Game semantics in the nominal model

Murdoch Gabbay

School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh, United Kingdom
www.gabbay.org.uk

Dan Ghica

School of Computer Science
University of Birmingham
Birmingham, United Kingdom

Abstract

We present a model of games based on *nominal sequences*, which generalise sequences with atoms and a new notion of *coabstraction*. This gives a new, precise, and compositional mathematical treatment of *justification pointers* in game semantics.

Keywords: Game semantics, nominal sets, nominal abstraction and coabstraction, equivariance

1 Introduction

Game semantics is a successful collection of techniques for giving denotations to logic and computation. It came to particular prominence by solving the open problem of full abstraction for PCF [2,19] and is widely used from philosophy and logic, to model checking and synthesis of digital circuits [22,13].

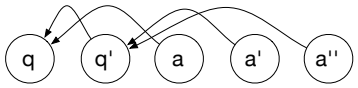
The game metaphor is a dialogue between *Proponent* and *Opponent*: a play of a game records interactions between a term (the Proponent) and its context (the Opponent), and how they are scheduled.

One way to model a play is as a labelled acyclic graph called a *pointer sequence*. Each node in the graph is a Proponent or Opponent move and edges in the graph represent the *justification* for that move. Thus, a pointer sequence records what moves were made and in what order, and also why.

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

We propose a model of games based on nominal sets, inspired by pointer sequences, with the difference that we model edges using *atoms* from nominal techniques (which we may also call *names*). Why this is useful will become clear in a moment.

Atoms are just a countably infinite set of distinct symbols: $a, b, c, \ldots$. A diagram shows how these can model pointer sequences. The pointer sequence on the left corresponds to the *nominal* sequence on the right:



corresponds to $\mathsf{q}c[a]\ \mathsf{q}'a[b]\ \mathsf{a}a\ \mathsf{a}'b\ \mathsf{a}''b$

Questions and answers are written q and a, and atoms are used as pointers. The symbols $[a]$ and $[b]$ can be thought of as naming the questions q and q' and are binders into the 'future'. So we see above that q justifies two moves: q' and a. Pointers (arrows, in the diagram above) are rendered as a pair of atoms. The tip of the arrow is represented by a *coabstraction* $[b]$ which must be unique (this is formalised by the condition $b \notin atoms(e)$ in Definition 2.8). The tail of the arrow, which need not be unique, is an occurrence of the name. This deals straightforwardly with *dangling pointers*, which are viewed just as *free names*; in the sequence above c is free.

Nominal sequences have the following good properties:

(i) A sub-sequence of a nominal sequence is a nominal sequence. A sub-graph of a pointer sequence is not a pointer sequence, because it might have 'dangling pointers'. In that sense, nominal sequences generalise pointer sequences and help talk easily about 'open sequences' (easy handling of open elements is a typical benefit of nominal techniques).

(ii) A concatenation of two nominal sequences is a nominal sequence; names link up and there are no reindexing isomorphisms. It is not so clear how to concatenate pointer sequences.

(iii) Nominal sequences are an inductive data-type and can be manipulated with standard tools (to fully benefit would require a mechanised nominal system [26] but we shall see our sequences simplify paper-and-pencil proofs too).

There is an important, specifically *nominal* advantage to using names in particular: it enables a particularly efficient management of *renaming* pointers to avoid 'accidental clash'. It is important and useful that we use *names* to name moves and not e.g. numbers, because names are by definition symmetric (i.e. can be permuted); not only can we use permutations to α-convert, but taking names and their permutative symmetry as *primitive* saves effort since permutations propagate *necessarily* to the things we build using them, such as plays and strategies. [1] This style of name

[1] A general statement of this is the *principle of equivariance* (see [7, Subsection 4.2], [12, Lemma 4.7]).
The principle of equivariance implies that, provided we permute names uniformly in all the parameters of our definitions and theorems, we then get another valid set of definitions and theorems. This is not true of numbers because our mathematical foundation equips numbers by construction with

management is characteristic of nominal techniques and we shall see that it is effective here.

We formalise game models for PCF [19,23] and Concurrent Algol [14], at low overhead. The decoration of sequences by atoms is no more of an overhead than decoration by pointers, and equivariance is a very efficient way to manage renaming, so the overhead is low and the advantages in precision and conciseness appear to be significant.

We cannot replicate all definitions and proofs from these two large papers in this conference paper but we hope that it will be entirely obvious to the reader how this could be done. We do not claim to make the work above trivial. However, we do claim that using our formulation, game semantics can be carried out more quickly, more accurately, and more transparently.

This is important for more than good practice: we speculate that by our formulation, *implementation* and *mechanisation* of game semantics proofs are significantly easier. The reader can compare the definitions in this paper with the original versions [19,14] and judge which would be easier to work with, in a prover like Isabelle. Furthermore, game semantics can provide theoretical foundations for program verification and for hardware synthesis, where the pen-and-paper style of much previous work must be augmented by machine-checked proofs, because of scale, or for safety, or both. Here, the compositionality, computational, and symmetry properties enumerated and discussed above really count. Finally, game semantics can reconcile the compositionality of denotational semantics with the effectiveness of operational semantics via communicating abstract machines [16]; here, the conventional representation of pointers is arguably actively counterintuitive, whereas the use of names as tags for messages carries immediate computational intuitions.

2 Nominal game semantics

2.1 Nominal sequences

Definition 2.1 Fix disjoint countably infinite set of **atoms** $\mathbb{A}$, and **constants**. a, b, c will range over distinct atoms (the **permutative convention**). $\mathsf{f}, \mathsf{g}, \mathsf{h}$ will range over constants, not necessarily distinct.

---

numerical properties such as *less than or equal to* $\leq$, which can be defined from first principles with no parameters.

So if we use numbers to model pointer sequences then we do not care about $\leq$ because we just needed a countable set of elements, but we repeatedly have to *prove* that we did not use an asymmetric property like $\leq$. In contrast, if we assume nominal foundations and use atoms, then we do not have to explicitly prove symmetry because we can just look at our mathematical foundation and note that it is naturally symmetric under permuting names; we reserve numbers for naturally *a*symmetric activities, such as counting.

Define **(nominal) sequences** by

$$e ::= \varepsilon \mid e\,a \mid e\,\mathsf{f} \mid e\,[a].$$

Remark 2.2 Call ε the **empty list** and write $e\,e'$ for list concatenation. Call $[a]$ a **coabstraction**. The reminds us of the *atoms-abstraction* of nominal techniques [12]—but in $e[a]$, a is bound 'in the future' in whatever e' we might concatenate after $e[a]$. We contrast the intended denotations of abstraction and coabstraction in the Conclusion.

Definition 2.3 Define **coabstracted** and **free** atoms $ca(e)$ and $fa(e)$ by:

$$\begin{array}{rlrl} ca(\varepsilon) &= \varnothing & fa(\varepsilon) &= \varnothing \\ ca(ea) &= ca(e) & fa(ea) &= fa(e) \cup (\{a\} \setminus ca(e)) \\ ca(e\mathsf{f}) &= ca(e) & fa(e\mathsf{f}) &= fa(e) \\ ca(e[a]) &= ca(e) \cup \{a\} & fa(e[a]) &= fa(e) \end{array}$$

Define the **atoms** in an expression $atoms(e)$ by $atoms(e) = fa(e) \cup ca(e)$.

Lemma 2.4 *$ca(e\,e') = ca(e) \cup ca(e')$ and $fa(e\,e') = fa(e) \cup (fa(e') \setminus ca(e))$.*

Definition 2.5 A **renaming** ρ is a function from atoms to atoms such that $dom(\rho) = \{a \mid \rho(a) \neq a\}$ is finite. Write id for the **identity** renaming such that $id(a) = a$ and $\rho' \circ \rho$ for **composition** such that $(\rho' \circ \rho)(a) = \rho'(\rho(a))$.

Call bijective ρ **permutations**. Following [12] let π range over permutations (the application of *renaming* in a nominal context goes back to [11]).

