found in [13]. If we take a morphism term, obtain its translation as a string diagram, and then topologically transform it into a diagram in a general position, the translation of the final diagram into a morphism term results in a morphism term isomorphic to the original one. Hence, putting a string diagram in general position does not the properties of the open game it represents.

Definition A.10 (Total input edges for a level)**.** We define the *total input edges* $\mathbf{I}(i)$ for level i recursively as:

$$\begin{aligned}\mathbf{I}\colon [N] &\to \mathbb{N}_0,\\ 1 &\mapsto S,\\ i &\mapsto \mathbf{I}(i-1) - I(i-1) + O(i-1), \text{ for } i \in 2,\dots,N.\end{aligned}$$

Definition A.11 (Total output edges for a level)**.** We define the *total output edges* $\mathbf{O}(i)$ for level i recursively as:

$$\begin{aligned}\mathbf{O}\colon [N] &\to \mathbb{N}_0,\\ 1 &\mapsto S - I(1) + O(1),\\ i &\mapsto \mathbf{O}(i-1) - I(i) + O(i), \text{ for } i \in 2,\dots,N.\end{aligned}$$

Remark A.12. Note that $\mathbf{I}(i+1) = \mathbf{O}(i)$ for all $1 \leq i < N$ as one would expect.

Definition A.13 (Right offset of a layer)**.** Similarly to how we have $L(i)$ designating the number of identity connections passing on the left of the morphism variable at level i we can define the number of identity connections passing on its right as $R(i) = \mathbf{I}(i) - L(i) - I(i) = \mathbf{O}(i) - L(i) - O(i)$.

Definition A.14 (Domain of an identity edge)**.** Let (S,N,L,I,O,M) be the encoding of a diagram Γ. Then the function $\tau\colon [N] \times \mathbb{N}_0 \to \Sigma_0$ assigns every identity connection its domain (which is the same as its codomain). For level i and identity connection $1 \leq j \leq L(i) + R(i)$ counted left-to-right, $\tau(i,j)$ is the domain and codomain of this connection.

Definition A.15 (Source of an identity edge)**.** Let (S,N,L,I,O,M) be the encoding of a diagram Γ. We denote by $\mu(i,j)$, with $0 \leq j \leq L(i) + R(i)$ the level of the morphism variable whose output edge is the the j-th identity edge at level i, counting left-to-right. If there is no such morphism variable, i.e. the edge is an open edge that leaves the diagram at the top, then $\mu(i,j)$ is undefined. We denote by $\eta(i,j)$, with $0 \leq j \leq L(i) + R(i)$ the position of the output edge of $\mu(i,j)$ that is the j-th identity edge of the morphism variable at level i. If $\mu(i,j)$ is undefined, then $\eta(i,j)$ is also undefined.

Definition 2.3 showed how every morphism term can be represented as a string diagram. We can also do the reverse translation: from a string diagram into a morphism term. Note, however, that these two processes do not form a bijection: there are multiple possible translations of the same diagram as a morphism term. For example, for every $n \in \mathbb{N}$ every identity edge can be represented as n successive compositions of the corresponding identity morphism. Nevertheless, all the possible translations of a diagram as morphism terms would result in isomorphic open games.

Definition A.16 (Translation of a general position diagram into a morphism term)**.** Given a diagram in general position Γ and its diagram encoding (S,N,L,I,O,M) we construct a morphism term as follows:

i. For every level i we build the morphism term T_i as

$$T_i = \mathrm{id}_{\tau(i,1)} \otimes \ldots \otimes \mathrm{id}_{\tau(i,L(i))} \otimes M(i) \otimes \mathrm{id}_{\tau(i,L(i)+1)} \otimes \ldots \otimes \mathrm{id}_{\tau(i,L(i)+R(i))},$$

with the identities being the respective identity terms on the pairs of sets corrsponding to the edges passing on the left and on the right, as in Definition A.14.

ii. The resulting morphsim term for Γ is:

$$T_1 \fatsemi \ldots \fatsemi T_N.$$

Remark A.17. Definition A.16 defines the translation process from diagrams to morphism terms only for diagrams in general position. For other diagrams, one would first need to put them in general position and then apply the conversion process. As every diagram can be put into general position, every diagram can be translated into a morphism term using the procedure in Definition A.16.