Definition 2.6 Define a **renaming action** $\rho{\cdot}e$ on sequences by:

$$\begin{array}{rl} \rho{\cdot}\varepsilon &= \varepsilon \\ \rho{\cdot}(ea) &= (\rho{\cdot}e)\rho(a) \\ \rho{\cdot}(e\mathsf{f}) &= (\rho{\cdot}e)\mathsf{f} \\ \rho{\cdot}(e[a]) &= (\rho{\cdot}e)[\rho(a)] \end{array}$$

2.2 Nominal game semantics

A game is an *arena* (Definition 2.7) along with some set of *legal plays* which are lists of moves by a *proponent* or *opponent*—precisely what classes of plays are legal, determines the type of game they play.

Definition 2.7 An **arena** is a tuple $\mathfrak{A} = (qst_{\mathfrak{A}}, ans_{\mathfrak{A}}, \lambda_{\mathfrak{A}}, \vdash_{\mathfrak{A}}, ini_{\mathfrak{A}})$ of:

- Disjoint sets of **questions** $\mathsf{q} \in qst_{\mathfrak{A}}$ and **answers** $\mathsf{a} \in ans_{\mathfrak{A}}$.
  Write $\mathsf{m} \in mvs_{\mathfrak{A}} = qst_{\mathfrak{A}} \uplus ans_{\mathfrak{A}}$ for short and call this the set of **moves**.
- A **polarity function** $\lambda_{\mathfrak{A}} : mvs_{\mathfrak{A}} \to \{\mathsf{O}, \mathsf{P}\}$. Write $\mathsf{O}^* = \mathsf{P}$ and $\mathsf{P}^* = \mathsf{O}$.
- An **enabling relation** $\vdash_{\mathfrak{A}} \subseteq mvs_{\mathfrak{A}} \times mvs_{\mathfrak{A}}$ where $\mathsf{m} \vdash_{\mathfrak{A}} \mathsf{m}'$ implies $\mathsf{m} \in qst_{\mathfrak{A}}$ and $\lambda_{\mathfrak{A}}(\mathsf{m}) = \lambda_{\mathfrak{A}}(\mathsf{m}')^*$.

- A set of **initial questions** $ini_{\mathfrak{A}} \subseteq qst_{\mathfrak{A}}$ such that:
  · $\lambda_{\mathfrak{A}}(\mathsf{i}) = \mathsf{O}$ for every initial $\mathsf{i} \in ini_{\mathfrak{A}}$.
  · If $\mathsf{q} \in qst_{\mathfrak{A}}$ and $\mathsf{i} \in ini_{\mathfrak{A}}$ then $\mathsf{q} \not\vdash_{\mathfrak{A}} \mathsf{i}$.

Definition 2.8 Define **proto-plays** e over an arena $\mathfrak{A}$ inductively by:

$$e ::= \varepsilon \mid e\,\mathsf{m}a[b] \qquad (b \notin atoms(e))$$

Recall that $\mathsf{m} \in qst_{\mathfrak{A}} \cup ans_{\mathfrak{A}}$. Write $pply_{\mathfrak{A}}$ for the set of all proto-plays of $\mathfrak{A}$.

Every proto-play is a sequence; not every sequence is a proto-play. It will always be clear what e ranges over.

Definition 2.9 Call $\mathsf{m}a[b]$ a **(named) move**; m will range over named moves.

Remark 2.10 • A proto-play consists of a sequence of named moves, each of which consists of a move m, a **justifying name** a and a coabstraction $[b]$ which we call the **name** of m. This b 'names' its move, so that a later named move's justifying name can point to m by its name b.
- The **freshness condition** $b \notin atoms(e)$ makes b name its move uniquely in the sequence. This is inefficient—we cannot reuse names even if somehow we know we could—but we optimise for mathematical convenience.
- In the games models of [19,23,14] only questions justify, so for the applications in this paper $\vdash_{\mathfrak{A}} \subseteq qst_{\mathfrak{A}} \times mvs_{\mathfrak{A}}$ and we can drop the coabstractions naming answers in protoplays (so: $\mathsf{q}a[b]$ but just $\mathsf{a}a$). However, this complicates definitions and loses generality, so we leave in (dummy) coabstractions and take $\vdash_{\mathfrak{A}} \subseteq mvs_{\mathfrak{A}} \times mvs_{\mathfrak{A}}$. Answers justifying moves are used to construct 'coproduct arenas' in game semantics for call-by-value languages [3].

Definition 2.11 Suppose e and e' are sequences. Write $e' \leq e$ when $e'\,e'' = e$ for some e''; call e' a **prefix** of e. Write $e' \subseteq e$ when $e'''\,e'\,e'' = e$ for some e''' and e''; call e' a **segment** of e.

Definition 2.12 Define $enabled(e)$ the moves **enabled** by $e \in pply_A$ by:

$$enabled(\varepsilon) = \varnothing$$
$$enabled(e\,\mathsf{m}a[b]) = enabled(e) \cup \{\mathsf{m}'b \mid \mathsf{m} \vdash_{\mathfrak{A}} \mathsf{m}'\}$$

Lemma 2.13 $enabled(e) = \bigcup\{\mathsf{m}b \mid \mathsf{m}'a[b] \subseteq e,\ \mathsf{m}' \vdash_{\mathfrak{A}} \mathsf{m}\}$.

Definition 2.14 Given $e \in pply_{\mathfrak{A}}$ define its **underlying sequence** $|e|$ by:

$$|\varepsilon| = \varepsilon$$
$$|e\mathsf{m}a[b]| = |e|\mathsf{m}$$

Intuitions for Definition 2.15 are discussed in Remark 2.16:

Definition 2.15 Suppose $\mathfrak{A}$ is an arena and $A \subseteq \mathbb{A}$.

(i) Call $e \in pply_{\mathfrak{A}}$ **justified** when $e'\,\mathsf{m}a \leq e$ and $\mathsf{m} \notin ini_{\mathfrak{A}}$ implies $\mathsf{m}a \in enabled(e')$.
(ii) Call $e \in pply_{\mathfrak{A}}$ **well-opened** when $e'\,\mathsf{i}a[b] \leq e$ implies $e' = \varepsilon$.
(iii) Call $e \in pply_{\mathfrak{A}}$ **strictly scoped** when $\mathsf{a}a[b]e' \subseteq e$ implies $a \notin fa(e')$, for every $e' \in pply_{\mathfrak{A}}$, $\mathsf{a} \in ans_{\mathfrak{A}}$, and atom a.
(iv) Call $e \in pply_{\mathfrak{A}}$ **strictly nested** when $\mathsf{q}a[b]\, e_2\, \mathsf{q}'b[c]\, e_3\, \mathsf{a}b \subseteq e$ implies $\mathsf{a}'c \subseteq e_3$ for some answering move $\mathsf{a}' \in ans_{\mathfrak{A}}$. [2]
(v) Call $e \in pply_{\mathfrak{A}}$ **alternating** when $\mathsf{m}\mathsf{m}' \subseteq |e|$ implies $\lambda_{\mathfrak{A}}(\mathsf{m}) \neq \lambda_{\mathfrak{A}}(\mathsf{m}')$.

Remark 2.16 Intuitively, Definition 2.15 means:

- e *is justified* when every non-initial move responds to a preceding move.
- e *is well-opened* when the initial move is unique and first in the sequence.
- e *is strictly scoped* [3] when a question can receive *at most* one answer. If we read games as processes, this means answering a question *stops* the process associated with that question.
- e *is strictly nested* when questions are answered in (reverse) order. This forbids starting a process b, then c from inside b, then stopping b before c.

The intuition of e *alternating* seems clear but it does not have directly to do with names and binding, so we will not consider it further.

Definitions 2.17 and 2.18 follow [23, Sec. 2.1]:

Definition 2.17 Given justified $e \in pply_{\mathfrak{A}}$ define the **proponent view** $\ulcorner e \urcorner$ and **opponent view** $\llcorner e \lrcorner$ by: [4]

$$
\begin{aligned}
\ulcorner \varepsilon \urcorner &= \varepsilon \\
\ulcorner e\mathsf{m}a[b] \urcorner &= \ulcorner e \urcorner \mathsf{m}a[b] && (\lambda_{\mathfrak{A}}(\mathsf{m}){=}\mathsf{P}) \\
\ulcorner e\mathsf{i}a[b] \urcorner &= \mathsf{i}a[b] \\
\ulcorner e\mathsf{q}a[b]e'\mathsf{m}b[c] \urcorner &= \ulcorner e \urcorner \mathsf{q}a[b]\mathsf{m}b[c] && (\lambda_{\mathfrak{A}}(\mathsf{m}){=}\mathsf{O}) \\
\\
\llcorner \varepsilon \lrcorner &= \varepsilon \\
\llcorner e\mathsf{m}a[b] \lrcorner &= \llcorner e \lrcorner \mathsf{m}a[b] && (\lambda_{\mathfrak{A}}(\mathsf{m}){=}\mathsf{O}) \\
\llcorner e\mathsf{q}a[b]e'\mathsf{m}b[c] \lrcorner &= \llcorner e \lrcorner \mathsf{q}a[b]\mathsf{m}b[c] && (\lambda_{\mathfrak{A}}(\mathsf{m}){=}\mathsf{P})
\end{aligned}
$$

Definition 2.18 We say that a justified proto-play $e \in pply_{\mathfrak{A}}$ satisfies **visibility** when $e'\mathsf{q}a[b]e''\mathsf{q}'b[c] \leq e$ implies that:

- if $\lambda_{\mathfrak{A}}(\mathsf{q}) = \mathsf{P}$ then $\mathsf{q}a[b] \subseteq \ulcorner e'\mathsf{q}a[b]e'' \urcorner$, and
- if $\lambda_{\mathfrak{A}}(\mathsf{q}) = \mathsf{O}$ then $\mathsf{q}a[b] \subseteq \llcorner e'\mathsf{q}a[b]e'' \lrcorner$.

Compare this to the more informal definition of visibility in [23, Sec. 2.1]:

[2] What is important here is the atom c.
[3] In [14, p. 7] 'strictly scoped' is called *fork* and 'strictly nested' is called *join*.
[4] The function $e \mapsto \llcorner e \lrcorner$ is not a total function because it is not defined on e of the form $\mathsf{i}a[b]e'$ where $\lambda_{\mathfrak{A}}(\mathsf{i}) = \mathsf{P}$. However, $\llcorner e \lrcorner$ is defined on all *justified* e, because $\mathsf{i}a[b]e'$ where $\lambda_{\mathfrak{A}}(\mathsf{i}) = \mathsf{P}$ is not justified.

A well formed sequence s is legal, or is a legal position, if it also satisfies the following visibility condition:

- *if $tm \sqsubseteq s$ where m is a P-move, then the justifier of m occurs in $\ulcorner t \urcorner$.*
- *if $tm \sqsubseteq s$ where m is a noninitial O-move, then the justifier of m occurs in $\ulcorner t \urcorner$.*

The difficulty here is that the taking of the view removes moves from a play, and so requires a complex reindexing if pointers are formalised using integers. Finding the justifier of a move in a view is not straightforward.

Visibility is subtle, typical of languages that are pure or have only ground-type state. We have shown above how to formalise it in our framework, but proofs of properties involving visibility are non-trivial for reasons other than the handling of names and binding, so we will not consider this property further.

Remark 2.19 We can now characterise the plays of **HO-games** (the games from [19]) and **GM-games** (those from [14]). Suppose $\mathfrak{A}$ is an arena and $e \in pply_{\mathfrak{A}}$ is a proto-play. Then:

- In HO-games, e is a **legal play** when $fa(e) = \{a\}$ for some $a \in \mathbb{A}$ and e is justified, well opened, alternating, strictly nested and satisfies visibility (see [19, Def. 4.2, Def. 4.4]).
- In GM-games, e is a **legal play** when $fa(e) = \{a\}$ as above and e is justified, well-opened, strictly scoped, and strictly nested (see [14, Def. 1]).

The condition $fa(e) = \{a\}$ implies e has one free atom a; one 'dangling pointer'. With being well-opened, this ensures a names the initial question.

How do we choose a above? We do not. It is a non-evident design decision that *proto-plays do not have α-conversion on coabstracted atoms.* This preserves compositionality: if $[a]a$ equals $[b]b$ then $[a]ab$ equals $[b]bb$, which is nonsense. [5] In our framework α-conversion lives in strategies (sets of proto-plays), which are subject to an equivariance (symmetry) condition up to the choice of atoms in the proto-plays they contain. So α-equivalence does not live in the elements, it lives in the sets of elements. More on this in Remark 5.3.

3 Operations on plays

3.1 Deletion of moves from a play

We often want to delete moves from pointer sequences, reflecting 'hiding' of irrelevant parts of a computation (see e.g. Definition 5.2). But pointers into and out of deleted moves need to be updated. Definition 3.1 and Proposition 3.4 make this formal for our nominal framework. The culminating result of this subsection is

[5] It is possible to reconcile α-conversion with proto-plays, by appending a 'future permutation', like so: $[a]_{\pi}$. Then $[a]_{id}a$ equals $[b]_{(b\ a)}b$, not $[b]_{id}b$. This is not needed here.

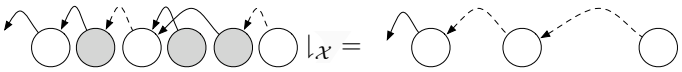
Theorem 3.11, which uses Proposition 3.4 amongst other constructions to show that properties of proto-plays are preserved by deletion.

Definition 3.1 Suppose $\mathcal{X} \subseteq mvs_{\mathfrak{A}}$ is some set of moves from an arena $\mathfrak{A}$, and $e \in pply_{\mathfrak{A}}$. Define **deletion** $e\lfloor\mathcal{X}$ inductively on e as follows, where we take inductively $(f, \rho) = e\lfloor\mathcal{X}$:

$$\begin{array}{lll} \varepsilon\lfloor\mathcal{X} = (\varepsilon, id) & & \\ (e\,\mathsf{m}a[b])\lfloor\mathcal{X} = (f\,\mathsf{m}\rho(a)[b], \rho) & & (\mathsf{m} \notin \mathcal{X}) \\ (e\,\mathsf{m}a[b])\lfloor\mathcal{X} = (f, \rho[b{:=}\rho(a)]) & & (\mathsf{m} \in \mathcal{X}) \end{array}$$

It will be convenient to write $e\lfloor\mathfrak{A}$ for $\pi_1(e\lfloor mvs_{\mathfrak{A}})$, that is, for deletion of the set of moves of $\mathfrak{A}$. Here π_1 is first projection.

Remark 3.2 Intuitively $e\lfloor\mathcal{X}$ is 'e with the moves in $\mathcal{X}$ deleted'. Some reindexing has to take place when we do this: e.g. if $\mathsf{q}a[b]$ is deleted then any pointers to b are 'reattached' so that they point to whatever a points to:



In the diagram above the shaded nodes (circles) are in $\mathcal{X}$ and are deleted.

$e\lfloor\mathcal{X}$ inductively generates a 'result' f and a 'reindexing renaming' ρ. It is tempting to dismiss ρ as a by-product, but ρ may be the more important information since f can be calculated from ρ and $\mathcal{X}$. This is Proposition 3.4, which is key to a nice proof of Theorem 3.11.

Definition 3.3 Suppose $e{\in}pply_{\mathfrak{A}}$ and $\mathcal{X}{\subseteq}mvs_{\mathfrak{A}}$. We define **naive deletion** e-$\mathcal{X}$ as follows:

$$\begin{array}{lll} \varepsilon\text{-}\mathcal{X} = \varepsilon & & \\ (e\,\mathsf{m}a[b])\text{-}\mathcal{X} = (e\text{-}\mathcal{X})\,\mathsf{m}a[b] & & (\mathsf{m} \notin \mathcal{X}) \\ (e\,\mathsf{m}a[b])\text{-}\mathcal{X} = e\text{-}\mathcal{X} & & (\mathsf{m} \in \mathcal{X}) \end{array}$$

Proposition 3.4 *Suppose* $e \in pply_{\mathfrak{A}}$ *and* $e\lfloor\mathcal{X} = (e', \rho)$. *Then* $e' = (\rho{\cdot}e)\text{-}\mathcal{X}$. [6]

Proof By induction on e. We consider the interesting case (it changes ρ):

- *The case* $e\,\mathsf{m}a[b]$ *where* $\mathsf{m} \in \mathcal{X}$. Suppose $e\lfloor\mathcal{X} = (e', \rho)$. Using the inductive hypothesis $(e\,\mathsf{m}a[b])\lfloor\mathcal{X} = ((\rho{\cdot}e)\text{-}\mathcal{X}, \rho[b{:=}\rho(a)])$.
  Now $(\rho{\cdot}e)\text{-}\mathcal{X} = (\rho{\cdot}(e\,\mathsf{m}a[b]))\text{-}\mathcal{X}$ since $\mathsf{m} \in \mathcal{X}$. Also, $\rho{\cdot}e = (\rho[b{:=}\rho(a)]){\cdot}e$ because by assumption in Definition 2.8 $b \notin atoms(e)$ (Definition 2.3). [7]

[6] $\rho{\cdot}e$ is a nominal sequence but it might not be a proto-play because coabstracted atoms need not be distinct. Also e-$\mathcal{X}$ need not be legal because naive deletion does not update links. Proposition 3.4 shows that Definition 3.1 calculates ρ and $\mathcal{X}$ such that if we do these two naive operations *together*, then we are all right.