Remark A.18. Let $\mathscr{S} = (\Sigma_0, \Sigma_1, \mathrm{dom}, \mathrm{cod})$ be a monoidal signature with Σ_0 a collection of pairs of sets, Σ_1 being open games, and dom and cod respecting the domain and codomain of the open games. Given a diagram Γ of a morphism term generated by $\mathscr{S}$, we will allow ourselves to abuse the notation and refer to the open game generated by the procedure in Definition A.16 also as Γ. This abuse of notation is well-behaved as there all open games generated when Definition A.16 is applied to Γ are isomorphic to the original game.

A.3 Proof of Theorem 5.11

Proof of Theorem 5.11. We will only show that if $A \in \mathscr{E}(\Gamma)$ then $A \in \mathscr{E}(\Gamma)[j \to m']$. The opposite follows directly as $\Gamma[j \to m'][j \to m] = \Gamma$.

From Definition 5.1 it holds that for $f(j)$ and for some x_j and some k_j mapping y to $\mathbf{C}_{\Gamma>j}(S_{j+1} \times \ldots \times S_N, y, k_{\mathrm{triv}})$ there is a $\sigma_{m'} \in \Sigma_{m'}$ such that

$$\mathbf{P}_m(f(j), x_j) = \mathbf{P}_{m'}(\sigma_{m'}, x_j) \tag{2}$$

$$\mathbf{C}_m(f(j), x_j, k_j) = \mathbf{C}_{m'}(\sigma_{m'}, x_j, k_j). \tag{3}$$

Hence, we can define a new assignment function

$$f' \colon [N] \to \left(\bigcup_{j \in [N] \setminus \{i\}} \sum_{m(j)} \right) \cup \Sigma_{m'}$$

$$i \mapsto \begin{cases} f(i) & \text{if } j \neq i, \\ \sigma_{m'} & \text{if } j = i. \end{cases}$$

The goal is to show that whether we have Γ with f assigning the strategy profiles or $\Gamma[i \to m']$ with f', all contexts and observations are the same, hence all labels must be the same too.

All contexts $k_j, \ldots, k_N$ for levels $j, \ldots, N$ are not affected by the substitution. If we show that the context for level $j-1$ is also unaffected, all context must also be the same after the substitution. The context k_{j-1} for level $j-1$ is the map

$$y \mapsto \mathbf{C}_{\Gamma \geq j}(S_j \times \ldots \times S_N, y, k_{\mathrm{triv}}).$$

To simplify the proof, we will abuse the notation to not include the identity games and will consider only the coplay function of the non-identity game at level j. In the case with f we have that co-play function to be:

$$k_{j-1}(y) = \mathbf{C}_m\left(f(j), y, \mathbf{C}_{\Gamma>j}\left(S_{j+1} \times \ldots \times S_N, \mathbf{P}_m(f(j), y), k_{\text{triv}}\right)\right),$$

and in the case with f':

$$k'_{j-1}(y) = \mathbf{C}_{m'}\left(\sigma_{m'}, y, \mathbf{C}_{\Gamma>j}\left(S_{j+1} \times \ldots \times S_N, \mathbf{P}_{m'}(\sigma_{m'}, y), k_{\text{triv}}\right)\right).$$

From Equations (2) and (3) it follows that $k_{j-1} = k'_{j-1}$ and, as all contexts above level $j-1$ depend only on $k_{j-1}, k_j, \ldots, k_N$, all contexts are unaffected by changing the morphism variable at level j from m to m' and its assigned strategy profile from $f(j)$ to $f'(j) = \sigma_{m'}$.

If the context for the topmost level k_1 is the same it will still have the same assigned strategy profile and produce the same action as it still has the trivial observation $\star$. The same recursively holds for all levels down to j. However, as from Equation (2) we know that under the same context and observation the strategy profile $f(j)$ of m produces the same action as the strategy profile $\sigma_{m'}$ of m', the same must hold for all levels. □