[7] This is the crux of the proof: because b is fresh, changing ρ to $\rho[b{:=}\rho(a)]$ does not change whatever we have calculated so far.

Lemma 3.5 $enabled(\rho{\cdot}e) = \rho{\cdot}enabled(e)$.
As an immediate corollary, $enabled(\rho{\cdot}e)\text{-}\mathcal{X} = (\rho{\cdot}enabled(e))\text{-}\mathcal{X}$.

We now examine the impact deletion has on the legality conditions of Definition 2.15. Legality is not preserved by arbitrary deletions, but deletion is usually used in a controlled way which ensures preservation. For instance deletion of moves forming an entire sub-tree in the arena, preserves legality properties. Other kinds of deletions can be dealt with similarly.

Lemma 3.6 *Suppose* $\mathcal{X} \subseteq mvs_{\mathfrak{A}}$ *and* $e \in pply_{\mathfrak{A}}$. *Write* $e\restriction\mathcal{X} = (f, \rho)$. *Then* $fa(f) \subseteq fa(e)$ *and* $ca(f) \subseteq \rho{\cdot}ca(e)$.

Lemma 3.7 *Suppose* $e \in pply_{\mathfrak{A}}$. *If* $\mathsf{m}a \in enabled(e)$ *then* $a \in atoms(e)$.

Proof By a routine induction on the proto-play e, using Definition 2.12.

Lemma 3.8 *If* $\mathsf{m}a[b] \subseteq e \in pply_{\mathfrak{A}}$ *and* $\mathsf{m}'b \in enabled(e)$ *then* $\mathsf{m} \vdash_{\mathfrak{A}} \mathsf{m}'$.

Proof By induction on e. We consider one case:

- *The case* $e\,\mathsf{m}a[b]$. Suppose $\mathsf{m}'b \in enabled(e\,\mathsf{m}a[b])$. By assumption in Definition 2.8 $b \notin atoms(e)$ and by Lemma 3.7 $\mathsf{m}'b \notin enabled(e)$. Unpacking Definition 2.12 it follows that $\mathsf{m} \vdash_{\mathfrak{A}} \mathsf{m}'$.

Definition 3.9 Call $\mathcal{X} \subseteq mvs_{\mathfrak{A}}$ **closed under** $\vdash_{\mathfrak{A}}$ when $\mathsf{m} \in \mathcal{X}$ and $\mathsf{m} \vdash_{\mathfrak{A}} \mathsf{m}'$ implies $\mathsf{m}' \in \mathcal{X}$.

Lemma 3.10 *Suppose* $\mathcal{X} \subseteq mvs_{\mathfrak{A}}$ *is closed under* $\vdash_{\mathfrak{A}}$. *Suppose* $e\restriction\mathcal{X} = (e', \rho)$. *Then if* $\mathsf{m}a \in enabled(e)$ *and* $\mathsf{m}a \notin \mathcal{X}$ *then* $\mathsf{m}\rho(a) \in enabled(e')$.

Proof By Lemma 3.5 it suffices to show that if $\mathsf{m}a \in enabled(e)$ then $\mathsf{m}\rho(a) \in enabled((\rho{\cdot}e)\text{-}\mathcal{X})$. We work by induction on e and consider one case:

- *The case of* $e\mathsf{m}'[a]$. Write $e\restriction\mathcal{X} = (e', \rho)$ and suppose $\mathsf{m}'a \in enabled(e\mathsf{m}'[a])$ and $\mathsf{m}' \notin \mathcal{X}$. By Lemma 3.8 $\mathsf{m} \vdash_{\mathfrak{A}} \mathsf{m}'$. Since $\mathsf{m}' \notin \mathcal{X}$ it follows by closure of $\mathcal{X}$ under $\vdash_{\mathfrak{A}}$ that $\mathsf{m} \notin \mathcal{X}$. So $(\rho{\cdot}(e\mathsf{m}'[a]))\text{-}\mathcal{X} = ((\rho{\cdot}e)\text{-}\mathcal{X})\,(\mathsf{m}\rho(a')[\rho(a)])$. By Definition 2.12, $\mathsf{m}'\rho(a) \in enabled(((\rho{\cdot}e)\text{-}\mathcal{X})\,(\mathsf{m}\rho(a')[\rho(a)]))$.

Theorem 3.11 *Suppose* $\mathcal{X} \subseteq mvs_{\mathfrak{A}}$ *and* $e\restriction\mathcal{X} = (f, \rho)$.

(i) *If* $\mathcal{X} \subseteq mvs_{\mathfrak{A}}$ *is closed under* $\vdash_{\mathfrak{A}}$ *then if* e *is justified then so is* f.
(ii) *If* $ini_{\mathfrak{A}} \cap \mathcal{X} = \varnothing$ *then if* e *is well-opened then so is* f.
(iii) *If* e *is strictly scoped then so is* f.
(iv) *If* $\mathcal{X} \subseteq mvs_{\mathfrak{A}}$ *is closed under* $\vdash_{\mathfrak{A}}$ *then if* e *is strictly nested then so is* f.

Proof

(i) Suppose $f'\mathsf{m}\rho(b) \leq f$ where $e\restriction\mathcal{X} = (f, \rho)$ and $\mathsf{m} \notin ini_{\mathfrak{A}}$. Using Proposition 3.4 $f'\mathsf{m}\rho(b) = ((\rho{\cdot}e')\text{-}\mathcal{X})\mathsf{m}\rho(b)$ for some $e'\,\mathsf{m}b \leq e$, and also $\mathsf{m} \notin \mathcal{X}$. Since e is justified, by Lemma 2.13 it must be that $\mathsf{q} \vdash_{\mathfrak{A}} \mathsf{m}$ for some $\mathsf{q}a[b] \subseteq e$.

Since $\mathcal{X}$ is closed under $\vdash_{\mathfrak{A}}$ we know $\mathsf{q} \notin \mathcal{X}$. It follows by Proposition 3.4 that $\mathsf{q}\rho(a)[\rho(b)] \subseteq f'$ and we are done.

(ii) By an easy argument using Proposition 3.4.

(iii) Suppose $\mathsf{a}\rho(a)f' \subseteq f$. Then $\mathsf{a}ae' \subseteq e$ for some $e' \in pply_{\mathfrak{A}}$. Since e is strictly scoped we know that $a \notin fa(e')$. By Lemma 3.6 also $a \notin fa(f')$.

(iv) Much as the previous case.

3.2 Restriction to a hereditarily justified sub-play

The structure of this subsection resembles that of Subsection 3.1. We have a more complex operation than deletion; extracting the *hereditarily justified* sub-pointer sequence. In our framework the definition is absolutely routine; we just take a sub-sequence. This is Definition 3.12; then Proposition 3.14 shows how to quickly calculate the relevant sub-sequence using names, and Theorem 3.18 expresses how properties are preserved.

Definition 3.12 Suppose $e \in pply_{\mathfrak{A}}$ and $A \subseteq \mathbb{A}$. Define the **hereditarily justified** proto-play $e{\restriction}A \subseteq pply_{\mathfrak{A}}$ as follows, where we take $(f, B) = e{\restriction}A$ and $a \in B$ and $a' \notin B$:

$$\begin{aligned} \varepsilon{\restriction}A &= (\varepsilon, A) \\ (e\,\mathsf{m}a[b]){\restriction}A &= (f\,\mathsf{m}a[b], B \cup \{b\}) \\ (e\,\mathsf{m}a'[b]){\restriction}A &= (f, B) \end{aligned}$$

Definition 3.13 Suppose $e \in pply_{\mathfrak{A}}$ and $A \subseteq \mathbb{A}$. Define $e@A$ as follows, where $a \in A$ and $a' \notin A$ (the resemblance with *atoms-concretion* from [12] is deliberate):

$$\begin{aligned} \varepsilon@A &= \varepsilon \\ (e\,\mathsf{m}a[b])@A &= (e@A)\mathsf{m}a[b] \\ (e\,\mathsf{m}a'[b])@A &= e@A \end{aligned}$$

Proposition 3.14 *If $e{\restriction}A = (f, B)$ then $e@B = f$.*

Corollary 3.15 *Suppose $e{\restriction}A = (f, B)$. Then:*

- *If $f'\mathsf{m}a[b] \leq f$ then e' exists such that $e'\mathsf{m}a[b] \leq e$ and $(e'\mathsf{m}a[b])@B = f'\mathsf{m}a[b]$.*
- *If $\mathsf{a}a[b]f' \subseteq f$ then e' exists such that $\mathsf{a}a[b]e' \subseteq e$ and $(\mathsf{a}a[b]e')@B = \mathsf{a}a[b]f'$.*
- *If $\mathsf{q}a[b]f_2\mathsf{q}'b[c]f_3\mathsf{a}b[d] \subseteq f$ then e_2 and e_3 exist such that*

$$(\mathsf{q}a[b]e_2\mathsf{q}'b[c]e_3\mathsf{a}b[d])@B = \mathsf{q}a[b]f_2\mathsf{q}'b[c]f_3\mathsf{a}b[d].$$

Corollary 3.16 *If $e{\restriction}A = (f, B)$ then $\{\mathsf{m}b \in enabled(e) \mid b \in B\} = enabled(f)$.*

Corollary 3.17 *$fa(e@B) = fa(e) \cap B$ and $ca(e@B) = ca(e) \cap B$.*

Theorem 3.18 *Suppose $e{\restriction}A = (f, B)$. Then if e is justified / well-opened / strictly scoped / strictly nested then so is f.*

Proof We consider each property in turn:

- *Justified.* Using part 1 of Corollary 3.15 and Corollary 3.16.
- *Well-opened.* Using part 1 of Corollary 3.15.
- *Strictly scoped.* From part 2 of Corollary 3.15 and Corollary 3.17.
- *Strictly nested.* From part 3 of Corollary 3.15.

4 Combining arenas

Definition 4.1 (i) Suppose f is a function on a set X and g is a function on a disjoint Y. Write $[f, g]$ for the **co-pairing** function on $X \cup Y$ such that $[f, g](x) = f(x)$ and $[f, g](y) = g(y)$ for $x \in X$ and $y \in Y$ respectively.

(ii) Suppose g is a function to $\{\mathsf{O}, \mathsf{P}\}$. Write g^* for the function mapping x to $g(x)^*$ (Definition 2.7).

Definition 4.2 Define **product** $\mathfrak{A} \times \mathfrak{B}$ and **arrow** $\mathfrak{A} \Rightarrow \mathfrak{B}$ of arenas by:

$$\mathfrak{A} \times \mathfrak{B} = (qst_{\mathfrak{A}} + qst_{\mathfrak{B}}, ans_{\mathfrak{A}} + ans_{\mathfrak{B}}, [\lambda_{\mathfrak{A}}, \lambda_{\mathfrak{B}}], \vdash_{\mathfrak{A}} + \vdash_{\mathfrak{B}}, ini_{\mathfrak{A}} + ini_{\mathfrak{B}})$$
$$\mathfrak{A} \Rightarrow \mathfrak{B} = (qst_{\mathfrak{A}} + qst_{\mathfrak{B}}, ans_{\mathfrak{A}} + ans_{\mathfrak{B}}, [\lambda^*_{\mathfrak{A}}, \lambda_{\mathfrak{B}}], \vdash_{\mathfrak{A}} + \vdash_{\mathfrak{B}} + ini_{\mathfrak{B}} \times ini_{\mathfrak{A}}, ini_{\mathfrak{B}})$$

- Above, the symbol $+$ denotes disjoint sets union (for convenience assume sets of moves of distinct arenas are distinct), and
- $ini_{\mathfrak{B}} \times ini_{\mathfrak{A}} = \{(\mathsf{i}', \mathsf{i}) \mid \mathsf{i}' \in ini_{\mathfrak{B}}, \mathsf{i} \in ini_{\mathfrak{A}}\}$. So $\vdash_{\mathfrak{A}} + \vdash_{\mathfrak{B}} + ini_{\mathfrak{B}} \times ini_{\mathfrak{A}}$ is the disjoint union of the enabling relations of $\mathfrak{A}$ and $\mathfrak{B}$, disjoint union $ini_{\mathfrak{B}} \times ini_{\mathfrak{A}}$.

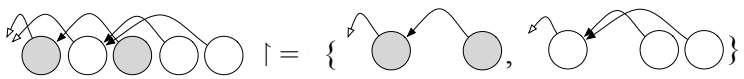
We show how from proto-plays in $\mathfrak{A} \Rightarrow \mathfrak{B}$ we recover proto-plays in $\mathfrak{A}$ and $\mathfrak{B}$. This is Lemmas 4.3 and 4.5. These state that two important operations on proto-plays—deletion and unravelling—preserve certain well-formedness properties which define the notion of HO and GM legal plays. These operations are key to formulating composition of strategies, so preservation of legality is essential to show that composition of HO or GM strategies is well-defined.

Lemma 4.3 *Suppose* $e \in pply_{\mathfrak{A} \Rightarrow \mathfrak{B}}$ *and* $e \restriction mvs_{\mathfrak{A}} = (f, \rho)$. *Then* $f \in pply_{\mathfrak{B}}$. *If* e *is justified, well-opened, strictly scoped, or strictly nested, then so is* f.

Proof For the first part, by Proposition 3.4 f contains only moves in $mvs_{\mathfrak{B}}$. The second part follows by Theorem 3.11 and we note that the enabling relation $\vdash_{\mathfrak{A} \Rightarrow \mathfrak{B}}$ restricted to the moves $mvs_{\mathfrak{B}}$, is just $\vdash_{\mathfrak{B}}$.

Definition 4.4 Define the **unravelling** of $e \in pply_{\mathfrak{A}}$ by $unravel(e) = \{e \lceil \{a\} \mid a \in fa(e)\}$.

Unravelling is key to constructing *exponential* games [23, Sec 2.4]. Intuitively, in a play in $\mathfrak{A} \Rightarrow \mathfrak{B}$ we can recover one play in $\mathfrak{B}$, by deleting the moves of $\mathfrak{A}$. Removing the moves in $\mathfrak{B}$ yields an interleaved *set* of plays of $\mathfrak{A}$. Unravelling separates these plays by following pointers, as illustrated:



It is easy to see that if e is justified then $unravel(e)$ captures the idea of "the set of threads in e", and if e is additionally well-opened then $unravel(e) = \{e\}$.

Lemma 4.5 *If $e \in pply_{\mathfrak{A}\Rightarrow\mathfrak{B}}$ then $unravel(e\restriction\mathfrak{B}) \subseteq pply_{\mathfrak{A}}$. If e is justified / well-opened / strictly-scoped / strictly-nested then so is every $f \in unravel(e\restriction\mathfrak{B})$.*

Proof Directly from Lemma 4.3.

5 Strategies

5.1 Strategies and equivariance

Definition 5.1 Call $\sigma \subseteq pply_A$ **equivariant** when $e \in \sigma$ implies $\pi \cdot e \in \sigma$ for every permutation π. Write $\sigma : \mathfrak{A}$ when σ is an equivariant subset of $pply_{\mathfrak{A}}$ and call σ a **strategy**.

(The notion of strategy is usually subject to further constraints; these are discussed below.)

Recall *deletion* $e\restriction\mathfrak{A}$ from Definition 3.1. We follow [23, Section 2.2.3]:

Definition 5.2 Suppose $\mathfrak{A}$, $\mathfrak{B}$, and $\mathfrak{C}$ are arenas on disjoint moves. Then for strategies $\sigma : \mathfrak{A} \Rightarrow \mathfrak{B}$ and $\tau : \mathfrak{B} \Rightarrow \mathfrak{C}$ define their **interaction** and **composition** by:

$$\sigma||\tau = \{e \in pply_{\mathfrak{A}\times\mathfrak{B}\times\mathfrak{C}} \mid e\restriction\mathfrak{C} \in \sigma \wedge e\restriction\mathfrak{A} \in \tau\}$$
$$\sigma;\tau = \{e\restriction\mathfrak{B} \mid e \in \sigma||\tau\}$$

This is the *linear* version of strategy composition; *exponential* games are constructed using the concept of unravelling introduced earlier (Definition 4.4). The use of proto-plays, which have almost no structure, simplifies the definition of interaction $(-||-)$ compared to the usual definition (c.f. [23, Section 2.2.3]) which needs the auxiliary concept of *interaction sequences*.

Remark 5.3 Equivariance is symmetry under permuting atoms. Names fulfil the function that links fulfil in e.g. [19,23,14]. Permutative symmetry of strategies amounts to saying 'we can α-rename'.

So proto-plays do not have α-equivalence in our framework but *sets* of proto-plays do (cf. [10]). Thus, Theorem 5.4 becomes a one-line argument by symmetry/equivariance. This avoids arguments about α-renaming, reindexing, or re-linking that would be needed if we used numbers or explicitly linked lists. So we have:

Theorem 5.4 *Suppose $\mathfrak{A}$, $\mathfrak{B}$, and $\mathfrak{C}$ are arenas and $\sigma : \mathfrak{A} \Rightarrow \mathfrak{B}$ and $\tau : \mathfrak{B} \Rightarrow \mathfrak{C}$ are strategies. Then the set $\sigma;\tau$ is equivariant, and thus is a strategy in $\mathfrak{A} \Rightarrow \mathfrak{C}$.*

Proof By Definition 5.1 a strategy is an equivariant set of protoplays, so σ and τ are equivariant. We note that the definitions involved in specifying $\sigma;\tau$ are all

symmetric in atoms, and so by assumption are the inputs to those definitions σ and τ, therefore by the *principle of equivariance* also $\sigma;\tau$ are symmetric. (A formal discussion of equivariance is elsewhere [7, Subsection 4.2].)

5.2 *Associativity of composition*

We will prove Theorem 5.5, that composition of strategies is associative:

Theorem 5.5 *Suppose $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}$, and $\mathfrak{D}$ are arenas on disjoint moves. Suppose $\sigma : \mathfrak{A} \Rightarrow \mathfrak{B}$, $\tau : \mathfrak{B} \Rightarrow \mathfrak{C}$, and $\mu : \mathfrak{C} \Rightarrow \mathfrak{D}$ are strategies. Then $(\sigma;\tau);\mu = \sigma;(\tau;\mu)$.*

This will follow immediately from Lemma 5.8. For us in this paper strategies are just sets of sequences of moves and names, and the proofs are just by routine induction and *name-chasing*, that is: unpacking definitions and noting that names end up in the same places on both sides of the equality (see the proof of Proposition 5.6).

Proposition 5.6 *Suppose $\mathfrak{C}$ is an arena and $\mathcal{X}, \mathcal{Y} \subseteq mvs_{\mathfrak{C}}$ are two disjoint sets of moves, and $e \in pply_{\mathfrak{C}}$. Suppose $e\lfloor\mathcal{X} = (e', \rho)$ and $e'\lfloor\mathcal{Y} = (e'', \rho')$ and $e\lfloor(\mathcal{X}\cup\mathcal{Y}) = (f, \rho'')$.*

Then $f = e''$ and $\rho'' = \rho' \circ \rho$ (where $\circ$ is functional composition, notation from Definition 2.5).

Proof By induction on e using Definition 3.1.

- *The case* $\varepsilon\ldots$ is easy.
- *The case* $e\,\mathsf{m}a[b]$ *where* $\mathsf{m} \in \mathcal{X}$. By Proposition 3.4 $e' = \rho{\cdot}e\text{-}\mathcal{X}$ and $e'' = \rho'{\cdot}e'\text{-}\mathcal{Y}$. It follows that $e'' = (\rho' \circ \rho){\cdot}e\text{-}(\mathcal{X}\cup\mathcal{Y})$. Also by Proposition 3.4 $f = \rho''{\cdot}e\text{-}(\mathcal{X}\cup\mathcal{Y})$. By inductive hypothesis $f = e''$ and $\rho'' = \rho' \circ \rho$.

  By Definition 3.1 we have that $e\mathsf{m}a[b]\lfloor\mathcal{X} = (e', \rho[b{:=}\rho(a)])$ and it follows that $\pi_1(e\mathsf{m}a[b]\lfloor\mathcal{X})\lfloor\mathcal{Y} = (e'', \rho' \circ (\rho[b{:=}\rho(a)]))$. Also, it is a fact of functions that $\rho' \circ (\rho[b{:=}\rho(a)]) = (\rho' \circ \rho)[b{:=}(\rho' \circ \rho)(a)]$.

  Using Definition 3.1 we have that $e\mathsf{m}a[b]\lfloor(\mathcal{X}\cup\mathcal{Y}) = (f, \rho''[b{:=}\rho''(a)])$, and the result follows.
- *The case* $e\,\mathsf{m}a[b]$ *where* $\mathsf{m} \in \mathcal{Y}$. Similar to the previous case, but simpler.

Remark 5.7 Our notion of strategy is simple and it does not rely on a notion of *legal* play. So to study properties of composition, any strategy over an arena $\mathfrak{A} \Rightarrow \mathfrak{B}$ is also a strategy over an arena $\mathfrak{A} \times \mathfrak{B}$, as the two arenas have the same sets of moves—the polarities of the moves and the justification structure are different between $\mathfrak{A} \Rightarrow \mathfrak{B}$ and $\mathfrak{A} \times \mathfrak{B}$, but this information is not used in the definition of composition.

Similarly for defining $\sigma||\tau$ for strategies $\sigma : \mathfrak{A} \Rightarrow \mathfrak{B}$ and $\tau : \mathfrak{B} \Rightarrow \mathfrak{C}$. An interaction can be viewed as a strategy in $(\mathfrak{A} \times \mathfrak{B}) \Rightarrow \mathfrak{C}$ or $\mathfrak{A} \Rightarrow (\mathfrak{B} \times \mathfrak{C})$, as convenient. This is correct because interaction preserves equivariance and the sets of moves in these arenas are the same.

Thus we will obtain a particularly simple proof of associativity of interaction, given below.

Lemma 5.8 *Suppose* $\sigma : \mathfrak{A} \Rightarrow \mathfrak{B}$, $\tau : \mathfrak{B} \Rightarrow \mathfrak{C}$, *and* $\mu : \mathfrak{C} \Rightarrow \mathfrak{D}$ *are strategies. Then* $(\sigma||\tau)||\mu = \sigma||(\tau||\mu)$.

Proof We unpack Definition 5.2 repeatedly: $e \in (\sigma||\tau)||\mu$ if and only if $e\lfloor \mathfrak{D} \in \sigma||\tau$ and $e\lfloor \mathfrak{A}\times\mathfrak{B} \in \mu$, if and only if $(e\lfloor \mathfrak{D})\lfloor \mathfrak{C} \in \sigma$ and $(e\lfloor \mathfrak{D})\lfloor \mathfrak{A} \in \tau$ and $e\lfloor \mathfrak{A}\times\mathfrak{B} \in \mu$. Using Proposition 5.6 this is equivalent to $e\lfloor \mathfrak{D}\times\mathfrak{C} \in \sigma$ and $e\lfloor \mathfrak{D}\times\mathfrak{A} \in \tau$ and $e\lfloor \mathfrak{A}\times\mathfrak{B} \in \mu$.

By similar reasoning, $e' \in \sigma||(\tau||\mu)$ is equivalent to $e'\lfloor \mathfrak{D}\times\mathfrak{C} \in \sigma$ and $e'\lfloor \mathfrak{D}\times\mathfrak{A} \in \tau$ and $e'\lfloor \mathfrak{A}\times\mathfrak{B} \in \mu$.

The result follows.

5.3 *Prefix- and opponent-closed*

Just as for proto-plays, GM and HO strategies are subject to constraints. In the rest of this section we sketch, sometimes in detail, how these can be expressed.

Two standard conditions on strategies are being *prefix-closed* and *opponent-closed*; see [19, Section 5] (where opponent-closed is called *contingent completeness*) or [14, Definition 4]. These are straightforward to formalise:

Definition 5.9 Call $\sigma \subseteq pply_{\mathfrak{A}}$ **prefix-closed** and **opponent-closed** respectively when:

$$\frac{e\,\mathsf{m}a[b] \in \sigma}{e \in \sigma} \qquad \frac{e \in \sigma \quad \lambda_{\mathfrak{A}}(\mathsf{m})=\mathsf{O} \quad e\,\mathsf{m}a[b] \in pply_{\mathfrak{A}}}{e\,\mathsf{m}a[b] \in \sigma}$$

5.4 *The asynchrony pre-order on proto-plays*

In [14] the authors were interested in modelling asynchronous concurrency. Accordingly strategies must be *saturated* under certain move swapping [14, Subsection 2.5] (the idea goes back to [25]).

Definition 5.10 Call a relation $\leq$ on sequences **compatible** when $e \leq e'$ implies $ef \leq e'f$ and $fe \leq fe'$. Define $\preceq$ on $pply_{\mathfrak{A}}$ to be the least compatible pre-order such that:

$$\frac{(b \notin fa(e),\ \lambda_{\mathfrak{A}}(\mathsf{m})=\mathsf{O})}{\mathsf{m}a[b]\ e \preceq e\ \mathsf{m}a[b]}\ (\mathrm{bmX}) \qquad \frac{(b \notin fa(e),\ \lambda_{\mathfrak{A}}(\mathsf{m})=\mathsf{P})}{e\ \mathsf{m}a[b] \preceq \mathsf{m}a[b]\ e}\ (\mathrm{bXm})$$

Call $\sigma \subseteq pply_{\mathfrak{A}}$ **$\preceq$-saturated** when $e \in \sigma$, $e' \in pply_{\mathfrak{A}}$ and $e' \preceq e$ imply $e' \in \sigma$.

Remark 5.11 It may be worth quoting the definition from [14] (text just before Definition 6) for comparison with Definition 5.10:

> *...we define a pre-order* $\preceq$ *on* $play_A$ *for any arena* $\mathfrak{A}$ *as the least reflexive and transitive relation satisfying* $s' \preceq s$ *for all* $s, s' \in play_A$
> (i) $s' = s_0 \cdot \mathsf{o} \cdot s_1 \cdot s_2$ *and* $s = s_0 \cdot s_1 \cdot \mathsf{o} \cdot s_2$, *or*
> (ii) $s' = s_0 \cdot s_1 \cdot \mathsf{p} \cdot s_2$ *and* $s = s_0 \cdot \mathsf{p} \cdot s_1 \cdot s_2$,

where o *is any O move and* p *is any P move and the justification pointers in s are "inherited" from* s' *...*

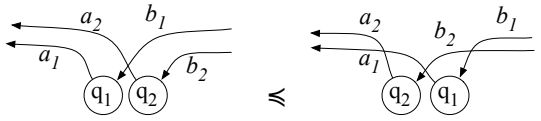
Nominal sequences help make these intuitions formal.

In [14, Lemma 7] a small step version $\preceq'$ is given and the equality $\preceq'{=}\preceq$ is claimed. With what we have so far, this is a routine inductive argument:

Definition 5.12 Give $\{\mathsf{O},\mathsf{P}\}$ a partial order such that $\mathsf{O}\leq\mathsf{O}$, $\mathsf{O}\leq\mathsf{P}$, and $\mathsf{P}\leq\mathsf{P}$. Define a pre-order $\preceq'$ on closed sequences to be the least reflexive transitive relation such that:

$$\frac{(\lambda_{\mathfrak{A}}(\mathsf{m}_1)\leq\lambda_{\mathfrak{A}}(\mathsf{m}_2))}{\mathsf{m}_1a_1[b_1]\ \mathsf{m}_2a_2[b_2]\preceq'\mathsf{m}_2a_2[b_2]\ \mathsf{m}_1a_1[b_1]}\ (\mathrm{smm}) \qquad \frac{(\lambda_{\mathfrak{A}}(\mathsf{m}_1)\leq\lambda_{\mathfrak{A}}(\mathsf{m}_2))}{\mathsf{m}_1a[b_1]\ \mathsf{m}_2a[b_2]\preceq'\mathsf{m}_2a[b_2]\ \mathsf{m}_1a[b_1]}\ (\mathrm{smm}')$$

Here is the asynchronous swapping rule (smm) interpreted for q_1, q_2:



Lemma 5.13 $\preceq'{=}\preceq$.

Proof We show $\preceq'{\subseteq}\preceq$ by induction on $\preceq'$:

- *Rule* (smm). By (bmX) if $\lambda_{\mathfrak{A}}(\mathsf{m}_1){=}\mathsf{O}$ and by (bXm) if $\lambda_{\mathfrak{A}}(\mathsf{m}_1){=}\mathsf{P}{=}\lambda_{\mathfrak{A}}(\mathsf{m}_2)$.
- *Rule* (smm'). By (bmX) if $\lambda_{\mathfrak{A}}(\mathsf{m}_1){=}\mathsf{O}$ and by (bXm) if $\lambda_{\mathfrak{A}}(\mathsf{m}_1){=}\mathsf{P}{=}\lambda_{\mathfrak{A}}(\mathsf{m}_2)$.

In both cases the side-condition $b_2\notin fa(\mathsf{a}_1a_1)$ is valid. Next we show that $\preceq{\subseteq}\preceq'$ by induction on $\preceq$ and the length of e:

- *Rule* (bmX). We use (smm) and (smm') to swap $\mathsf{m}a[b]$ with the leftmost move in e. The condition $b{\notin}fa(e)$ matches the distinctness condition $b_2{\notin}\{a,a_1,b_1\}$.
- *Rule* (bXm). We use (smm) and (smm') to swap $\mathsf{m}a[b]$ with the rightmost move in e.

5.5 *Innocence*

An important notion in HO games is *innocence* [19, Definition 5.2], which characterises side-effect-free sequential computation. For us this is Definition 5.14 and with the tools we have built so far, it is quite compact: [8]

Definition 5.14 Suppose m and m' are named moves (Definition 2.9). Given HO-legal plays emm', $e'm$ in $\mathfrak{A}$, where $|emm'|$ has even length, $ca(m')\cap atoms(e')=\emptyset$ and $\ulcorner em\urcorner=\ulcorner e'm\urcorner$, there is a unique renaming $\rho=(c\mapsto c')$ with $c\in fa(m')$ and $c'\in ca(e')$ such that $\ulcorner emm'\urcorner=\ulcorner e'm(\rho{\cdot}m')\urcorner$. Call $\sigma:\mathfrak{A}$ **innocent** when

$$emm',e'\in\sigma\wedge e'm\ \text{HO-legal}\ \wedge\ulcorner em\urcorner=\ulcorner e'm\urcorner\implies e'm(\rho{\cdot}m')\in\sigma.$$

[8] We use McCusker's equivalent formulation [23, Subsection 2.2.4].

In [19, Definition 5.2] Hyland and Ong must write in English about the manipulation of pointers, and that this has to be done throughout their work (and this is typical of similar papers). We propose nominal techniques as a way to deal with this quickly and elegantly.

In HO games, moves can be repeated, which leads to a need to identify particular occurrences of moves in sequences. This goes away in our setting because every question or answer is uniquely identified by a name: the coabstracted name that it introduces. So implicit in our framework is a separation of 'move' versus 'occurrence', removing a significant source of ambiguity. In this paper we have been able to implicitly identify an occurrence of a move in a proto-play with the named move in which the occurrence appears, since coabstracted atoms in proto-plays are distinct (Definition 2.8).

6 Conclusions

We have seen how pointer sequences can be modelled as nominal sequences. Pointers are split into a coabstraction $[b]$ corresponding to the head of the arrow, and a (free) atom b corresponding to its tail. Unlike pointers, a name carries its identity with it; b points to $[b]$ wherever we put it.

Furthermore, unlike e.g. numbers, a name is permutatively symmetric, so reindexing / renaming can be expressed at a high level of abstraction. Because of this, nominal sequences are easy to break apart, compose, and reindex.

We have considered some non-trivial operations, like deletion and hereditarily justified sub-sequences; and some important definitions, like strategy composition, asynchronous reordering, and innocence. We have seen how these operations and definitions become straightforward and precise, if we choose the right machinery. This is attractive, but we also believe it will be almost a prerequisite for the kind of mechanised treatment of game semantics that is required for games to be applied in the second author's research programme.

We have discussed pointer sequences [19,14,23]. In contrast, the *Abramsky-Jagadeesan-Malacaria* (AJM) games [2] rely on tags instead of pointers. These do not raise the problems of pointers and are fully formalised, but they are a more restricted formalism which was only used for PCF. For languages with effects the flexibility of pointers was required.

Another strategy is to become more abstract: so [5,17,24,21] revise the whole game semantic paradigm *per se*, in categorical terms. Some readers will instinctively believe that this categorical generalisation obsoletes any concrete realisation, but this is incorrect; there will always be a need for concrete models—especially if we want to implement or mechanise theorems. We seek convenient reformulations of the impressive collection of existing game models to make them more suitable for our intended applications. The work cited above is complementary, but also orthogonal.

Representations of pointer games [18] and games models of nominal languages [20]

exist, including work by the second author with others on game semantics for nominal or nominal-related languages [15,1]. However, there has been no *nominal* representation of pointer sequences themselves. The closest the literature gets is in the Introduction to [24] where Melliès discusses representing pointers using integer indexes acted on by two group actions.

There is more to this paper than representing pointers. We use *atoms* in FM sets, which have structure that ZF sets do not. Functions, predicates, and subsets have symmetry (equivariance) properties and apartness (freshness) structure which make it relatively more convenient to handle distinctness conditions (like in Definition 2.8) or to deduce symmetry properties (as in Theorem 5.4), and so on (a very general treatment is in [6, Section 5]).

In this paper, coabstraction is a syntactic token in sequences. We give a denotational intuition how this differs from nominal atoms-abstraction: suppose X is a nominal set with an internal atoms-abstraction $[\mathbb{A}]\mathsf{X} \to \mathsf{X}$ written $[a]x$ (for definitions see [12,7]). Suppose $\mathcal{R} \subseteq \mathsf{X} \times \mathsf{X}$ is a relation on X. Then (briefly) $\mathcal{R}[\mathbb{A}]$ is the least relation such that if $x\ \mathcal{R}\ y$ and $a\#x, \mathcal{R}$ then $x\ \mathcal{R}\ [a]y$, and $\mathcal{R}[a]$ is the least relation such that if $x\ \mathcal{R}\ y$ and $a\#x, \mathcal{R}$ then $x\ \mathcal{R}\ [a]y$. This is coabstraction. Nominal terms admit a similar generalisation; we would admit freshness $a\#X$ and *cofreshness* $a\%X$ conditions. More on this in a later paper.

We can read this paper as an exciting, if only partially articulated, commentary on semantics. The issue of dangling pointers and compositionality has *not* been properly addressed in the games literature and it remains to understand where the nominal model will take us. The nominal model of this paper exists in a larger context of nominal sets, substitution models, and some sophisticated logical and semantic theory [7,9], including abstract treatments of metavariables and renaming [8,11] and even e.g. trees with pointers [4]; the fruit of applying this theory, remains to be discovered.

Acknowledgements. The first author acknowledges the support of the Leverhulme trust.

References

[1] Samson Abramsky, Dan R. Ghica, Andrzej S. Murawski, C.-H. Luke Ong, and Ian D. B. Stark. Nominal games and full abstraction for the nu-calculus. In *Proceedings of the 19th IEEE Symposium on Logic in Computer Science (LICS 2004)*, pages 150–159. IEEE Computer Society Press, 2004.

[2] Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Information and Computation*, 163(2):409–470, 2000.

[3] Samson Abramsky and Guy McCusker. Call-by-value games. In *Proceedings of Computer Science Logic (CSL'97)*, volume 1414 of *Lecture Notes in Computer Science*, pages 1–17, 1998.

[4] Luca Cardelli, Philippa Gardner, and Giorgio Ghelli. Manipulating trees with hidden labels. In *Proceedings of the 6th international conference on Foundations of software science and computational structures (FoSSaCS 2003)*, pages 216–232, 2003.

[5] Vincent Danos and Russell Harmer. The anatomy of innocence. In *Proceedings of the 10th EACSL Annual Conference on Computer Science Logic (CSL 2001)*, pages 188–202, 2001.

[6] Murdoch J. Gabbay. A General Mathematics of Names. *Information and Computation*, 205(7):982–1011, July 2007.

[7] Murdoch J. Gabbay. Foundations of nominal techniques: logic and semantics of variables in abstract syntax. *Bulletin of Symbolic Logic*, 17(2):161–229, 2011.

[8] Murdoch J. Gabbay. Meta-variables as infinite lists in nominal terms unification and rewriting. *Logic Journal of the IGPL*, 2012.

[9] Murdoch J. Gabbay. Nominal terms and nominal logics: from foundations to meta-mathematics. In *Handbook of Philosophical Logic*, volume 17. Kluwer, 2012.

[10] Murdoch J. Gabbay and Vincenzo Ciancia. Freshness and name-restriction in sets of traces with names. In *Foundations of software science and computation structures, 14th International Conference (FOSSACS 2011)*, volume 6604 of *Lecture Notes in Computer Science*, pages 365–380. Springer, 2011.

[11] Murdoch J. Gabbay and Martin Hofmann. Nominal renaming sets. In *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2008)*, pages 158–173. Springer, November 2008.

[12] Murdoch J. Gabbay and Andrew M. Pitts. A New Approach to Abstract Syntax with Variable Binding. *Formal Aspects of Computing*, 13(3–5):341–363, July 2001.

[13] Dan R. Ghica. Applications of game semantics: From software analysis to hardware synthesis. In *Proceedings of the 24th IEEE Symposium on Logic in Computer Science (LICS 2009)*, pages 17–26, 2009.

[14] Dan R. Ghica and Andrzej Murawski. Angelic semantics of fine-grained concurrency. *Annals of Pure and Applied Logic*, 151(2-3):89–114, 2008.

[15] Dan R. Ghica and Andrzej S. Murawski. Compositional model extraction for higher-order concurrent programs. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2006)*, pages 303–317, 2006.

[16] Dan R. Ghica and Nikos Tzevelekos. A system-level semantics. *Electronic Notes in Theoretical Computer Science (to appear)*. Mathematical Foundations of Computer Science XXVIII, Bath, UK, 2012.

[17] Russell Harmer, Martin Hyland, and Paul-André Melliès. Categorical combinatorics for innocent strategies. In *Proceedings of the 22nd IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 379–388. IEEE Computer Society, 2007.

[18] Martin Hyland and C.-H. Luke Ong. Pi-calculus, dialogue games and PCF. In *FPCA*, pages 96–107, 1995.

[19] Martin Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. *Information and Computation*, 163(2):285–408, 2000.

[20] Jim Laird. A game semantics of names and pointers. *Annals of Pure and Applied Logic*, 151(2-3):151–169, February 2008. First Games for Logic and Programming Languages Workshop.

[21] Jim Laird, Giulio Manzonetto, and Guy McCusker. Constructing differential categories and deconstructing categories of games. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP (2)*, volume 6756 of *Lecture Notes in Computer Science*, pages 186–197. Springer, 2011.

[22] Ondrej Majer, Ahti-Veikko Pietarinen, and Tero Tulenheimo. *Games: Unifying Logic, Language, and Philosophy*. Springer, 2009.

[23] Guy McCusker. Games and full abstraction for FPC. *Information and Computation*, 160(1-2):1–61, 2000.

[24] Paul-André Melliès. Asynchronous games 2: The true concurrency of innocence. In *Proceedings of the 15th International Conference on Concurrency Theory (CONCUR 2004)*, pages 448–465, 2004.

[25] Jan Tijmen Udding. A formal model for defining and classifying delay-insensitive circuits and systems. *Distributed Computing*, 1(4):197–204, 1986.

[26] Christian Urban. Nominal reasoning techniques in Isabelle/HOL. *Journal of Automatic Reasoning*, 40(4):327–356, 2008.